PRENTICE HALL

# Joomla!™ 1.6: A User's Guide

## BUILDING A SUCCESSFUL JOOMLA! POWERED WEBSITE

THIRD EDITION

BARRIE M. NORTH

## Praise for Previous Edition of *Joomla!: A User's Guide*

"A complete guide to the powerful features of Joomla! 1.5, this book takes a holistic approach to building a Joomla!-powered website—from the CMS itself to its many extensions, search engine optimization, and even building your own tableless template. The novice reader is eased into the subject and confidently guided through the basic principles and on to the more advanced features. This guide empowers the user not only to build a professional website but to also to make it a success."

—Russell Walker, CEO, Netshine Software Limited
(Joomla! Development Consultancy)

"If you've been using or following Joomla! in the past years, you've most likely seen the name Barrie North or Joomlashack. Barrie has been a member of the community for a long time and, as such, my expectations for this book were pretty high. Besides explaining how Joomla! works from a usability point of view, there is valuable information for people who want to learn serious template building, and readers can stand out of the crowd by using Barrie's steps to make their (X)HTML and CSS optimized for accessibility and SEO. All in all, this book is a great guide that comes at the right time for newcomers and more experienced Joomla! users and developers alike. Well done, Barrie!"

—Arno Zijlstra, Joomla! cofounder, custom template specialist,
www.alvaana.com

"In a time when solid, real-life Joomla! 1.5 information is rarely available, this book is a thirst-quenching oasis of knowledge. The abundant and clear examples in the book make Joomla! 1.5 websites within anyone's reach. I heartily recommend *Joomla! 1.5: A User's Guide* by Barrie North."

—Tom Canavan, author of *Dodging the Bullets: A Disaster Preparation Guide for Joomla! Based Web Sites*

"Refreshing! After reading many how-to books, this one is a step beyond the rest because of its focus on examples based on live sites. This book is well crafted for beginners to advanced users with a well-organized overview that walks you through the entire Joomla! CMS."

—Steven Pignataro, corePHP, www.corephp.com

"As a long-time Joomla! end-user and developer, I had low expectations for anything new I might learn from this book. However, I was pleasantly surprised to find it a great refresher course, especially since the book is logically organized, leading beginners from the most basic Joomla! concepts and continuing through to more complex ones, such as tableless template design and how to write a template for Joomla! 1.5. In summary, Barrie North has produced the gold-standard print reference for Joomla! 1.5. I highly recommend this book for novice and intermediate users if you want to make the most of Joomla!"

—Vicor Drover, http://dev.anything-digital.com

# Joomla!™ 1.6: A User's Guide

## Building a Successful Joomla! Powered Website

**Barrie M. North**


Prentice Hall
Upper Saddle River, NJ · Boston · Indianapolis · San Francisco
New York · Toronto · Montreal · London · Munich · Paris · Madrid
Cape Town · Sydney · Tokyo · Singapore · Mexico City

*Library of Congress Cataloging-in-Publication Data:*

North, Barrie M.
Joomla! 1.6 : a user's guide : building a successful Joomla! powered website / Barrie M. North.
p. cm.
ISBN 978-0-13-248706-1 (pbk. : alk. paper)
1. Joomla! (Computer file) 2. Web sites--Authoring programs. 3. Web site development.
I. Title.
TK5105.8885.J86N67 2011
006.7'8--dc22

2010051011



ISBN-13: 978-0-132-48706-1
ISBN-10: 0-132-48706-3

Text printed in the United States on recycled paper at RR Donnelley in Crawfordsville, IN.

First printing February 2011

**Editor-in-Chief**
Mark Taub

**Executive Editor**
Debra Williams Cauley

**Development Editor**
Songlin Qiu

**Marketing Manager**
Stephane Nakib

**Managing Editor**
Kristy Hart

**Project Editor**
Anne Goebel

**Copy Editor**
Geneil Breeze

**Indexer**
Heather McNeill

**Proofreader**
Kathy Ruiz

**Technical Reviewers**
Robert P. J. Day
Torah Bontrager

**Publishing Coordinator**
Kim Boedigheimer

**Cover Designer**
Chuti Prasertsith

**Compositor**
Nonie Ratcliff

*For Sarah*

# Contents

# 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—version 1.6. This release is an important update that includes some key new features such as better Access Control Levels (ACL).

## This Book's Target Audience

This book primarily targets people using Joomla 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. You can also use the book as a reference. If you need some quick ideas of what newsletter extensions are available, for example, head to Chapter 6, "Extending Joomla!" Finally, the appendixes contain valuable information about various aspects of Joomla.

### 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 that do. The idea of powering websites with a CMS has been around for some time, but only recently with the advent of high-quality open source CMS scripts like Joomla have we seen these powerful CMS tools coming into the hands of 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 "Joomlasphere," 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 find and 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 going 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 all the content in your site. It does this through content articles. These discrete pieces of content must be organized into a hierarchy of categories.

This chapter provides an in-depth tutorial that explains 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 them 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 also determine the layout of what a dynamic page looks like 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 this 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, plug-ins, and languages. Many hundreds of extensions are available both 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 backend. Part of the attraction of Joomla is the ability to easily add and edit content through a What You See Is What You Get (WYSIWYG) editor.

In this chapter, we look 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 also discuss a more holistic marketing plan including 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 use of tables. This is a desirable goal as it means that the template code is easier to validate to World Wide Web Consortium (W3C) standards. It also tends to load faster, be easier to maintain, and perform better in search engines. 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!

