HOW TO BE A ROCKSTAR WordPress Designer

Build Themes and Use WordPress to Create Amazing Sites

Collis Ta'eed & Harley Alexander

Rockablepress.com
Envato.com

4

5

Foreword

So often it is the simplest concepts that are the most compelling. Email is just sending a message, XML is just a way of wrapping information up and blogging is just regularly updating content. But what might be simple to explain often has far-reaching and complex consequences.

In a few short years blogging has gone from geek-speak to mainstream web, appearing in online media portals, social pages, corporate sites and of course the thousands upon thousands of blogs.

To blog however, you need something to blog with. And though there are scores of options available, it is a handful of platforms that stand out - Blogger, Typepad, and of course WordPress.

That WordPress excels at blogging is obvious. It's fast, quick to learn, exceedingly easy to use and takes next to no time to install. Moreover it is everything one could hope for in open source software: well supported, well loved and well used.

But what makes WordPress extra special, so much so that I find myself writing a book about it, is that above all else WordPress is customizable.

Of course a lot of software lays claim to being customizable, but often this amounts to little more than changing some colors and rudimentary options. With WordPress, extensibility reaches from the admin system to the front end to the very functionality of the software.

Like many web users I am pretty handy with a bit of HTML and CSS, and I even know a little of languages like PHP, but there are unfortunately plenty of things that are way beyond me to build on

my own. The reason I love WordPress is that using my regular skill-set I am able to build much, much more than I could before. In this book Harley and I will show you how you can flex themes, code hacks and plugins to fit your needs. We'll take WordPress from blogging platform to flexible content management system.

I hope that you get as much fun and utility out of WordPress as both Harley and I have!

Collis Ta'eed

9

Notes: Example Themes and Files

Packaged with this book you will find a directory of example files made available for your reference. These files correspond to the example designs, HTML and WordPress themes used in the book. Specifically you will find:

- Photoshop files for the Creatif Design
- HTML/CSS files for the example Creatif build
- Creatif Blog Theme for WordPress
- Creatif Portfolio Theme for WordPress
- Creatif Site Theme for WordPress
- Example plugin from Chapter 6

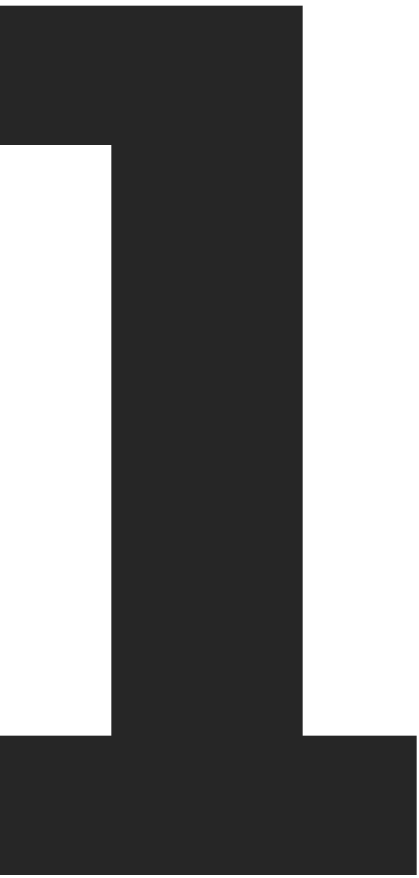
These files and themes may be used freely in your projects both commercial and non-commercial. However they may not be redistributed or resold in any way.

As you work through the book you may choose to either construct your own set of files from scratch, or to look through the example files as a guide.

Themes Online

In addition to the example files, you can find the three working themes setup on working WordPress installations online. These are made available so that you can see what the working end-result should / could look like. You will find the themes at:

Creatif Blog	http://superpreviewer.com/creatifblog
Creatif Portfolio	http://superpreviewer.com/creatifportfolio
Creatif Site	http://superpreviewer.com/creatifsite



Getting Familiar with WordPress

In this chapter we'll skim through the basics of WordPress, getting an install, how it works, where you find out more and the technical skills you need to work with WordPress as a web designer/developer. If you are already familiar with the software you can skip ahead to Chapter 2 or read on for a quick refresher.

Introduction

WordPress is an open source, content management system made specifically for blogging. That means it is a program that you install, free of charge, on your server to build a website – usually a blog.

Once installed WordPress has two parts. First the password-protected admin system where you can put up posts, edit content and manage the site. And second the public site where people can view the posts and content.

WordPress is written in the PHP language and has had 2 major version releases to date as well as lots of sub-releases. At the time of writing we are up to 2.7.x, though this number updates frequently.

What Makes WordPress Popular

Though initially falling in the shadow of SixApart's Moveable Type and Typepad products, WordPress has been steadily gaining in popularity and is now arguably the most popular blogging platform for serious bloggers.

This popularity is due to a number of factors, most notably WordPress' ease of use and its flexibility and customization. Installation of the software takes just minutes and a blogger with very little savvy can get up and running in next to no time.

Once installed, WordPress offers a huge range of customization options from the default modifications to custom themes and plugins.

WordPress.com vs WordPress.org

One common confusion arises between the two main WordPress sites, WordPress.com and WordPress.org. To understand the difference you first need to know about Automattic.

Automattic is the company behind WordPress. Although the software is open source, Automattic manages the servers, the distribution and the management of the product. They also operate WordPress.com which is a hosted version of WordPress.

So if you didn't have your own server or didn't know anything about web development you might sign up there and get an account which would give you access to your own WordPress blog. The accounts are very limited in options for customization however, so we won't be too concerned with WordPress.com in this book.

Fig 1-1 – The WordPress.com landing page.

WordPress.com actually runs a special version of something called WordPress Multi-User (WPMU) which allows you to operate a hosted blog platform. Automattic turns a profit by, among other things, offering upgrades and occasional ads on WordPress.com.

So in other words WordPress.com is a service that runs the WordPress software.

WordPress.org on the other hand is the home of that software, in the form of an open source project. This is where you can chat to other people about WordPress, download the latest install, find plugins and themes and access the official help and documentation – also known as the Codex.

In this book we are only interested in WordPress.org and the software you download and install on your own server.

Getting a WordPress Install

In order to get acquainted with WordPress you will first need a copy to experiment with. There are three ways you can do this:

Method 1: Get an Account with a Web Host with Auto-Install for WordPress

Because WordPress is so popular, many web hosts have begun packaging auto-installers with their services. You can find some well-known web hosts doing this via WordPress.org's hosting page: http://wordpress.org/hosting.

Once you have an account with one of these web hosts you generally log into their control panel and find the list of auto-installers. Installation is then simply a matter of clicking through a form or two. If you run into any problems with this method you can simply contact your web host's support team.

If you use an auto-installer, it's a good idea to check the version of WordPress being used to ensure it's an up to date edition.

Method 2. Install WordPress Manually onto an Existing Web Host

WordPress can be installed on any web host supporting a reasonably recent edition of PHP and MySQL. Assuming your web host doesn't provide an auto-install facility, you will need to do the work yourself.

You will need to first download the latest release of WordPress from: http://wordpress.org/download

You can find full installation instructions for WordPress at:

http://codex.wordpress.org/Installing_WordPress

The installation process is relatively easy, though you will need to create a database on your server. This can usually be done through the control panel your web host provides.

Method 3. Install MySQL, PHP and WordPress on Your Local Computer

Your final option is to create a local WordPress installation on your computer. This is great for quick testing and development of themes and plugins.

Depending on your platform (Mac/Win/Linux) there are a selection of tutorials on how to get setup available at: http://codex.wordpress.org/Installing_WordPress

After Install

Once you have your copy of WordPress installed, log in to the WP-Admin dashboard and click around to get familiar with the system. If this is your first time using WordPress we recommend you try creating a post, updating your profile, logging out, adding a comment to the post you made, logging back in and then approving it. Once you've done that, try clicking on *Appearance* > *Themes* and change the theme you are using. WordPress comes with two themes by default and often more if you have used an auto-installer.

This book will make a lot more sense if you are familiar with the basic mechanics of how WordPress works. So it's really worth spending a couple of hours trying things out.

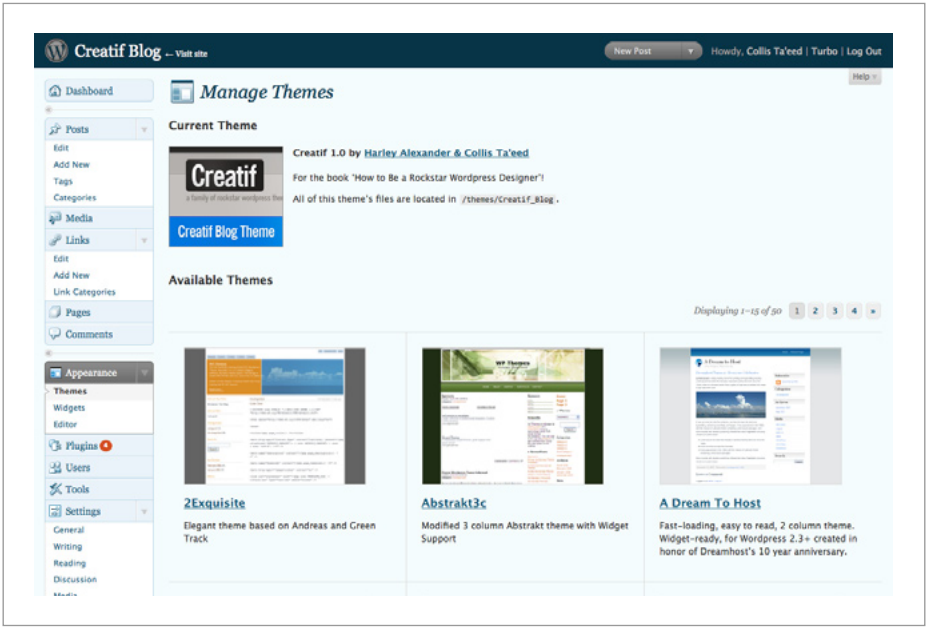


Fig 1-2 – WordPress' Appearance page lets you switch themes with just a few clicks.

Main Concepts

WordPress is fundamentally a blogging platform. Although later in the book we'll look at how to make WordPress do other things, initially we'll just assume that everything you are doing is for the purposes of creating a blog.

There are a few core concepts in WordPress, they are:

Posts
A Post is a time stamped piece of content. Posts are what a blog is made of and appear on the homepage in sequential order. They can also be categorized and tagged, have an author, a date and comments attached. Posts are the building block of WordPress. Throughout this book we will capitalize the word Posts when referring to an actual WordPress Post.

Pages
WordPress Pages are for high level content that is more permanent and static. Things like about and contact pages are usually made with a Page. Pages don't have a timestamp and appear separate from Posts. Again throughout this book we will capitalize the word Pages when referring to WordPress Pages.

Comments
Comments are small messages left on a blog by users or the public. Comments are comprised of a name, email address, URL and comment message. Comments have an approval system that by default lets through comments from any registered user and from anyone who has previously had a comment approved. First time commenters will have their comments held for approval. You will need to log in and check through the list to sort the spam from the legitimate discussion.

A very well-known plugin that will make your comment approval life much easier is *Akismet*. Akismet is made by Automattic and is a very good spam filter. WordPress comes installed with Akismet by default, although you need to get a (free) key from WordPress.com.

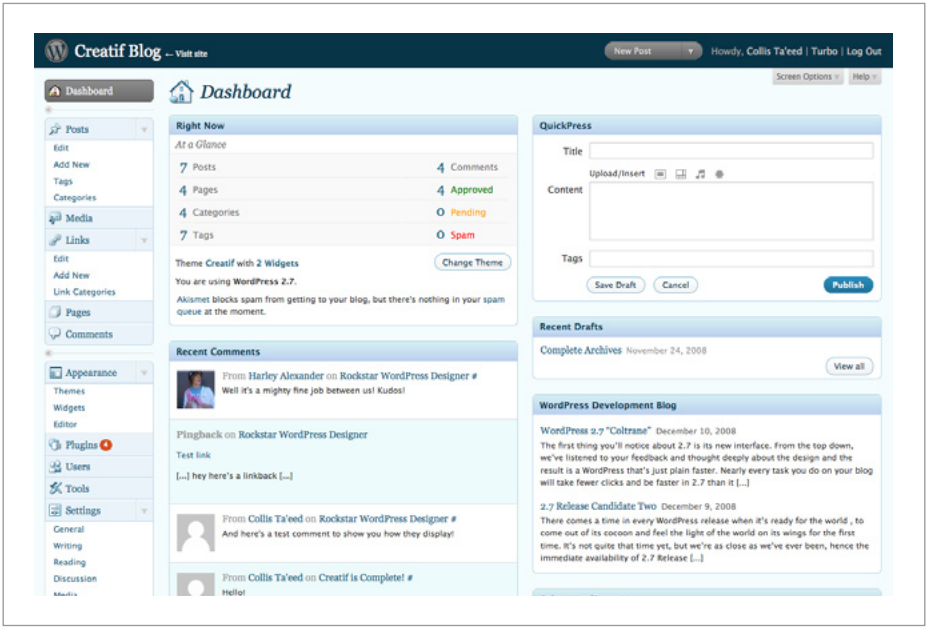


Fig 1-3 – WordPress' WP-Admin Dashboard

WP-Admin

When you install WordPress there is the front-end of the site which everyone sees and an admin area for managing and updating the site called WP-Admin, or sometimes referred to as the Dashboard, or the admin area. If you have a blog at `example.com`, then the admin will be at `example.com/wp-admin`.

Users

A WordPress install can have multiple registered users who each have different roles. The first user will always be the *admin* account having the role of Administrator. Other possible roles are: User,

Contributor, Author, Editor, Administrator. Depending on a user's role, they will be able to do different things like post, edit, and approve comments.

Users can be created in WP-Admin or if you make it possible, from the front end. So for example you could choose to make a blog such that anyone wishing to comment would need to register. You might also then hire an author or two and create user accounts for them set to Author. Finally you might hire a WordPress developer to fix a bug and give them Administrator access.

Themes

A theme is a package of PHP and CSS files that determines the front-end design of your WordPress install. When you install WordPress it uses a default theme. This book will show you how to make your own themes.

Themes can radically alter how a WordPress install works and are one of the main tools a WordPress developer uses.

To install a theme you simply place a directory containing the theme files into the `wp-content/themes/` directory of your WordPress install.

Plugins

Plugins are add-ons or extensions that change or add to WordPress' existing functionality. Examples of plugins are: *Akismet* which helps you filter spam comments and *WP-Cache* which caches your pages to make your install run faster under heavy load.

There are thousands of plugins, almost all available freely on the web. Along with Themes, Plugins are one of the main tools you will use to customize WordPress.

To install a plugin you upload the relevant files to the `wp-content/plugins` directory. If the plugin is packaged in a directory, you simply place the entire directory in the `wp-content/plugins` directory.

Linkbacks / Pingbacks / Trackbacks
When another blog links to your site, WordPress will register a linkback, sometimes referred to as a pingback or trackback. Generally linkbacks appear mixed in with a Post's comments. Linkbacks serve as a way to interconnect sites and continue conversation from one blog to another.

Blogroll
Along with posts, WordPress lets you post links to other sites in your Blogroll. A Blogroll is simply a set of categorized links. Traditionally these links would be to other blogs which is where the name comes from.

Tags
WordPress Posts can be tagged with keywords to help users find similar posts. A user will click on a tag and see all posts that have been tagged with that word. Tagging is a recent addition to the default categorization that WordPress uses.

HTML, CSS & Basic PHP

This book is aimed at web designers and developers looking to make use of WordPress as a flexible content management system. As such there is some coding knowledge required. In particular there are three principal types of code we will deal with when using WordPress: HTML, CSS & PHP.

HTML & CSS

Because this book deals with the creation of themes and websites using WordPress you'll need a very solid understanding of HTML and CSS. In particular you should be used to coding in a text editor. It is difficult to use a WYSIWIG editor like Adobe Dreamweaver on WordPress themes because the pages are broken up into multiple files.

So if you're not used to hand coding, it's a good idea to start doing so. Try coding some static HTML websites before you start building WordPress themes. If you're really unfamiliar with coding websites, you can find many great HTML and CSS tutorials on the web.

Basic PHP

WordPress is written in PHP, however you don't need to know a great deal simply to work with themes and even plugins. This is because much of the hard functionality like database access is handled by WordPress.

Nonetheless when editing a WordPress theme you will need to work with and around small snippets of PHP code.

If you've never used or seen PHP, you should read through an introductory course and make some basic programs to get familiar with how it looks and feels. Although you don't need to know much, it will make working with PHP tags a lot less stressful and avoid small unnecessary mistakes like deleting a character or statement without understanding the consequences.

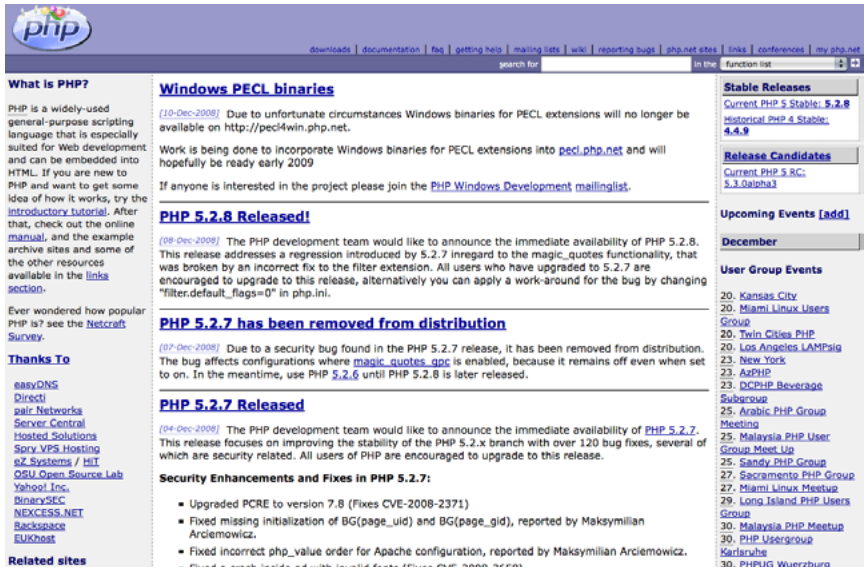


Fig 1-4 – PHP.net is a good source for syntax and help on PHP.

Fortunately PHP is a simple language and you only need to understand a basic subset of how it works. Most of what you do in theme work is to call functions, place variables and very occasionally work in a loop or if statement.

Some important things you need to know:

1. **What PHP tags look like**
   PHP is code wrapped in either `<? … ?>` or `<?php … ?>`. When the server sees these tags it knows to interpret all code in between as PHP and execute it on the server before serving up the page to the end-user's browser.

2. **How If Statements work**
   PHP, like most programming languages uses special statements that check whether something is true or

not and depending on the outcome then execute different pieces of code. The most common variety of these so-called conditional statements are if/else statements – http://www.php.net/manual/en/control-structures.else.php – however you may also run into switch/case statements – http://php.net/switch.

3. **How Loops Work**
   Another essential piece of the puzzle are loops. A loop tells the server to execute the same bit of code a set number of times usually with a changing variable. Common loop types include for loops – http://www.php.net/manual/en/control-structures.for.php, while loops – http://www.php.net/manual/en/control-structures.while.php, and foreach loops – http://www.php.net/manual/en/control-structures.foreach.php.

There are many aspects of PHP that you can get by without knowing including: how to work with databases, how to read files and advanced topics like object oriented programming.

Naturally the more you know, the easier your life will be. Happily PHP is quite an intuitive language and you can expect however to pick up a lot about PHP by working with themes. In this book you will find a medium level of explanation for the PHP code we use, so you need not be as familiar with the language as you should be with HTML and CSS.

The WordPress Codex

The WordPress Codex is the documentation and help for WordPress. It is stored at WordPress.org and is extensive and helpful.

To access the Codex visit http://codex.wordpress.org

You will use the Codex mostly to look up function names and find out how things work. For example if you were working in a theme file and you wanted to show who a blog post's author was, you might search the Codex for the words 'Post Author' and find the page: http://codex.wordpress.org/Author_Templates which tells you that to show the author you simply need this piece of code:

```
<p>Written by: <?php the_author_posts_link(); ?></p>
```

There are often a few functions to achieve the same results, and the Codex pages will link to related functions for a specific topic. So for example when looking up functions for authors, you will see links to other functions to display author information.

Using the Codex is often a good way to find out things you didn't know you could do. For example you might start reading about author functions and realize that you can link to a bio page for authors. When you are getting started with theming, it's worth making it a habit to always take some time and looking up a few related pages as an exercise in broadening your knowledge of WordPress.

Preparing WordPress for Use Checklist

After installing WordPress there are several things worth doing to get WordPress ready to go. These are just options to change in the WP-Admin system, one or two essential plugins to install and some other modifications and checks. Here's a checklist, along with explanations of what each does:

Activate Akismet Spam Protection

Akismet is a spam filter plugin for comments. It is automatically installed with WordPress, so you simply need to go to your WP-Admin Dashboard and click on *Plugins*, find Akismet in the list and click *Activate*. A message will appear asking for your WordPress.com API key, simply click the link and create a WordPress.com account and then copy and paste the key back into your Dashboard.

Install WP-Cache Plugin

When a page is accessed on your site, WordPress performs some database lookups and executes a bunch of code. Ordinarily this isn't a problem, however if you happen to have a lot of traffic all at once, this can slow your site right down. WP-Cache is a plugin that caches your pages periodically to dramatically reduce the load on the WordPress server. You can find the plugin at http://wordpress.org/extend/plugins/wp-cache/

Once installed, click on *Plugins* in the Dashboard and click Activate. You then need to click on Options in the dashboard menu and find *WP-Cache* in the menu. This leads you to a page where you can enable the caching. Don't enable just yet, because it's best to do this after you install your theme and have everything working, otherwise you can sometimes have trouble testing changes.

Note that to get WP-Cache working you may need to create some directories for it to write to. The plugin will give you details of what to do.

Set Permalinks

Every post you create has a permanent web address that WordPress creates called a *permalink*. You can select how WordPress should structure your Permalinks by clicking on *Settings* in the dashboard menu and then clicking on *Permalinks*.

There are five options to choose from:

1. **Default** – The default URL structure is a mix of odd characters and ID numbers

2. **Day and Name** – Suitable for news and time centric blogs

3. **Month and Name** – Suitable for news and time centric blogs

4. **Numeric** – An incrementing number rather than the post's actual title.

5. **Custom** – By specifying a variable in %% signs you can choose the structure

A good format to use is to click on *Custom* and write:

```
/%category%/%postname%/
```

This will create links that have the category of the post and title of the post in the URL. These links will not only be more readable and memorable, but are also better for search engines.

WordPress uses a file called `.htaccess` to create the permalinks, if this is not writable by the server, WordPress will give you instructions on creating the file yourself.

Update Your Profile

Click on *Users* > *My Profile* in the WP-Admin menu and take a moment to fill out your profile. You may wish to untick *'Use Visual Editor When Writing'* if you prefer to write in HTML as the Visual Editor has been know to mess with code, though more recent versions of WordPress are getting better in this department. This is just a preference however.

After you click Update, go back and find the select box which reads *"Display Name as"* and select a new option other than 'admin'.

Make Sure File Uploads are Working

One of the neatest things about using WordPress is that when you are creating a Post you can upload files directly in WordPress and not need to ever touch your FTP program. This is great if you have multiple writers or non-tech writers, however you may need to double check that uploads are working on your server.

In *WP-Admin*, click on *Posts* > *Add New* to bring up the Add New Post screen then click the image icon to add assets using WordPress' Uploader. Enter a file and try uploading. If all goes to plan the file will appear there successfully. If not, you will most likely need to create a directory for the upload facility to upload to, or set permissions for the directory to be writable. You can do this through your FTP program, the error message will tell you what folders to create.

Options for the uploads are controlled via *Settings* > *Miscellaneous* in the Dashboard.

Install Sitemap Plugin

The Sitemap plugin automatically creates an XML sitemap prepared to Google's specification and then contacts the Google servers every time you create a new post or change the sitemap. You can download the plugin from http://tinyurl.com/2bbwmg.

Clear WordPress and Create Categories

WordPress comes with a lot of default content in the form of a sample Post, a comment, categories and a blogroll. Although these can be quite good for testing, generally speaking you will want to go through, delete them all and then create your own content. This is particularly true in the later chapters of this book where we will need to match content with our themes.

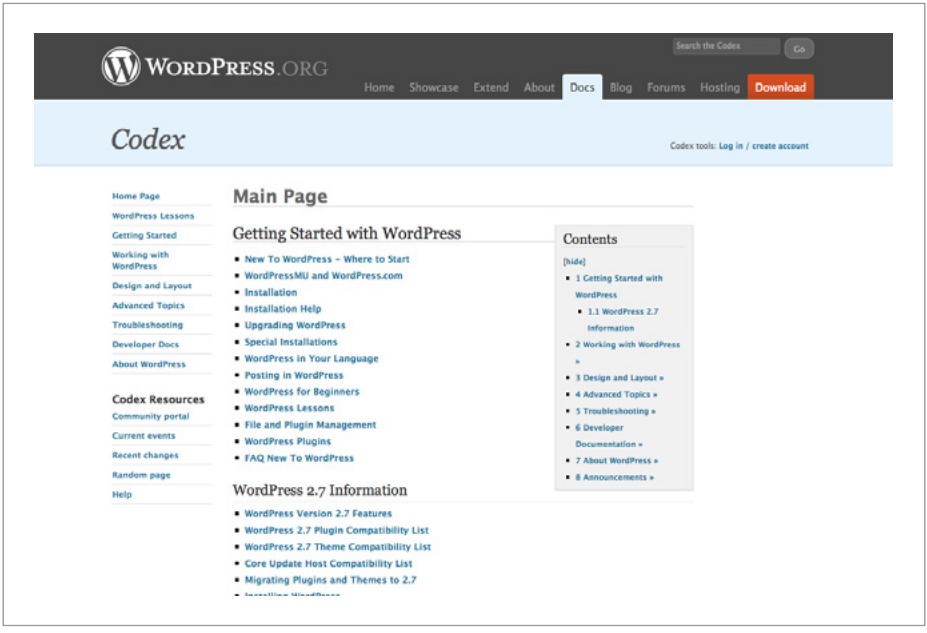


Fig 1-5 – The WordPress Codex, a definite bookmark in any theme developers favorites.

Further Resources for Getting Started with WordPress

You can find resources on FTP, installation, and blogging with WordPress at:

Changing Permissions:
http://codex.wordpress.org/Changing_File_Permissions

FTP:
Transmit (Mac) http://www.panic.com/transmit/
SmartFTP (Win) http://www.smartftp.com/
FileZilla (Cross Platform) http://filezilla-project.org/

Basic WordPress Lessons:
http://codex.wordpress.org/WordPress_Lessons

Absolute Blogging Basics:
http://www.problogger.net/archives/2005/02/05/what-is-a-blog/
http://www.problogger.net/archives/2006/02/14/blogging-for-beginners-2/

2

Blog Design

Designing a blog, like designing any other website, is all about the site's users. What do they want to do? And what do you want them to do? In this chapter we'll look at some of the important issues in designing blogs, what trends have emerged and how you can design for usability.

How Blog Design is Evolving

Blogging is a fast changing medium. The way internet users view blogs today is vastly different to how it was just two or three years ago. What began as almost simple journals have in some instances expanded into magazines, portals and communities. Even sites that hold no larger aspirations have had to cope with new features, widgets and technologies.

Conventions on how to design a simple blog are strong. Most users have a firm idea of what a blog should look like, with sidebars, a posting column, a footer and navigation/header. As in any form of design, these conventions can be broken, but it's important to understand why they are there and why they work well.

The biggest challenge currently in blog design is that as blogs evolve into new formats, add new content types, expand from single author to editorial team, and fundamentally grow, the designs that house them also need to change.

There are three trends in blogging that have interesting and important implications for blog design:

Blogs as Communities

Blogs increasingly serve as spaces for niche (and sometimes mainstream) communities. These communities naturally grow around a shared interest and capitalize on one of the key features of a blog – reader commentary.

Since bloggers, like almost any website owner, are primarily concerned with keeping users around, increasing pageviews and building loyalty, it is a natural progression to include additional community features.

These features include user profiles, ratings, social network tie-ins, galleries and an increasing number of external widgets. To date WordPress hasn't hugely grown its off-the-shelf featureset in this department, at least not for the standalone version of WordPress, however competing provider Six Apart – makers of Typepad and Moveable Type – have. No doubt WordPress won't be far behind.

Some larger blogs have gone the route of building custom platforms, most notably the tech blog Mashable – http://mashable.com – which provides a small social network to its users.

For WordPress users today, there are a huge range of plugins to extend the core functionality and in Chapter 9 we'll showcase some of them with explanations on how they can build community features into a WordPress blog.

Blogs as Magazines

While a single, hobby blogger may only produce one or two posts a day. Increasingly professional blogging outfits are building sites that publish up to 20 posts a day. When presenting that much information on a daily basis, showing it in a single downward column may not be the best solution.

The recent trend to 'magazine' style blog themes shows that bloggers are looking for designs that improve access to their content, emulating traditional media in format and style. These designs often have feature posts and sub posts, present a greater number of short snippets and use different layout and positioning to engage the reader.

Fig 2-1 – FreelanceSwitch is a blog that has added forums, a job board and a variety of additional elements to its basic WordPress format.

Blogs as Portals

Another trend we are seeing in blogs is to grow from housing just a blog to adding job boards, forums, link directories, classifieds and other types of content. In many cases, these efforts come from bloggers looking to increase their revenue by building in revenue streams that fit in with their blog.

Design Implications

The implications of all three trends – blog communities, magazines and portals – is similar. Namely to find ways to fit in more information in a way that doesn't overload the user, stays easy to navigate and understand, and looks good doing it!

The best way to keep up with trends in how blog design is changing is to look at what other blog designers are doing. A useful practice for blog designers it to spend time researching how other designers approach typical blog information design.

You can find lots of great blog designs at these sites:

BestWebGallery – http://bestwebgallery.com
A gallery of mostly non-flash design that includes a large number of blogs.

WebCreme – http://webcreme.com
Similar to BestWebGallery, WebCreme is another great source of blog designs.

SmashingMagazine – http://smashingmagazine.com
Smashing is a blog itself, and although about web design in general, devotes a large number of posts to rounding up great blog designs and great Wordpress themes.

Usability

Usability is about designing a website that lets its users do what they want to do in the most logical and intuitive manner. Navigation, for example, is an important part of usability, because your readers shouldn't be confused about where they are or how to get where they want to go. If they can't figure these out easily, chances are the experience will negatively impact their opinion of the site.

There are many topics, like navigation, which apply to blogging in much the same way as they do to other types of web design. You can learn about general web usability at AListApart – http://alistapart.com – one of the largest and most successful blogs around.

Here are three usability issues that are particularly important for blog design:

Understanding and Following (Blog) Conventions

Over time blog designers have established a number of conventions that users have grown accustomed to. A straightforward example is that when you are on a blog homepage and you click on a post title, you expect to be taken to the post's page where you will see the post in full, complete with comments and trackbacks.

It is important to know and follow these conventions. You can of course break some, however if you go against the grain everywhere you risk confusing users.

If you are reading this book and building Wordpress themes, chances are you are already familiar with many blog conventions, however it might be worth exploring some blogs with this topic specifically in mind. Think about where items are placed – for example social media icons tend to always appear at the bottom of a post so the user can click them after reading the post. Also take note of when something isn't in the place you expected it to be. It's natural that many blog designers get things wrong, and this can teach as much about what to do as what not.

Think Like a User

The cornerstone of usability design is to think like a user. Approach your blog design as if you are a reader rather than a designer and ask yourself questions like, what are you trying to accomplish on the site? What information is important to you? What information is irrelevant?

As an example, in many Wordpress themes there is a block of information called “Meta Information”. This shows links to the admin area, whether the page is valid xHTML and so on. In other words, information that isn’t important or even necessary for anyone except possibly the owner of the site. If you wanted to have it on the page then, it should be off somewhere discreet, instead of cluttering up an already busy sidebar.

When you design blogs, try to come at it as a user of the site and design so that common user tasks like searching archives, reading content and placing comments are easy to accomplish.

Consider how your design gives focus to different elements on a page. What are the first things the user’s eye sees? Where do they instinctively click? If possible watch what typical readers (including yourself) do when visiting a blog.