This chapter looks at the entire process of creating a small business website, in this case a restaurant website, from scratch. Starting from an analysis of needs, this chapter shows you 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 as a way to shape perception of who they are and what they do. Chances are, if you go to a company's website today, you will find a link to its blog somewhere on the site. What is becoming more common on websites now, is a section of the site that is 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 from scratch using Joomla.

### Appendix A: Getting Help

Stuck with Joomla? A tremendous amount of information is available on the Web, as well as many active communities to ask for help.

### Appendix B: A Guide to Joomla! 1.6 ACL

Access Control Levels dictate what users can perform what tasks in your Joomla website. This brief guide helps you understand how ACL has been changed and improved in Joomla 1.6.

### Appendix C: A Quick Introduction to SEO

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

### Appendix D: Installing WampServer

This appendix provides 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 separate content from its presentation. Its main features are the ease of creation and editing of content and dynamic web pages. CMSs are usually 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 the responsibilities involved in developing a website. A web designer can be concerned with the design, and nontechnical 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 usually can find more details about the tip at compassdesigns.net.

**NOTE**
The note boxes denote cautions about an aspect of the topic. They are not applicable to all situations, but you should check whether a note applies to your site.

**THE LEAST YOU NEED TO KNOW**
Explanations of key critical concepts can be found in the Least You Need to Know boxes. These are worth circling in a big red pen or writing out for yourself on a cheat sheet.

**CAUTION**
Cautions provide critical information.

## 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 two, I've kept the exclamation point in heads but dropped it in the 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 maintenance releases with slight changes can often 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, chances are it is because of these minor updates. To stay informed of recent changes to Joomla, consult the forum at www.joomlabook.com where you can find discussions of Joomla versions.

# Acknowledgments

Without the continuing support of my wife, Sarah, this book would not have been possible. Sarah let me frequently slip off to work on the manuscript. Part of my thanks also goes to the three boys who (mostly) managed not to bug me while I was writing.

I'd also like to thank the third-party developers I frequently annoyed on Skype with questions about this or that.

Finally, many thanks to the guys who live on the trunk—the many developers who selflessly contribute code to the Joomla project on a daily basis.

# About the Author

**Barrie M. North** has more than 20 years of experience with the Internet as a user, designer, and teacher. He has spent more than 8 years in the education field, becoming steadily more involved in web technology, teaching web design classes to students and technology integration to teachers. Most recently, he worked as an IT consultant for two new schools pioneering the use of technology. As well as web design, he has provided web marketing/SEO, usability, and standards compliance expertise to his clients.

He is a founder of Joomlashack.com, one of the oldest and most popular Joomla template providers, and SimplWeb.com, a service that provides easy-to-use, turnkey Joomla hosting for those new to Joomla. He also maintains a blog about all things Joomla at CompassDesigns.net. When not working, he can frequently be found on the Joomla community boards, and he has written many free tutorials for using Joomla. His combination of Joomla expertise, educational skills, and engaging writing has produced a book accessible to everyone. Barrie lives in South Strafford, Vermont.

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 chapter. This chapter 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 1.6 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 you have the skills needed to make a Joomla template, I strongly advise grabbing a free template from compassdesigns.net. A good way to learn is to grab one of my 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.

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 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 (ttp://net2.com/nvu/download.html). You can even download the source code and make special changes, if you want.

**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.

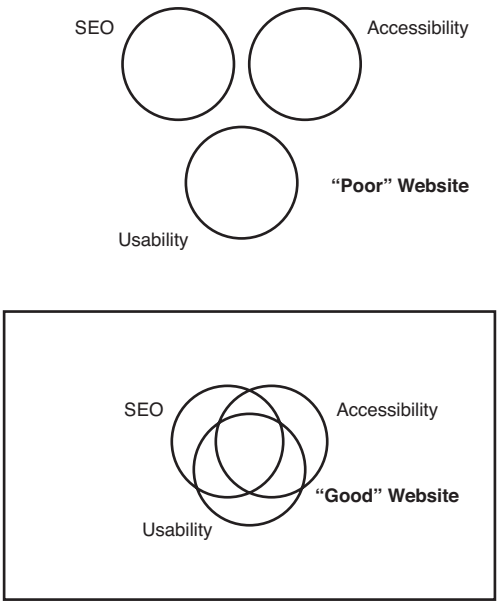


FIGURE 9.2 The overlap between usability, accessibility, and SEO.

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 introduced new features, and the manufacturers came up with different, sometimes proprietary, tags (names) for those features. Each browser tends 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. Consequently, web designers must design their websites to support older browsers rather than new ones. Designers and website owners often decide that it's important that a web page appear properly in these "legacy" browsers. The W3C standards outlined for web page code were 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. 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: 960TemplateTutorialStep1

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, the `index.php`:

`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 for a template called Element:

```
/element/template_thumbnail.png
/element/params.ini
/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 Example 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. |
| `/element/params.ini` | A text file that would store the values of any parameters the template has. |
| `/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 want. 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 back-end 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, 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"?>
<!DOCTYPE install PUBLIC "-//Joomla! 1.6//DTD template 1.0//EN"
➥"http://www.joomla.org/xml/dtd/1.6/template-install.dtd">
<install version="1.6" type="template">
    <name>960TemplateTutorialStep1</name>
    <creationDate>1/10/10</creationDate>
    <author>Barrie North</author>
    <authorEmail>contact@compassdesigns.net</authorEmail>
    <authorUrl>http://www.compassdesigns.net</authorUrl>
    <copyright>Copyright (C) 2005 - 2010 Barrie North</copyright>
    <license>GPL</license>
    <version>1.6.0</version>
    <description>The first of 4 tutorial templates from
➥Joomla 1.6 - A User's Guide</description>
    <files>
        <filename>index.php</filename>
        <filename>templateDetails.xml</filename>
        <filename>params.ini</filename>
        <folder>images</folder>
        <folder>css</folder>
    </files>
    <positions>
        <position>breadcrumbs</position>
        <position>left</position>
        <position>right</position>
        <position>top</position>
        <position>footer</position>
        <position>debug</position>
    </positions>
```

```
    <config>
        <fields name="params">
            <fieldset name="basic">
                <field
                    name="colorVariation"
                    type="list"
                    default="white"
                    label="Color Variation"
                    description="Base Color of template">
                    <option
                        value="blue">blue</option>
                    <option
                        value="red">red</option>
                </field>
            </fieldset>
        </fields>
    </config>
</install>
```

Let's look at what some of these lines mean:

- **`<install version="1.6" 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 1.6.
- **`<name>960TemplateTutorialStep1 </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>`**—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>`**—This is the name of the author of this template—most likely your name.
- **`<copyright>`**—Any copyright information goes in this element.

- **<authorEmail>**—This is the email address at which the author of this template can be reached.
- **<authorUrl>**—This is the URL of the author's website.
- **<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>params.ini</filename>
    <folder>images</folder>
    <folder>css</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>**—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.
- **<config>**—This section describes the parameters that can be set in the back-end 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, if you haven't already you should create the template folder in `/templates/960TemplateTutorialStep1` and then create the `index.php` and `templateDetails.xml` files in that folder.

As we have added the files directly, you need to run the Discovery process to install them. Select Extensions>Extension Manager>Discover.

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 top of every web page. At the top of our page, put this in the template:

```
<?php
/**
 * @copyright    Copyright (C) 2005 - 2010 Barrie North.
 * @license        GPL
 */

defined('_JEXEC') or die;
?>
<!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 shows the copyright/license and 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 To 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 6 or less 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
/**
 * @copyright    Copyright (C) 2005 - 2010 Barrie North.
 * @license        GPL
 */
defined('_JEXEC') or die;
$app = JFactory::getApplication();
?>
<!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="<?php echo $this->baseurl
➥?>/templates/system/css/system.css" type="text/css" />
<link rel="stylesheet" href="<?php echo $this->baseurl
➥?>/templates/system/css/general.css" type="text/css" />
<link rel="stylesheet" href="<?php echo $this->baseurl ?>/templates/<?php
➥echo $this->template ?>/css/template.css" type="text/css" />
</head>
```

What does all this mean?

We 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.

`$app = Jfactory::getApplication();` is a variable that allows you to grab various parameters, like the name of the site and use them in the template.

The next line is for including more header information:

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

This code snippet inserts in the generated page (that is, your frontend) all 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="rights" content="" />
  <meta name="language" content="en-GB" />
  <meta name="description" content="Joomla! -
➥the dynamic portal engine and content management system" />
  <meta name="generator" content="Joomla! 1.6 -
➥Open Source Content Management" />
  <title>Home</title>
  <link href="/Joomla_1.6/index.php?format=feed&amp;type=rss" rel="alternate"
➥type="application/rss+xml" title="RSS 2.0" />
  <link href="/Joomla_1.6/index.php?format=feed&amp;type=atom"
➥rel="alternate" type="application/atom+xml" title="Atom 1.0" />
```

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, and any RSS-feed URLs.

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="<?php echo $this->baseurl
➥?>/templates/system/css/system.css" type="text/css" />
<link rel="stylesheet" href="<?php echo $this->baseurl
➥?>/templates/system/css/general.css" type="text/css" />
<link rel="stylesheet" href="<?php echo $this->baseurl ?>/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. For example, the following code detects and adds an additional CSS file that targets the quirks of Internet Explorer 6 (we'll leave it out of our working example here):

```
<!--[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="<?php echo $this->baseurl ?>/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/960TemplateTutorialStep1/css/red.css"
➥type="text/css" />
```

**NOTE**

If you want to load the MooTools JavaScript library, commonly used in Joomla functions and extensions, you'll need to load it in the head, adding two lines like this:

```
<?php
/**
 * @copyright    Copyright (C) 2005 - 2010 Barrie North.
 * @license        GPL
```

```
 */
defined('_JEXEC') or die;
JHTML::_('behavior.mootools');
$app = JFactory::getApplication();
?>
```

### The Joomla! Page Body

Still in the `index.php` file, 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 $app->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" />
<jdoc:include type="modules" name="debug" />
</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
- A debug module

At this point (if you preview it, make sure it's the default template), the site does not look very awe inspiring (see Figure 9.3). Note I applied the template to the dataset I continued with from Chapter 7, "Expanding Your Content: Articles and Editors."


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:

```
<body>
<?php echo $app->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" />
<jdoc:include type="modules" name="debug" />
</body>
```

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" />
```

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.

**960TEMPLATETUTORIALSTEP1**

At this point, you have a very bare template.

I have created an installable template that is available from www.joomlabook.com: `960TemplateTutorialStep1.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 "fixed" 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.

A few years ago, fluid width templates were all the rage. Accessibility guys loved them, and it was cool to grab the corner of your browser window and see all that content slide around.