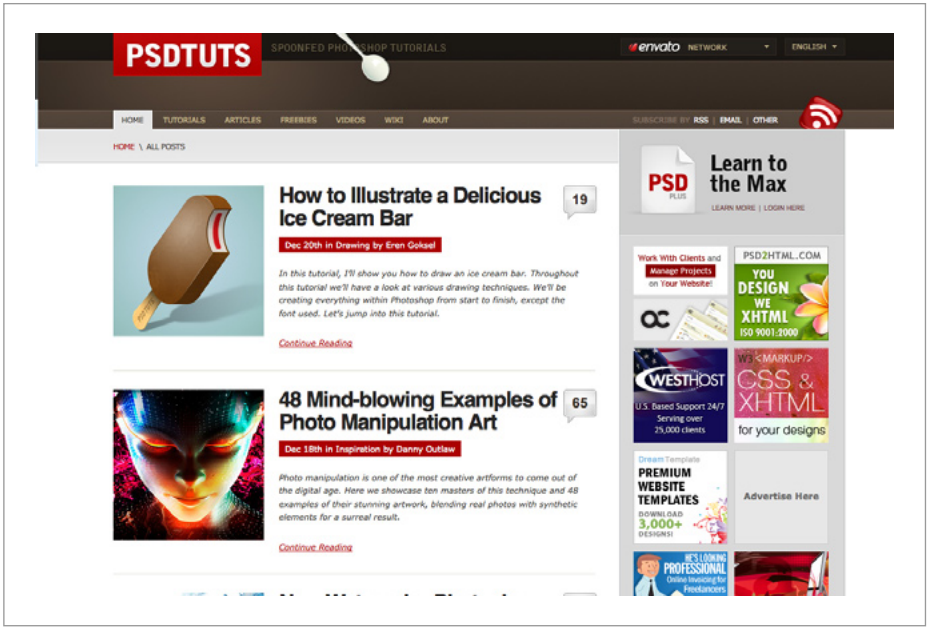


Fig 2-2 – PSDTUTS is a large blog that is a good example of dealing with the huge amount of clutter that blogs accumulate.

Order and Clutter

One particular affliction that most blogs suffer is the overwhelming number of elements on a single page. In a typical sidebar you will have categories, a search box, archives, adverts, links, banners, and possibly even navigation. The content area, besides having the blog post itself, will often include numerous links, social media icons, headings, comments, trackbacks and so on.

Most blogs are cluttered with information as is, when you add in the three trends outlined earlier, the overall effect can be overwhelming. As a blog designer your challenge is to make order out of it all. A user should be able to find everything they expect and want, but not feel overwhelmed or confused.

The best way to achieve order is to expect and plan ahead for the many elements that will appear on the page. Use size, color, position and other design tools you have at your disposal to make them fit together into a coherent page for the user.

Planning ahead to provide space for bloggers to add widgets, adverts and other extras to your blog can ensure your theme continues to look good, even after it leaves your hands.

Making Room for Advertising

Many blogs today run advertising. The more successful the blog, the more likely this will be the case. As a web designer, advertising presents a challenge because it creates a space that is beyond your control. Below are some tips for designing a blog that is going to run advertising.

Plan for Advertising

It's tempting to design your page and then think about how to fit adverts in later, but this only makes the problem worse. If you realize advertising is going to be run on the blog, then design spaces to fit.

What you want to avoid is adverts that look like they were just placed in whatever empty space the blogger could find. As a designer your job is to make the advert look like it's meant to be there.

Place Ads in Your Mockups

When you are designing up your page, avoid simply leaving blank spaces where ads will be placed. Instead find actual adverts and mock them into your designs. Empty spaces are deceptive and will usually look better than actual advertising. Advertising often has a lot of visual noise and usually needs space around it to balance it out. You won't realize this or make space to accommodate unless you actually place real ads in your design. Moreover, it's often best to plan for the worst case scenario – somehow that's often the one that eventuates!

Blend In, Stand Out

Advertising works best when it stands out because it catches the user's attention. Unfortunately it often looks the worst when standing out the most. Your aim should be to design in advertising that blends in with the interface whilst at the same time standing out.

To accomplish this, aim to make the advertising spaces a part of the interface. For example if your design was made up of a series

of boxes, advertising would make most sense if it had its own box rather than just appearing in an empty space.

Avoid the temptation to move ads off to areas of the page that users won't see, like the footer. When working with text adverts, avoid coloring them so that they blend in completely, but do choose a palette that compliments the rest of the page.

Use Standard Ad Sizes

As a web designer it's tempting to use ad sizes that match your design – adding them in after the initial layout is done. If a column happens to be 178px wide, then the advert should be 178px right? This might make life easier when designing, but is harder for the blogger and ad buyer to manage. Learn standard Internet Advertising Bureau (IAB) ad sizes and work your designs to fit them, not the other way around.

Converting Visitors into Readers

A blog has one underlying aim: to capture as large an audience as possible. To do this you must not only create traffic, but additionally convert that traffic into regular readers. It is this second task that a blog designer should pay attention to.

To turn a casual visitor into a regular reader, a blog needs to engage the visitor as much as possible. To a large extent this is a function of the blog's content, however a good website design can make this task much simpler. Below are some ideas to consider.

Focus on Content

As a designer it can be tempting to add a lot of visual noise to a page in the form of graphics and interface. However it's vital that

a blog's design be focused on the content of the blog, not the form. This is particularly true for visitors arriving from services like StumbleUpon and Digg, where the reader often gives the site only a cursory glance before moving on. It is therefore important to get straight to the heart of things, with content, big headings, perhaps a bold image and not too much else. If the page looks too distracting, chances are the visitor will move on, after all something more compelling is usually only a click away.

Giving the Reader Somewhere to Go

The natural time for a user to leave a site is once they've finished what they were doing there. So if a user is reading a blog post, the natural time for them to leave the site is when they get to the end of the post.

It is therefore important to ensure the user has somewhere to go which keeps them on the site. There are many ways to do this including:

- **Related Posts**
  At the end of blog posts, give the user links to related posts either using a Wordpress plugin (http://wordpress.org/extend/plugins/wordpress-23-related-posts-plugin/) or by running a query to show other posts in the same category or other posts by the same author.

- **Archive and Category Links**
  At the end of blog posts, give the user links to search the archives or browse the post's category.

- **Popular Posts**
  Popular posts are popular for a reason and are a great way to engage new readers and show them why the site is worth bookmarking. Providing easy links to these

is a good way to direct the user to the best the site has to offer.

Make Browsing Archives Easy

As a blog gets older and larger, much of its content gets hidden away. Often there is content tucked away in the archives that would be of great interest to a new, casual visitor. Standard blogs will generally have a list of category or date based archives, however as a blog designer you can go further and look at additional means for users to browse older posts:

- **Popular Posts**
  The more popular a post is, the more likely a new visitor will enjoy it. The design of a blog can help users find a blog's popular posts through a simple sidebar element that is either hand picked by the blog author, or automated using a formula like most *comments = most popular.*

  Recently blogs have begun using Javascript to create changeable menus of post links. The user is able to view for example popular posts of all time, recently popular, the editor's picks or recently commented posts. When they select a new option, the list of posts shown changes using Javascript.

- **Random Posts**
  Using a freely available Wordpress plugin (http://wordpress.org/extend/plugins/random-posts-plugin/) a blog design can show a list of random posts to the user. This can be useful for not just casual, first time visitors but even for regular readers as it is a powerful way to help them explore parts of a blog they may not have seen before.

- **Sneeze Pages**
  A sneeze page is an author edited list of links organized up according to blog topics. The idea is that from a sneeze page, your readers can be sent off to different parts of a blog, according to their interest.

  As a blog designer, you might create a sidebar area titled "Explore the Blog" with a list of main topics. Each topic would then link off to a page with a brief introduction to that topic and a list of post links to relevant articles.

- **Browsable Archives**
  Almost every blog has some sort of archive, however blog designers often don't pay that much attention to these pages. Archives are however amongst the most important pages on a site and should be well thought out. A good blog archive should have at the very least a hierarchical listing of posts according to date, category or author, a sitemap of non-post pages and a search form in case the user can't find what they need.

Help the User Subscribe

The mainstay of blog readership are subscribers. A blog should be designed in such a way as to encourage readers to become subscribers easily. There are many ways a design can do this:

- **Email Subscriptions**
  Although traditionally subscriptions are through RSS, thanks to services like Feedburner (http://feedburner.com) offering a choice of RSS or email subscription is simple. This should be an obvious first step in ensuring subscription is easy.

- **Help with RSS**
  Not everyone knows what RSS is, particularly on non-tech blogs. So it makes sense to have a link or page explaining what RSS is and how you can subscribe to a blog.

- **Ask Them to Subscribe!**
  The best time to ask a reader to subscribe is at the end of a post. It's easy to add a button or sentence asking a user to subscribe in a panel at the end of a post.

- **Utilize Current Subscriber Numbers**
  For larger blogs, the social proof of having current subscriber numbers can provide a "It must be good!" kind of effect. Feedburner provides a simple daily updating image that shows the subscriber count, however it's possible to use a WordPress plugin for even more customization (http://tinyurl.com/customrss)

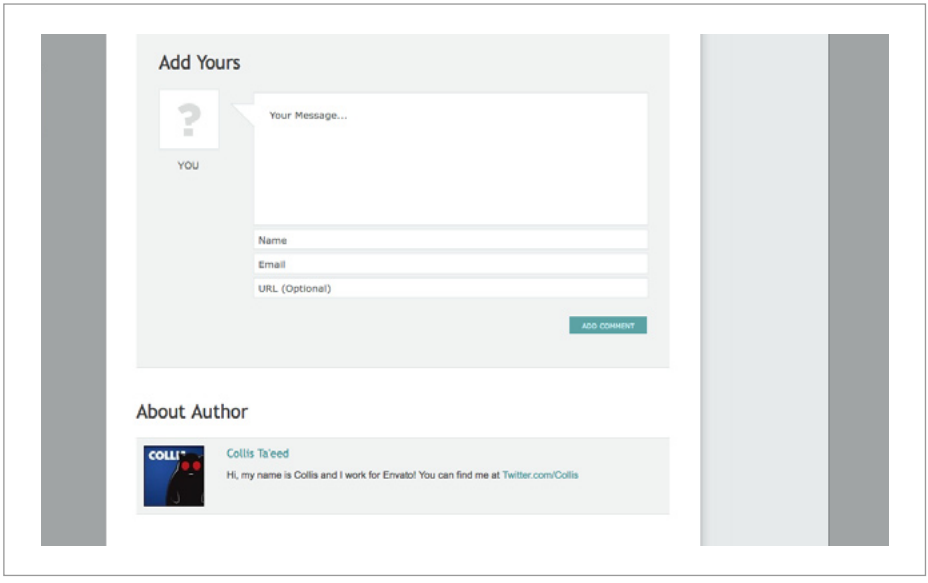


Fig 2-3 – The comment form on Creattica Daily.

Engage the Reader in Discussion

Most blogs include comments, and a great way to turn a casual visitor into a regular visitor is to engage them in the discussion that takes place in those comments. Some of the ways that a blog design can help get a reader involved include:

- **Make It Easy to Comment**
  Commenting should be as easy as possible. When a post finishes, you want as little noise between the post and comments as possible. Headings and comment instructions should be clear and large so the user knows where to go and what to do.

  In most instances it is a good idea to avoid making membership a requirement for commenting – though generally this decision is out of the hands of the blog designer.

- **Show Recent Comments**
  Showing recent comments on a blog's homepage or sidebar is a good way to get casual visitors involved in a discussion they might not have seen. It also gives the user more incentive to comment as there is more emphasis on discussion on the blog.

- **Include Comment Subscriptions**
  Wordpress makes it easy to provide RSS feeds for comment threads and this feature will keep commenters hooked to a blog far more tightly than if they rely on remembering which posts they wanted to check the discussion on. A plugin is also available to allow users email updates on a discussion (http://wordpress.org/extend/plugins/subscribe-to-comments/)

- **Use Gravatars**
  Gravatar is a simple, lightweight system to provide user's with images or avatars to appear next to their comments. Since WordPress 2.5, Gravatars have become native to WordPress and it's a good idea to make use of them. They help make a discussion more personable by giving a face to people's comments helping draw in users to chat on a blog.

Tips for Public Theme Design

Most blog design is done for custom sites, however some blog designers create WordPress themes for public release. When you design a theme for public release, there are several extra issues to consider.

- **Use text wherever possible**
  When designing a custom blog design it's possible to use graphics for things like sidebar headings or logos. However doing this in a public theme means the bloggers using the theme won't be able to customize those parts. Therefore unless it's absolutely essential you should avoid using graphics and stick to plain HTML text or for a more advanced look, Flash and sIFR – http://en.wikipedia.org/wiki/Scalable_Inman_Flash_Replacement. You may not be able to get quite the look you wanted, but working within the constraints can still get a good result.

- **Make the design as flexible as possible**
  People using your theme will inevitably try to do things you never thought of. This means you need to keep a theme as flexible as possible. For example if you rely on a sidebar being a certain height in order not to break the page's structure you are heading for trouble. Users

will undoubtedly try to fit extra things in, or have text and links that are odd sizes. Try to make a design that is hard to break, where overflow content just moves down the page and doesn't throw things out of alignment.

- **Design the WHOLE theme**
  In a custom blog design you can get away with only designing the parts you are going to use. For example if you don't want archives, your archives page doesn't need to be designed (or even exist), however in public theming, some users of your theme will have different requirements. So it's important to design every page and make sure the theme is complete.

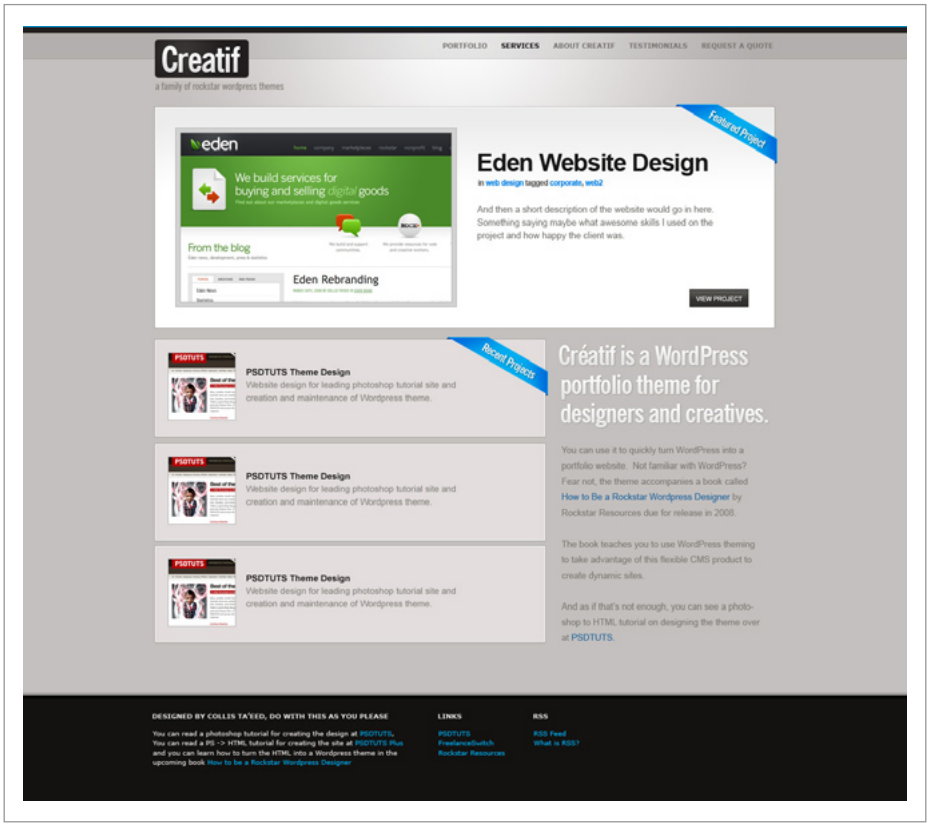
3

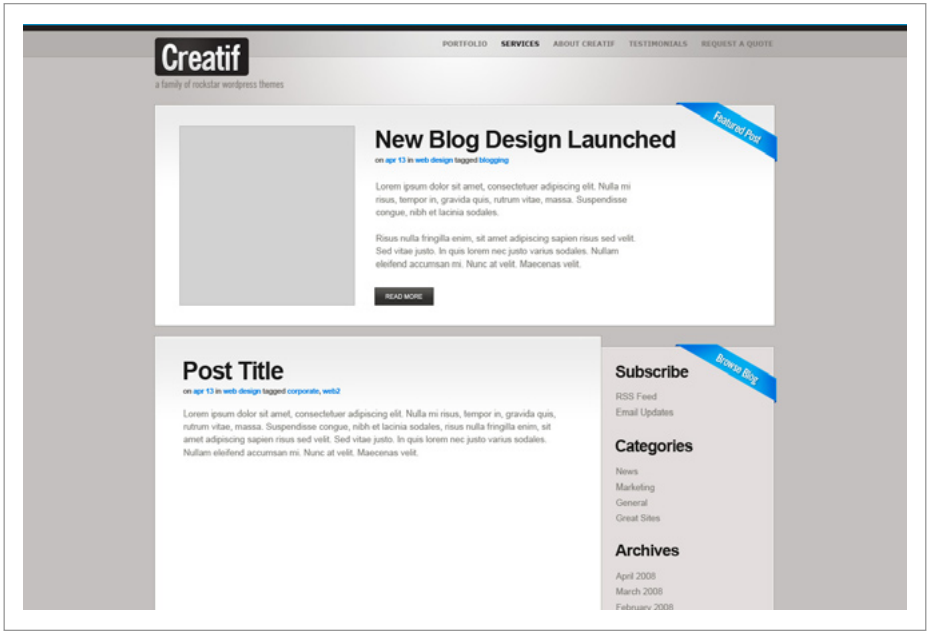
Meet Creatif

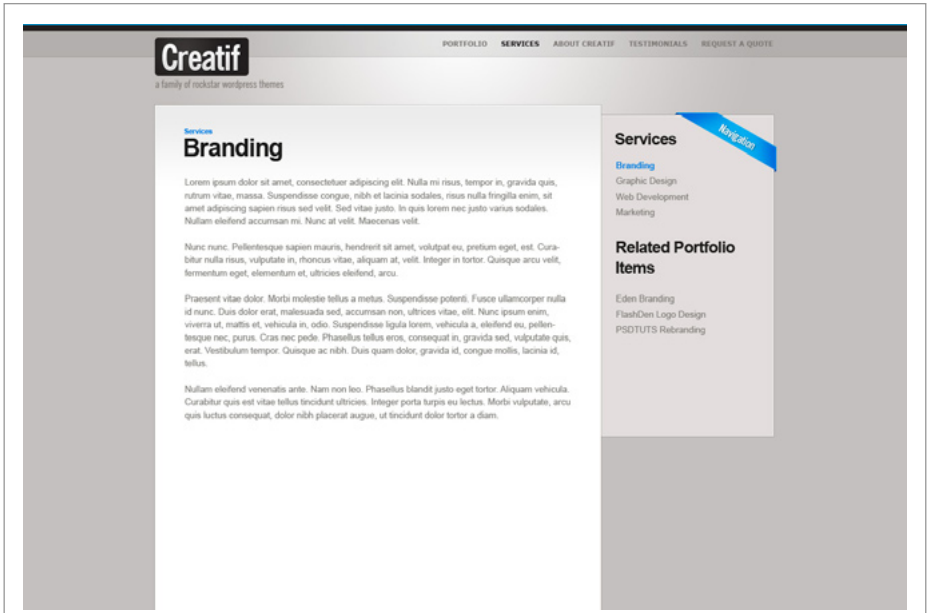
The best way to learn about theming WordPress is to have an example to follow along. In this chapter we'll look at the example set of designs we're going to be working with, run through a design tutorial on putting them together in Adobe Photoshop and then a second tutorial on taking the Photoshop PSD files and building a set of HTML mockups.

Our Example Set of Designs

The example set of themes we're going to be using in this book are called Creatif. The designs were done specifically to suit the purposes of this book and include three homepages, a post page, a portfolio page and a general page, as well as an alternate color scheme. Here's what they all look like:




Creatif
a family of rockstar wordpress themes
PORTFOLIO
SERVICES
ABOUT CREATIF
TESTIMONIALS
REQUEST A QUOTE
Featured Post
New Blog Design Launched
on apr 13 in web design tagged blogging
Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Nulla mi risus, tempor in, gravida quis, rutrum vitae, massa. Suspendisse congue, nibh et lacinia sodales.
Risus nulla fringilla enim, sit amet adipiscing sapien risus sed velit. Sed vitae justo. In quis lorem nec justo varius sodales. Nullam eleifend accumsan mi. Nunc at velit. Maecenas velit.
READ MORE
Post Title
on apr 13 in web design tagged corporate, web2
Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Nulla mi risus, tempor in, gravida quis, rutrum vitae, massa. Suspendisse congue, nibh et lacinia sodales, risus nulla fringilla enim, sit amet adipiscing sapien risus sed velit. Sed vitae justo. In quis lorem nec justo varius sodales. Nullam eleifend accumsan mi. Nunc at velit. Maecenas velit.
Browse Blog
Subscribe
RSS Feed
Email Updates
Categories
News
Marketing
General
Great Sites
Archives
April 2008
March 2008


Creatif
a family of rockstar wordpress themes
PORTFOLIO
SERVICES
ABOUT CREATIF
TESTIMONIALS
REQUEST A QUOTE
Services
Branding
Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Nulla mi risus, tempor in, gravida quis, rutrum vitae, massa. Suspendisse congue, nibh et lacinia sodales, risus nulla fringilla enim, sit amet adipiscing sapien risus sed velit. Sed vitae justo. In quis lorem nec justo varius sodales. Nullam eleifend accumsan mi. Nunc at velit. Maecenas velit.
Nunc nunc. Pellentesque sapien mauris, hendrerit sit amet, volutpat eu, pretium eget, est. Curabitur nulla risus, vulputate in, rhoncus vitae, aliquam at, velit. Integer in tortor. Quisque arcu velit, fermentum eget, elementum et, ultricies eleifend, arcu.
Praesent vitae dolor. Morbi molestie tellus a metus. Suspendisse potenti. Fusce ullamcorper nulla id nunc. Duis dolor erat, malesuada sed, accumsan non, ultrices vitae, elit. Nunc ipsum enim, viverra ut, mattis et, vehicula in, odio. Suspendisse ligula lorem, vehicula a, eleifend eu, pellentesque nec, purus. Cras nec pede. Phasellus tellus eros, consequat in, gravida sed, vulputate quis, erat. Vestibulum tempor. Quisque ac nibh. Duis quam dolor, gravida id, congue mollis, lacinia id, tellus.
Nullam eleifend venenatis ante. Nam non leo. Phasellus blandit justo eget tortor. Aliquam vehicula. Curabitur quis est vitae tellus tincidunt ultricies. Integer porta turpis eu lectus. Morbi vulputate, arcu quis luctus consequat, dolor nibh placerat augue, ut tincidunt dolor tortor a diam.
Navigation
Services
Branding
Graphic Design
Web Development
Marketing
Related Portfolio Items
Eden Branding
FlashDen Logo Design
PSDTUTS Rebranding

Provided below are two tutorials on building the photoshop layout and HTML/CSS for Creatif. Most of the styles and pages we need to build our themes are covered in these, however you should use the final files provided with this book when building Creatif. A few extra styles not mentioned below, are added in the CSS that are needed in the WordPress development.

The Creatif Design Tutorial – Layout in Photoshop

We're going to begin building the Creatif themes by putting together a design in Adobe Photoshop. You can begin from scratch by creating a new canvas in Photoshop, sized at around 1300px wide x 1400px high. Alternatively you will find completed Photoshop PSD files in the files that came with this book.

Colour Palette

A good place to begin a design is by selecting an appropriate colour palette. Sites like COLOURlovers (http://colourlovers.com) and

Adobe's Kuler (http://kuler.adobe.com) are great for choosing a nice set but often you can just come up with your own by experimenting. A simple formula that works sometimes is to choose a set of neutral shades and a single highlight colour to lift the palette. In this instance we're going to use a beigy-grey colour palette with a really bright light blue as the highlight colour.

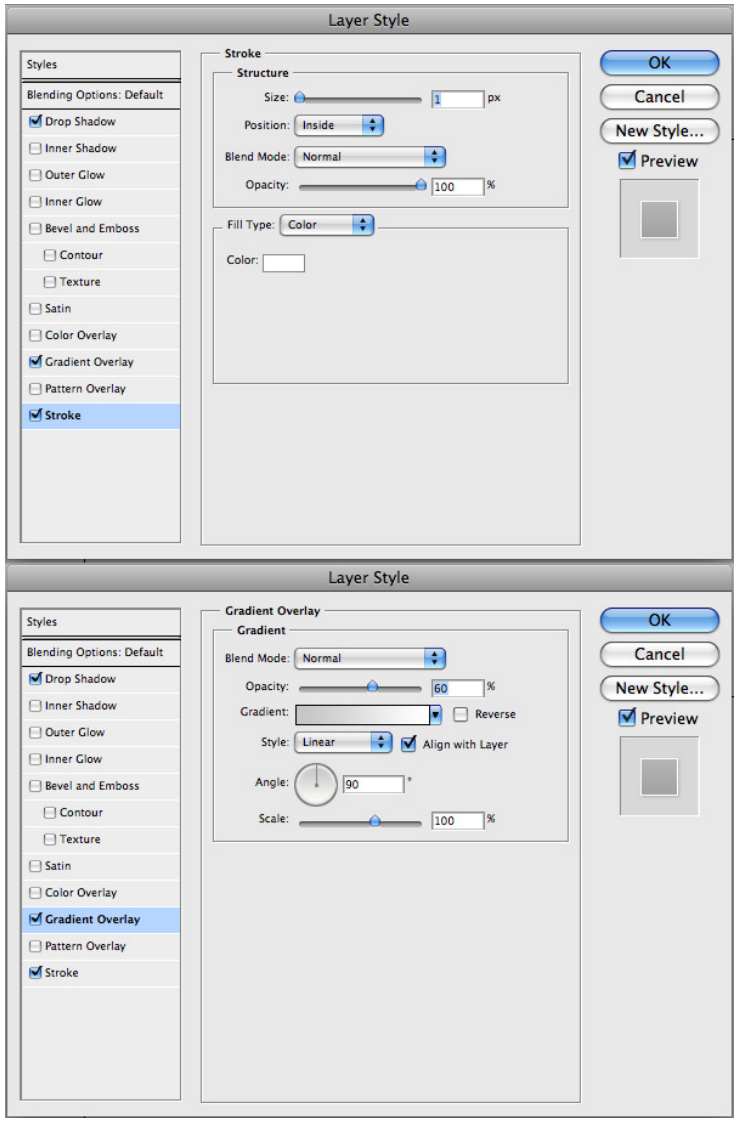
Step 1

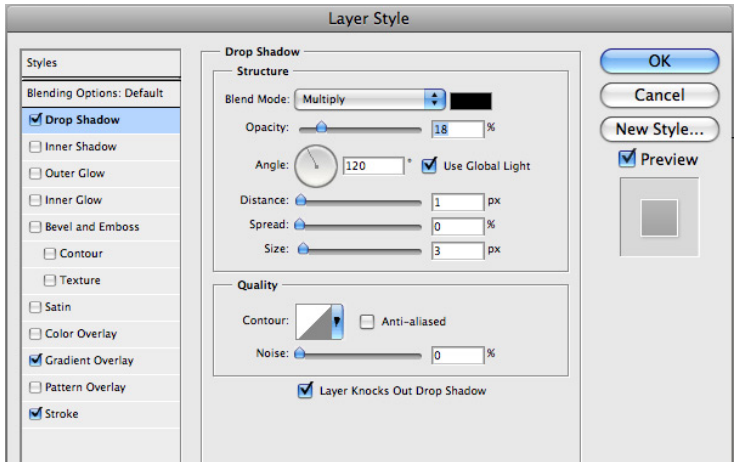
We begin the tutorial with a little logo. While logo design is generally a complex process, in this instance we just want a little graphic to anchor the page.

Here we've used the font *News Gothic Condensed Bold*. We'll add a simple *Layer Style* (settings for which can be found in the next step) to give the text a bit of kick. It uses a faint gradient, subtle shadow and a 1px border to lift the type off the page.

Step 2

Here are the settings for the Layer Style:





Step 3

Next we add a rounded rectangle behind the text. You can create this with the *Rounded Rectangle Tool (U)*. As you can see in the image below I have added a faint gradient to the box as well. You can do this by *CTRL-clicking* the box layer to select its pixels, going to *Select* > *Modify* > *Contract* and contracting by *1px* and then in a new layer drawing a *Radial Gradient* from a lighter version of the dark colour and fading to transparency.

Step 4

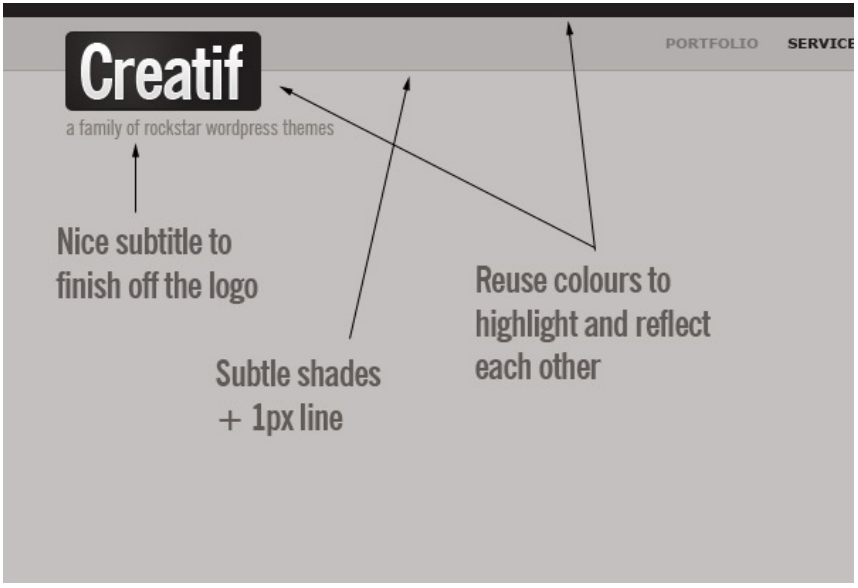
The canvas we're using is 1300px wide x 1400px high. In reality all the content is within 1000px so that it will be viewable on a 1024 x 768 screen. It's a good idea to have a wider canvas so we can plan for what happens when the viewer has a larger resolution.

In the image below, we've added the basics of the header, namely a dark bar along the top, a darker shade of the background colour as my menu bar, a 1px line to seal off the menu bar, and some subtext under the logo (in News Gothic Condensed again).

There are two things to note:

1. It's always nice to use shades of your colour palette in your design. Here we have beige as our background colour, then the menu bar, the menu items and the logo subtext are all varying, darker shades of that colour. This gives a nice, smooth, non clashing feel. Of course if you only use shades it gets pretty boring, that's why we introduce our highlight colour a little later. Different design styles will call for more variation in colour, but in our case we want mostly matching hues and shades with one strong highlight.

2. Additionally it's nice to reflect your colour across your design. So here we have the beige background colour and then our darker colour appears in three spots – the logo, the top bar and the highlighted menu link. This creates a visual balance and alignment between the three elements. Balance is important in creating a pleasing aesthetic.



Step 5

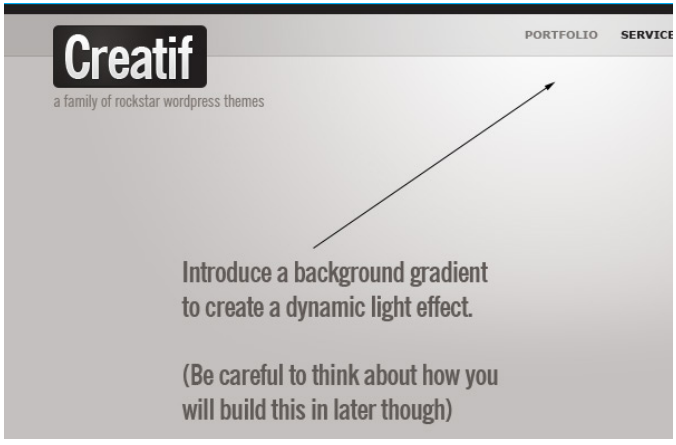
Here we add the first bit of our highlight colour. It's a really subtle 1px line along the top. Later as we add more elements the highlight colour will appear again in different spots, and will pull those elements together to unify them into a single, slick looking design. Because there's not much to this design except well placed elements and colour, it's very important to get the colouring right.

Step 6

Now the page is looking a bit flat, so here we have added a layer just above the background layer. In that layer we have a Radial Gradient going from the dark beige/grey colour to transparency, with a layer blending mode set to Colour Dodge to lighten the background. Because the menu bar is in fact drawn in with transparency the lightening effect shows through the menu bar too.

It's vital to remember though that you need to build this design into HTML later. For that reason you'll notice that by the time you get to the edges of the 1000px viewable area we're back to monotone colours. This means later on we'll be able to create a single image slice and use it as a CSS background image. Then we'll have another background image with the big highlight area and this will be a background image in the main content body.