But now, I don't make fluid templates, but focus on fixed width templates. I firmly believe they are the best fit on today's Web. Four years ago, many people were still using 800px width screens. The main point of a fluid width was that you could have a web page that looked okay in a 1024px screen, but still could shrink down to the smaller screens still used.

Now, the trend in screens is the opposite. People are getting huge screens; 32% of people browsing Joomlashack.com are doing so with resolutions over 1440px!

With these big screens and a fluid width layout, you get a new problem—readability. Studies have shown that readability onscreen drops off as you go over 960px. So a fluid width will fill that big screen and a) look daft and b) slow down your reading.

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 a lot 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.

**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.

When it comes to CSS layouts, there has been a trend toward what have been coined *frameworks*. The idea is that a consistent set of CSS is used to create the layout, and then that set is maintained for various issues like browser compatibility. For this template we are going to adopt the 960 grid system developed by Nathan Smith (http://960.gs/). 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.

In Figure 9.4, each column—left, middle, and right—is given its own element and grid size. With the 960 grid system, you merely have to specify with a class how big you want the grid to be. In this example, I am using a 12-column grid, so for the header to run across the full width of 960px, in the `index.php` use:

```
<div id="header" class="container_12">.
```


FIGURE 9.4 Basic template layout.

For our three columns, we add grids inside a container like this:

```
<div id="content" class="container_12">
  <div id="" class="grid_3 ">
    <jdoc:include type="modules" name="left" />
  </div>
  <div id="" class="grid_6">
    <jdoc:include type="modules" name="breadcrumbs" />
    <jdoc:include type="component" />
  </div>
  <div id="" class="grid_3">
    <jdoc:include type="modules" name="right" />
  </div>
</div>
```

Notice that there is already some breathing room to the content with a 10px column spacing, commonly called *gutter*. This is all automatically done by the clever 960 CSS grid framework, and all browser issues (yes, we mean you, Internet Explorer) are dealt with.

**NOTE**
The 960 grid works only in "modern browsers." These are defined as A grade http://developer.yahoo.com/yui/articles/gbs/.

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

```
<body>
<div id="header" class="container_12">
  <?php echo $app->getCfg('sitename');?><br />
  <jdoc:include type="modules" name="top" />
</div>
<div id="content" class="container_12">
  <div id="sidebar" class="grid_3 ">
    <jdoc:include type="modules" name="left" />
  </div>
  <div id="maincolumn" class="grid_6">
    <jdoc:include type="modules" name="breadcrumbs" />
    <jdoc:include type="component" />
  </div>
  <div id="sidebar-2" class="grid_3">
    <jdoc:include type="modules" name="right" />
  </div>
</div>
<div id="footer" class="container_12">
  <jdoc:include type="modules" name="footer" />
</div>
<jdoc:include type="modules" name="debug" />
</body>
```

In this example, I renamed the CSS file to `layout.css`. With the 960 grid framework, we will rarely need to touch this file and can compress it as much as possible. The critical parts of the `layout.css` file look like this:

```
.container_12 {
margin-left:auto;
margin-right:auto;
width:960px;
}
.alpha {
margin-left:0 !important;
}
.omega {
margin-right:0 !important;
}
.grid_1,.grid_2,.grid_3,.grid_4,.grid_5,.grid_6,.grid_7,.grid_8,.grid_9,
➥.grid_10,.grid_11,.grid_12,.grid_12 {
```

```
display:inline;
float:left;
position:relative;
margin-left:10px;
margin-right:10px;
}
.container_12 .grid_1 {
width:60px;
}
.container_12 .grid_2 {
width:140px;
}
.container_12 .grid_3 {
width:220px;
}
.container_12 .grid_4 {
width:300px;
}
.container_12 .grid_5 {
width:380px;
}
.container_12 .grid_6 {
width:460px;
}
.container_12 .grid_7 {
width:540px;
}
.container_12 .grid_8 {
width:620px;
}
.container_12 .grid_9 {
width:700px;
}
.container_12 .grid_10 {
width:780px;
}
.container_12 .grid_11 {
width:860px;
}
.container_12 .grid_12 {
width:940px;
}
```

Quite simply, everything is floated left, and the various grid sizes are set based on their desired width. It's a 12-column grid, so, for example `grid_6` means six columns, which would be 460px—the full width minus the padding. This simple layout is a good one to use for learning about how to use CSS with Joomla because it shows two of the advantages of CSS over table-based layouts: It is less code, and it is easier to maintain.

However, this simple layout is ordered in the code in the sequence in which you see content on the screen. It is 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.

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. Many commercial templates, for example, Joomlashack's, develop this source-ordered concept further.

## Default CSS

So far, all the CSS has been only about layout, which makes a plain page. So let's add some formatting, placing the CSS in a new file called `typography.css`. Remember to add it to the `index.php` file!

As you begin working on typography with CSS, you should set some overall styles and include a simple *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.


FIGURE 9.5 A basic template with typography.

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 some technique to give the columns an equal height. One common solution is to use a JavaScript script to calculate and set the heights on the fly.

### 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 if 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 if 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 if we use the `"xhtml"` option:

```
<div class="moduletable<?php echo $params->get('moduleclass_sfx'); ?>">
        <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 if 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 1.6.

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

```
<body>
<div id="header" class="container_12">
  <?php echo $app->getCfg('sitename');?><br />
  <jdoc:include type="modules" name="top" style="xhtml" />
</div>
<div class="clear"></div>
<div id="content" class="container_12">
  <div id="sidebar" class="grid_3 ">
    <jdoc:include type="modules" name="left"style="xhtml"  />
  </div>
  <div id="maincolumn" class="grid_6">
    <jdoc:include type="modules" name="breadcrumbs" style="xhtml" />
    <jdoc:include type="component" />
  </div>
  <div id="sidebar-2" class="grid_3">
    <jdoc:include type="modules" name="right" style="xhtml" />
  </div>
</div>
<div class="clear"></div>
<div id="footer" class="container_12">
  <jdoc:include type="modules" name="footer" style="xhtml" />
</div>
<jdoc:include type="modules" name="debug" />
</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 1.6, 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.

We add the following to the typography. Your CSS file should now look like this:

```
#header{
   font-size:2em;
   }
#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*/
ul.actions li{
float:right;
list-style:none;
border:0;}
ul.actions li a img{
border:0;}
```

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.

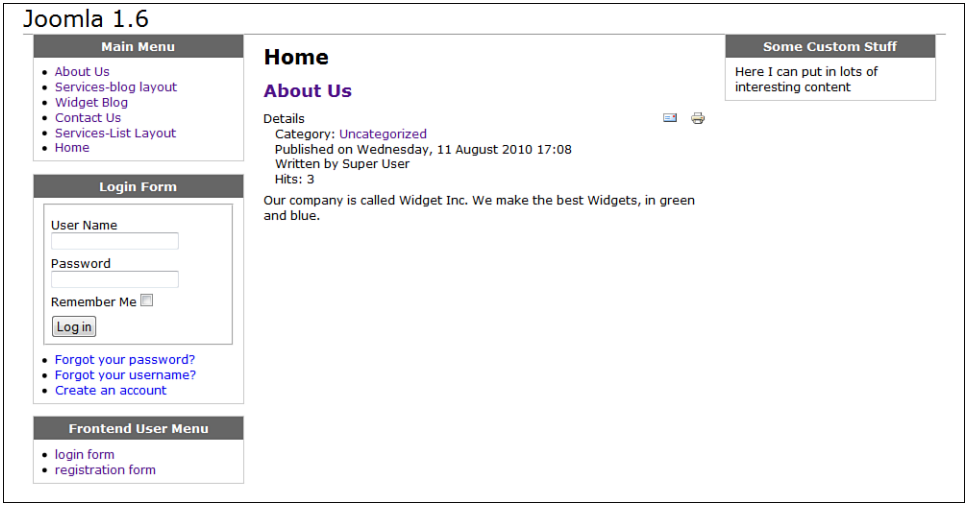


FIGURE 9.6 A basic template with module and title styling.

## 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.

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 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`.

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 looks 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 produces a menu like what's shown in Figure 9.7.

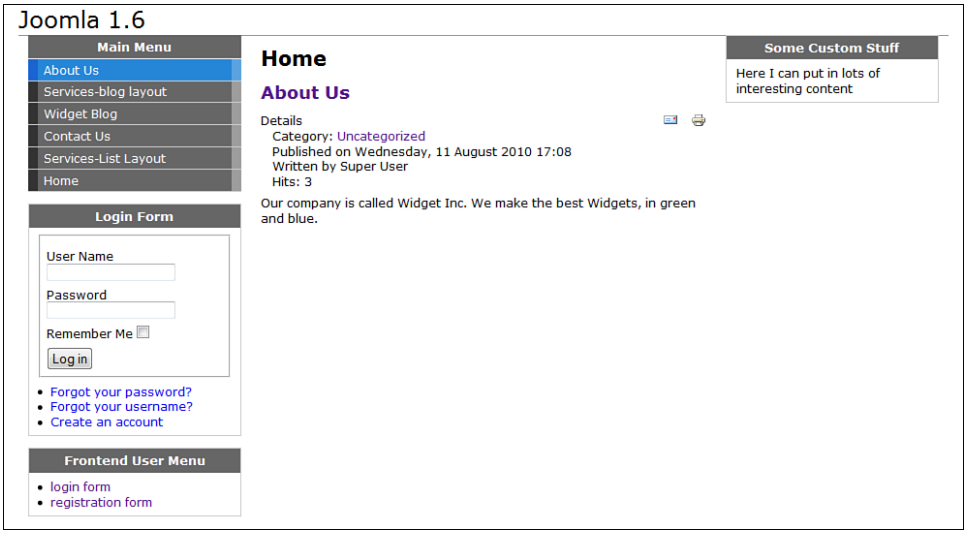


FIGURE 9.7 A basic template with menu styling.

**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="menu">
  <li id="current" class="active item101"><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. Also note that the class of `active item101` has the 101 determined by the ID of that menu item. It will be different for other menu items.

## 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 content in any column, 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 1.6 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 `3` because there are three 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).
- **`countModules('left or right')`**—This returns `1` because there is a module in the left or right position. Both tests are true (>0).
- **`countModules('left + right')`**—This returns `4` 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. We are going to take advantage of the grid system and simply pass the sizes of the grid based on some calculations.