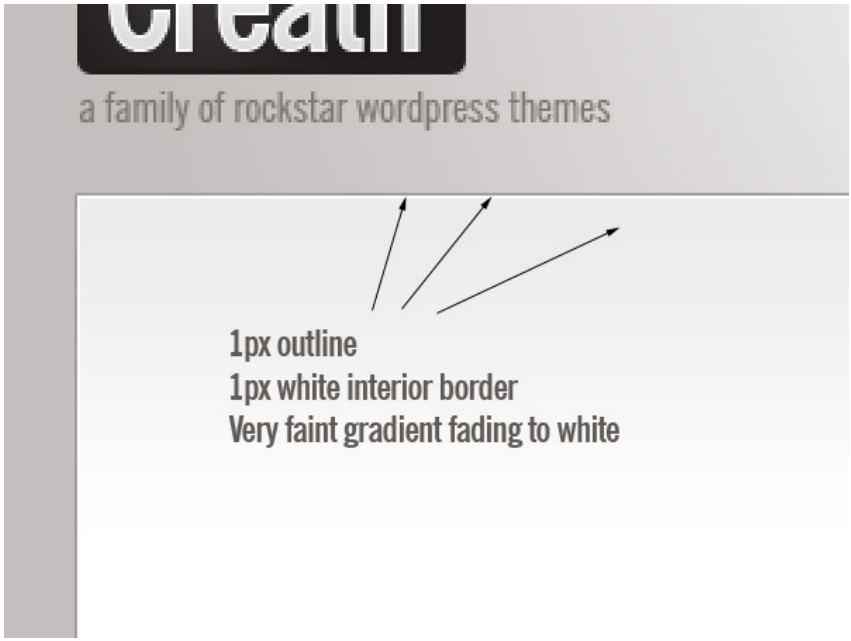
It's important to know about building sites so that you can design them in such a way as to avoid complications later down the track. This mostly comes from experience and learning what design decisions can make life troublesome during the build. Here things will be much easier if we have an easily repeatable background outside the 1000px viewable area.



Step 7

Next it's time to start adding the first white content block area. Here we have used a 1px outline of a darker version of the background colour, then a 1px interior border and finally a faint beige gradient going downwards. This style matches our logo. Additionally by having the darker outline, followed by lighter interior outline we get a very sharp look to the page. Visual sharpness or clarity comes from contrast – e.g the dark to light between the two lines.

Clarity is important in web design and is a key difference between traditional print work and on-screen graphics. So it's important to pay attention to detail and keep elements on the page looking clear and crisp.

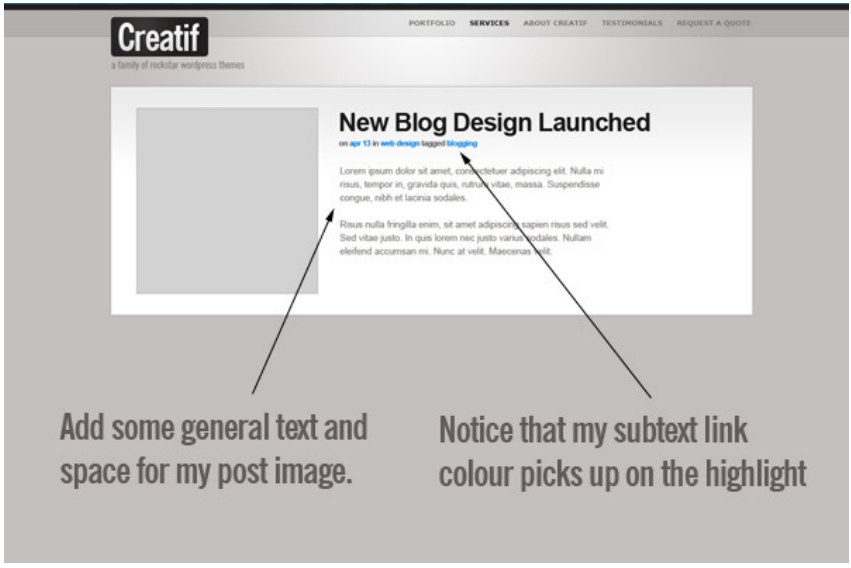


Step 8

Now I add some mock content in here. Because this text has to be HTML text it's important to choose your fonts carefully. There's nothing more depressing than choosing some nice fonts and then remembering later on that they aren't default fonts and so consequently your design is going to look totally different to how you had previously imagined. Here we've used Helvetica for the bold headline and Arial for the text.

In Photoshop it's a good idea to set the *Anti-Aliasing* to *"Sharp"* to mimick how the text will look in the browser. In the old days we would have used *"None"*, but these days most PCs and all Macs use that ClearType algorithms to smooth fonts.

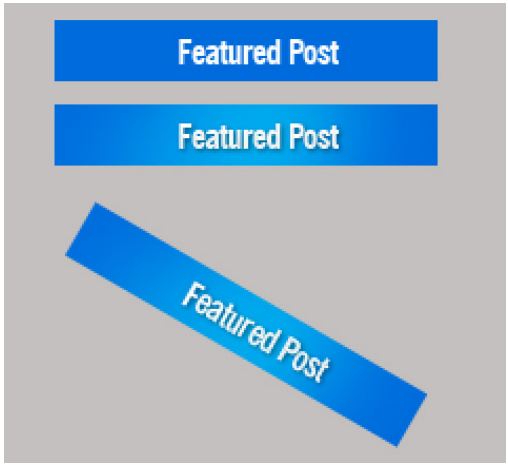
Note that the subtext links use our highlight blue, picking up on the 1px headling line we added earlier.



Step 9

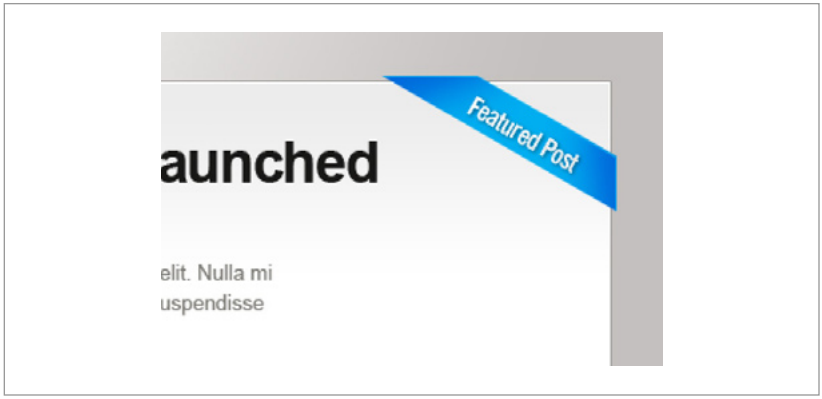
Next we will add a small, but eye-catching design element in the form of a message strip along the top right corner of the boxes. In a simple design like this (where it's mostly simple lines and boxes), it is a good idea to have one or two elements in the design that really leap out. In this case we're going to use our sharp blue colour combined with a 45' angle to make a great highlight for our design.

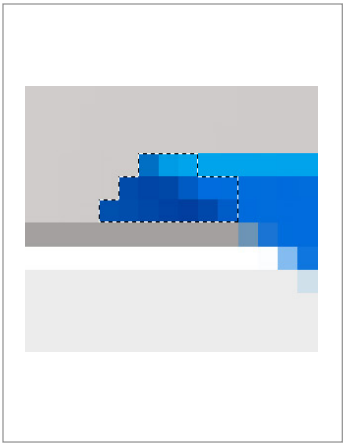
So we draw a rectangle and add some text over the top. Then use the *Dodge Tool (O)* to lighten the middle part, and add a Layer Style to give the text a bit of shadow. Then select both layers together, hit *CTRL-T* to transform and rotate 45'.



Step 10

After placing the strip over our box, we cut away the edges as shown. Now you'll notice we could have placed it so it was aligned with the box, but to add extra visual interest we're going to make it look like this strip is wrapping around the box, so instead we move it about 4px off the box to the right and top.





Step 11

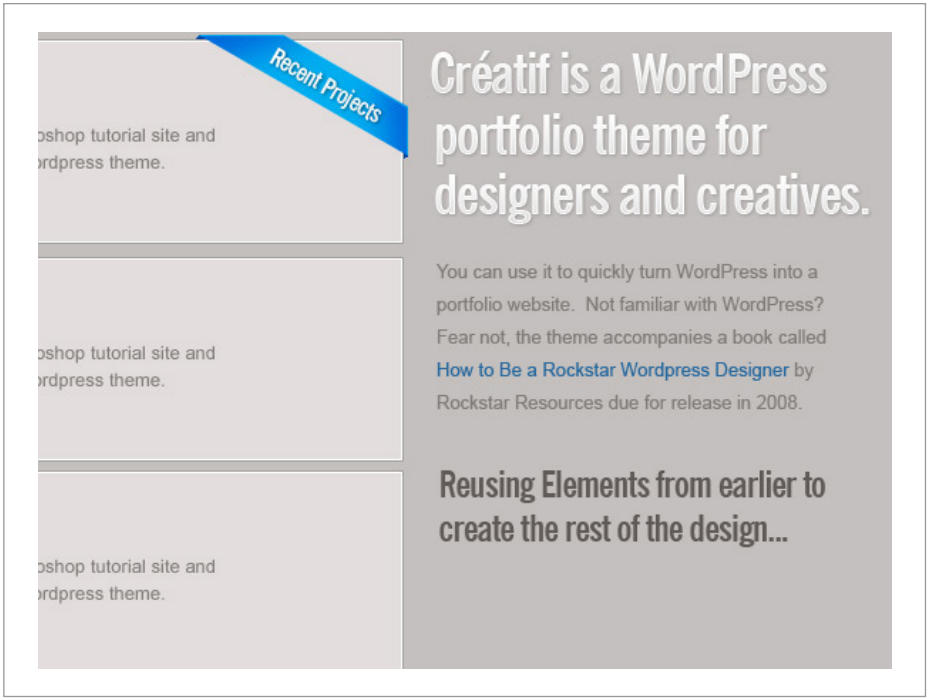
Next we manually select the pixels in the pattern you see below, create a layer below the message strip layer and draw in a darker blue colour. It's darker so it looks like the back of the message strip, and you'll notice that it is darker towards the right where the pixels are in a sort of mock shadow cast by the main strip.

Step 12

We then duplicate the wrap element, rotate it 90 degrees and place it on the right hand side of the box as well, as shown. And voila, we have our design element!

Step 13

Next we'll create some more elements. There's not much new here. We are basically reusing the same design elements – the same text style, the same message strip, the same boxes – carefully arranged as shown.



Step 14

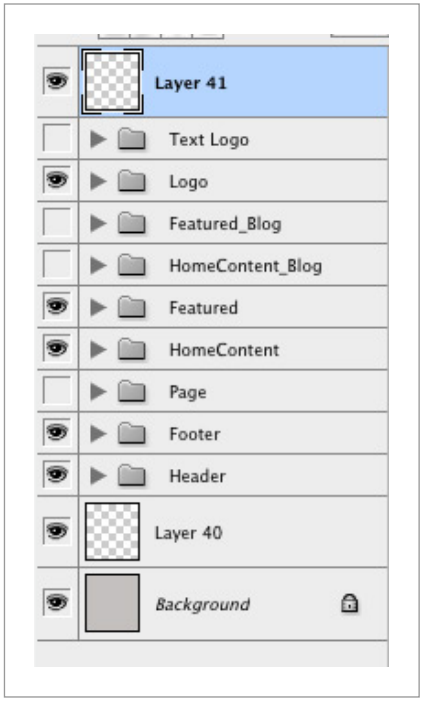
Next we'll add a footer area. Again we'll make use of the same colours as used in the top bar to reflect them yet again and in this case seal off the design.

Step 15

Now because we'll be creating a WordPress theme it's not a bad idea to create a version of the logo that could be created with plain text. You can see it below.

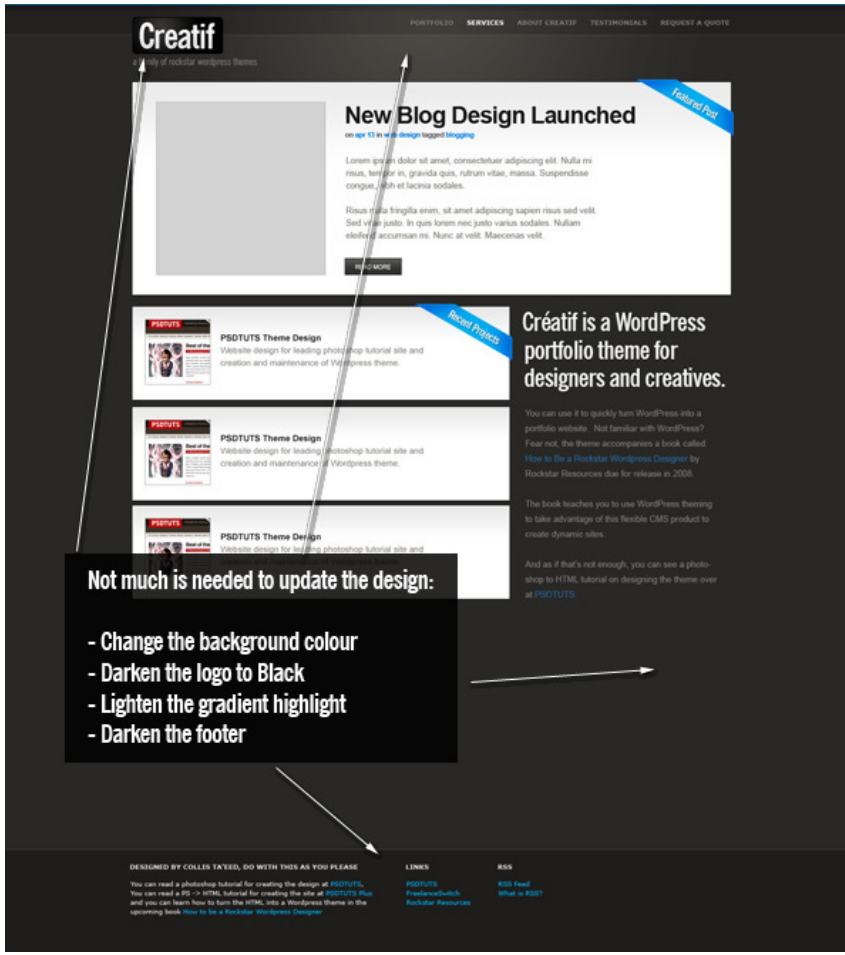
Step 16

At this point let's take a quick look at the Layers Palette so far. As you can see, we're taking advantage of grouping layers into sets. Here the design for the logo vs text logo, the blog vs portfolio and the internal page are all in the same PSD file, just in different layer sets. So we can switch them on and off and get different arrangements. This is useful because if we suddenly decide to move the logo 2px to the left, it's not necessary to open up three files and move it 2px in each or risk having discrepancies. Additionally it's just nice and ordered and will have you feeling all warm and fuzzy just looking at it!



Step 17

Finally, we'll put together a second alternate vesion of the design using a dark brown background. Although it looks quite different, there isn't actually much that needs to change, we simply darken the logo to black so it stands out still and adjust a few other colours to make the design all make sense.



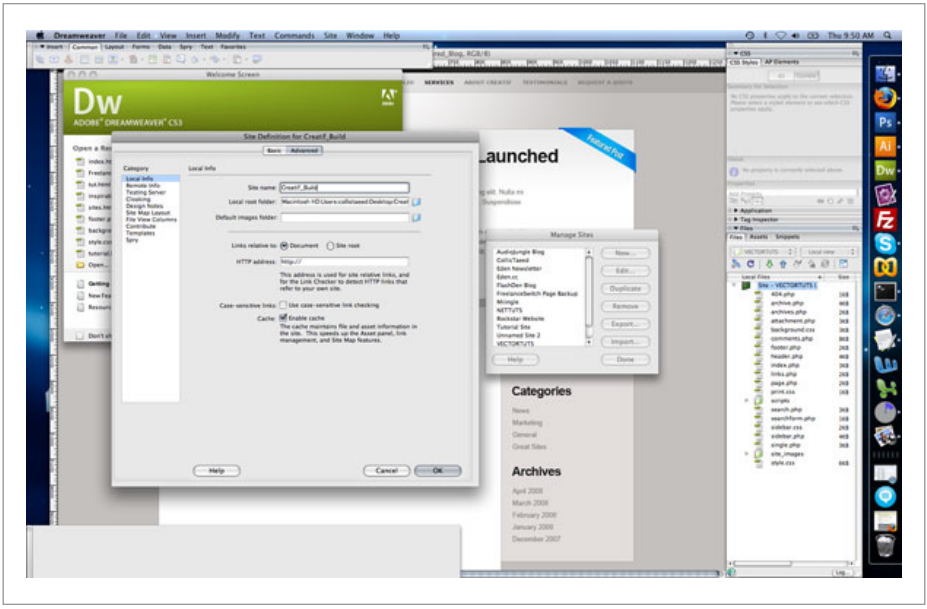
The Creatif HTML Tutorial – From PSD to HTML

Now that we have a complete set of designs in Photoshop, it's time to transform our mockups into a fully functioning HTML site. Then in the following chapters we'll add WordPress functionality to the HTML to create our final themes.

Again you can grab the finished HTML from the files that came with this book, or alternately follow along. We're going to approach the build in stages. First we'll do the framework, then the first page, then alternate pages, then finally an alternate colour scheme.

Step 1 – Getting Ready

So first of all we boot up our code editor of choice. Additionally it's good to set up a directory structure that includes an `/images/` directory and a `/scripts/` directory.



Step 2 – Quick Early Layout

The first thing we'll do is a quick overall layout in HTML with some barebones CSS just to make sure we've got a solid foundation. We can also check it in the major browsers (IE7, IE6, Firefox, Safari) just to make sure we're on a solid footing. There is nothing worse

than coming back all the way to the beginning to fix browser compatibility issues. It's much better to do it as you go.

So we're building the first mockup, we can see a few things:

1. The design is centred. That immediately tells us we have to wrap it in a container and then centre that container.

2. Essentially the design is a series of horizontal blocks. Sometimes the blocks have two columns, sometimes one. So we can do it as a series of `<div>`'s. This is good because we can then mix and match elements into different pages as you'll see later.

3. We have a footer which is a different colour. This means the background needs to be that colour, in case the users browser stretches. So the footer will need to sit in a different container to the main stuff.

So here's a HTML layout:

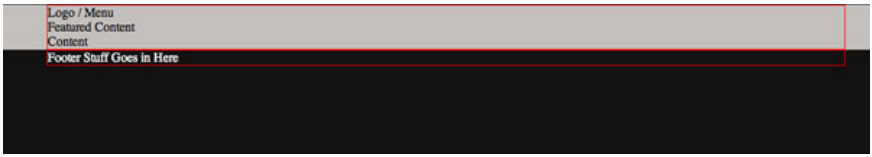
```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <meta http-equiv="Content-Type" content="text/html;
    charset=UTF-8" />
    <title>Creatif</title>
    <link href="style.css" rel="stylesheet" type="text/css" />
</head>
<body>
    <div id="main">
            <div class="container">
                    <div id="header">
                    Logo / Menu
            </div>
```

```
            <div id="block_feature">
                Featured Content
            </div>
            <div id="block_content">
                Content
            </div>
        </div>
    </div>
    <div id="footer">
            <div class="container">
                Footer Stuff Goes in Here
            </div>
        </div>
</body>
</html>
```

As you can see there are two segments: the `#main` area and the `#footer` area. Inside each we have a `<div class="container">` element which will be fixed width and centred. Then inside the main container we just have a sequence of `<div>`'s. Now let's add a little CSS as follows:

```
body {
    margin:0px; padding:0px;
    background-color:#131211;
}
#main {
    background-color:#c4c0be;
}
#footer {
    color:white;
}
.container {
    width:950px;
    margin:0 auto;
    border:1px solid red;
}
```

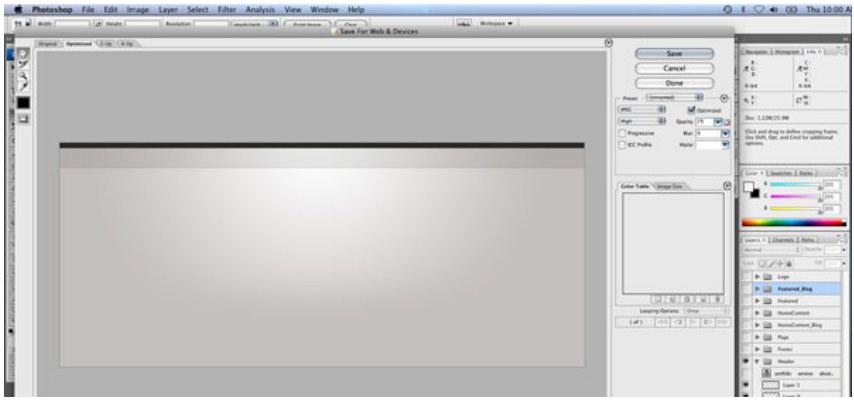
So we're setting the body's background colour to the dark brown of the footer. Then the `#main` area has the lighter background. Finally you can see the `.container` elements have a width of 950px and are centred using `margin: auto`. We've also added a red border just so you can see where the elements are on the page.



Step 3 – Add Some Background Images

So our layout is looking ship shape. With the main elements positioned, it's just a matter of going through and styling it all up. The first thing we need are some images. You can make these yourself or just grab them from the demo files.

Here's a screenshot of Photoshop, saving the first image – a large background JPG. This large background image will be used to get that radial gradient highlight, then we'll use a thin 1px slice to fill out the left and right sides so it extends off.



Similarly we'll create a background image for the footer to tile along as a border between it and the main area (you can find that image in the ZIP file, it's called *background_footer.jpg*). Now we'll update the CSS file to remove that red border and add our new background images, as follows:

```
@charset "UTF-8";
/* Background-Styles */
body {
    margin:0px; padding:0px;
    background-color:#131211;
}
#main {
    background:#c4c0be url(images/background_light_slice.
    jpg) repeat-x;
}
#main .container {
    background-image:url(images/background_light.jpg);
    background-repeat:no-repeat;
    min-height:400px;
}
#footer {
    background-image:url(images/background_footer.jpg);
    background-repeat:repeat-x;
    color:white;
    padding:40px;
}
.container {
    width:950px;
    margin:0 auto;
    position:relative;
}
```

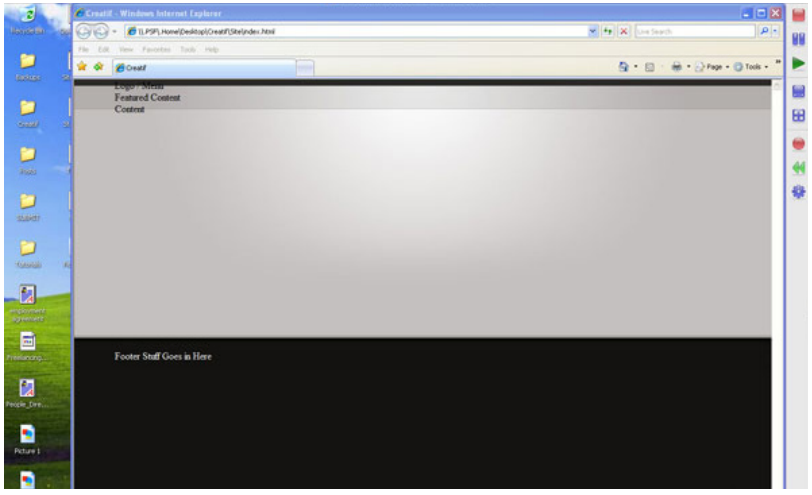
Two things to note:

1. There are multiple ways to set a background. In `#main` we've used a single selector which sets three properties – colour, image, image repeat. But you can also set each property individually as we have done in `#main .container` and `#footer`.

2. Notice that because we want to apply the *"background_light.jpg"* image to the `<div class='container'>` which is inside `#main`, but not to the one that is inside `#footer`, we've written `#main .container`. In other words, we apply it only to elements with the `class='container'` that are inside elements with `id='main'`.

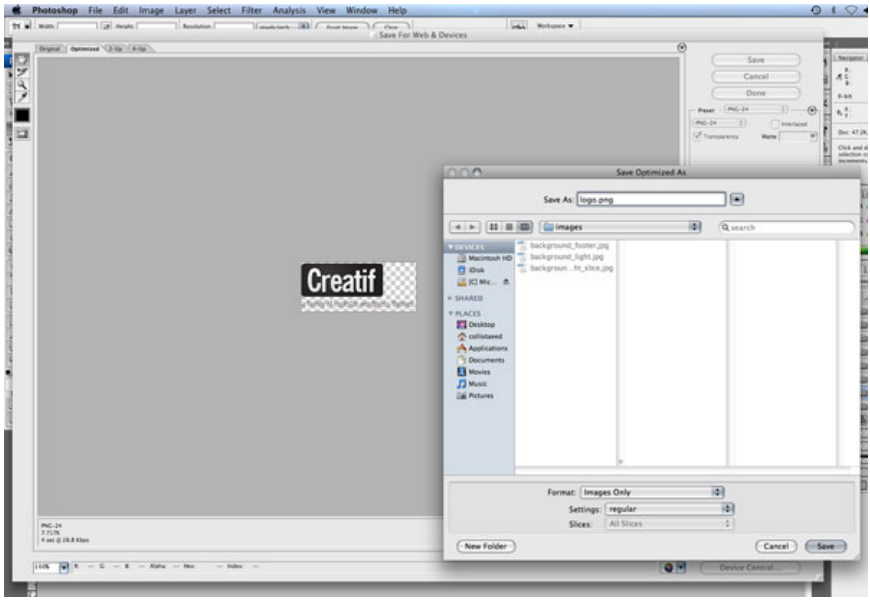
Step 4 – Testing in Browsers

So far so good. Don't forget to test in different browsers. Here you can see in IE7 it's looking fine!



Step 5 – Making a Transparent Logo

Next we've created the logo element. Because later on we'll be running an alternate colour scheme we're going to use a transparent background PNG file. You can make these by switching off the background in Photoshop and then going to *File* > *Save for Web and Devices* and selecting *PNG-24*. You should be aware that PNG-24 produces pretty high file sizes. It's OK for a small image like this, but for larger ones they can get quite big.



Now we'll add our logo and also a menu with this HTML:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
```

```
        <meta http-equiv="Content-Type" content="text/html;
        charset=UTF-8" />
        <title>Creatif</title>
        <link href="step_2.css" rel="stylesheet" type="text/
        css" />
        <link rel="shortcut icon" href="images/favicon.ico" />
    </head>
    <body>
        <div id="main">
                <div class="container">
                <div id="header">
                        <ul id="menu">
                        <li><a href="">Portfolio</a></li>
                        <li><a href="">Services</a></li>
                        <li><a href="">About</a></li>
                        <li><a href="">Testimonials</a></li>
                        <li><a href="">Request a Quote</a></li>
                </ul>
                <div id="logo">
                        <h1>Creatif</h1>
        <small>A Family of Rockstar Wordpress
        Themes</small>
                </div>
        </div>
        <div id="block_feature">
                Featured Content
        </div>
        <div id="block_content">
                Content
        </div>
        </div>
    </div>
        <div id="footer">
                <div class="container">
                        Footer Stuff Goes in Here
                </div>
```

```
        </div>
    </body>
    </html>
```

and this extra CSS:

```
    #header {
        padding-top:20px;
    }
    #logo h1, #logo small {
        margin:0px;
        display:block;
        text-indent:-9999px;
    }
    #logo {
        background-image:url(images/logo.png);
        background-repeat:no-repeat;
        width:194px;
        height:83px;
    }
    ul#menu {
        margin:0px; padding:0px;
        position:absolute;
        right:0px;
    }
    ul#menu li {
        display:inline;
    }
```

Some things to note:

1. Rather than just placing the logo image in the HTML, we've created a `<div id="logo">` and inside that placed a `<h1>` with the title. Then using CSS to create a massive text indent we've made the text vanish and swapped it for the logo image. This has some SEO benefits.

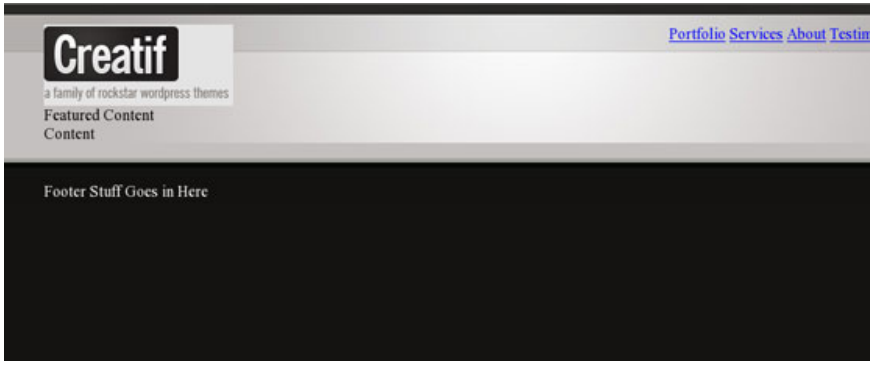
2. We've placed a very quick, unstyled menu using an unordered list. By setting the display property to inline for the `<li>` elements, the list changes to a horizontal set of elements.

3. Finally because our `<div class="container">` element has `position:relative`, we can now use absolute positioning inside and set right:0px for the menu and it will be aligned to the right. This is great for a WordPress theme because as the person creates new pages the menu will extend, and this way it will stay right aligned.

Step 6 – Fixing Transparency in IE6

Now the one problem with transparent PNGs is that our friend Internet Explorer 6 doesn't support them! Fortunately that's relatively easily fixed with a quick hack via this website: http://bjorkoy.com/past/2007/4/8/the_easiest_way_to_png/. We just download a script and add this line in our CSS:

```
/* Fix up IE6 PNG Support */
img, #logo { behavior: url(scripts/iepngfix.htc); }
```

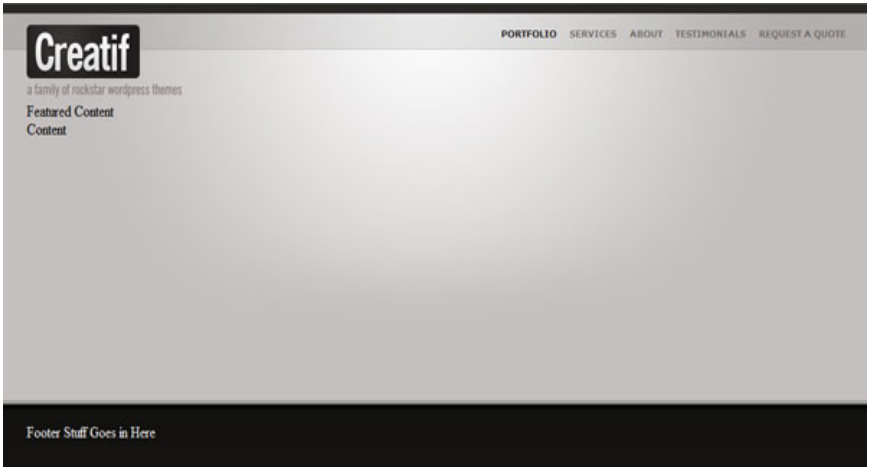


Step 7 – Fixing up the Menu

Now our menu is still looking pretty ugly, so let's add a few styles to finish it off, as follows:

```
ul#menu {
    margin:0px; padding:0px;
    position:absolute;
    right:0px;
}
ul#menu li {
    display:inline;
    margin-left:12px;
}
ul#menu li a {
    text-decoration:none;
    color:#716d6a;
    font-family:Verdana, Arial, Helvetica, sans-serif;
    font-size:10px;
    font-weight:bold;
    text-transform:uppercase;
}
ul#menu li a.active, ul#menu li a:hover {
    color:#211e1e;
}
```

Nothing very exciting here except that we've defined an "active" style which is the same as the `:hover` style (namely it's a darker shade). That means we can write `<a href="" class="active">` and the link will darken.



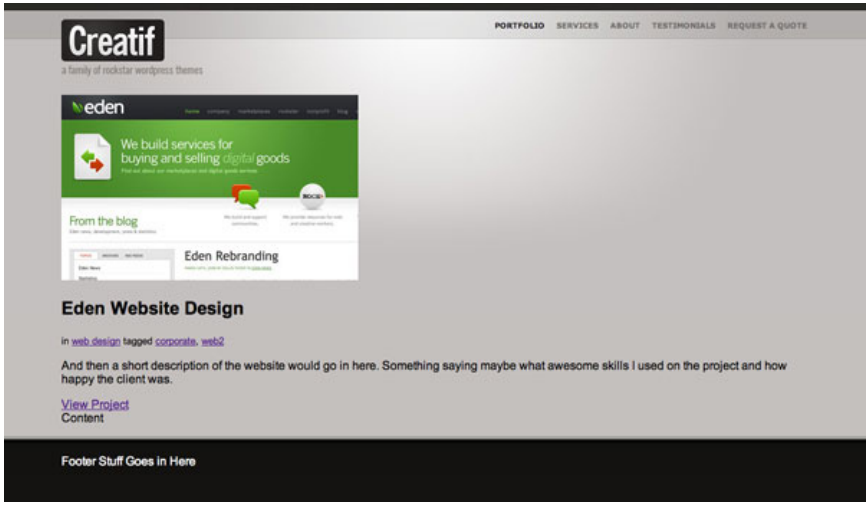
Step 8 – Adding the Featured Portfolio Item Content

Now we have the base of our page laid out, it's time to start adding the content blocks. As mentioned earlier we are going to make this site as a series of interchangeable content blocks. The first one is the "Featured Project" block. So let's add some HTML:

```
<div id="block_featured" class="block">
    <span class="block_inside">
            <div class="image_block">
            <img src="images/sample_feature.jpg" />
    </div>
    <div class="text_block">
            <h2>Eden Website Design</h2>
            <small>in <a href="">web design</a> tagged
            <a href="">corporate</a>, <a href="">web2</a></
            small>
            <p>And then a short description of the website
            would go in here. Something saying maybe what
```

```
                awesome skills I used on the project and how
                happy the client was.</p>
                <br />
                <a href="" class="button">View Project</a>
                </div>
        </span>
</div>
```

So that code goes below the `<div id="header"></div>` code from the previous steps. And unstyled it looks like this:



There are two important things to note here:

1. You will see that we have a `<div class="block">` followed immediately by a `<span class="block_inside">`. This is because the boxes we are drawing have a double border, first there is a 1px dark grey border, then inside that a 1px white border. So having two elements means we can have a border on each.

2. Where we have the View Project button, instead of using an image, we're going to create a 'button' class and then apply it to regular text links. This makes for a very simple, reusable button look and feel.

Step 9 – Adding some Basic Styles

Now we apply some basic styling like this:

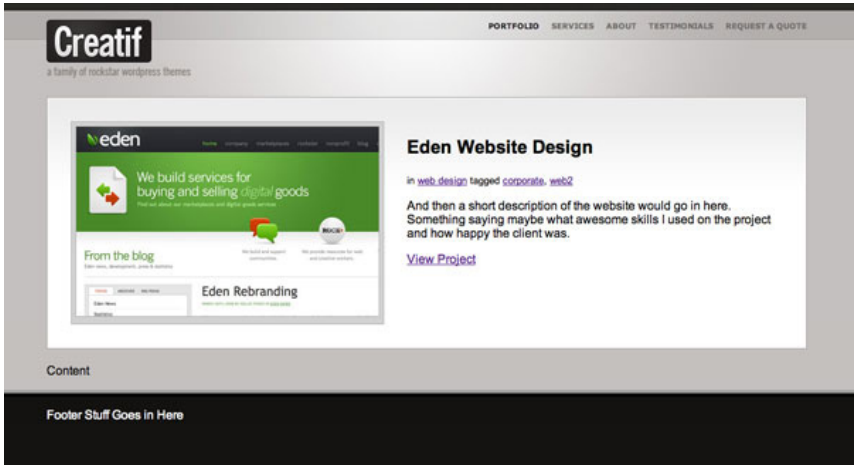
```
/*
    Block-Styles
*/
.block {
    border:1px solid #a3a09e;
    background-color:#ffffff;
    margin-bottom:20px;
}
.block_inside {
    display:block;
    border:1px solid #ffffff;
    background: #ffffff url(images/background_block_slice.jpg)
    repeat-x;
    padding:30px;
    overflow:auto;
}
.image_block {
    border:1px solid #b5b5b5;
    background-color:#d2d2d2;
    padding:5px;
    float:left;
}
.image_block img {
    border:1px solid #b5b5b5;
}
.text_block {
```

```
        float:left;
        width:430px;
        margin-left:30px;
    }
```

So as mentioned above we have the `.block` class which just sets a border and bottom margin. Then immediately inside we have the `.block_inside` element which has a white border, thin slice background (to give it that faint gradient), some padding and finally an overflow value.

We have `overflow:auto` because we are going to have two floated elements inside. Then inside we have an `.image_block` class which gives our image a double border (one on the `<div>` and one on the `<img>` itself) and which is floated left with our `main .text_block` also floated left to form a mini columned layout.

So our layout now looks like this:



Step 10 – Adding Text Styles

Now the text styling is all over the place at the moment. It sort of looked OK in the previous screenshot because Firefox (used for the screenshot) defaults to a Sans-Serif font. But if we'd screenshotted Internet Exploere you would have seen a Serifed typeface instead. So we should get the text sorted out now. We'll add these bits of CSS to our stylesheet:

```
body {
    margin:0px; padding:0px;
    background-color:#131211;
    font-family:Arial, Helvetica, sans-serif;
    color:#7f7d78;
    font-size:13px;
    line-height:19px;
}
/*
    Text-Styles
*/
h2 {
    margin:0px 0px 10px 0px;
    font-size:36px;
    font-family:Helvetica, Arial, Sans-serif;
    color:#000000;
}
small {
    color:#595856;
    font-weight:bold;
    font-size:11px;
    display:block;
    margin-bottom:15px;
}
a {
    color:#007de2;
    text-decoration:none;
}
```

```
a:hover { text-decoration:underline; }
p { margin: 0px 0px 15px 0px; }
a.button {
    background:#32312f url(images/button_bg.jpg) repeat-x;
    padding:5px 10px 5px 10px;
    color: #ffffff;
    text-decoration: none;
    border:1px solid #32312f;
    text-transform:uppercase;
    font-size:9px;
    line-height:25px;
}
a.button:hover {
    background:#007de2 url(images/button_bg_o.jpg) repeat-x;
    border-color:#007de2;
}
```

Some notes on the code:

1. First we've updated the body tag to have a default font, colour, size and line-height.

2. Then we've created a `<h2>` style which fixes the margins and sets the font to Helvetica.

3. We've also created a `<small>` style for subheadings (like what category a post is in).

4. We've created a link style and `link:hover` style.

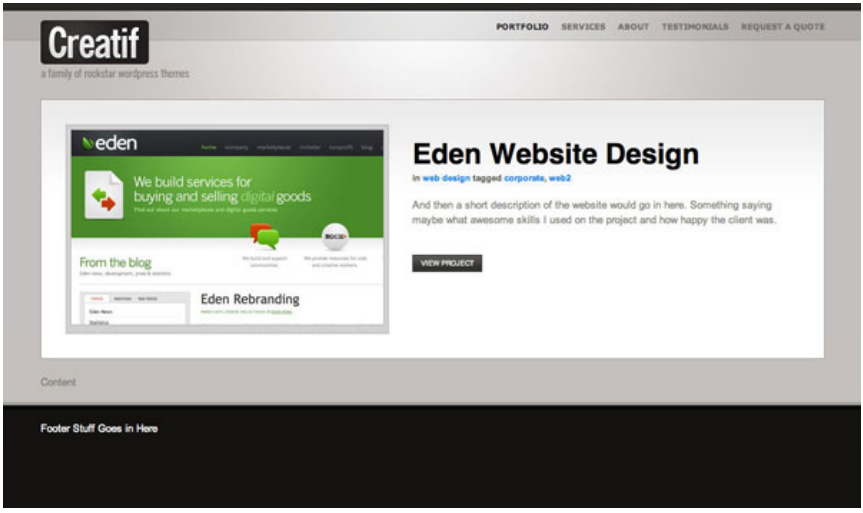
5. We've reset the `<p>` styling so that the margins are fixed from the stupid defaults.

6. Finally we've created that button class. Note that we've defined it as "`a.button`", or in other words all `<a>` tags with the `class = "button"`. Why didn't we just make

it "`.button`"? Well later on there is a good chance that we will make a second button class for `<input>`'s and it will be slightly different. So this way they won't accidentally interact.

7. In the button class you will see we've set some padding, a border, a background image, a hover style and a line-height attribute ... wait a line-height attribute? Yes unfortunately this is a fix for IE which otherwise cuts off the button.

With our extra styling, the page is starting to take shape!



Step 11 – Adding the Ribbon

One of the neat things about this design is the little blue ribbon strips in the right corner. Thanks to a mix of CSS, transparent PNG files and absolute positioning, these are really easy to add. So first we need to make the image. Once again we create an image with a transparent background and save it as PNG-24, here's the image:

Next we need to place the image in our HTML, we can do it like this:

```
        <div class="block">
                <img src="images/ribbon_featured.png" class="ribbon"/>
        <span class="block_inside">
        <div class="image_block">
        <img src="images/sample_feature.jpg" />
</div>
<div class="text_block">
        <h2>Eden Website Design</h2>
        <small>in <a href="">web design</a> tagged <a
        href="">corporate</a>, <a href="">web2</a></small>
        <p>And then a short description of the website would go
        in here. Something saying maybe what awesome skills I
        used on the project and how happy the client was. </p>
        <br />
        <a href="" class="button">View Project</a>
        </div>
</span>
</div>
```

So you can see the `<img>` tag there on the second line. Note we've given it a `class="ribbon"` and put it inside the `.block` element, but outside the `.block_inside` element. That's because if we do it inside `.block_inside` it messes up the `overflow:auto` property we set earlier. Right now this will just mess up our layout, so let's add some styling:

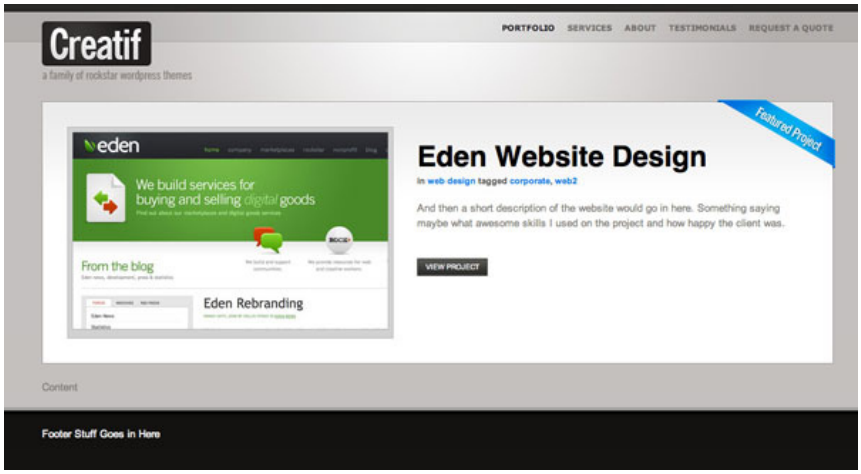
```
.block {
    border:1px solid #a3a09e;
    background-color:#ffffff;
    margin-bottom:20px;
    position:relative;
}
.ribbon {
    position:absolute;
    top:-3px;
    right:-3px;
}
```

You can see that we've:

1. Added a `position:relative` attribute to the `.block` element. This is so that we can use absolute positioning inside and have it relative to the `.block` element (and not the whole page).

2. Then we've set the image to appear 3px past the right edge and 3px past the top edge.

Easy! Back in the day, we would have had to use some super complicated `<table>` layout to achieve that same effect. Here's how it's looking now:



Step 12 – Creating the Second Block

With the ribbon added, our first block element is complete! Now it's time to start on the next `<div>` block. This one will have that text about the theme and the recent projects list. So first we add some HTML:

```
        <div id="block_portfolio">
  <div id="portfolio_items">
        <div class="mini_portfolio_item">
        <div class="block_inside">
            <img src="images/sample_mini_portfolio.jpg"
           class="thumbnail" alt="PSDTUTS"  />
            <h3>PSDTUTS Theme Design</h3>
            <p>Website design for leading photoshop
           tutorial site and creation and maintenance of
           Wordpress theme.</p>
            <a href="#" class="button">View Project</a>
        </div>
    </div>
```

```
            <div class="mini_portfolio_item">
            <div class="block_inside">
                <img src="images/sample_mini_portfolio.jpg"
               class="thumbnail" alt="PSDTUTS"  />
                <h3>PSDTUTS Theme Design</h3>
                <p>Website design for leading photoshop
               tutorial site and creation and maintenance of
               Wordpress theme. </p>
                <a href="#" class="button">View Project</a>
            </div>
        </div>
        <div class="mini_portfolio_item">
            <div class="block_inside">
                <img src="images/sample_mini_portfolio.jpg"
                class="thumbnail" alt="PSDTUTS"  />
                <h3>PSDTUTS Theme Design</h3>
                <p>Website design for leading photoshop
                tutorial site and creation and maintenance of
                Wordpress theme. </p>
                <a href="#" class="button">View Project</a>
            </div>
        </div>
    </div>
    <div id="text_column">
            <h2 id="text_title">Creatif is a WordPress
            portfolio theme for designers and creatives</h2>
            <p>You can use it to quickly turn WordPress into a
            portfolio website.  Not familiar with WordPress?
            Fear not, the theme accompanies a book called <a
            href="#">How to Be a Rockstar Wordpress Designer
            </a> by Rockstar Resources due for release in
            2008.</p>
            <p>The book teaches you to use WordPress theming
            to take advantage of this flexible CMS product to
            create dynamic sites.</p>
```

```
            <p>And as if that's not enough, you can see a
            photoshop to HTML tutorial on designing the theme
            over at <a href="http://psdtuts.com">PSDTUTS</a>
            and <a href="http://nettuts.com">NETTUTS</a>.</p>
                </div>
    </div>
```

So that looks like lots of code, but it's not really. Let's go through it:

1. First we've created a container `<div id="block_portfolio">` to wrap up the code segment.

2. Next we've got a `<div id="portfolio_items">` which contains three identical `<div class="mini_portfolio_item">`'s. We'll talk about these in a second.

3. Next we have a `<div id="text_column">` which is filled with some text and a `<h2>` heading.

4. What we are going to do is float the text column and portfolio items side by side to form two columns of content.

5. We're going to replace that `<h2>` with a background image.

6. And we'll style up those mini_portfolio_item divs to look nice using a similar double border effect as we did earlier.

Here's the CSS:

```
/*
    Portfolio-Home-Styles
*/
#block_portfolio {
    overflow:auto;
    margin-bottom:20px;
}
#portfolio_items {
    width:615px;
    margin-right:25px;
    float:left;
}
#text_column {
    float:right;
    width:310px;
}
#text_column h2#text_title {
    text-indent:-9999px;
    background-image:url(images/creatif.jpg);
    background-repeat:no-repeat;
    width:310px;
    height:129px;
}
.mini_portfolio_item {
    border:1px solid #a3a09e;
    margin-bottom:10px;
}
.mini_portfolio_item .block_inside {
    background:none; background-color:#e2dddc;
    padding:25px 30px 15px 30px;
}
.mini_portfolio_item .thumbnail { float:left; margin-
right:20px; border:1px solid #979390; }
```

OK again, looks like a lot, but it's not too bad. Let's go through it step by step:

1. First we've again used `overflow:auto` on the main `#block_portfolio` element. That's because we again have two floated columns and if we don't do this, they'll run over the footer.

2. Next we've set `#portfolio_items` to float to the left, have a margin to separate it from the text column and a width of 615px.

3. The `#text_column` is set to float to the right with a width of 310px.

4. Inside the text column we've again done that trickery with our `<h2>` tag where we use a massive text-indent to make the text disappear and then instead use a background image.

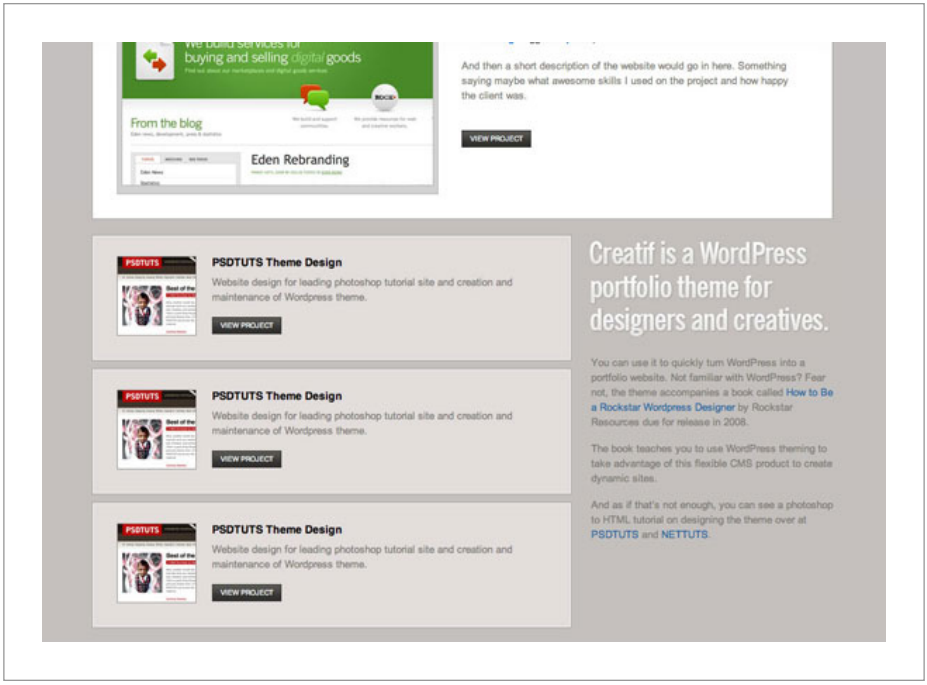
Next we have three style definitions for the *mini_portfolio_item* elements as follows:

1. First we set a 1px dark border and a margin between them.

2. Next we redefine the `.block_inside` styles to suit these elements. Remember `.block_inside` was defined earlier when we did the Featured Project area. So here we are overriding the background image, changing the background colour and changing the padding.

3. Finally we make the thumbnail images float left and have a border.

So all in all it's looking like this:



Step 13 – Adding a Ribbon.

Now we want to add a "Recent Projects" ribbon to the top most item. To do this we simply slot it in, in the same position in the HTML as previously, like this:

```
<div class="mini_portfolio_item">
<img src="images/ribbon_recent.png" class="ribbon"
alt="Recent Projects"/>
<div class="block_inside">
    <img src="images/sample_mini_portfolio3.jpg"
    class="thumbnail" alt="AudioJungle" />
    <h3>AudioJungle Site Design</h3>
    <p>Website design for leading photoshop tutorial site
```

```
        and creation and maintenance of Wordpress theme. </p>
        <a href="#" class="button">View Project</a>
    </div>
</div>
```

Then we add a `position:relative` attribute to the *mini_portfolio_item* element like this:

```
    .mini_portfolio_item {
        border:1px solid #a3a09e;
        margin-bottom:10px;
        position:relative;
    }
```

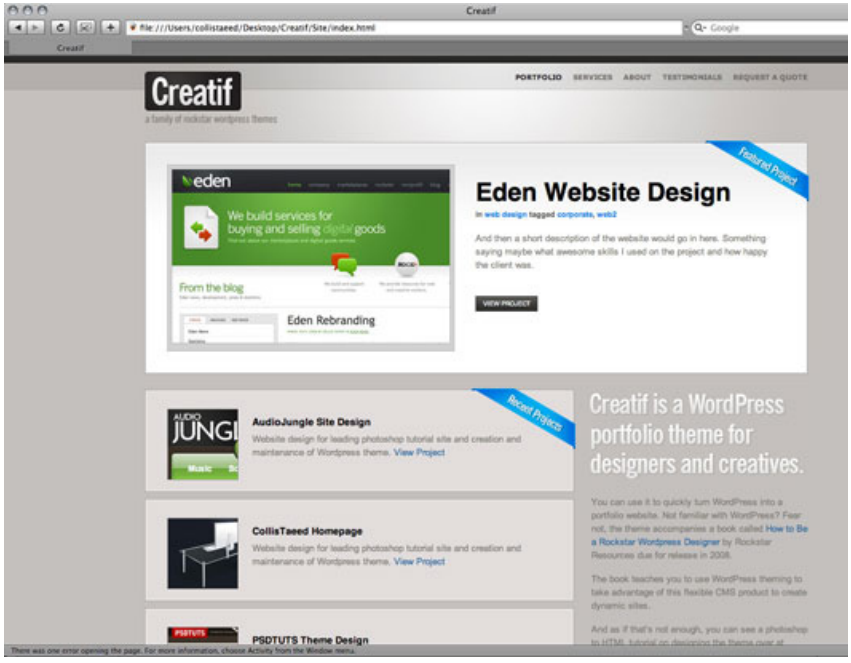
But something strange happens: while the right hand side looks correct, the top is getting cut off, as you can see in the screenshot:

The reason is that the element that our *mini_portfolio_item* is sitting inside is cutting it off. So we check up and see that the *mini_portfolio_item*'s are all inside a `<div id="portfolio_items">`. So the solution is pretty easy, we add 3px of padding to the top which is just enough space for our ribbon to show through. Here's the adjusted CSS:

```
#portfolio_items {
    width:615px;
    margin-right:25px;
    float:left;
    padding-top:3px;
}
```

Step 14 – Finishing off the Portfolio Items

Finally we've swapped in a few images and titles so we can see how the page looks with 3 different items instead of the same one repeated. Then we've also gotten rid of the View Project button and gone with just a text link. This looks a bit cleaner and less busy. So here's the final portfolio items section (shown in Safari, don't forget to keep testing in different browsers!)



Step 15 – Adding Footer Content

Now there is just one more section to our page: the footer! Let's add some text content to it:

```
<div id="footer">
<div class="container">
            <div class="footer_column long">
            <h3>Designed by Collis Ta'eed, do with this as
            you please</h3>
            <p>You can read a photoshop tutorial
            for creating the design at <a href="http://
            psdtuts.com">PSDTUTS</a>, You can read a
            PS->HTML tutorial for creating the site at
            <a href="http://nettuts.com">NETTUTS</a> and
            you can learn how to turn the HTML into a
            Wordpress theme in the upcoming book
            <a href="http://freelanceswitch.com/book">How to
            be a Rockstar Wordpress Designer</a></p>
                   </div>
            <div class="footer_column">
            <h3>More Links</h3>
                          <ul>
                <li><a href="http://vectortuts.
                com">VECTORTUTS</a></li>
                   <li><a href="http://flashden.
                   net">FlashDen</a></li>
                   <li><a href="http://audiojungle.
                   net">AudioJungle</a></li>
                   <li><a href="http://freelanceswitch.
                   com">FreelanceSwitch</a></li>
                   <li><a href="http://faveup.com">FaveUp
                   </a></li>
            </ul>
         </div>
         <div class="footer_column">
```

```
            <h3>RSS</h3>
                        <ul>
                <li><a href="">RSS Feed</a></li>
                <li><a href="">What is RSS?</a></li>
            </ul>
        </div>
    </div>
</div>
```

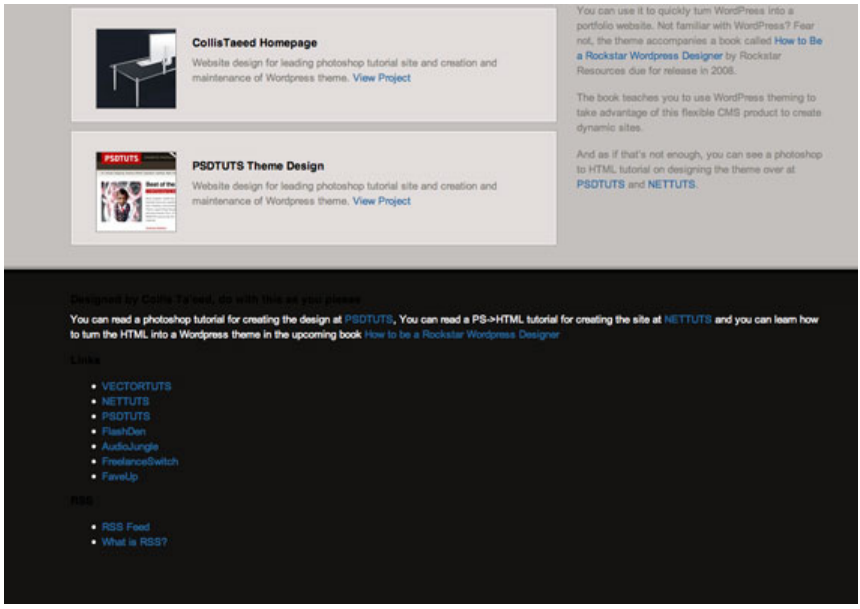
A few things to note:

1. We've created three `<div class="footer_column">`'s to house the content of the footer, we'll float these into place in a second.

2. Since the first column is a different width we'll give it a second class called *"long"*. Note that you set two classes like this: `class="class1 class2"`, not like this: `class="class1" class="class2"` which is invalid markup.

3. Inside the columns I've used `<ul>` lists and `<h3>` tags for the headings. It's always good to use nice semantic markup, both because it makes it more readable, and because search engines like to see those headings and lists all laid out properly.

Here's how it's looking!



Step 16 – Styling the Footer

Styling the footer is a pretty simple job, here's the code we need:

```
/*
    Footer-Styles
*/
#footer {
    font-family:Verdana, Arial, Helvetica, sans-serif;
    font-size:10px;
}
.footer_column {
    float:left;
    width:120px;
    margin-right:30px;
}
```

```
#footer .long {
    width:610px;
}
#footer h3 {
    color:#e2dddc;
    text-transform:uppercase;
    font-size:10px;
}
.footer_column ul li, .footer_column ul {
    list-style:none;
    margin:0px;
    padding:0px;
}
```

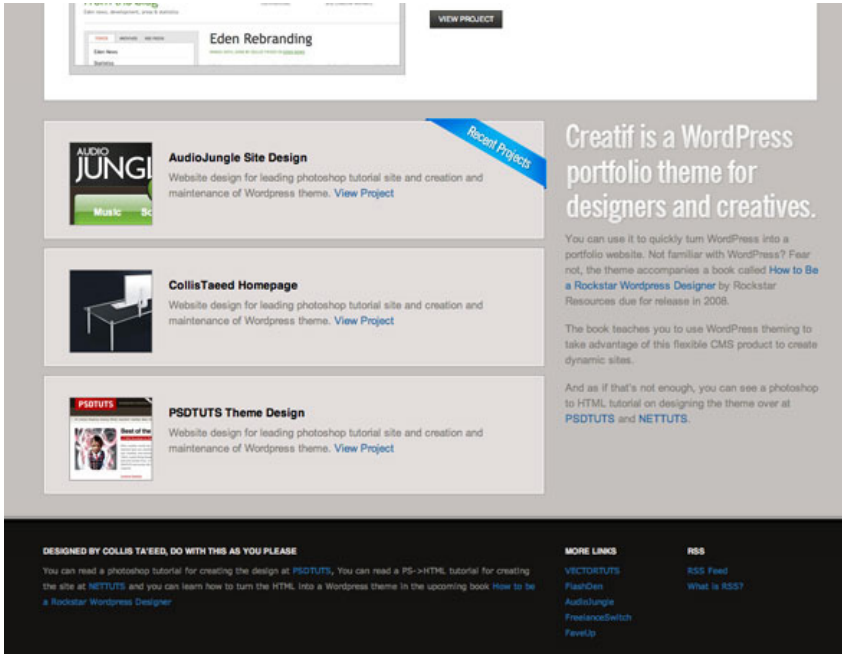
Going through:

1. First we set the fonts for the `#footer` area.

2. Then we set all the columns to float with a default width of 120px.

3. We override this width for the `.long` column. Notice that we've set "`#footer .long`" instead of just "`.long`". The reason we do this is that "long" is the kind of generic name that might get used again later somewhere else, so it is a good idea to make the definition more specific to avoid confusion.

4. Finally the `<h3>` and `<ul>` tags get some simple styles.

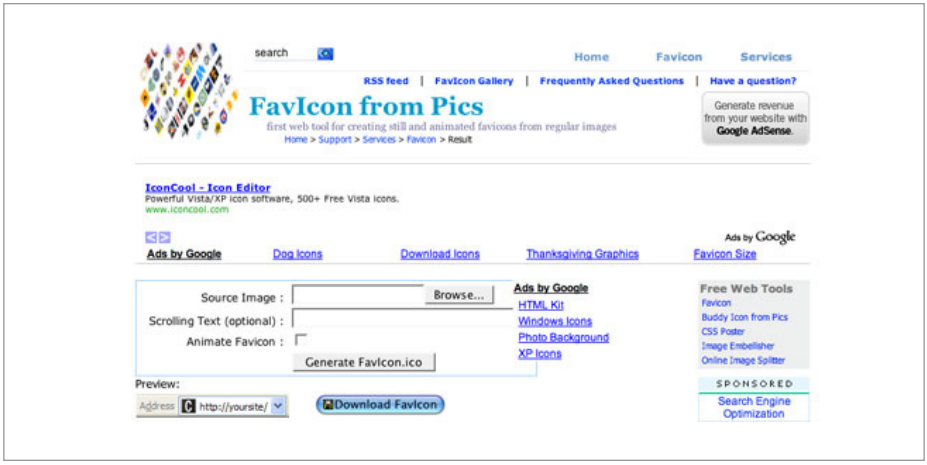


Step 17 – Adding a Favicon!

We're almost finished our first page. It's time to add a few small details. First we'll create a favicon – one of those little icons that appear in your browser bar. This little black square with a C for Creatif will do nicely. So first we create a square image of what we want in Photoshop.

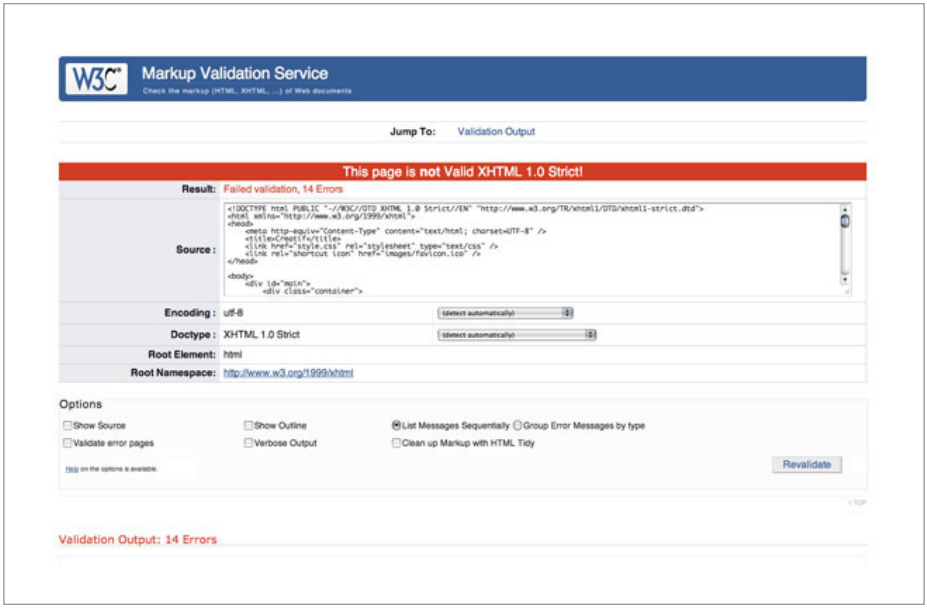
There are lots of sites to make Favicons, I've used this one: http://www.html-kit.com/favicon/. You simply upload the image and hit *Generate Favicon.ico*. Place the *.ico* file into `your /images/ directory` and then hook it up with this line of HTML:

```
<link rel="shortcut icon" href="images/favicon.ico" />
```



Step 18 – Validating!

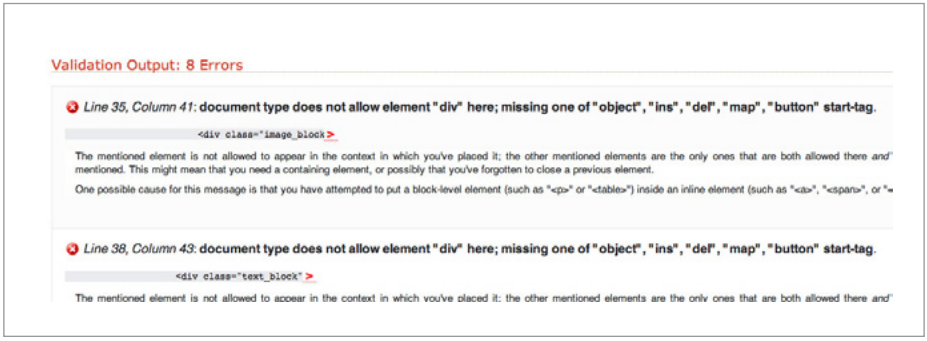
Now it's time to check that our markup is w3c valid! So we go to the Validator put in our code and cross fingers ... and pah-bow, we are not valid, yet.



Looking down there are 14 errors. The number one problem is that there is no alt text on any of the images... whoops! So going back and adding them like so:

```
<img src="images/ribbon_featured.png" class="ribbon"
alt="Featured Project"/>
```

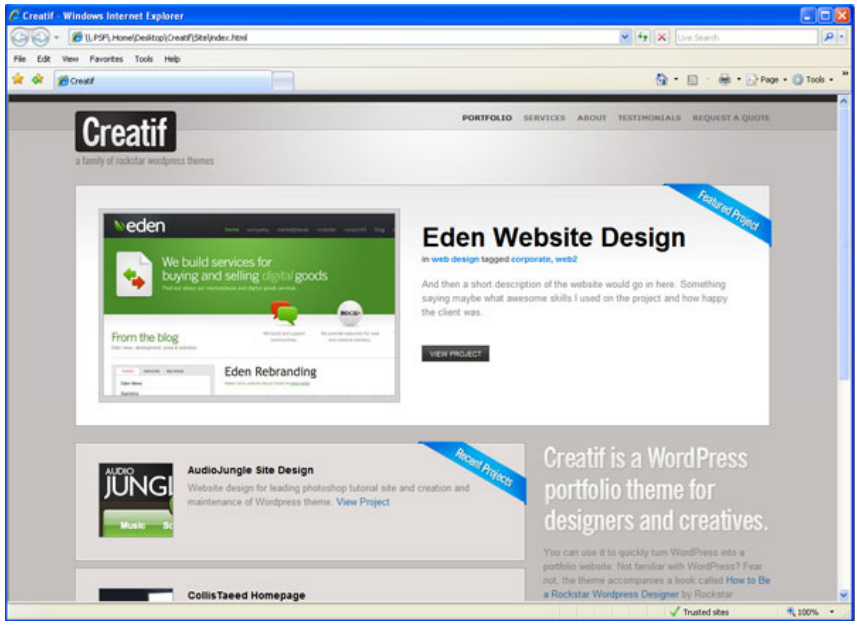
That should fix up lots of the errors. So now we run it again and ... still invalid. OK this one looks a bit trickier:



Fortunately the remaining 8 errors are actually the same problem. Basically we've used an inline element (specifically a `<span class="block_inside">`) and then tried to put block level elements like `<div>`'s inside. And that's not allowed. Luckily it's easily fixed, we just change every instance of `<span class="block_inside">` to a `<div class="block_inside">`. And ... we pass!

Completed the First Page

OK we have successfully made our basic page! Here you can see me testing it in IE7 and thankfully there are no bugs.



With our basic framework in place we are now ready to build the extra pages and the alternate colour scheme. We've lain a good foundation and will be able to make use of a lot of the code we've already written. This is why it's really important to plan ahead. If you don't plan you can easily wind up with a lot of duplication, extra code and other folly.

Step 19 – Building the Blog Homepage

The next page we're going to build is the blog homepage. This is similar to the portfolio homepage in that it will have a featured blog post and then a series of blog posts below. Eventually these will become two related WordPress themes – one for portfolios, one for blogs.

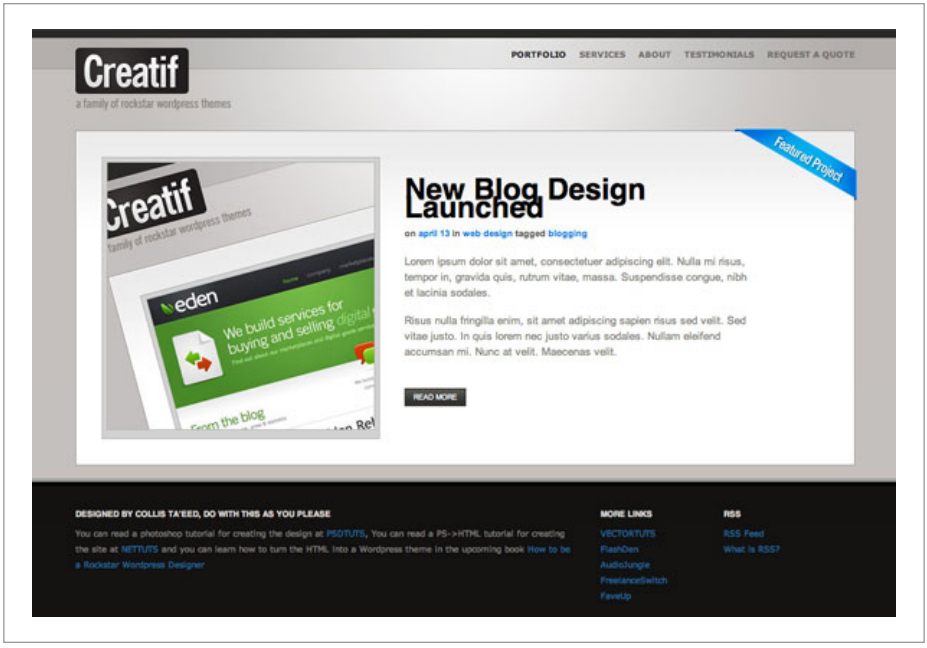
So first we duplicate our *index.html* – the file we've been working on up 'til now, and call the new file *blog.html*.

In our *blog.html* we first delete the whole `<div id="block_portfolio">`. We're going to replace that block with a different one shortly. Then we replace the `<div id="block_featured">` with a new block for featured blog posts which is just slightly different and looks like this:

```
            <div id="block_featuredblog" class="block">
                    <img src="images/ribbon_featuredblog.png"
                    class="ribbon" alt="Featured Project"/>
      <div class="block_inside">
            <div class="image_block">
            <img src="images/sample_blog.jpg"
            alt="New Blog"/>
          </div>
          <div class="text_block">
            <h2>New Blog Design Launched</h2>
            <small>on <a href="">april 13</a> in
            <a href="">web design</a> tagged
            <a href="">blogging</a></small>
            <p>Lorem ipsum dolor sit amet, consectetuer
            adipiscing elit. Nulla mi risus, tempor in,
            gravida quis, rutrum vitae, massa. Suspendisse
            congue, nibh et lacinia sodales. </p>
                    <p>Risus nulla fringilla enim, sit amet
                    adipiscing sapien risus sed velit. Sed
```

```
                    vitae justo. In quis lorem nec justo
                    varius sodales. Nullam eleifend accumsan
                    mi. Nunc at velit. Maecenas velit. </p>
                    <br />
                    <a href="" class="button">Read More</a>
                    </div>
        </div>
    </div>
```

So really all we've done is change the id tag to be `block_featuredblog`, the ribbon image and the content. Essentially though it's the same layout. So let's take a look and see how it's looking:



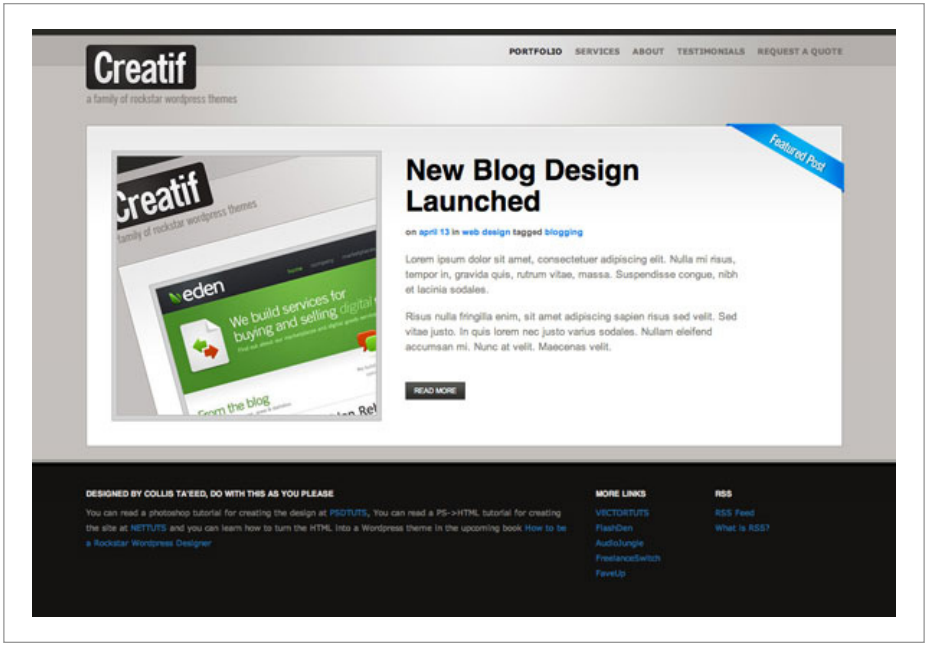
Step 20 – Adjusting some CSS

So that pretty much works as is, we'll just make a couple of small adjustments to the CSS like this:

```
#block_featuredblog .text_block { padding-top:5px;
width:490px;}
h2 {
    margin:0px 0px 10px 0px;
    font-size:36px;
    font-family:Helvetica, Arial, Sans-serif;
    color:#000000;
    line-height:39px;
    letter-spacing:-1px;
}
```

Here we've adjusted the "`text_block`" class but only when it's in the `#block_featuredblog` element. It now has a tiny bit of padding at the top and is wider.

Also we've added an appropriate line-height to the heading and on a whim adjusted the text kerning by -1px. And we're finished with this element, that was easy!



Step 21 – Making the Main Content Area

Making this content area is the last big thing we need to do really. It will form not only the bottom of this page, but also the whole basis of the generic page (with some adjustments of course!). So first let's put in some really basic HTML:

```
                    <div id="block_content">
            <div id="content_area" class="block">
                    <div class="block_inside">
                                        Content
                </div>
              </div>
              <div id="sidebar">
                    <div class="block_inside">
                    Sidebar Content
                </div>
              </div>
          </div>
```

Basically what we've created is a container element – `<div id="block_content">` and then inside that we've got two blocks which we're going to float to either side. You'll see we're making use of our rather handy `<div class="block_inside">` elements to add the double border. Here's the CSS to make them sit correctly:

```
/*
    Block-Content-Styles
*/
#block_content {
}
#content_area {
    width:665px;
    float:left;
}
#sidebar {
    float:left;
```

```
        width:281px;
        position:relative;
        left:-1px;
        margin-top:15px;
        background-color:#e2dddc;
        border:1px solid #a3a09e;
    }
    #sidebar .block_inside {
        background:none;
        background-color:#e2dddc;
    }
```

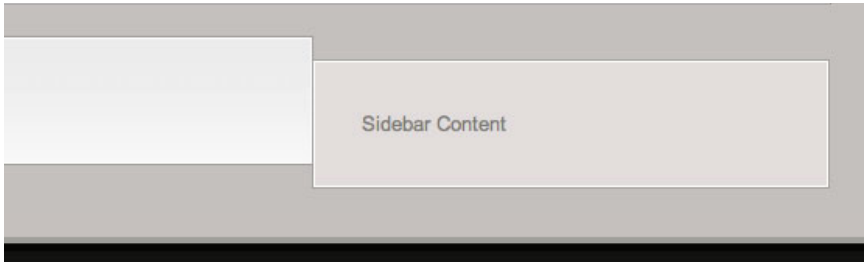
Going through the styles:

1. Then we've given the `#content_area` box and the `#sidebar` box each a width and a float.

2. Next we've moved the sidebar to the left by 1px using `a position:relative`. We have done this so that the left border will overlap and it will look like it's jutting out.

3. Additionally we've added a 15px top margin so that the sidebar isn't top-aligned. Currently it looks a bit odd, but when we add some content it will look great.

4. Finally we've redefined the `.block_inside` in the `#sidebar` element to override the background image and instead give it that beigey colour for a background.



Step 22 – Adding Content

Now we add some content to our two elements to style:

```
<div id="block_content">
    <div id="content_area" class="block">
        <div class="block_inside">
            <h2>Working on a New Project</h2>
            <small>on <a href="">april 13</a> in
            <a href="">web design</a> tagged
            <a href="">blogging</a></small>
            <p>Lorem ipsum dolor sit amet,
            consectetuer adipiscing elit. Nulla
            mi risus, tempor in, gravida quis, rutrum
            vitae, massa. Suspendisse congue, nibh et
            lacinia sodales. </p>
            <p>Risus nulla fringilla enim, sit amet
            adipiscing sapien risus sed velit. Sed
            vitae justo. In quis lorem nec justo
            varius sodales. Nullam eleifend accumsan
            mi. Nunc at velit. Maecenas velit.
            <a href="#">Read More</a></p>
        <div class="separator"></div>
            <h2>Design Awards!</h2>
            <small>on <a href="">april 13</a> in
            <a href="">web design</a> tagged
            <a href="">blogging</a></small>
            <p>Lorem ipsum dolor sit amet,
            consectetuer adipiscing elit. Nulla
            mi risus, tempor in, gravida quis, rutrum
            vitae, massa. Suspendisse congue, nibh et
            lacinia sodales. </p>
            <p>Risus nulla fringilla enim, sit amet
            adipiscing sapien risus sed velit. Sed
            vitae justo. In quis lorem nec justo
            varius sodales. Nullam eleifend accumsan
```

```
                    mi. Nunc at velit. Maecenas velit.
                    <a href="#">Read More</a></p>
            <div class="separator"></div>
                    <h2>This Site is Almost Complete
                    Finally...</h2>
                    <small>on <a href="">april 13</a> in
                    <a href="">web design</a> tagged
                    <a href="">blogging</a></small>
                    <p>Lorem ipsum dolor sit amet,
                    consectetuer adipiscing elit. Nulla
                    mi risus, tempor in, gravida quis, rutrum
                    vitae, massa. Suspendisse congue, nibh et
                    lacinia sodales. </p>
                    <p>Risus nulla fringilla enim, sit amet
                    adipiscing sapien risus sed velit. Sed
                    vitae justo. In quis lorem nec justo
                    varius sodales. Nullam eleifend accumsan
                    mi. Nunc at velit. Maecenas velit.
                    <a href="#">Read More</a></p>
    </div>
</div>
<div id="sidebar">
                    <img src="images/ribbon_browse.png"
                    class="ribbon" alt="Featured Project"/>
                    <div class="block_inside">
                    <h3>Subscribe</h3>
                    <ul>
                            <li><a href="">RSS Feed</a></li>
                            <li><a href="">Email Updates</a></li>
                                    </ul>
                    <h3>Categories</h3>
                    <ul>
                    <li><a href="">News</a></li>
                    <li><a href="">Marketing</a></li>
                    <li><a href="">General</a></li>
```

```
                    <li><a href="">Great Sites</a></li>
                                        </ul>
                    <h3>Archives</h3>
                    <ul>
                    <li><a href="">June 2008</a></li>
                    <li><a href="">May 2008</a></li>
                    <li><a href="">April 2008</a></li>
                    <li><a href="">March 2008</a></li>
                                        </ul>
                    </div>
            </div>
    <!-- a Clearing DIV to clear the DIV's because overflow:
    auto doesn't work here -->
        <div style="clear:both"></div>
    </div>
```

OK there are three important things to mention here:

1. First in the content area you'll see we've added three dummy blog posts and in between each is an empty `<div class="separator">` that we'll style shortly into a thin line with some spacing.

2. Next we've added a ribbon image to the sidebar in much the same way as previously.

3. Finally we've used a clearing `<div>` at the bottom. Now previously in this tutorial we've been using `overflow: auto`; to deal with floated columns, but when we add the margin-top in the previous step to move the sidebar down it messes with the overflow and creates a scrollbar. So since there may be occasions when the sidebar will be longer than the content box we're going to use this method of clearing floating `<div>`'s instead.

Now we'll add some basic styling to fix it all up as follows:

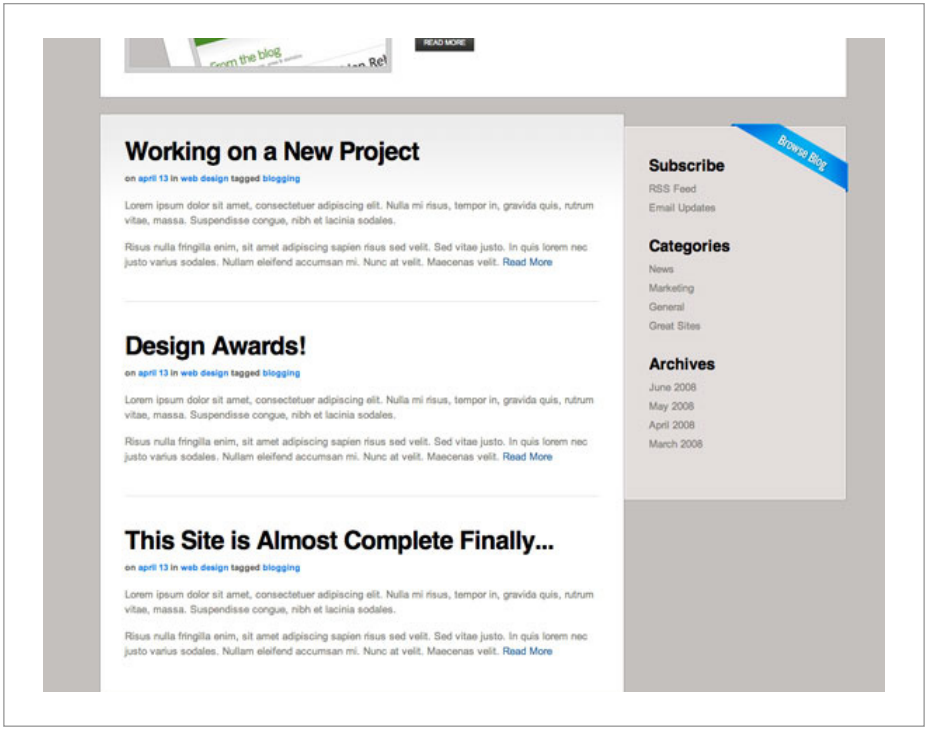
```
#sidebar h3 {
    font-size:20px;
    line-height:23px;
}
#sidebar ul { margin:10px 0px 30px 0px; padding:0px; }
#sidebar ul li { list-style:none; margin:0px 0px 5px 0px;
padding:0px; }
#sidebar ul li a { color:#7f7d78; }
#sidebar ul li a:hover { color:#0172dd; text-decoration
none; }
#content_area h2 { font-size:32px; line-height:31px; }
#content_area .separator {
    border-top:1px solid #e3e3e3;
    margin-top:40px;
    padding-top:40px;
}
```

Two things to note:

1. We've formatted the `<ul>` lists in the sidebar to remove the bullet points and space them out nicely

2. We've also created a separator style using margin and padding along with 1px border

And that's it, our `#block_content` element is complete!



Step 23

Making our final page is a piece of cake now. We just duplicate our *blog.html* and call it *page.html* this time. Then remove the featured blog post and alter the HTML of the `#block_content` area as follows:

```
        <div id="block_content">
        <div id="content_area" class="block">
                <div class="block_inside">
                        <h4>Services</h4>
                                <h2>Branding</h2>
                        <br />
                        <p>Lorem ipsum dolor sit amet,
                        consectetuer adipiscing elit. Nulla
                        mi risus, tempor in, gravida quis, rutrum
```

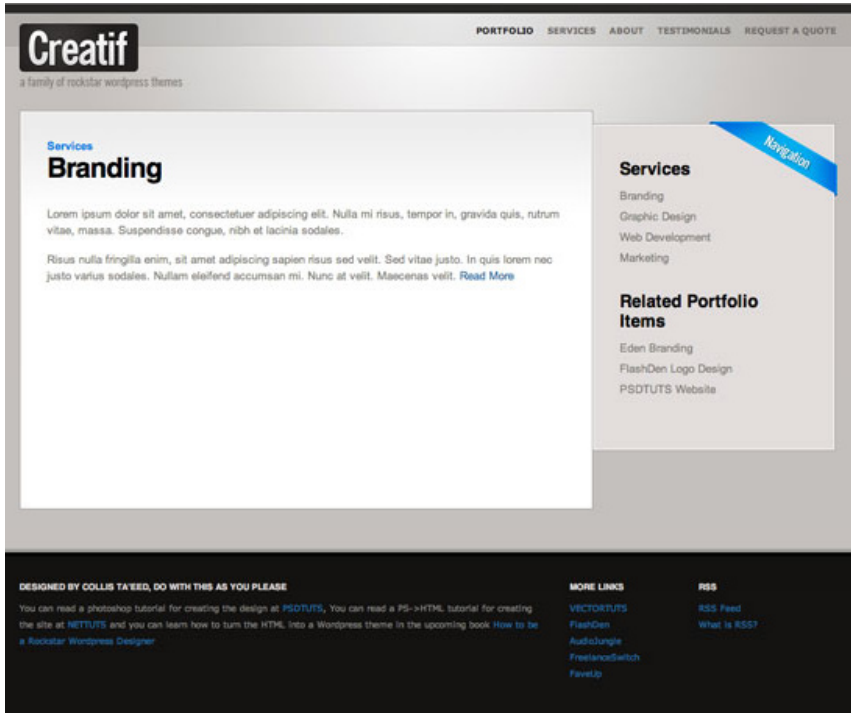
```
                    vitae, massa. Suspendisse congue, nibh et
                    lacinia sodales. </p>
                    <p>Risus nulla fringilla enim, sit amet
                    adipiscing sapien risus sed velit. Sed
                    vitae justo. In quis lorem nec justo
                    varius sodales. Nullam eleifend accumsan
                    mi. Nunc at velit. Maecenas velit. <a
                    href="#">Read More</a></p>
            </div>
      </div>
      <div id="sidebar">
            <img src="images/ribbon_navigation.png"
            class="ribbon" alt="Featured Project"/>
                    <div class="block_inside">
                    <h3>Services</h3>
                    <ul>
                            <li><a href="">Branding</a></li>
                            <li><a href="">Graphic Design
                            </a></li>
                            <li><a href="">Web Development
                            </a></li>
                            <li><a href="">Marketing</a></li>
                                        </ul>
                    <h3>Related Portfolio Items</h3>
                    <ul>
                            <li><a href="">Eden Branding
                            </a></li>
                            <li><a href="">FlashDen Logo
                            Design</a></li>
                            <li><a href="">PSDTUTS Website
                            </a></li>
                                        </ul>
                    </div>
            </div>
   <!-- a Clearing DIV to clear the DIV's because overflow:
   auto doesn't work here -->
```

```
        <div style="clear:both"></div>
    </div>
```

Which is pretty much the same HTML as previously just with some different text and a new ribbon. The only real change is that now we have a title and above that a subtitle wrapped in an `<h4>` tag. So we can style that with a couple of lines of CSS as follows:

```
h4 {
    color:#007de2;
    margin:0px 0px 0px 0px;
}
```

And that is that!



Step 24 – It don't matter if it's Black or White!

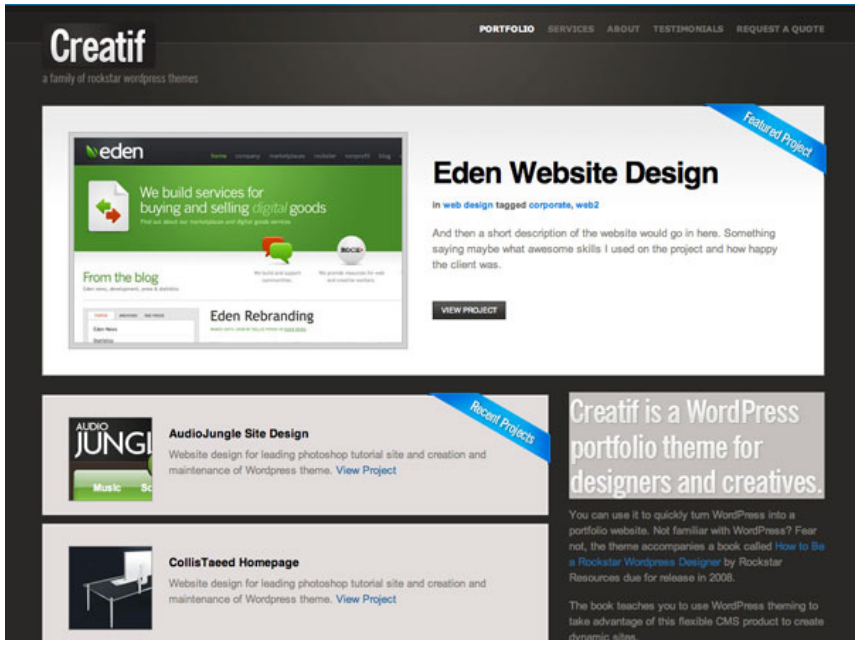
Now we're going to do some very simple CSS to switch the site from light to dark. What's neat about this is the only HTML we need to alter is this one line:

```
<body id="dark">
```

That's it! With that one bit of extra HTML code we can make all the CSS adjustments necessary. This means later we can make a little Javascript button that switches the stylesheet. The way it's going to work is for any class which needs to change we just add an extra style beginning with `body#dark`. So first of all we say:

```
body#dark {
    background-color:#1e1d1b;
}
body#dark #main {
    background:#292826 url(images/background_dark_slice.
jpg) repeat-x;
}
body#dark #main .container {
    background-image:url(images/background_dark.jpg);
}
body#dark #footer {
    background-image:url(images/background_dark_footer.
jpg);
}
body#dark ul#menu li a.active, ul#menu li a:hover {
    color:#ffffff;
}
```

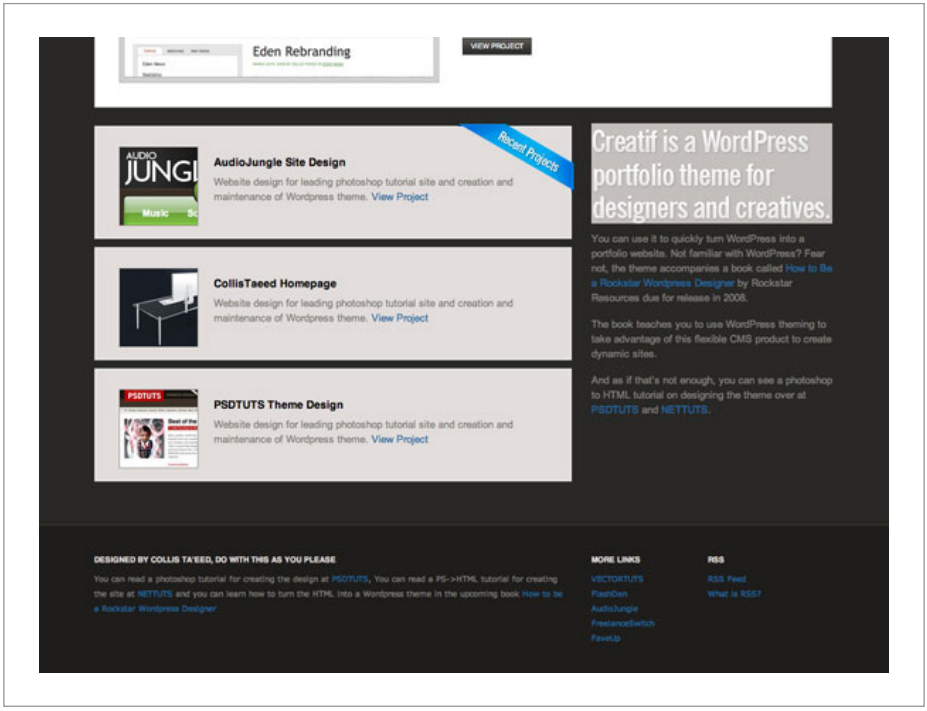
And that tells the browser that if `<body id="dark">` then to override the styles for `#main`, `#main .container`, `#footer`, and the active and hover states of the menu, swapping in some new background images and changing the text colour to white.



Step 25 – Borders and Fixing the Text

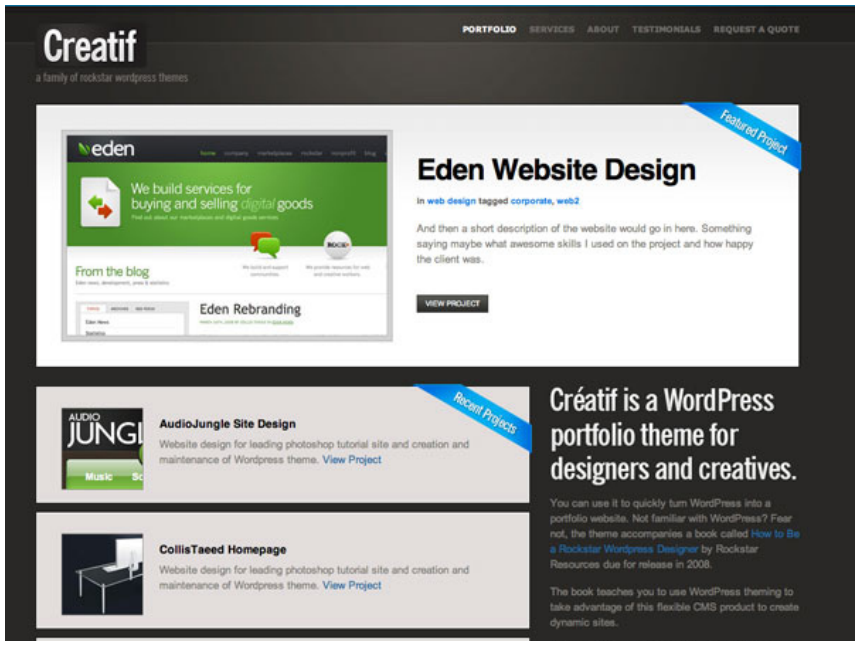
As you can see in the image below our footer is fixed thanks to the new background image and colour, there's just two more fixes: the "Creatif is a WordPress ..." text and the borders around the boxes which are quite light and should be dark now. So we do this:

```
body#dark .block, body#dark .mini_portfolio_item {
    border-color:#1b1a19;
}
body#dark #text_column h2#text_title {
    background-image:url(images/creatif_dark.jpg);
}
```



Step 26 – Alternate Colour!

And with those adjustments, we now have an alternate colour scheme all controlled by a single id tag on the `<body>` element. That's the magic of transparent PNG files and CSS at work.



Further Resources on HTML & Photoshop

You can find additional help on designing with Photoshop and building websites at:

PSDTUTS – http://psdtuts.com
NETTUTS – http://nettuts.com
CSS-Tricks – http://css-tricks.com
WebDesignerWall – http://webdesignerwall.com

4

Introduction to Themes

A theme can completely transform your WordPress site, by changing graphics, showing or removing features, adding extra functionality or simply by rearranging the page.

If you understand HTML, then most themes are fairly simple to understand. In this chapter we'll review some theme basics that we'll need to use to build our Creatif themes in the following chapters.

Finding and Installing Themes

Themes reside in a directory in your WordPress install. If you FTP into your server you will see a directory called `wp-content`, inside of which sit three important directories: uploads, themes and plugins.

If you open up the Themes directory you should see at least two more folders inside. Each folder represents a theme and by default WordPress comes with the Classic and Default (Kubrick) themes.

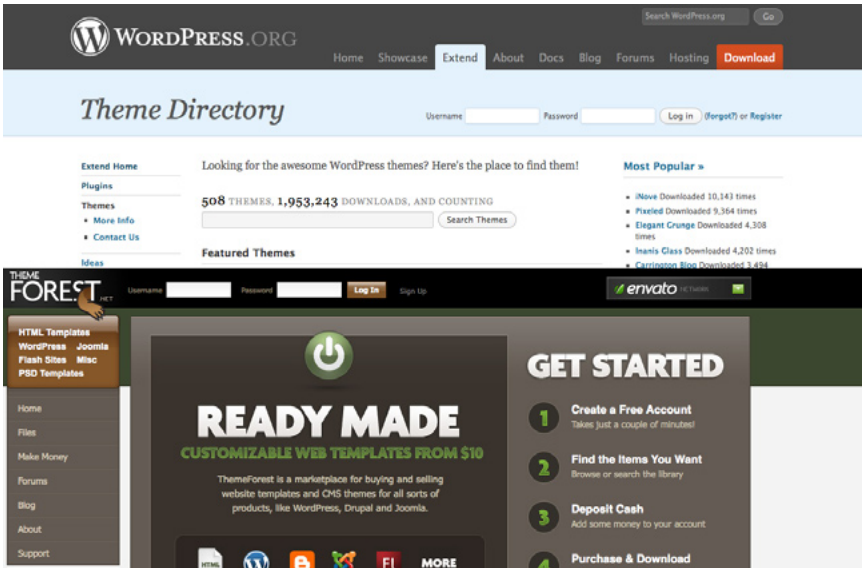


Fig 4-1 – WordPress offer a large selection of free themes, whilst ThemeForest lets WordPress theme designers sell their work.

You can find other themes online at sites like:

- **WordPress.org (Free Themes)** – http://wordpress.org/extend/themes/
- **ThemeForest (Paid Themes)** – http://themeforest.net

Once you download a theme, simply upload its folder into your themes directory on the server. Then log into *WP-Admin*, click on *Appearance* and you will see a sequence of theme previews. Click the preview for your new theme and it will be installed.

Editing Theme Files through WordPress

It is possible to edit theme files through the WP-Admin dashboard. Simple click on *Appearance* > *Editor*. You will see a listing of all the files in the theme and you can edit pages and CSS files by clicking on the file and editing in the text area.

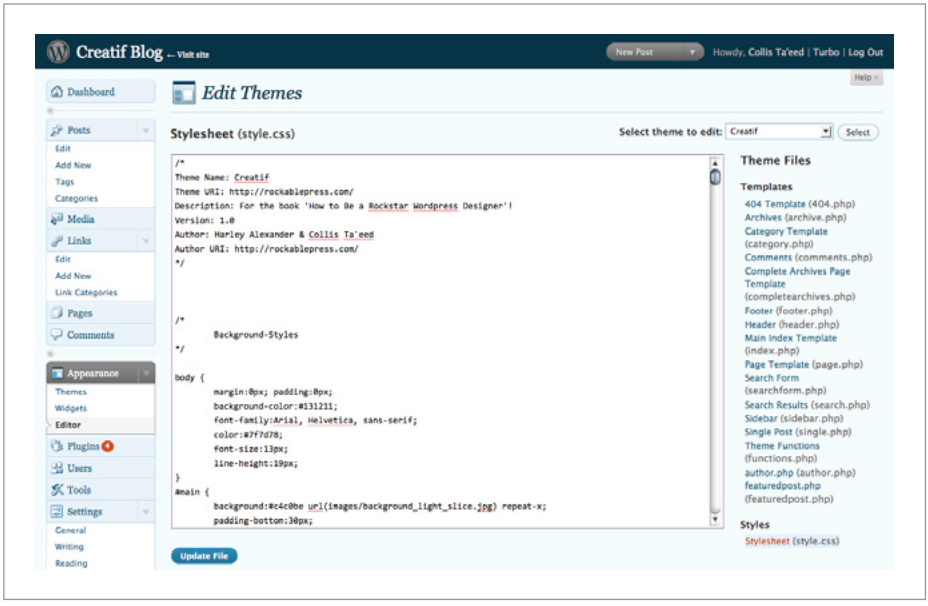


Fig 4-2 – WordPress' WP-Admin Theme Editor.

To be able to edit a theme however the files must have the correct permissions set, specifically the files for the theme must be writable. You can update these permissions through your FTP program.

How a Theme Works

A WordPress theme is a set of PHP files, images and CSS, bundled up in a directory. The PHP files are mostly HTML and structured so that when stitched together they create a single HTML document. So for example if you grabbed the `header.php`, `index.php`, `sidebar.php` and `footer.php` and stitched their code together into one file, you'd get a single HTML page with some little bits of PHP code here and there. Separate files are called in using special statements called includes. For common items like the sidebar, footer and header, they look like this:

```
<?php get_sidebar(); ?>
```

So to stitch a file together, you would get the main file – `index.php` – and then wherever there is an include line like the one shown above, you would swap in the contents of the included file, for example `sidebar.php`. We'll look more at includes later.

When a page on your site is called for, WordPress displays the theme page and wherever you've placed PHP code, it executes that code and fills in the appropriate content. So for example if your theme had the code:

```
<?php wp_list_categories(); ?>
```

WordPress would substitute the code for an unordered list `<ul>` of categories before displaying the page. These commands are called *Template Tags*. You can find lists of Template Tags in the WordPress

Codex along with their syntax and usage details – http://codex.wordpress.org/Template_Tags.

WordPress themes are essentially just the final layer between WordPress and the end user. All the main work is done by WordPress behind the scenes, retrieving posts, inserting comments and so on. Then the theme acts like a filter outputting the results in neat HTML and CSS.

How to Make a Theme in 60 Seconds

The simplest way to get your head around a theme is to build one! So here's a sixty second example:

1. Create a New Directory, call it *BasicTheme*
2. Inside the *BasicTheme* directory, create a file called `style.css` but leave it empty
3. Next create a file called `index.php` and enter the following HTML code into the new file:

```
<html>
<head>
    <title>Test Theme</title>
</head>
<body>
</body>
</html>
```

Now upload the theme to your server and you should see it appear in the Presentation section. Click on *BasicTheme* to choose it and take a look at your blog.

Add a Little Bit of WordPress PHP

OK so your theme isn't doing very much, but you should see that the HTML you created is showing through.

Now edit your `index.php` file and change the title tag to:

```
<title><?php bloginfo('name'); ?></title>
```

Upload this file again and test your theme out now. You'll see that the WordPress blog's name is appearing in the title bar of your browser.

That's How Themes Work

You've now seen the basic mechanics of a theme. You place some HTML into a bunch of PHP files, then you replace the bits you want to be live data from WordPress with short WordPress/PHP commands.

In this case `bloginfo('name')` tells WordPress to show the title of the blog.

There are hundreds of Template Tags to do different things in WordPress. Using different Template Tags you can make posts, categories, links and just about anything else appear.

Everything is there waiting in WordPress, but when you create a theme you determine what gets shown and how it gets shown.

So in our basic theme we decided to only show the blog title. That's not to say that posts, comments and all the other stuff aren't there, we have just chosen not to make them appear on the page.

The most useful thing to learn in order to make themes are the Template Tags you write in PHP to tell WordPress what to output.

Template Tags

Template Tags are in fact PHP functions. They have been created in the main WordPress codebase and when you write one onto the page, WordPress executes the function code to produce some results.

As you know from basic PHP, a function looks like this:

```
functionname(parameter 1, parameter 2, ...);
```

or with the addition of the PHP tags:

```
<?php functionname(parameter 1, parameter 2, ...); ?>
```

The parameters are different options that are passed to the function to help it decide what to output. For example with the `wp_list_categories()` Template Tag you can choose to list categories with the number of posts inside each category by writing this:

```
<?php wp_list_categories('showcount=1'); ?>
```

Alternatively if you wanted the number of posts to be hidden, you would write this:

```
<?php wp_list_categories('showcount=0'); ?>
```

Functions can also have no variables passed to them, in which case the code between the brackets is just empty like this:

```
<?php wp_list_categories(); ?>
```

If you call a function without giving it any inputs, and it requires some, the function will generally fall back on default values. For example the potential input parameters with their default values for

the wp_list_categories Template Tag are:

```
'show_option_all' => '',
'orderby' => 'name',
'order' => 'ASC',
'show_last_update' => 0,
'style' => 'list',
'show_count' => 0,
'hide_empty' => 1,
'use_desc_for_title' => 1,
'child_of' => 0,
'feed' => '',
'feed_image' => '',
'exclude' => '',
'hierarchical' => true,
'title_li' => __('Categories'),
'echo' => 1,
'depth' => 0
```

So when you write in wp_list_categories you can have all or none of those values in the brackets. If you have more than one parameter that you are passing, you simply separate them with an & character like this:

```
<?php wp_list_categories('showcount=1&hide_empty=0'); ?>
```

To figure out what all these values do and mean, you can refer to the wp_list_categories page in the WordPress Codex – http://codex.wordpress.org/Template_Tags/wp_list_categories

Note: *In reality everything in the quotation marks is one PHP parameter (a string) which is why it doesn't fit the parameter 1, parameter 2 syntax that normal functions follow. WordPress Template Tags instead are passed a single string and that string is then broken up wherever a & character appears. This is done to make Template Tags easier to use, read and understand.*

Finding and Understanding Template Tags

There are many Template Tags available for use in your themes, and the best way to learn them is simply to look them up in the WordPress Codex. Each listing contains a specification of the way the Template Tag should be used – this is called the syntax – as well as examples of its usage.

The Loop

A loop is a piece of code that is executed multiple times, with some variables that change each time. For example if you wanted to output the numbers 1 to 10, you might write this PHP code:

```
for ( $counter = 1; $counter <= 10; $counter += 1) {
    echo $counter;
}
```

Here the code between the `{` brackets `}` is executed and each time the value of `$counter` changes by +1 until the end condition is met – namely that `$counter` becomes less than or equal to 10.

That loop is called a *For Loop*. Another type of Loop is a *While Loop*. While Loops execute a set of code *while* a condition is true. So the same example written with a While Loop would look like this:

```
$counter = 1;
while ( $counter <= 10 ) {
    $counter += 1;
}
```

In WordPress themes there is one major loop that appears on almost all pages. In WordPress terminology it's called *The Loop* and it processes a list of any Posts that are meant to be shown on that

page. Inside the loop you can write code that shows those Posts in different ways.

Here's the simplest possible version of *The Loop*:

```
<?php while (have_posts()) { ?>

    <?php the_post(); ?>
    <h2><?php the_title(); ?></h2>
    <?php the_content(); ?>

<? } ?>
```

So the code says that while `have_posts()` is returning true – or in other words while there are Posts to be shown – we should repeat the code in the middle over and over.

The code in the middle has three parts:

1. `the_post()` is a special function that does a bit of behind the scenes work to prepare the post to be outputted in the next lines.

2. `the_title()` outputs the post's title

3. `the_content()` outputs the post's main body

So in plain English, our loop says while we have posts, go through each one and show the post title as a heading and then the post content. That's it!

A More Common Version of The Loop

The while loop above uses the most common way of writing loops and statements in PHP. However there is an alternate syntax that is

often used when writing PHP mixed in with HTML. Instead of using { and } brackets like this:

```
<?php while(conditions) { ?>
... HTML CODE ...
<?php } ?>
```

You can use colons and an `endwhile` statement, like this:

```
<?php while(conditions) : ?>
... HTML CODE ...
<?php endwhile; ?>
```

Both versions do exactly the same thing. The second one is a little easier to work with in HTML because it's obvious what the `endwhile` is referring to, whereas `{}` are a little more ambiguous.

The version of The Loop we looked at just above follows a regular way of writing loops in PHP. In practice in WordPress, most people write the loop using the alternate syntax. Additionally before you run the while loop, it's a good idea to check if there are even any posts around, so here is a more common version of The Loop that you are likely to come across:

```
<?php if ( have_posts() ) : while ( have_posts() ) : the_
post(); ?>
    <h2><?php the_title(); ?></h2>
    <?php the_content(); ?>
 <?php endwhile; else: ?>
    <p>Sorry, no posts matched your criteria.</p>
 <?php endif; ?>
```

So here we are first checking if there are any posts, if so then while there are posts we show the title and content of each, and if there aren't any posts we print out a little statement to that effect.

As you work with more themes you will see other variants of The Loop. The important thing to note is that it is simply a way of querying the WordPress server to get your posts to show.

Files and What They Do

Although `index.php` and `style.css` are the only totally mandatory files needed to make a WordPress theme, there are a number of files that have special meaning to WordPress. These different files sit in a hierarchy and when WordPress is asked to show a page, it runs through the hierarchy searching for the right file to show. At the bottom of the chain is `index.php`, so if nothing else is available that's the one that gets shown.

So let's say you have a WordPress install sitting at www.example.com. Now for example a URL a reader might visit when going to your WordPress site is:

Example Post URL

www.example.com/?p=1
www.example.com/post_title (with Permalinks set)

Now for a Post, WordPress first looks to see if there is a file in the theme directory called `single.php`, if that exists, that is the template file that will be used to display this page. If `single.php` isn't there, then WordPress defaults to showing `index.php`.

So the hierarchy goes:

```
single.php > index.php
```

Now another URL a reader might visit is:

Example Page URL

www.example.com/?page_id=2
www.example.com/page_title (with Permalinks set)

In this case WordPress first looks to see if the Page in WordPress has a special Page Template selected for it (this is set on the Edit Page screen). So for example, let's say there is a template file called `some_template.php`. If the Page isn't meant to use a special Page Template, then WordPress checks to see if there is a generic page template with the file name `page.php`, if that's not there either, then as usual WordPress falls back and displays our good old `index.php` file.

So in this case the hierarchy goes:

```
some_template.php > page.php > index.php
```

The File Hierarchy

Here is a list of all the different file hierarchies that WordPress looks for:

Date Archives – www.example.com/2008/12

```
date.php > archive.php > index.php
```

Category Archives – www.example.com/category/category_title

```
category-id.php > category.php > archive.php > index.php
```

Tag Archives – www.example.com/tag/tag_name

```
tag-slug.php > tag.php > archive.php > index.php
```

Author – www.example.com/author/author_name

```
author.php > archive.php > index.php
```

Home – www.example.com

```
home.php > index.php
```

Single Post – www.example.com/post_title

```
single.php > index.php
```

Page – www.example.com/page_title

```
some_template.php > page.php > index.php
```

Search – www.example.com/?s=search_word

```
search.php > index.php
```

404 – www.example.com/something_that_isnt_there

```
404.php > index.php
```

Including Files

Because some parts of a site are repeated over and over, it's simpler to place the code for them in a separate file one time, and then write an include statement many times. A good example of this is the sidebar. If you had for example an archives page, an author page, a homepage and a single Post page and all of them shared the same sidebar, it makes sense to have a single sidebar file and include it each time. That way any changes to the sidebar are only made in one spot.

You can include any file you create into another using this line of code:

```
<?php include (TEMPLATEPATH . '/filename.php'); ?>
```

This is a regular PHP Include, and `TEMPLATEPATH` is a special variable in WordPress that points to the theme folder.

Additionally there are four special files – `sidebar.php`, `footer.php`, `header.php` and `comments.php` that have slightly different includes that you can use:

```
<?php get_header(); ?>
<?php get_footer(); ?>
<?php get_sidebar(); ?>
<?php comments_template(); ?>
```

Of course for any of these you could simply use the regular PHP Include mentioned above instead, but since the header, footer, sidebar and comments are the most common they have their own more readable versions.

Conditional Tags

So you may be wondering, if all the different file types default back to `index.php` in the end, how can that one file handle all the different cases. For example how does it know if it is meant to show content for the homepage, a Page, a single Post or something else?

The answer is by using a special type of Template Tag called a Conditional Tag. Here is an example Conditional Tag to check if the page being shown is a single Post page:

```
<?php is_single(); ?>
```

Let's say for example this line of code is in the `index.php` file. And let's say there are no other theme files and the URL example.com/post_title has been accessed, then that Conditional Tag would return `True`. Now on its own that isn't very useful, but we can use the code in an if statement like this:

```
<?php if(is_single()) : ?>
```

```
        ... Some code to show on single Posts
    <?php else: ?>
        ... Some code to show on other pages
    <?php endif; ?>
```

So the code above checks if we're meant to be displaying a single Post, if the answer is yes then we execute the code below, else we show some other code.

There are lots of conditional tags that you can use, some examples include:

`is_home()` – Is this the homepage?

`is_page()` – Is this a Page?

`is_single()` – Is this a single Post?

`is_category()` – Is this a category archive?

`is_author()` – Is this an author archive?

You can find more conditional tags as well as details on how you can use them in different ways in the Codex – http://codex.wordpress.org/Conditional_Tags

The WordPress Default Theme – Kubrick

Now that you've got the basics of how WordPress themes work, it's a good idea to inspect a real world theme and see how it's made up. Kubrick, WordPress' default theme, comes with every install and is a great place to learn about and experiment with themes.

Fig 4-3 – WordPress' Default Theme – Kubrick.

1. **Download Kubrick**
   Log in via FTP to your WordPress install and download the theme files from `wp-content/themes/default` to a new directory on your hard drive. Or if you are using a local copy, simply open the directory.

2. **Look Through the Files**
   Browse through the files and note how they come together. The main file is `index.php`. If you look through you should quickly see how the file has a number of includes to pull in the files `header.php`, `sidebar.php` and `footer.php`.

3. **Rename the Theme and Experiment**
   Rename the folder so that you don't override the original Default theme, then re-upload it via FTP to the themes directory. Or if you are using a local copy, simply duplicate the folder and give it a new name. Log

in to WP-Admin, click on *Appearance* and select your renamed theme. You can then try modifying parts of the files and testing out what they do to your WordPress when you upload them back. Don't be afraid to break things, you can always retrieve the original Kubrick theme and fix it back up if need be.

Using Kubrick as a Base

The WordPress Codex is great for looking up code and finding out how to do things when making or adjusting themes. An alternative method is to simply use another theme as a base to build off.

The best themes to use for this are the default themes – Kubrick and Classic. These two themes are simple, clean and adhere to good practices and conventions.

Not only can copy and paste segments of code to save you time, but they also include all the main files that you should replicate in your own themes.

Further Resources on Theming Basics

You can find additional help on the basics of themes by visiting these resources:

1. **WPDesigner's Theming Tutorials** – http://www.wpdesigner.com/category/tutorials/
2. **NETTUTS' WordPress Tutorials** – http://nettuts.com/category/tutorials/wordpress/
3. **WordPress Codex Theme Development** – http://codex.wordpress.org/Theme_Development

5

Building a Basic Theme: Creatif Blog

In this book we'll be making use of WordPress to power all sorts of sites, however it is first and foremost designed to be a blogging platform. So the best place to begin our theming work is with a blog theme.

WordPress has produced one of the easiest to use theming engines around. The amazing way in which WordPress works allows for a huge level of customization. There is virtually nothing you can't show, hide, feature, dynamically add or change to your liking.

In this chapter we'll go through all you need to know to create your own, custom blog theme from scratch. We'll make use of much of what you learnt in Chapter 4, including Template Tags, file hierarchies and The Loop, as these will form the backbone of your theme!

Setting up WordPress

Before we get into the theme development, the first thing to do is get a copy of WordPress ready to theme. You should go through and add a few categories and some sample Posts. Doing this now means that as you work on your theme you'll be able to see the results of your efforts as you go along.

There is one particular thing that you need to do before we build the Creatif Blog theme and that is to add a featured post image to the most recent Post. To do this, create the Post, then while editing scroll down to the area marked Custom Fields. These fields allow us to add extra information to a Post. We'll cover them in detail in Chapter 6. For now simply create a new Custom Field with the name `large_preview` and under `value` paste in the URL of an image to use for your featured post. If you don't have an image online, you can upload one using WordPress' regular upload functionality and then copy the URL over.

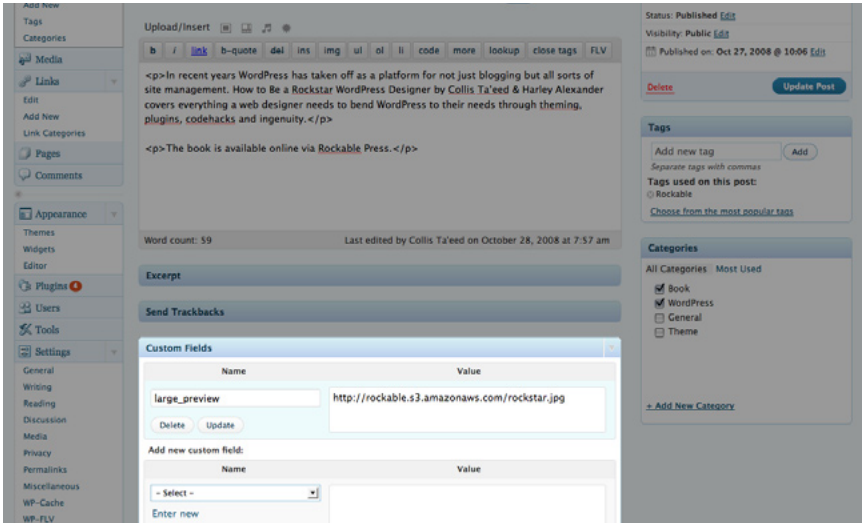


Fig 5-1 – Adding Custom Fields.

Setting up the Theme

Two files are absolutely necessary for theme development – `index.php` and `style.css`. In Chapter 4, we looked at how you can create an extremely basic theme just by adding a relatively blank `index.php` file and `style.css` file to your theme directory. Of course our basic theme didn't do very much, so we'll now go through constructing a more functional theme.

When developing a theme, `style.css` is a good place to start. It sets up a number of definitions that make the theme easy to identify using a regular CSS comment, here's an example from WordPress' documentation:

```
/*
Theme Name: Rose
Theme URI: the-theme's-homepage
Description: a-brief-description
Author: your-name
Author URI: your-URI
Template: use-this-to-define-a-parent-theme--optional
Version: a-number--optional
.
General comments/License Statement if any.
.
*/
```

So we've got:

- The theme name
- A URI (or URL) usually to a general download site for the theme
- A short description

- The author's details
- And some general theme development details – for example if you have built your theme using the default Kubrick theme as a base, you might list that here.

Setting these details not only provides information for users of your theme but also creates the basic theme listing in your WordPress WP-Admin area.

So to begin building our first Creatif theme, create a new directory for the theme. Then create a file called `style.css` in your favorite text editor and place this information in:

```
/*
Theme Name: Creatif Blog
Theme URI: http://rockablepress.com
Description: For the book 'How to Be a Rockstar WordPress
Designer!'
Author: Collis Ta'eed & Harley Alexander
Author URI: http://rockablepress.com
Version: 1.0
.
This theme can be used for commercial or non-commercial use
so long as it is not redistributed or resold in any way.
.
*/
```

Next we'll create a simple `index.php` file, just type in the words `Hello World!` and add the theme folder to your WordPress install.

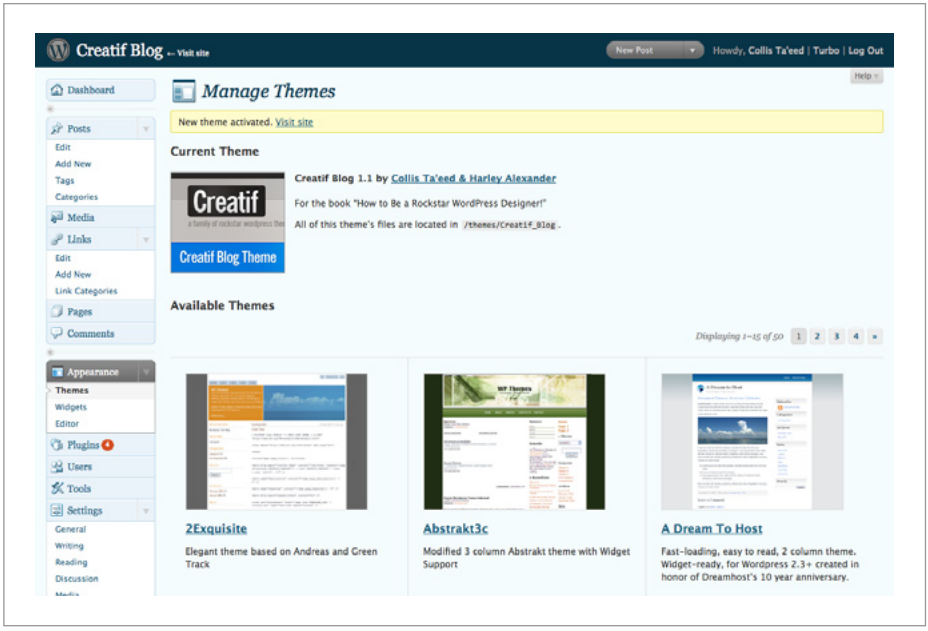


Fig 5-2 – The Creatif Theme on the Appearance page.

As you know activating a theme is straightforward. Once `index.php` and `style.css` are created, open up WP-Admin and click on Appearance. Notice the details we set in the CSS file are showing here on the theme's listing. Select the new theme by clicking on it and then press OK through the preview to get it activated on your WordPress site.

If you've activated your theme correctly, saved the correct files and written the right things, you can now go to your WordPress install in a browser and see the words: *Hello World!*

Absolute URLs

Ordinarily when developing a website, you use what are known as Relative URLs to reference images, stylesheets and Javascript includes. These are relative to your main HTML document, however

in WordPress, the theme files (such as `index.php`) are applied as a template to files in a whole other directory structure. In other words any relative links will break when applied in a theme. Instead you need to have Absolute URLs to your assets. So instead of:

```
/images/sample.jpg
```

You need to write:

```
http://example.com/wp-content/themes/theme_name/images/
sample.jpg
```

Of course if you were to change the theme folder that your files have been placed in, say from `theme_name` to `theme_name2`, without editing your theme files, all your Absolute URLs would break. Fortunately, WordPress is equipped with a simple Template Tag to make life easier.

bloginfo – As the name suggests, the `bloginfo` template tag gets some basic information about your blog. Specifying what information is needed is as simple as adding a parameter. If you need the main URL of a WordPress blog, just write `'url'`:

```
<?php bloginfo('url'); ?>
```

So going on this, one option for linking to our image might be:

```
<?php bloginfo('url'); ?>/wp-content/themes/theme_name/
images/sample.jpg
```

However that is still quite cumbersome and again if your user happens to install the theme in a different folder, everything will break. Fortunately WordPress provides an alternate parameter: `template_directory`. This fancy bit of code does all the hard

work, and returns the directory of your currently active WordPress theme:

```
http://example.com/wp-content/themes/theme_name
```

So to grab our image and place it into an HTML `<img>` tag, we simply write:

```
<img src="<?php bloginfo('template_directory'); ?>/images/
sample.jpg" alt="Example Image!" />
```

To see bloginfo's other options, consult the Codex: http://codex.WordPress.org/Template_Tags/bloginfo

Bringing your HTML into WordPress

So in this chapter we're going to take the HTML for the Creatif Blog theme from Chapter 3 and add WordPress functionality to create a working theme.

As you will recall, the design has several sections:

- the header – with logo and menu;
- the featured post;
- the main content area;
- the sidebar; and
- a footer.

What we'll be doing is going through each of these sections of the HTML and swapping the dummy content we placed during the markup phase with WordPress Template Tags, includes, loops and functions.

Now the HTML we created was for a few different Creatif themes that we're building in this book. In this chapter we are modifying the *blog* version of the site.

So to bring over the HTML from Chapter 3:

1. Copy the contents of `blog.html` and paste it into `index.php` in your theme folder.

2. Next copy the contents of `style.css` and paste it into `style.css` in your theme folder (below the comment header we added earlier)

3. Finally copy over the images directory into your theme folder

If you now test your theme you should see a mangled version of the `blog.html` page. Note that all our image URLs have broken because they were Relative URLs, and all our styles have vanished because the CSS file is also linked with a Relative URL.

So our process from here on is basically to work top down and fix each section in turn with the appropriate bits of WordPress code.

Updating <head> for WordPress

The `<head>` section of WordPress themes includes a lot of important information, not only the regular HTML title tag, links to the stylesheet and so on, but also a few special WordPress instructions to set up RSS feeds, provide plugins with space to add their own code and a pingback URL.

In this section, you can assume that any code we cover is meant to be placed within the `<head></head>` part of the `index.php` file.

Currently our code looks like this:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <meta http-equiv="Content-Type" content="text/html;
charset=UTF-8" />
    <title>Creatif</title>
    <link href="style.css" rel="stylesheet" type="text/css"
/>
    <link rel="shortcut icon" href="images/favicon.ico" />
</head>
```

Page Information

So we'll begin with the `Doctype`. Defining the Doctype is important because it tells the browser how to process the HTML. Our Doctype is fine so we'll ignore it.

Next there's the meta information provided for the browser and robots processing this page. We'll change that to:

```
<meta http-equiv="Content-Type" content="<?php
bloginfo('html_type'); ?>; charset=<?php
bloginfo('charset'); ?>" />
<meta name="generator" content="WordPress <?php
bloginfo('version'); ?>" />
```

There's the handy `bloginfo` template tag appearing three times over, with parameters `html_type`, `charset` and `version` respectively.

The Page Title

Next we have our `<title></title>` tag, currently it just says ‘’Creatif”, but this should change when the reader visits different pages, right? Not only will it make bookmarking and reading more meaningful, but it will also help the blog get indexed well in search engines.

There are a variety of ways to format your site titles, in our example we’re just going have each page print out the page’s title followed by the blog’s name. This name is defined in the Dashboard by the user under *Settings* > *General*.

The code we need is:

```
<title><?php bloginfo('name'); ?><?php wp_title(); ?></title>
```

Once again we’ve used `bloginfo` this time to grab the blog’s name, and coupled it with a template tag that outputs the page’s title. The `wp_title` tag will display the name of the Post if viewing a single Post, the name of the category if viewing by category, the searched term if showing the results of a search, and so on. It’s very versatile, and very handy!

Stylesheets and RSS

Next comes the URL to our stylesheet. As we discussed previously this Relative URL doesn’t work given the different virtual locations this code will be accessed by. So we’ll swap it out for another variant of the very useful `bloginfo` tag that provides the URL for the default stylesheet:

```
<link rel="stylesheet" href="<?php bloginfo('stylesheet_url'); ?>" type="text/css" media="screen" />
```

Since we want our blog to have an RSS feed, we're going to need to define the URL and information for it. Once again, the fantastic `bloginfo` has the information needed to link the page to the URL of our feed:

```
<link rel="alternate" type="application/rss+xml" title="RSS
2.0" href="<?php bloginfo('rss2_url'); ?>" />
```

This is important in browsers such as Safari that have RSS readers built in – it provides, in their URL bar a small RSS icon appears allowing the user to quickly hook the RSS feed into their reader preferences.

Trackbacks

Pingbacks (or trackbacks or linkbacks as they are sometimes called) are important in creating connections in the blogging world. Pingbacks allow the author to know when another site has mentioned the URL of any page within your blog. Here's the code we need to make this happen:

```
<link rel="pingback" href="<?php bloginfo('pingback_url');
?>" />
```

Favicon Image

Like our stylesheet, our favicon image is being referenced with a Relative URL, so to get it showing we need to update with an Absolute URL as follows:

```
<link rel="shortcut icon" href="<?php bloginfo('template_
directory'); ?>/images/favicon.ico" />
```

Leaving Space for Plugins

The final piece of code we'll add is a vital part of any WordPress theme:

```
<?php wp_head(); ?>
```

When your theme is processed for viewing, WordPress replaces this tag with any extra code needed to run plugins that the user has setup. So if you install a plugin that for example needs to run some Javascript – as many do – when you open the site in a browser and check the source code you will see the tag has been replaced by a set of script includes. Similarly some plugins will add CSS code, extra HTML, meta tags or other snippets of code. If `wp_head` is not present your theme will simply not be compatible with many common WordPress plugins.