In the header, let's define a couple of variables to make sure they have some default value.

```
$leftcolgrid    = "3";
$rightcolgrid    = "3";
```

In the HTML of the template, we can then use these variables to set the `grid` class:

```
<div id="content" class="container_12">
  <div id="sidebar" class="grid_<?php echo $leftcolgrid;?>">
    <jdoc:include type="modules" name="left"style="xhtml"  />
  </div>
  <div id="maincolumn" class="grid_<?php echo
➥(12-$leftcolgrid-$rightcolgrid);?>">
    <jdoc:include type="modules" name="breadcrumbs" style="xhtml" />
    <jdoc:include type="component" />
  </div>
  <div id="sidebar-2" class="grid_<?php echo $rightcolgrid;?>">
    <jdoc:include type="modules" name="right" style="xhtml" />
  </div>
</div>
```

You'll notice we are echoing out the `colgrid` values and then doing a simple calculation to find the main column, as we know they must total 12.

We then can use the `countModules` function to find some value. In our head we insert:

```
<?php
if ($this->countModules('left') == 0):?>
<?php $leftcolgrid = "0";?>
<?php endif; ?>
<?php
if ($this->countModules('right') == 0):?>
<?php $rightcolgrid = "0";?>
<?php endif; ?>
```

Note that we are checking to see whether the left and right positions have zero modules as we have already set the default grid size to 3. We could have also have done this check with a true/false check rather than a numerical value (zero).

**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 $(12-$leftcolgrid-$rightcolgrid); ?>% 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" class="grid_<?php echo $leftcolgrid;?>">
    <jdoc:include type="modules" name="left"style="xhtml"  />
  </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
/**
 * @copyright    Copyright (C) 2005 - 2010 Barrie North.
 * @license      GPL
 */
defined('_JEXEC') or die;
$app = JFactory::getApplication();
$leftcolgrid    = "3";
$rightcolgrid   = "3";
?>
<!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="<?php echo $this->baseurl
➥?>/templates/system/css/system.css" type="text/css" />
<link rel="stylesheet" href="<?php echo $this->baseurl
➥?>/templates/system/css/general.css" type="text/css" />
<link rel="stylesheet" href="<?php echo $this->baseurl ?>/templates/<?php
➥echo $this->template ?>/css/layout.css" type="text/css" />
<link rel="stylesheet" href="<?php echo $this->baseurl ?>/templates/<?php
➥echo $this->template ?>/css/typography.css" type="text/css" />
<?php
if ($this->countModules('left') == 0):?>
<?php $leftcolgrid     = "0";?>
<?php endif; ?>
<?php
if ($this->countModules('right') == 0):?>
<?php $rightcolgrid    = "0";?>
<?php endif; ?>
</head>
<body>
<div id="header" class="container_12">
  <?php echo $app->getCfg('sitename');?><br />
  <jdoc:include type="modules" name="top" style="xhtml" />
</div>
<div class="clear"></div>
<div id="content" class="container_12">
<?php if($this->countModules('left')) : ?>
  <div id="sidebar" class="grid_<?php echo $leftcolgrid;?>">
    <jdoc:include type="modules" name="left"style="xhtml"  />
  </div>
  <?php endif; ?>
  <div id="maincolumn" class="grid_<?php echo
➥(12-$leftcolgrid-$rightcolgrid);?>">
    <jdoc:include type="modules" name="breadcrumbs" style="xhtml" />
    <jdoc:include type="component" />
  </div>
  <?php if($this->countModules('right')) : ?>
  <div id="sidebar-2" class="grid_<?php echo $rightcolgrid;?>">
    <jdoc:include type="modules" name="right" style="xhtml" />
  </div>
  <?php endif; ?>
</div>
<div class="clear"></div>
```

```
<div id="footer" class="container_12">
  <jdoc:include type="modules" name="footer" style="xhtml" />
</div>
<jdoc:include type="modules" name="debug" />
</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.

**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.

**960TEMPLATETUTORIALSTEP2**

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: `960TemplateTutorialStep2.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! 1.6 Template: 960TemplateTutorialStep3

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, we'll use a design by Dan Cedarholm from his book *Bulletproof Web Design* (see Figure 9.8). I heartily recommend this book, as it's provides an outstanding foundation in some CSS techniques that are useful in creating Joomla templates. We'll use some of these techniques to build this real-word template.