The <head> Code

So here is our complete `<head></head>` section

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <meta http-equiv="Content-Type" content="<?php
bloginfo('html_type'); ?>; charset=<?php
bloginfo('charset'); ?>" />
    <meta name="generator" content="WordPress <?php
bloginfo('version'); ?>" />
    <title><?php bloginfo('name'); ?><?php wp_title(); ?>
</title>
    <link rel="stylesheet" href="<?php bloginfo('stylesheet_
url'); ?>" type="text/css" media="screen" />
    <link rel="alternate" type="application/rss+xml"
```

```
title="RSS 2.0" href="<?php bloginfo('rss2_url'); ?>" />
    <link rel="pingback" href="<?php bloginfo('pingback
url'); ?>" />
    <?php wp_head(); ?>
    <link rel="shortcut icon" href="<?php
bloginfo('template_directory'); ?>/images/favicon.ico" />
</head>
```

Image URLs

With the `<head>` section all set to go, we'll next quickly go through and fix all our broken image links. As we discussed earlier, these need to be migrated from their current Relative URLs to Absolute URLs like this:

```
<img src="<?php bloginfo('template_directory'); ?>/images/
sample.jpg" alt="Example Image!" />
```

The fastest way to make this change is to use your text editor to run a *Find + Replace* on your code. Simply search for the string:

```
src="/images/
```

And replace all instances with:

```
src="<?php bloginfo('template_directory'); ?>/images/
```

With that change and the fix to your stylesheet include, your theme should be looking almost identical to the original `blog.html`. You can test your theme at any time by loading the latest version of the theme folder into `wp-content/themes` on your WordPress server and (assuming the theme has already been activated) refreshing your site.

Dynamic Navigation and Adding Pages

The next part after the `<head>` section that needs some fixing up is the navigation. At the moment it's static and leads to nothing. Here we will use another Template Tag to list all Pages present.

Simply take out all the `<li>`'s within the `<ul id="menu">`, and replace the code with this:

```
<ul id="menu">
<li><a href="<?php bloginfo('url'); ?>" title="Home">Home</
a></li>
<?php wp_list_pages('title_li='); ?>
</ul>
```

So there are two parts to this code. The first `<li>` just links back home – a necessity for all websites really. The second line, however is the real magic. The template tag `wp_list_pages` does exactly that – lists all the pages, only we have some power over how they are displayed.

Here we pass a value for the `title_li` attribute, saying that it's going to be blank. This is written as: `'title_li='` and tells WordPress that we want to modify it's default output – which is to place a title wrapped in `<li></li>`'s before listing all the pages (which we don't want).

Like most Template Tags, there are loads of parameters one can use. To add values for additional parameters you simple separate as many as you wish with an & character. For our needs though, one is sufficient. View the Codex reference for `wp_list_pages` at: http://codex.wordpress.org/Template_Tags/wp_list_pages

If we now test the theme you'll see that only 'Home' and 'About' are showing up, what's with that? That's because you don't have any Pages added on your site! Open up the WordPress Dashboard, and select *Pages* > *Add New*, add a title, fill in some dummy content, click publish and if you flip back to your site that page will now magically appear in your navigation. Say hello to the power of a dynamically managed site!

Note that instead of using `wp_list_pages`, it is actually possible to simply place static links by getting the Permalink for each page and manually coding them in, like this for example:

```
<li><a href="http://example.com/about/"
title="about">About</a></li>
```

However you should generally only do this if the theme is for personal use as you have no control over whether other blog users will create pages with the exact same link structure, and your theme may end up with broken links in the navigation.

Creating a Featured Post with WP_Query

Working down again, we now come to our Featured Post block. You're about to come into contact with WordPress' famous loop – and some additional customising using a very handy piece of WordPress code called `WP_Query`.

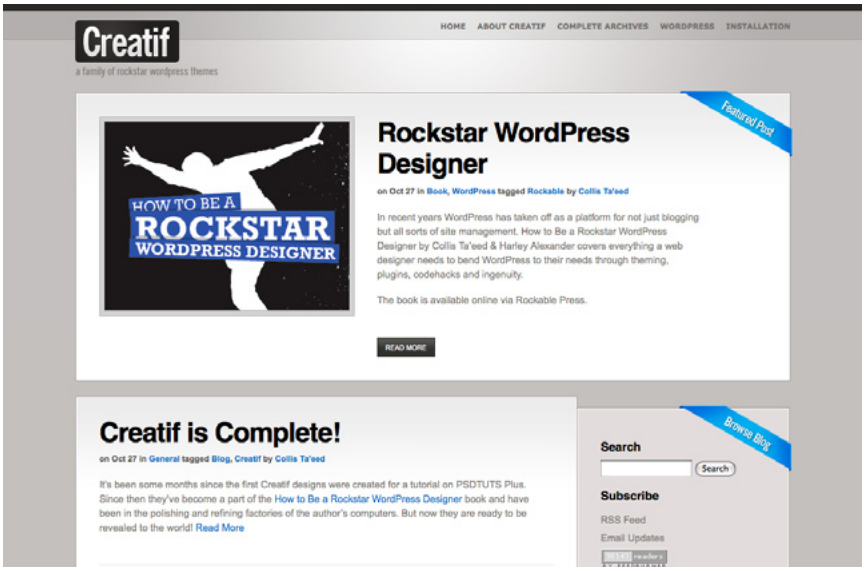


Fig 5-3 – The featured post section.

So, what we're going to do is get rid of everything between the initial `<div class="block_inside">` to the `</div>` just below the Read More button, and replace the code with this:

```
<div class="block_inside">

    <?php
        $featured = new WP_Query();
        $featured->query('showposts=1');
        while($featured->have_posts()) : $featured->the_
post();

            $wp_query->in_the_loop = true;
            $featured_ID = $post->ID;
    ?>
        <?php if (get_post_meta($post->ID, 'large_preview',
true)) { ?>
```

```
                <div class="image_block">
                    <img src="<?php echo get_post_meta($post-
>ID, 'large_preview', true); ?>" alt="Featured Post" />
                </div>
            <?php } ?>
            <div class="text_block">

                <h2><a href="<?php the_permalink(); ?>"
title="<?php the_title(); ?>"><?php the_title(); ?></a></
h2>
                <small>on <?php the_time('M d'); ?> in <?php
the_category(', '); ?> tagged <?php the_tags(''); ?> by
<?php the_author_posts_link(); ?></small>
                <?php the_content('Read More'); ?>

            </div>

        <?php endwhile; ?>
    </div>
```

"Whoah", you say. What on EARTH is this?

Basically what we've written here is a custom query. Because WordPress generally just runs one big loop through the codebase, and we want one special featured post, we need to query the database for that first post to feature. Later we'll use the regular WordPress loop (i.e. The Loop) to show the rest of the posts on the page.

Featured posts are actually quite an advanced bit of theming, so as you read through the explanation below, don't stress if it's not 100% clear. There are a few advanced PHP concepts used in that bit of code, and for the most part you can just copy and paste it without really needing to understand too deeply. Later as you go through other parts of the book we'll come back to the methods used here and things will all fit into place!

So let's go through line by line:

1. The first PHP line tells WordPress that we're going to run a new WordPress Query. This line is creating a PHP *object*. An object is like a regular variable on steroids, it can do lots of things like hold variables inside it, execute special functions (called methods) and even have other objects derived from it. But don't worry, object-oriented programming is not what we're interested in here! The main thing you need to know is that an object is a sort of container of data and functions, that has been defined somewhere else for us to make use of here.

   So we're calling our object `$featured`. We create it by running a special method called a constructor, written as new `WP_Query()`. After that line is called we can then use the `$featured` object to do its various bits of cleverness!

2. In the next line of PHP we run one of `$featured`'s methods called `query()`. Methods are just like regular functions – just like those Template Tags we've been using all this time. The difference is that a method is specially attached to an object. So here the object is `$featured` and whenever you run the `query()` method, it does stuff to the data held inside `$featured`. In this case it queries the database and stores the results in our object.

   So the `query()` method like many of the Template Tags we've used can take parameters. In this case we'll give a value for the parameter `showposts=1`. Unsurprisingly this parameter tells the query to only get 1 post.

There are many different parameters, and like Template Tags such as `wp_list_pages`, these parameters can be separated by an `&`. You can view a full list of parameters on the query_posts codex page: http://codex.WordPress.org/Template_Tags/query_posts.

3. The next line begins a `while` loop. As you know a while loop executes a block of code over and over again while a certain condition is true. So here the code:

```
while($featured->have_posts()) ... endwhile;
```

will execute everything in between as long as `have_posts()` – another method on the `$featured` object – returns true.

As you will see in between these two bits of code we have our HTML with some WordPress Template Tags outputting the Post title, content and associated information like tags and date posted.

Now because earlier when we ran the `query()` method and we specified it to only grab one post, this while loop is only going to execute one time. So strictly speaking we don't actually need a while loop. However it's a useful code snippet because sometimes you might want to feature more than one post, in which case you simply change the value of `query('showposts=1')` to `query('showposts=2')` or more!

4. Below that we have a line of code that reads:

```
$wp_query->in_the_loop = true;
```

This line is actually to clear up a small issue that arises when using `WP_Query`. Namely that the Template Tag `the_tags('')` won't work without this being here. You can safely ignore this line as being an oddity that just needs to appear here.

5. The next line is:

```
$featured_ID = $post->ID;
```

This line creates a variable called `$featured_ID` and gives it the value of the Post's ID number. Later on when we are showing all the rest of the Posts on the page we'll check them against this ID number to make sure we don't print this same Post out again.

You might have guessed from the syntax used that `$post` is an object. It's a different object type to the `WP_Query` object we created earlier. In this case the `$post` object contains lots of information about the Post in variables like `ID` which can be referenced with the `->` notation. For the most part though we'll use Template Tags like `the_title` and `the_time` to extract that information because they are a bit simpler to use.

6. The next four lines check if there's a featured post image and show it if it's there. They use those Custom Fields we setup earlier to store the image URL. We'll cover Custom Fields in detail in the next Chapter.

7. Finally we have the post heading, text and information. To fit with the HTML we have, we need to display the title wrapped in a link to the actual page. Then display some meta information wrapped in a `<small>` tag in the format of: 'on Month, Day in category, category tagged tag, tag, tag'.

The reason `the_time` is used, and not the similar template tag: `the_date` is because the latter only shows the date once per day – if you posted numerous times a day, only the most recent would show the date. Finally, we have the content with the text of the read more link.

And with that we have a featured post. Between the `WP_Query` and the use of a Custom Field, this is some of the most advanced code you'll ever need to use, so don't worry if that seemed a little complex, we're just throwing you in the deep end!

All you really need to understand is that we're asking the database for some information, telling it we only want one Post, and then while we have that Post outputting all the relevant details of it into our specially formatted HTML.

Showing the Rest of the Posts

With the featured Post sorted, next we need the rest of the blog's posts to be displayed in the main content area. We do this in a similar way to the previous bit of code, except this time we don't need to create a special query, we can use WordPress' default query and loop over it – this is commonly known as *The Loop* – which we discussed in Chapter 4.

So moving down the page from the featured Post, replace everything within the `<div class="block_inside">` with this code:

```
<?php if(have_posts()) : while(have_posts()) : the_post(); ?>
    <? if ($post->ID != $featured_ID) { ?>
        <h2><a href="<?php the_permalink(); ?>"
title="<?php the_title(); ?>"><?php the_title(); ?></a></h2>
        <small>on <?php the_time('M d'); ?> in <?php the_
```

```
category(', '); ?> tagged <?php the_tags(''); ?> by <?php
the_author_posts_link(); ?></small>
        <?php the_content('Read More'); ?>
        <div class="separator biggap"></div>
    <? } ?>
<?php endwhile ?>
    <div id="posts_navigation">
        <?php previous_posts_link(); ?>
        <?php next_posts_link(); ?>
    </div>
<?php else : ?>
    <h2 class="center">Not Found</h2>
    <p class="center">Sorry, but you are looking for
something that isn't here.</p>
    <?php include (TEMPLATEPATH . '/searchform.php'); ?>
<?php endif; ?>
```

This code is basically The Loop that we looked in Chapter 4. As you'll see we have two bits of PHP in quick succession in the first line. First we run a quick `if()` statement to see if we even have any Posts, after all there is no point executing all that code if there wasn't any!

So everything between:

```
if(have_posts()):
```

and

```
else:
```

is executed only if `have_posts()` returns true – i.e. there are some Posts to show. Otherwise we show a short message saying there are no Posts found.

The second bit of code that comes right after the if statement is a while loop. As we've discussed before, a while loop executes everything in between the while and the `endwhile`, so long as the condition is true (the condition again being `have_posts()`).

Now in the featured Post part of the code we had to actually query the database to get some Posts for our while loop to go through. That was all that `WP_Query()` code, remember? But this time we will take advantage of the fact that every WordPress page executes one default query and so we can simply loop over it using the loop code without bothering to create that special object we did previously. This is commonly termed as The Loop.

So for every post we go through and output the Post title, the date it was created, the tags it has, the author, an excerpt, and a separator `<div>` element.

Note however that we've wrapped this output code in a big `if` statement like this:

```
<? if ($post->ID != $featured_ID) { ?>
...
<? } ?>
```

This statement checks if the current post's ID matches the featured Post's ID that we stored earlier. The `!` = mark means *not equal*. So in plain English we're saying if the Post's `ID` doesn't match the featured Post's `ID`, then do everything in between. That way we don't reprint the same featured Post again.

The number of Posts outputted by The Loop is actually set inside WordPress' Dashboard under *Settings* > *Reading*. Depending on how many posts you have to show, you may need buttons at the end to show the previous or next set of posts. For that we have this code:

```
<div id="posts_navigation">
    <?php previous_posts_link(); ?>
    <?php next_posts_link(); ?>
</div>
```

These links only appear if they need to. So if there are no more Posts to go to, or no more Posts that haven't already been shown on a previous page, then the respective links won't display.

Building the Sidebar

With the main Posts appearing, it's time now to look at our sidebar. Sidebars in WordPress themes can contain a variety of different elements and are an important part of a blog theme.

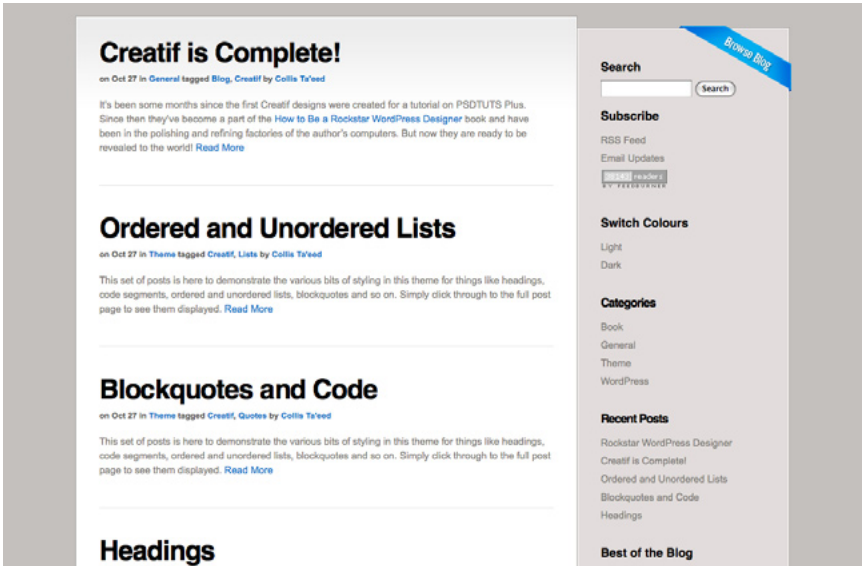


Fig 5-4 – Sidebars are an important part of any blog theme.

At the moment, the sidebar simply consists of a `<div class="block_inside">` within the wrapping sidebar `<div>`. So get rid of everything within the inside block, and create a new unordered list:

```
<ul>
</ul>
```

An unordered list is the best way to organise a sidebar, we simply make each part of the sidebar into a list item, those parts containing their own child unordered lists for their respective information.

Search

For sites containing more than a few pages, it is a good idea to include a search tool. WordPress comes with built-in search functionality that is surprisingly easy to implement. All you need is a simple search form.

Because the search form is likely to appear in multiple places – the sidebar, the archives, the 404 and so on – it's a good idea to create a separate PHP file containing the code and then using an PHP Include to place it wherever required.

So create a new file called `searchform.php` inside the theme directory. Inside this file, place the following code to create our form:

```
<form method="get" id="searchform" action="<?php
bloginfo('url'); ?>/">
<p>
<input type="text" value="<?php echo wp_specialchars($s,
1); ?>" name="s" id="s" size="15" />
<input type="submit" id="search_submit" value="Search" />
</p>
</form>
```

In the `<form>` tag we have defined two important things. The first is that the form will post its results via the `get` method. In other words the variables the person types in will be passed as a URL. The second is that the `action` for where the form is to go, is set as the blog's main URL. Also note that in the text input field, the id is the letter `'s'`.

Put this together and you'll find that if a person were to type in `'search_word'` into the field, the browser would be directed to this URL:

```
http://www.example.com/?s=search_word
```

WordPress processes this type of URL as a search query. From the file hierarchy in Chapter 4, we know that it will first attempt to show the file `search.php`, then since that file isn't present, it will fall back and show `index.php`.

Finally note that the `<input>` field has a value set to:

```
<?php echo wp_specialchars($s, 1); ?>
```

In PHP the word `echo` simply means to print out the following statement onto the page. That statement is a function to strip out any special characters from the variable it is passed – in this case `$s`, which has a value only if a search query has just been executed, in which case it will display the previously searched term.

Including the search form in the sidebar

With our search form file created, we now only need to include it in our sidebar. Remember each widget in the sidebar is to be wrapped in an `<li></li>` element, so to include the search form we need this code:

```
<li id="search" class="widget"><h3>Search</h3>
    <?php include(TEMPLATEPATH.'/searchform.
php'); ?>
</li>
```

You will recall the middle line is a PHP Include, discussed in Chapter 4.

RSS and FeedBurner

RSS or Really Simple Syndication is a way to package the site's content for use elsewhere. People generally use RSS feeds in RSS readers to stay up to date with a site. However they may also use the feed to syndicate content onto another website or application.

An RSS feed is basically just a specially formatted text file that is updated to contain the latest content wrapped in some XML tags to standardize the information. WordPress has a built-in system for producing RSS feeds for both recent Posts and comments on Posts.

Although WordPress comes equipped with the basics of what we need, your RSS feeds are more powerful with the addition of Google Feedburner which can make your RSS feed more compatible, provide statistics and add special formatting and advertising. Additionally FeedBurner can repackage a feed as email updates for those users not up to speed with RSS.

In this section we'll discuss how to push your feed through FeedBurner, add email subscriptions and create an icon to display the number of readers.

Burning your RSS Feed

First up, you're going to need to register your RSS Feed with FeedBurner. There are two ways to find your feed. If you have permalinks switched on you can use this URL:

```
http://example.com/feed/
```

Or without permalinks, your feed URL will look like:

```
http://example.com/?feed=rss2
```

Now that you have your Feed URL, you can now head over to http://www.feedburner.com/, and register. Once completing the short registration, you "burn" your feed. During the process, you can select a couple of extra options to monitor your feed by; such as clicks, downloads of enclosed files, and so on.

Visit your feed by clicking on *My Feeds*, and visiting your feed (click the tiny RSS icon next to its name)! Great, you can now select the URL of your feed at FeedBurner.

Creating Email Subscriptions

What you really want to focus on is the options you're now shown. Most importantly, *Publicize*. You're given a wide range of ways to publicize your feed, including email subscriptions.

So back in our `index.php`, another widget needs to be created to accommodate the RSS and Email Subscription links. Just as we did with the Search widget, create a new `<li></li>` element in the sidebar and inside it we'll place another unordered list, like this:

```
<li id="subscribe" class="widget"><h3>Subscribe</h3>
    <ul>
            <li></li>
    </ul>
</li>
```

You can now fill the first `<li>` with a link to the FeedBurner URL you just created. As for the second list item, it will require a special link – the *Email Subscription* link. Go back to your FeedBurner Dashboard, where you were looking at the page *Publicize* and select *Email Subscriptions*. Select this, and activate it. Select your desired language, and copy all the code provided in the second text box, titled *Subscription Link Code*.

Create another `<li>`, this time paste the code from the text box on FeedBurner. Feel free to change the text value between the `<a>` tags, from `'Email Updates'` to whatever you wish. Your code should now look something like this:

```
<li id="subscribe" class="widget"><h3>Subscribe</h3>
    <ul>
            <li><a href="http://feeds.feedburner.com/
yourFeed">RSS Feed</a></li>
            <li><a href="http://www.feedburner.com/fb/a/emai
lverifySubmit?feedId=1234567&amp;loc=en_US">Email Updates</
a></li>
    </ul>
</li>
```

Adding a Subscription Count

Finally, we want to add a Subscription Counter to show off how many subscribers we have. So head back to FeedBurner, once again to the *Publicize* page, this time go to *FeedCount*. Activate the feed count, then select some colours of your preference. To fit with

the Creatif theme, I'd suggest using `#7f7d78` as the body colour, and `#eeeeee` as the text colour.

You'll be given some code that looks like this:

```
<p><a href="http://feeds.feedburner.com/CreatifBlog"><img
src="http://feeds.feedburner.com/~fc/CreatifBlog?bg=7f
7d78&amp;fg=eeeeee&amp;anim=0" height="26" width="88"
style="border:0" alt="" /></a></p>
```

As you've probably guessed, we now type in a new `<li>`, and place this code within it.

Updating or Redirecting the Feed URL in <head>

Now currently the feed URL in our `<head>` area is still the main WordPress feed URL. Remember it's using this code:

```
<link rel="alternate" type="application/rss+xml" title="RSS
2.0" href="<?php bloginfo('rss2_url'); ?>" />
```

This feed still works, and can be accessed through a browser address bar, but it means that FeedBurner's statistics won't be tracking people using this URL. You could just change the value of href and hardcode in your new FeedBurner URL, but a more elegant solution is to use the FeedSmith plugin to redirect your feed. You can grab this from: http://www.google.com/support/feedburner/bin/answer.py?answer=78483

Doing More with Subscriptions

As you can see, it is relatively quick to integrate FeedBurner to your WordPress blog. It provides a simple and easy way to give readers options to subscribe to your blog and read content regularly, without causing them a fuss!

To view stats for your feed, you can click on the Analyze tab, and it will show you a range of information that is picked up by FeedBurner. This is great in figuring out how many people subscribe, how many people visit your site, and so on.

Recent Posts

Although recent Posts are always going to be displayed in the main content area, it can be handy to have a short list of their titles in the sidebar. Using `WP_Query` again, we'll now create a widget that displays a list of a number of recent Posts, however many you wish. For our example we'll show the most recent seven. Here's the code:

```
<li id="recent_posts" class="widget"><h3>Recent Posts</h3>
    <ul>
        <?php
            $recent = new WP_Query();
            $recent -> query('showposts=7');
            while($recent -> have_posts()) : $recent ->
the_post();
        ?>
            <li><a href="<?php the_permalink(); ?>"
title="<?php the_title(); ?>"><?php the_title(); ?></a></
li>
        <?php endwhile; ?>
    </ul>
</li>
```

So you'll notice that this is a very similar loop to the one used to grab the featured Post earlier. The only difference is that we've set the query to grab 7 Posts instead of 1. We've then avoided showing the content of each Post and just printed a linked title. If you wished you could easily add a small segment of the Post, or almost any other detail of the Posts.

Using `WP_Query` is a very flexible method of getting recent Posts. An alternative method is to use the Template Tag `wp_get_archives()`, a tag usually used to get monthly archives (see below). By passing some alternative parameters we can grab the most recent post titles instead.

Here's the code:

```
<li id="recent_posts" class="widget">
    <ul><?php wp_get_archives('type=postbypost&limit=7');
?></ul>
</li>
```

Monthly Archives

Date based archives show the reader a list of dates and allow the reader to click through and see all Posts from those dates. You can choose whether to display monthly, weekly, daily or yearly archives. Like most blogs, we're going to use monthly.

The Template Tag used here will again be `wp_get_archives()`. Two parameters need to be set – 'type' and 'limit'. We want to show by the month and a maximum of seven:

```
<li id="archives" class="widget"><h3>Archives</h3>
    <ul><?php wp_get_archives('type=monthly&limit=7'); ?></ul>
</li>
```

This will simply display an unordered list of the seven more recent months – displayed as *'December 2008'*. Later we'll look further in depth at archives, we're only scratching the surface with this little beauty!

Codex Page for `wp_get_archives`: http://codex.wordpress.org/Template_Tags/wp_get_archives

Categories

Another common way to browse older content is via Post categories. Like archives, categories have a very simple Template Tag used to call for them: `wp_list_categories()`. It works a little differently to archives however, and is much more similar to `wp_list_pages`, in the sense we need to rid of the `<h2>` title it generates by default:

```
<li id="categories" class="widget"><h3>Categories</h3>
    <ul>
            <?php wp_list_categories('title_
li=&orderby=name'); ?>
    </ul>
</li>
```

The title is hidden, and they are ordered in alphabetical order! There are more parameters to control whether you want to exclude a certain category, show the number of Posts and other behavior.

Codex Page for `wp_list_categories`: http://codex.wordpress.org/Template_Tags/wp_list_categories

Completed Sidebar

Our sidebar is now completely customized. You could leave your sidebar this way, completely hard-coded, however in the next section we'll turn the sidebar into a widgetized sidebar – more appropriate for public release.

Widgetizing the Sidebar

Widgetization is a concept integrated into WordPress 2.3 and basically is a way of making the sidebar editable in WordPress'

dashboard. Creating a widgetized sidebar is not difficult, and is great when releasing your theme to a public who may not be very code savvy. It means they can then add widgets, rearrange the order and change some options all from WP-Admin. A widgetized theme presents an extra option in the *Appearance* tab labelled *Widgets*. From this page the user can add, edit, rearrange or simply remove widgets.

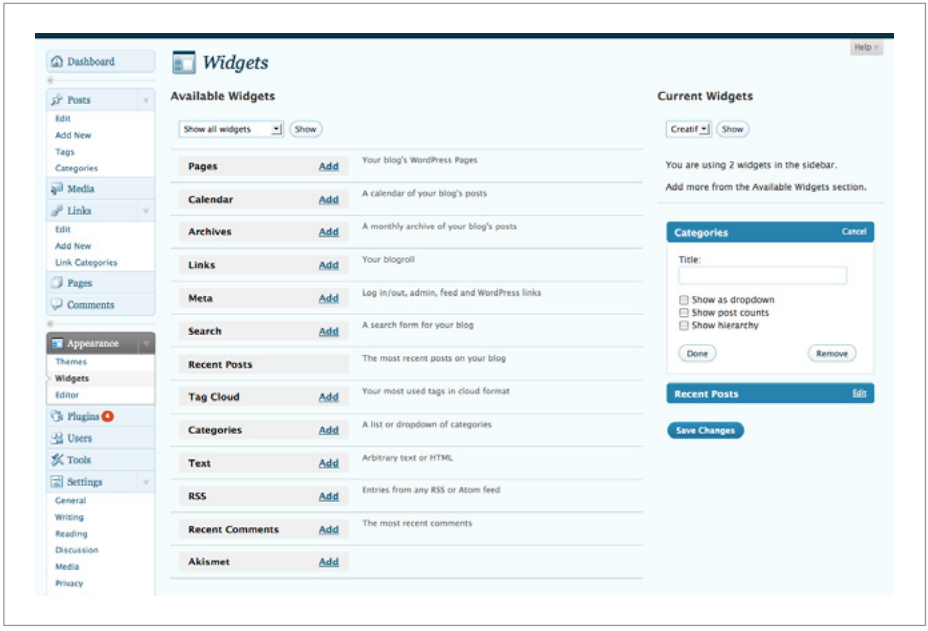


Fig 5-5 – Widgets enable the user to easily change sidebar content.

Widgetizing a sidebar requires a special function, so you must first create a new file and call it `functions.php`. Inside this file, paste the following code:

```
<?php
    if(function_exists('register_sidebar')){
            register_sidebar(array('name' => 'Creatif'));
    }
?>
```

That's it. It's important to re-name the sidebar within the array above, so that in the admin area you can select your specific sidebar to edit.

Next we need to edit the `index.php` file to check if a dynamic sidebar exists. If it does then WordPress will swap it in here and if not it will simply display everything in between the if statement.

```
<?php if ( !function_exists('dynamic_sidebar') ||
!dynamic_sidebar() ) : ?>
... old sidebar code
<?php endif; ?>
```

If you have some fixed, hardcoded elements – for example the FeedBurner widget, you would leave those in above or below the widgetized code and they will always display.

Note: *Dynamic Sidebar headers use* `<h2>` *'s not* `<h3>` *'s, so you may need to change some CSS to accommodate the widgets.*

So our final sidebar code looks like this:

```
<div id="sidebar">
    <img src="<?php bloginfo('template_directory'); ?>/
images/ribbon_browse.png" class="ribbon" alt="Featured
Project"/>
    <div class="block_inside">
        <ul>
            <li id="search" class="widget"><h3>Search</h3>
                <?php include(TEMPLATEPATH.'/searchform.
php'); ?>
            </li>

            <li id="subscribe" class="widget"><h3>Subscribe</h3>
                <ul>
                    <li><a href="<?php bloginfo('rss2_
```

```
url'); ?>">RSS Feed</a></li>
                    <li><a href="http://feedburner.google.
com/fb/a/mailverify?uri=psdtuts">Email Updates</a></li>
                    <li><a href="http://feeds.feedburner.
com/psdtuts"><img style="border:0" src="http://feeds.
feedburner.com/~fc/psdtuts?bg=a6a2a0&amp;fg=ffffff&amp;anim=0"
alt="" width="88" height="26" /></a></li>
                </ul>
            </li>
            <li id="search" class="widget"><h3>Switch
Colours</h3>
                    <ul id="color_switch">
                        <li id="switch_light"><a>Light</
a></li>
                        <li id="switch_dark"><a>Dark</a></li>
                    </ul>
            </li>
            <?php if ( !function_exists('dynamic_sidebar')
|| !dynamic_sidebar() ) : ?>
                <li id="recent_posts" class="widget"><h3>
Recent Posts</h3>
                    <ul>
                        <?php
                            $recent = new WP_Query();
                            $recent ->
query('showposts=7');
                            while($recent -> have_posts())
: $recent -> the_post();
                        ?>
                            <li><a href="<?php the_
permalink(); ?>" title="<?php the_title(); ?>"><?php the_
title(); ?></a></li>
                        <?php endwhile; ?>
                    </ul>
                </li>
                <li id="archives" class="widget"><h3>Archives
```

```
    </h3>
                        <ul><?php wp_get_archives('type=monthly
    &limit=7'); ?></ul>
                    </li>
                    <li id="categories" class="widget"><h3>
    Categories</h3>
                        <ul>
                            <?php wp_list_categories('title_
    li=&orderby=name'); ?>
                        </ul>
                    </li>
                <?php endif; ?>
            </ul>
        </div>
     </div>
```

You can learn more about widgetizing themes including how to specify your own custom versions of widgets here: http://codex.wordpress.org/Widgetizing_Themes

The Footer

With the sidebar sorted, it's time to work on the footer. The sidebar has a text blurb, normally you would just leave this written in the HTML, however instead we're going to move the text into a text file called `about.txt` and then include it with this line:

```
<?php include(TEMPLATEPATH.'/about.txt'); ?>
```

As you can see PHP Includes can be used to include other types of files, not just PHP ones.

Next there are some links to be shown – but who wants to edit the `index.php` every single time you partner up with a new site? Instead we'll use WordPress' Blogroll functionality.

The blogroll is a set of links and link categories set up in WordPress' dashboard. To display them in the footer, replace the links code with:

```
<ul>
    <?php wp_list_bookmarks('title_li=&categorize=0'); ?>
</ul>
```

This is a similar Template Tag to `wp_list_pages` which we used in the menu, in the sense that we need to get rid of the `title_li` however there is an additional `categorize=0` option needed. Without these two settings, WordPress will output a `<h2>` heading and wrap the list in `<ul></ul>` elements.

In the last column, there are a few RSS links. Head back up the page to your manual sidebar, or go back to your FeedBurner page and grab the URL of your feed to replace those.

Lastly we need to add another hook for WordPress' plugin API. As you recall in the header we added the `wp_header()` function, here just before the end of the HTML we add:

```
<?php wp_footer(); ?>
```

So our final footer code look like this:

```
    <div id="footer">
        <div class="container">

        <div class="footer_column long">
            <h3>About Rockable Press</h3>
            <?php include(TEMPLATEPATH.'/about.txt'); ?>
              </div>
         <div class="footer_column">
            <h3>More Links</h3>
                    <ul>
```

```
                    <?php wp_list_bookmarks('title_
li=&categorize=0'); ?>
                </ul>
            </div>

            <div class="footer_column">
                <h3>RSS</h3>
                        <ul>
                    <li><a href="<?php bloginfo('rss2_
url'); ?>">RSS Feed</a></li>
                        <li><a href="http://whatisrss.
com">What is RSS?</a></li>
                </ul>
            </div>
        </div>
    </div>
<?php wp_footer(); ?>
</body>
</html>
```

Splitting the Page Up

So at this point, our theme is working, at least the homepage is. The next step is to start to make all the other pages – search results, archives, single Posts, Pages and so on. But before we do that, it's time to do a little housekeeping.

Currently we have one major file – `index.php`, and some auxiliaries – `about.txt`, `style.css`, `searchform.php`. If we were to start making pages like those mentioned above, we would need to create a lot of duplication between `index.php` and the new files. Namely the header, sidebar and footer will usually remain exactly the same no matter what page is displaying.

So instead we are going to create three new files:

1. **header.php** – Contains everything up to the end of the `<div id="header"></div>`

2. **sidebar.php** – Contains all the code shown at the end of the sidebar section

3. **footer.php** – Contains all the code shown at the end of the footer section

Back in `index.php`, we replace the code we've removed with these three lines of code (placed respectively where each piece of code has been removed)

```
<?php get_header(); ?>
<?php get_sidebar(); ?>
<?php get_footer(); ?>
```

As you recall from Chapter 4, these are three include functions provided by WordPress for including the three most common files. They work in exactly the same way as the usual PHP Includes we used for the search form and about text.

`Header.php`, `sidebar.php` and `footer.php` are essential to any reasonably complex theme – if you don't create these, altering the smallest detail in the footer of your site can be much more troublesome as you have to trawl through all your template files to make sure you haven't missed any instances. With our universal header, sidebar and footer, changing anything is simply a matter of hitting a single file each time.

Featuredpost.php

This methodology of moving code segments into sub template files is also good practice for making your theme neat and tidy. So next

we'll create a new file – `featuredpost.php` – and paste in all the code inside `<div id="block_featuredblog">`.

We then replace the cut code with:

```
<?php if(is_home()){ include(TEMPLATEPATH.'/featuredpost.
php'); } ?>
```

As you recall from Chapter 4, the first part is a Conditional Tag that checks to see if the page currently being displayed is the homepage. If it is, then we display our featured Post, if not, we skip right on past.

Creating the Single Post and Single Page

In Chapter 4 we looked at the file hierarchy and how WordPress will always check to see if certain files are available in the theme directory to decide what to serve up in any given scenario. In particular if a reader visits a single post page:

Single Post Example – www.example.com/post_title

```
single.php > index.php
```

So next we'll create a `single.php` file that will be shown instead of the index. In our new file we can add things that are specific to viewing a single Post, in particular we will later add comments!

First create a file called `single.php` and copy in the contents of `index.php`. Since we don't need featured Posts, we can remove the include line. Next we currently have:

```
<?php the_content('Read More'); ?>
```

But we don't need a Read More link, so we can replace this line with:

```
<?php the_content(); ?>
```

Code for a Single Post

So the full code of `single.php` is:

```
<?php get_header(); ?>
    <div id="block_content">
            <div id="content_area" class="block">
                <div class="block_inside">
                    <?php if(have_posts()) : while(have_
posts()) : the_post(); ?>
                        <h2><a href="<?php the_permalink();
?>" title="<?php the_title(); ?>"><?php the_title(); ?></a></h2>
                        <small>on <?php the_time('M d'); ?>
in <?php the_category(', '); ?> tagged <?php the_tags(' ');
?> by <?php the_author_posts_link(); ?></small>
                        <?php the_content(); ?>
                    <?php endwhile; ?>
                    <?php else: ?>
                                        <p>There are no
posts to display. Try searching:</p>
                          <?php include(TEMPLATEPATH.'/
searchform.php'); ?>
                    <?php endif; ?>
                </div>
             </div>
            <?php get_sidebar(); ?>
             <div style="clear:both"></div>
            </div>
        </div>
    </div>
<?php get_footer(); ?>
```

You can already see how handy it is that we made our footer, sidebar and header into separate sub template files!

Single Page

Now before we go and add comments to our Post, let's first duplicate the `single.php` file and name the new version `page.php`. As you recall from Chapter 4, when WordPress needs to display a Page, it checks for these files:

Page – www.example.com/page_title