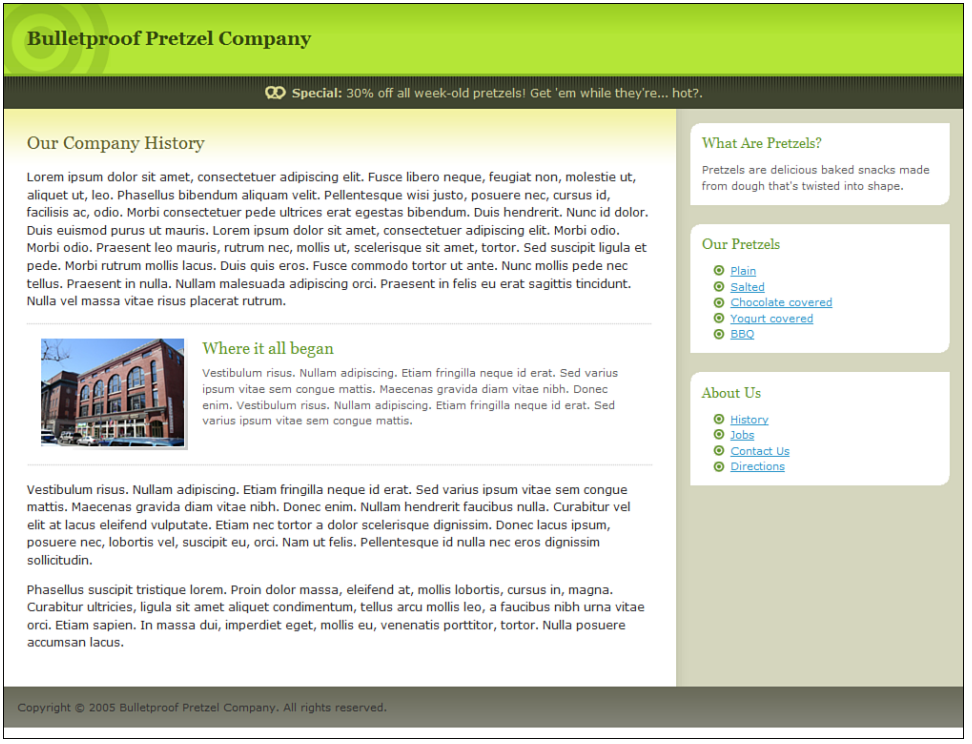


FIGURE 9.8 A design from *Bulletproof Web Design*.

### 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 could probably fill a whole book by itself. To get a sense of how to slice up a design, you can look at the images folder and see the slices.

### Header

The header image has a faint gradient 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 shows up on the black background.

```
Here is the CSS we must add to style the header#header {
    border-bottom: 3px solid #87B825;
    background: #B4E637 url(../images/header-bg.gif) repeat-x top left;
    }
#header h1 {
    margin: 0;
    padding: 25px;
    font-family: Georgia, serif;
    font-size: 150%;
    color: #374C0E;
    background: url(../images/bulls-eye.gif) no-repeat top left;
    }
```

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 Banner/Message Module

We use our "top" module location from the last template for a message. To give it some styling, we can add

```
#message {
    font-size: 90%;
    color: #cc9;
    text-align: center;
    background: #404530 url(../images/message-bg.gif) repeat-x top left;
    }
#message .moduletable {
padding:1px 10px;
}
```

The header now looks as shown in Figure 9.9.

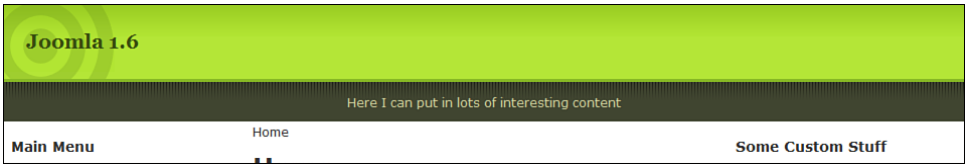


FIGURE 9.9 Header image background.

Next, you need to implement a technique to show a background on the side columns.

## 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. This is a weakness of grid-based systems; we would have to use some JavaScript to get a background on the side columns.

There are many scripts out there that calculate the height of columns and make them equal. We'll use one from Dynamic Drive: http://www.dynamicdrive.com/csslayouts/equalcolumns.js.

Note that we must change the columns/elements referred to in the script to match ours. We are also going to add another containing block element, `"maincolbck"` to hold the yellow faded background for the top of the content in the main column.

Our main content code in the `index.php` looks like this:

```
<div id="content" class="container_12">
<div id="maincolbck">
  <?php if($this->countModules('left')) : ?>
  <div id="sidebar" class="grid_<?php echo $leftcolgrid;?>">
    <jdoc:include type="modules" name="left"style="xhtml" />
  </div>
  <?php endif; ?>
  <div id="maincolumn" class="grid_<?php echo
➥(12-$leftcolgrid-$rightcolgrid);?>">
    <jdoc:include type="modules" name="breadcrumbs" style="xhtml" />
    <jdoc:include type="component" />
  </div>
```

```
  <?php if($this->countModules('right')) : ?>
  <div id="sidebar-2" class="grid_<?php echo $rightcolgrid;?>">
    <jdoc:include type="modules" name="right" style="xhtml" />
  </div>
  <?php endif; ?>
</div>
<div class="clear">
</div>
```

Let's also put a background onto the footer element while we are adding these. Our added CSS is

```
#content {
    font-size: 95%;
    color: #333;
    line-height: 1.5em;
    background: url(../images/content-bg.gif) repeat-x top left;
    }
#maincolbck {
    background: url(../images/wrap-bg.gif) repeat-y top right;
    }

#footer {
    background: #828377 url(../images/footer-bg.gif) repeat-x top left;
    padding:1px 0;
    }
```

This now gives us a gradient background for the right column:

```
#footer {
    background: #828377 url(../images/footer-bg.gif) repeat-x top left;
    padding:1px 0;
    }
```

**NOTE**

As is, the right column background is on a `div` that contains all three columns, so it will be there even if there is no actual content in the right column. It would be easy to make this template a little more robust and flexible to enclose the opening and closing of the `div` with a conditional statement that checks whether there are any modules in the right column.

### Flexible Modules

When designing modules, you need to consider whether they will stretch vertically (if more content is in them), horizontally, or both. Here we use the principles of bulletproof design contained in Dan's book. We use a couple of simple background images to create a module background that stretches in both axes. We place one background on the containing `div`, and the other one for the opposite corner on the `h3` header.

As this design does not have a horizontal menu, we also take care of menu styling as we consider the side modules.

Our CSS looks like this:

```
#sidebar .moduletable,#sidebar-2 .moduletable {
    margin: 10px 0 10px 0;
    padding: 0 0 12px 0;
    font-size: 85%;
    line-height: 1.5em;
    color: #666;
    background: #fff url(../images/box-b.gif) no-repeat bottom right;
    }
#sidebar  h3,#sidebar-2  h3 {
    margin: 0;
    padding: 12px;
    font-family: Georgia, serif;
    font-size: 140%;
    font-weight: normal;
    color: #693;
    background: url(../images/box-t.gif) no-repeat top left;
    }
#sidebar p,#sidebar-2 p,sidebar ul,#sidebar-2 ul {
    margin: 0;
    padding: 0 12px;
    }
sidebar ul li,#sidebar-2 ul li {
    margin: 0 0 0 12px;
    padding: 0 0 0 18px;
    list-style: none;
    background: url(../images/li-bullet.gif) no-repeat 0 3px;
    }
```

Now let's focus on some of the typography.

## Typography

The CSS for typography is greatly simplified in Joomla 1.6. Earlier versions of Joomla had unique classes for various parts of the output, such as "contentheading". In Joomla 1.6, the output uses more recognized classes like `H1`, `H2`, and so on, and is completely tableless.

Let's style these elements:

```
h1, h2, h3, h4, h5, h6 {
    font-family: Georgia, serif;
    font-size: 150%;
    color: #663;
    font-weight: normal;
}
h1 {font-size:2em;line-height:1;margin-bottom:0.5em;}
h2 {font-size:1.5em;margin-bottom:0.75em;}
h3 {font-size:1.25em;line-height:1;margin-bottom:1em;}
h4 {font-size:1.1em;line-height:1.25;margin-bottom:1.25em;}
```

We can also add some handy icon treatment for special classes that can be applied to content:

```
p.info {
    background: #F8FAFC url(../images/info.png) center no-repeat;
    background-position: 15px 50%; /* x-pos y-pos */
    text-align: left;
    padding: 5px 20px 5px 45px;
    border-top: 2px solid #B5D4FE;
    border-bottom: 2px solid #B5D4FE;
}

p.warn {
    background: #FFF7C0 url(../images/warn.png) center no-repeat;
    background-position: 15px 50%; /* x-pos y-pos */
    text-align: left;
    padding: 5px 20px 5px 45px;
    border-top: 2px solid #F7D229;
    border-bottom: 2px solid #F7D229;
}
```

```
p.alert {
    background: #FBEEF1 url(../images/exc.png) center no-repeat;
    background-position: 15px 50%; /* x-pos y-pos */
    text-align: left;
    padding: 5px 20px 5px 45px;
    border-top: 2px solid #FEABB9;
    border-bottom: 2px solid #FEABB9;
}

ul.checklist li {
    list-style:none;
    background: url(../images/tick.png) no-repeat 0 4px;
    line-height: 24px;
    padding-left: 20px;
    }
```

The finished template should look as shown in Figure 9.10.

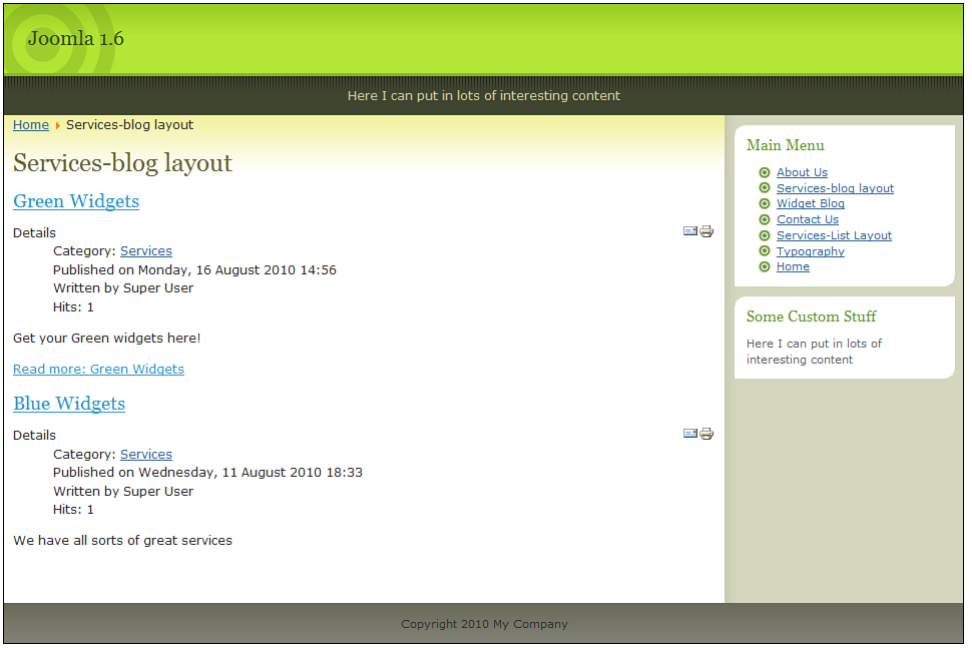


FIGURE 9.10 An advanced template with typography.

**THE LEAST YOU NEED TO KNOW**

Creating a production Joomla template is more a question of graphical design and CSS manipulation than some special Joomla knowledge.

**960TEMPLATETUTORIALSTEP3**

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 slick backgrounds with gradients.

I created an installable template that is available from www.joomlabook.com: `960TemplateTutorialStep3.zip`.

## Summary

This chapter worked through four examples of templates, each time increasing 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 consistently outputs 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 pre-built 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 use 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.

# Index

## Numbers



## A

## B

## D

## E



## F

## G–H



## I

## M

## N

## Q–R



## S

## T

## U

## V



## W

## X–Z