```
page_title.php > page.php > index.php
```

So now we're using our single Post template as our Page template as well. If you wanted to you could make adjustments to this file that would only show up when a visitor was viewing a single Page. And in fact you could make a special version of the `page.php` file on top of that and name it after a specific Page to even further customize. As it stands though, our simple duplicate will do us just fine.

Adding Comments

A fundamental part of our blog is missing from the single Post page, that's right, comments! What blog is a blog without comments on blog posts?

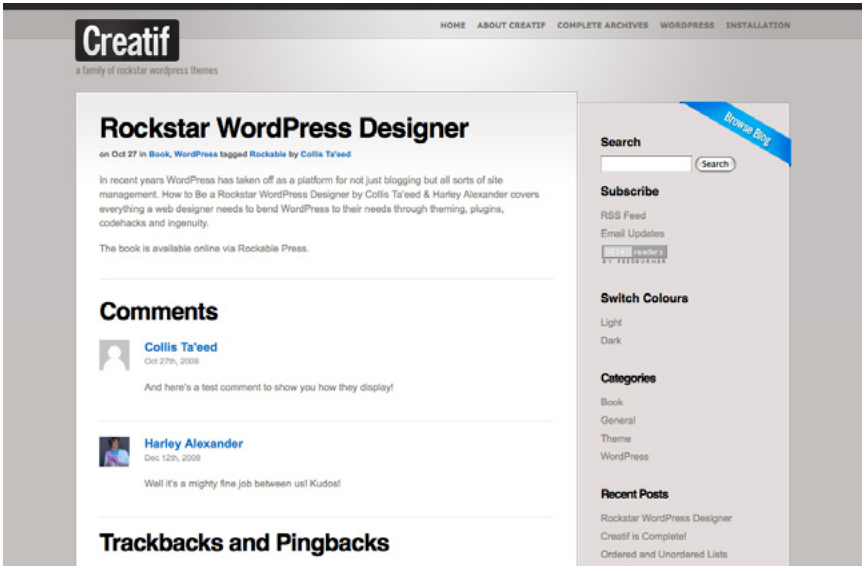


Fig 5-6 – Comments on a blog post.

So begin by creating a file called `comments.php` and then go back to `single.php` and just after the `endif;` paste in this code:

```
<div id="comments_template">
    <?php comments_template(); ?>
</div>
```

Along with the header, sidebar and footer, comments have a special include tag that works just like a regular include. You should also note that you can in fact add comments to Pages as well, but in our theme we won't be doing that.

Now `comments.php` can be one of the more intimidating theme files, but we'll go through it piece by piece and you'll find it's not too scary in the end. The file is comprised of three parts:

1. A check to see if the comments are password protected
2. A loop through the comments
3. An "Add your Comment" form

So let's look at each of these in turn.

Checking for Password Protection and an Attached Post

The first lines we add to the `comments.php` file are always the same:

```
<?php
if ('comments.php' == basename($_SERVER['SCRIPT_
FILENAME'])) die ('Please do not load this page directly.
Thanks!');
if (!empty($post->post_password)) {
    if ($_COOKIE['wp-postpass_' . COOKIEHASH] != $post-
>post_password) {
?>

<h2>This Post is Password Protected!</h2>
<p>Please enter the password to view Comments.</p>

<?php return;
    }
}
?>
```

What this code does is first, it checks to see if a person has loaded up `comments.php` directly – i.e. without loading up `single.php` or some other template file. If that happens, the page simply stops and returns an error message. Later on down the page we'll be

referencing a PHP object called `$post`, this has to be set in the file that has included `comments.php`. So if we simply called the file directly this object would be blank and the whole file would result in errors.

Next we check to see if the page has been password protected. WordPress gives authors the option to password protect a Post or Page when writing. It is essential to check for this, to avoid confusion. So the code checks if the Post's password field is empty, and if it isn't then we check if the user has logged in, else we give them an error message.

For the most part you don't need to worry about this code snippet, just copy+paste it in to your comments files when theming.

The Comment Loop

```
<?php if ($comments) : ?>

    <h2>Comments</h2>
    <ol id="comments">
            <?php foreach ($comments as $comment) : ?>
                    <!-- Looping content -->

            <?php endforeach; ?>
    </ol>
<?php else : ?>

    <!-- If there are no comments... -->
<?php endif; ?>
```

The next part of the code is the main comment loop. Shown above is the structure of the loop, where we've replaced some segments of code with an HTML comment to simplify it for explanation.

The code begins by checking if there are any comments, if there are we're going to show them as an ordered list `<ol>`, and if not then we'll later add some code to tell the user.

The loop through the comments is similar to the for loops we discussed in Chapter 4. However it is a slightly different variant called a *Foreach Loop*. This loop cycles through a set of items and then acts on each one.

So in our case we have the set of `$comments`, and we're going to go through each one in turn display them. With each cycle of the loop we take the next comment and place it in `$comment` so that we can run some standard template tags to extract the comment information.

For Each Comment

So for each comment we will display an `<li></li>` element. Inside the element we will use some standard Template Tags for extracting comment details, namely:

`comment_ID()` – Outputs the ID number of the comment

`comment_author_link()` – Outputs the commenter's name with a link to their specified URL

`comment_date()` – Outputs the date/time of the comment

`comment_text()` – Outputs the actual comment text

We will also use the `get_avatar` function to add Gravatars to the comments. Gravatars are small images that users can create at http://gravatar.com and which are associated with a comment by means of an email address. Since 2.5 WordPress has integrated Gravatar functionality and adding them is extremely simple. We just need this code:

```
<?php if (function_exists('get_avatar')) {
    echo get_avatar(get_comment_author_email(),'40');
} ?>
```

The code first checks that the `get_avatar` function exists or in other words that this theme is running on an instance of WordPress newer than 2.5. If it does then we pass the email address of the commenter as well as the size in pixels that we want the image. The output will be an `<img>` tag with a `class="avatar"` so that we can format it correctly!

Here's how each comment will look:

```
<li id="comment-<?php comment_ID(); ?>">
    <h4><?php comment_author_link(); ?></h4>
    <small><a href="#comment-<?php comment_ID(); ?>">
            <?php comment_date('M jS, Y'); ?>
    </a></small>
    <div class="the_comment">
            <?php comment_text(); ?>
    </div>
    <?php if (function_exists('get_avatar')) {
            echo get_avatar(get_comment_author_
email(),'40');
    } ?>
</li>
```

Alternating Comment Classes

Now currently every comment looks the same, but we want them to alternate their CSS class so that we can differentiate visually between comments. To do this, we first need to set a variable before the loop begins like this:

```
<?php $altcomment = 'alt'; ?>
```

Then in the loop we'll edit the `<li></li>` element slightly to set its class like this:

```
<li class="<?php echo $altcomment; ?>" id="comment-<?php
comment_ID(); ?>">
```

So the first <li> element will have the class "alt". Now we need to add some code to make this variable switch back and forth on each alternate pass. So just before the `<?php endforeach; ?>` statement we add this code:

```
<?php
    if ($altcomment == 'alt') {
            $altcomment = '';
    } else {
            $altcomment = 'alt';
    }
?>
```

This if statement swaps the value of `$altcomment` between "alt" and nothing each time.

Note that if you're not used to PHP, in the condition we are checking, you need to use a double == sign. A single = tells PHP to assign the value to the variable, whilst a double == check if they are equivalent. This is an important distinction.

If there are No Comments

Displaying comments is all set, but what if there are no comments to display? We need to tell WordPress what to do when that happens, so that users are informed rather than left hanging. In some instances the post may not even have comments enabled, while in others there simply might not be any yet. So in the event there are no comments, we'll use this code:

```
<?php if ($post->comment_status == 'open') : ?>
    <p>There are no comments yet, add one below.</p>
<? else : ?>
    <p>Comments are closed.</p>
<?php endif; ?>
```

So here we check if the `comment_status` is set to open and depending on the outcome display one of two messages to the user.

The Full Comment Loop Code

So our final loop code looks like this:

```
<?php $altcomment = 'alt'; ?>
<?php if ($comments) : ?>
            <h2>Comments</h2>
    <ol id="comments">
            <?php foreach ($comments as $comment) : ?>
                    <li class="<?php echo $altcomment; ?>"
id="comment-<?php comment_ID(); ?>">
                            <h4><?php comment_author_link();
?></h4>
                            <small><a href="#comment-<?php
comment_ID(); ?>">
                                    <?php comment_date('M jS,
Y'); ?>
                            </a></small>
                            <div class="the_comment">
                                    <?php comment_text(); ?>
                            </div>
                            <?php if (function_exists('get_
avatar')) {
                                     echo get_avatar(get_
comment_author_email(),'40');
```

```
                                    } ?>
                        </li>

                        <?php
                                if ($altcomment == 'alt') {
                                        $altcomment = '';
                                } else {
                                        $altcomment = 'alt';
                                }
                        ?>
                <?php endforeach; ?>
        </ol>
    <?php else : ?>

        <?php if ($post->comment_status == 'open') : ?>
                <p>There are no comments yet, add one below.</p>
        <? else : ?>
                <p>Comments are closed.</p>
        <?php endif; ?>

    <?php endif; ?>
```

The Leave a Comment Form

The final part of the comments file is a form for readers to add their own comments. This part looks a lot more complicated because we have to check if the reader needs to be logged in, if they are logged in, or if they are not logged in. Just remember though, it's basically just a form.

So let's go through piece by piece:

```
    <?php if ($post->comment_status == 'open') : ?>
                <h3>Leave a Comment</h3>
        <?php if ( get_option('comment_registration') &&
```

```
!$user_ID ) : ?>
            <p>You must be <a href="<?php bloginfo('url');
?>/wp-login.php?redirect_to=<?php the_permalink();
?>">logged in</a> to post a comment.</p>
    <?php else : ?>
```

First we check to see if comments are open. Assuming they are, we show a title saying "Leave a Comment", and then check if registration / login is necessary to comment. This is a setting that the blog owner sets in WordPress' dashboard. If the user does need to login, we send them to the `wp-login` page with a redirect value set to bring them back afterwards.

If they are either already logged in, or it's not necessary to be registered, then we start showing the comment form:

```
<form action="<?php bloginfo('url'); ?>/wp-comments-post.
php" method="post" id="commentform">
<?php if ( $user_ID ) : ?>
    <p>Logged in as <a href="<?php echo get_
option('siteurl'); ?>/wp-admin/profile.php"><?php echo
$user_identity; ?></a>.
    <a href="<?php echo get_option('siteurl'); ?>/wp-login.
php?action=logout" title="Log out of this account">Logout</
a></p>
<?php else : ?>
```

First we have a `<form>` tag defining what action to take. Then we check if the reader is a registered/logged in user, because if they are then we don't need to ask for much information, instead we can print their identity and give them a logout link.

Now if the reader isn't logged in, we know we don't need to show them a register link, because we already checked for that previously. So instead we can assume that registration is not necessary and show them some extra form fields to fill in for name, email and URL:

```
<?php else : ?>
<p><input type="text" name="author" id="author"
value="<?php echo $comment_author; ?>" size="50" />
<label for="author"><small>Name <?php if ($req) echo
"(required)"; ?></small></label></p>
<p><input type="text" name="email" id="email" value="<?php
echo $comment_author_email; ?>" size="50" />
<label for="email"><small>Mail (will not be published)
<?php if ($req) echo "(required)"; ?></small></label></p>
<p><input type="text" name="url" id="url" value="<?php echo
$comment_author_url; ?>" size="50" />
<label for="url"><small>Website</small></label></p>
```

This code snippet is mostly just HTML form fields. Note that we check a PHP variable called `$req` which WordPress sets to lets us know if a field is mandatory. Also you will see that the `value`'s of the fields are set in case the form has to be displayed again (for example a required field wasn't filled out) or if we are dealing with a returning commenter – to make their life easier.

Finally, a `textarea` is needed to put the actual comment. Because this `textarea` applies to both registered and unregistered users, we need an `endif` statement first.

```
<?php endif; ?>
<p><textarea name="comment" id="data" cols="60" rows="7"
tabindex="4"></textarea></p>
<p><input name="submit" type="submit" id="submit"
tabindex="5" value="Submit Comment" />
<input type="hidden" name="comment_post_ID" value="<?php
echo $id; ?>" />
</p>
<?php do_action('comment_form', $post->ID); ?>
</form>
<?php endif; >
```

The hidden input is the ID of the new comment being submitted to the Database and `do_action()` is another hook function to let user-installed WordPress plugins add extra functionality to the comments section. For example a live comment preview plugin would output some code there.

Wrap up of the Comments.php File

So with that we have a basic comments file. There is a lot more PHP involved in the comments page, but as you work with it you'll find it's not too complicated. In the next chapter we'll do some more advanced work on the file to separate trackbacks from comments. We'll also look at how you can add threaded comments – a feature that has only just been introduced since WordPress 2.7. But for now, we're all done here.

Customizing a Search Results Page

As we discussed earlier, when a user searches for something in WordPress, the following file hierarchy is used:

Search – www.example.com/?s=search_word

```
search.php > index.php
```

To make a custom search results page, we simply copy everything from `index.php` to `search.php` and then make a few changes to customize the page.

First we'll delete the featured post include and change `the_content('Read More')` to `the_excerpt()`. The difference between these two functions is that the first will add a "Read More" only if the writer has specified it in the Post using the `<!--more-->`

tag. The second function will *always* display an excerpt, regardless of the writer's actions. Since we are displaying the results of a search, it's important to ensure only excerpts are displayed.

Just above the search results, we'll add a page title to remind the user what they searched for. The title is a simple matter – a value is saved every time a search is submitted as the variable `$s`. So to display that search term, we use an `echo` statement:

```
<h4>Search results for '<?php echo($s); ?>'.</h4>
<div class="separator"></div>
```

This is just a little bit of raw PHP, just displaying the value of `$s`.

And that's it, we now have a custom search results page.

The Archives

There are many ways a user may wish to browse the archives of a blog – by date, by category, by author or by tag. The hierarchies for these four archive types look like this:

Date Archives – www.example.com/2008/12

```
date.php > archive.php > index.php
```

Category Archives – www.example.com/category/category_title

```
category-id.php > category.php > archive.php > index.php
```

Tag Archives – www.example.com/tag/tag_name

```
tag-slug.php > tag.php > archive.php > index.php
```

Author – www.example.com/author/author_name

```
author.php > archive.php > index.php
```

As usual all four default back to `index.php` at their most basic, but note that also all four fall back to `archive.php`. So if we have a clever `archive.php` we can handle all four types of archives, so let's create that now!

First save `search.php` as `archive.php` then replace the title `<h4>` line with:

```
<h4>Archive of <?php wp_title(' ', true, '');?></h4>
```

Next replace the following lines up to `<?php endwhile; ?>` with this code:

```
<ul>
    <?php if(have_posts()) : while(have_posts()) : the_
post(); ?>
            <li><a href="<?php the_permalink(); ?>"
title="<?php the_title(); ?>"><?php the_title(); ?></a></li>
    <?php endwhile; ?>
</ul>
```

This creates an unordered list of all Post titles in the archive. We now have a nicely formatted list based archive that will be shown for any of the four types of archives.

Adding a Custom 404 Page

A 404 error message is shown when a file, Post, Page or URL is wrong or has gone missing. A lot of the time, you'll see the some pretty unhelpful 404 pages around. A few simple things can be displayed here to improve the user experience dramatically.

A short message, search form and sitemap will give the user all the tools they need to find what they were looking for. So create a new file called `404.php` and paste in this code:

```
<?php get_header(); ?>
    <div id="block_content">
            <div id="content_area" class="block">
                <div class="block_inside">
                 <h1>404 - page not found</h1>
                 <p>Sorry, the page you're looking for has
gotten lost! Either keep looking, or try to find what you
were after below.</p>
                 <div class="separator"></div>
                 <h3>Search the Site</h3>
                 <?php include(TEMPLATEPATH.'/searchform.
php'); ?>
                 <div class="separator"></div>
                 <h3>Sitemap</h3>
                 <ul><?php wp_get_archives('type=postbypost
'); ?></ul>
                 <div class="separator"></div>
                 <h3>Pages</h3>
                 <ul><?php wp_list_pages('title_li='); ?></ul>
                </div>
             </div>
            <?php get_sidebar(); ?>
             <!-- a Clearing DIV to clear the DIV's because
overflow:auto doesn't work here -->
             <div style="clear:both"></div>
            </div>
        </div>
    </div>
<?php get_footer(); ?>
```

This code follows the same basic structure as all our other template files, but our content includes a general error message, a search form, using our regular include:

```
<?php include(TEMPLATEPATH.'/searchform.php'); ?>
```

And a sitemap made using two template tags we've seen previously: `wp_get_archives` and `wp_list_pages`. You can test out your 404 page by going to this URL:

```
http://example.com/index.php?error=404
```

Or by visiting any page that doesn't exist! If your server isn't showing WordPress' 404 page for missing pages, you may need to edit the `.htaccess` file to specify where the 404 page is. You can get details on doing this from the Codex: http://codex.wordpress.org/Creating_an_Error_404_Page

Author Pages and Multiple Authors

One of the great things about WordPress is that it can be setup to work with multiple authors, whether it's a buddy hired to write extra content, or a professional organization with multiple staff. New authors can be set up via the WP-Admin dashboard under *Users*. Once setup each author will have posts they've written, a bio, an author URL so it makes sense to make a special page to display their information.

As we know when an author page is accessed, WordPress follows this hierarchy of pages:

Author – www.example.com/author/author_name

```
author.php > archive.php > index.php
```

We've already got that general purpose archive, but it'd be nice if we also listed the author's details on their page. So create a file called `author.php` and copy over the contents of the `archive.php` file as a base to work off. Replace the heading area with the following code:

```
<?php
    if(isset($_GET['author_name'])) :
        $curauth = get_userdatabylogin($author_name);
        else :
        $curauth = get_userdata(intval($author));
    endif;
 ?>
 <h1>About <?php echo $curauth->display_name; ?></h1>
 <div id="auth_desc"><?php echo $curauth->description; ?></div>
 <div class="separator"></div>
 <?php if ($curauth->user_url): ?>
 <p><a href="<?php echo $curauth->user_url; ?>"
class="button">Visit <?php echo $curauth->display_name;
?>'s Website</a></p>
 <?php endif; ?>
<div class="separator"></div>
```

Displaying author information isn't as nicely coded as most other parts of WordPress theming and requires a bit of unusual looking code. We first check that the author_name has been passed via the URL – e.g. www.example.com/author/author_name – and then depending on the outcome, we use one of two `get_userdata` functions to extract information about the author and create an object called `$curauth` to hold that information. We can then access data like the authors `display_name`, `description` and `user_url`.

You can get more information on coding author pages from the Codex: http://codex.wordpress.org/Author_Templates

Wrap up of Creatif Blog

In this chapter you have learnt the skills to make a completely customized blog theme. Most WordPress theming can be achieved using the information in this chapter, so you should now be able to download other people's themes and understand how they work.

With the basics of theming out of the way, we'll spend the remaining chapters of this book looking at advanced theming, taking WordPress beyond the blog and into use as a flexible CMS. You'll also get more comfortable with general theming practices. So if you found this has been a lot of information to absorb all at once, don't worry too much. Just keep reading and experimenting then come back and go through this chapter again and it should all be much clearer.

Adding a Screenshot!

But before we move on, it's time to give our faithful blog a screenshot, so that users can preview it before they apply the theme.

So simply navigate to the homepage of your blog, take a screenshot using your preferred method and save it as *screenshot.png*, placed in your themes directory. And now in the dashboard under Appearance you should see your favorite site appearing all nice and professional looking.

6

Tools for Advanced Theming

In the previous chapters we covered the bulk of theming basics, enough to build and deploy themes for most blogging projects. In this chapter we'll look at some more advanced WordPress theming techniques that will allow you to not only build more complex blog themes, but to use WordPress as a content management system for a variety of non-blog projects.

We will do this by extending WordPress' functionality through plugins, Custom Fields, clever PHP, and by reorienting existing WordPress functionality into alternate uses. Although this is more advanced work, it is nonetheless easy to grasp and use.

Tools for Advanced Theming

In the previous chapters we covered the bulk of theming basics, enough to build and deploy themes for most blogging projects. In this chapter we'll look at some more advanced WordPress theming techniques that will allow you to not only build more complex blog themes, but to use WordPress as a content management system for a variety of non-blog projects.

We will do this by extending WordPress' functionality through plugins, Custom Fields, clever PHP, and by reorienting existing WordPress functionality into alternate uses. Although this is more advanced work, it is nonetheless easy to grasp and use.

If Statements and Conditionals

Conditional statements such as `if` statements check if certain conditions are true and depending on the outcome execute different pieces of code. They can be extremely useful and in this section we'll look at a few more uses to show how they can be applied in different scenarios.

Scenario 1: Degrading Gracefully

When you design WordPress themes for the public, you can never be sure what your users are going to do with their WordPress installs. Plugins may or may not be installed, comments may or may not be switched on, specific pages may or may not be there. How your theme handles different non-optimal situations can be improved by some clever planning and use of conditional statements. Let's look at an example.

Imagine you have created a theme and for the archives you ideally want the user to install a certain hypothetical plugin called

extended_archives. In fact you've written it into the download and installation instructions. But you happen to know that there is another very popular archives plugin called *regular_archives*. Moreover you also know there are a lot of people who are going to install your theme and never bother with plugins at all, or perhaps simply not know how to install them correctly! What do you do?

Option 1 – Just Paste in The Code

Since you specifically stated that the user should have the *extended_archives*, you could just paste in the code (let's say hypothetically it looked like the code below)

```
<li class="widget">
    <h3>Browse Extended Archives</h3>
    <?php extended_archives(); ?>
</li>
```

However if the user hasn't installed this plugin, or does so incorrectly, then at best your page will have an empty spot, and at worst it will cause errors on the page or simply cause the page not to show. So this is not a very good idea.

Option 2 – Check If It Exists ...

A better solution is to check if the function exists (i.e. the plugin is installed) before attempting to run it, like this:

```
<li class="widget">
    <?php if(function_exists('extended_archives')) { ?>
            <h3>Browse Extended Archives</h3>
            <?php extended_archives(); ?>
    <?php } ?>
</li>
```

With a neat if statement we have removed any chance of seeing errors or killing our page. Now the only possibilities are the plugin appearing or there being an empty spot.

Option 3 – Check If It Exists ... Else If ...

Still a big ugly gap isn't really a good scenario either. Since we know the alternate *regular_archives* plugin is also popular, it'd be good to check if that's installed as a fallback, and for that we can extend our if statement with an `elseif` – providing for an alternate case:

```
<li class="widget">
    <?php if(function_exists('extended_archives')) { ?>
            <h3>Browse Extended Archives</h3>
            <?php extended_archives(); ?>
    <?php } elseif(function_exists('regular_archives')) { ?>
            <h3>Browse Regular Archives</h3>
            <?php regular_archives(); ?>
    <?php } ?>
</li>
```

So here we check if the first plugin exists, and if it doesn't, then we check if the second plugin exists. You can have as many `elseif`'s as you want. Note also that it's important to use an `elseif` and not just another if statement, because a user who had both plugins installed should only have one used. With two if statements, both plugins would appear, but with an `elseif`, the second is only considered if the first condition has failed.

Option 4 – Check If It Exists ... Else If ... Else ...

Still if the user doesn't have either plugin, we are still left with a blank spot. So we need a fallback for when all else fails and there are no plugins installed. In that case, we'll revert back to

WordPress' regular archives. We can add this default case with a final `Else`:

```
<li class="widget">
    <?php if(function_exists('extended_archives')){ ?>
            <h3>Browse Extended Archives</h3>
            <?php extended_archives(); ?>
    <?php } elseif(function_exists('regular_archives')) { ?>
            <h3>Browse Regular Archives</h3>
            <?php regular_archives(); ?>
    <?php } else { ?>
            <h3>Browse Wordpress Archives</h3>
            <ul><?php wp_get_archives(); ?></ul>
    <?php } ?>
</li>
```

This process of checking for different scenarios and handling them in turn is called degrading gracefully. It is an important use of conditional statements, particularly in situations where your themes will be used out in the wild and you have little or no control over what your users will do.

Scenario 2: Splitting Comments and Trackbacks

Trackbacks (or Pingbacks) are links back to one blog from another. They encourage conversation between sites and are an integral part of blogging. WordPress views trackbacks as a kind of comment and will automatically list incoming trackbacks to a post amongst its comments. This isn't ideal as people reading comments are generally interested in reading all the comments together. So it would be much better to list the trackbacks separately either before or after the comments.

As you recall from Chapter 5, our `comments.php` file consists of a big loop going through all the comments. Now if we could somehow determine on each pass whether we were dealing with a comment

or a trackback, then we could decide whether to show it or not. Then we could do a second pass and only show the reverse type.

The function `get_comment_type()` does exactly what we need. Using an `if` statement we can compare the comment type with a string. As you know from PHP there are several comparison operators: `==` equivalent, `!=` not equivalent, `<` less than, `>` greater than.

Open up `comments.php` from our Creatif Blog theme from Chapter 5. Before editing the file, you may wish to save a backup copy. Now find the main `foreach` loop. Just inside this loop we add an if statement, so below the `foreach` line and above the `endforeach` line, that checks if the comment type is equivalent to `'comment'`:

```
<?php foreach ($comments as $comment) : ?>
    <?php if(get_comment_type() == 'comment'): ?>
            <li class="<?php echo $altcomment; ?>"
id="comment-<?php comment_ID(); ?>">
                    <h4><?php comment_author_link(); ?></h4>
                    <small><a href="#comment-<?php comment_
ID(); ?>">
                    <?php comment_date('M jS, Y'); ?></a></
small>
                    <div class="the_comment">
                            <?php comment_text(); ?>
                    </div>
            </li>
            <?php
            if ($altcomment == 'alt') {
                    $altcomment = '';
            } else {
                    $altcomment = 'alt';
            } ?>
    <?php endif; ?>
<?php endforeach; ?>
```

So now in each cycle of the `foreach` loop the block of code is executed only if it's a comment. So now comments show, but what about trackbacks? We simply create another loop and check for the opposite case this time – i.e. that the comment type is not `'comment'`. So just below the end of the `<ol></ol>` list, we create a second heading and an `<ol></ol>` list, like this:

```
<h2>Trackbacks and Pingbacks</h2>
    <ol>
            <?php foreach ($comments as $comment) :
            $comment_type = get_comment_type();
            if($comment_type != 'comment') { ?>
                    <li><?php comment_author_link() ?></li>
            <?php } endforeach; ?>
    </ol>
```

That's it! Your comments area is now be successfully split to display trackbacks below the regular comments, all with the magic of if statements!

Scenario 3: Switching the Title with Conditional Tags

In Chapter 4 we looked at Conditional Tags. These are functions that return true if a certain page is being viewed. For example `is_category()` returns true only if the page being viewed is a category listing.

Conditional Tags are great for switching content around without creating multiple files. A good example of this can be found in our header.php file where we are setting the page title. Because the `<title></title>` tag is so important to search engines it makes sense to optimize it as much as possible on each page.

As you recall, in our Creatif Blog example, our title tag looks like this:

```
<title><?php bloginfo('name'); ?><?php wp_title(); ?></title>
```

Now that's OK, but wouldn't it be better to display information specific to each page? With an if statement and our knowledge of WordPress Template Tags, this is easily done, here's an example:

```
<title><?php
if(is_home()) {
    echo bloginfo('name').' - Home';
} elseif(is_category()) {
    echo 'Browsing the Category ';
    wp_title(' ', true, '');
} elseif(is_archive()){
    echo 'Browsing Archives of';
    wp_title(' ', true, '');
} elseif(is_search()) {
    echo 'Search Results for "'.$s.'"';
} elseif(is_404()) {
    echo '404 - Page got lost!';
} else {
    bloginfo('name'); wp_title('-', true, '');
}
?></title>
```

This is a good example also of how `elseif` can be used over and over. In different situations we've either outputted using an `echo` statement, the value of a Template Tag such as `bloginfo`, or used the `wp_title` tag in different ways. One useful bit of PHP knowledge is that you can join two strings together with a `.` character. So in the search example:

```
echo 'Search Results for "'.$s.'"';
```

We are displaying the string `Search Results for "` followed by the search term, followed by ". So if a person searched for keyword, the title would be:

```
Search Results for "keyword"
```

Choosing Specific Pages / Posts

It is also possible to display content on very specific pages. Say you wanted the title for the about page on your blog to be "All About Me", but you didn't want to change the title in WordPress itself for some reason. Different Conditional Tags have different ways of specifying Pages, Posts, categories and so on. You can check them all in the Codex: http://codex.wordpress.org/Conditional_Tags

To choose a specific Page you can use any of these three:

1. **ID** – The ID number of the Post or Page – e.g. 3
2. **Post Title** – The title in WordPress – e.g. About Me
3. **Post Slug** – The string WordPress uses in the Post or Page's URL – e.g. about-me

So let's say we chose to use the ID method. To find the ID of your page go to *Pages* > *Edit* in your admin area, and select *Edit* on the About Page. If you look in the address bar of your browser, you'll see the URL will have an ID at the end of it. This is that Page's ID number. Let's say it is 3.

In that case you can simply append the following `elseif` to our previous title if statement:

```
elseif(is_page('3')) {
    echo 'All About Me';
}
```

Threaded Comments and WordPress 2.7

The release of WordPress 2.7 has brought a whole new set of comment functionality including threading and a new, simpler method for looping through comments. In this section we'll create a new `comments.php` file that uses the new features, then use `functions.php` to switch between the new and legacy versions of our comments depending on what version of WordPress the theme is installed on.

***Note:** 2.7 is backwards compatible, so the code we developed in Chapter 5 and earlier in this chapter will in fact still work in WordPress 2.7*

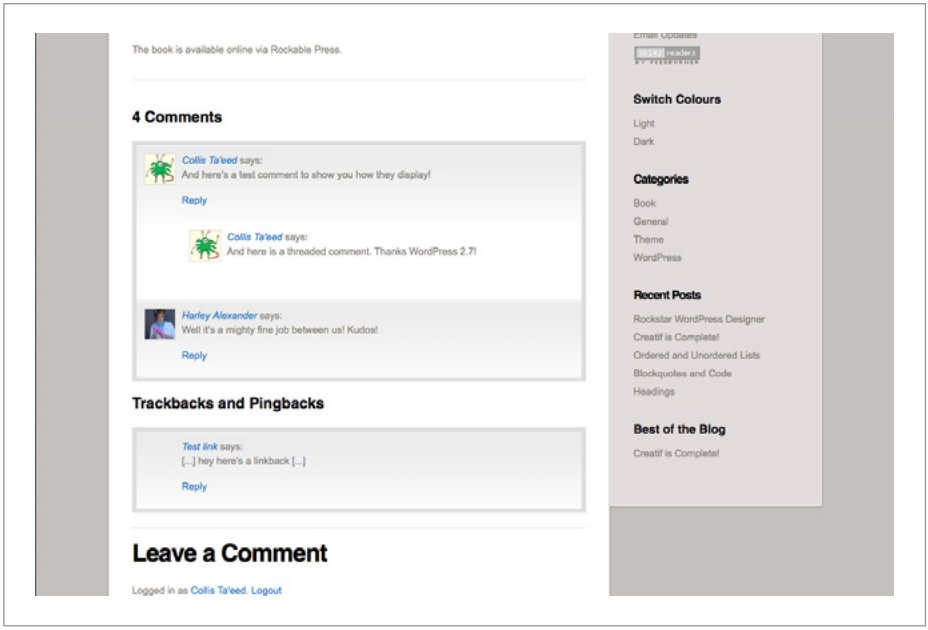


Fig 6-1 – WordPress 2.7 introduced threaded commenting.

Duplicating and Switching

So the first step is to duplicate the current `comments.php` file and name the new file `old-comments.php`. Then in `functions.php` file add these lines below our widgetization functions from Chapter 5.

```
add_filter('comments_template', 'legacy_comments');
function legacy_comments($file) {
    if ( !function_exists('wp_list_comments') )
            $file = TEMPLATEPATH . '/old-comments.php';
    return $file;
}
```

Here we are creating a function that checks if the Template Tag `wp_list_comments` exists. If it does then we know that WordPress 2.7 or higher is present and we can use `comments.php`. But If it doesn't, then we intercept the `comments_template()` call in our `single.php` and use `old-comments.php` instead!

Upgrading the Comment Loop

The biggest change to comments is the introduction of a simpler loop. Our previous loop, that is everything between:

```
<?php if ($comments) : ?>
...
<?php else : ?>
```

Can now be replaced by

```
<?php if ( have_comments() ) : ?>
    <h2>Comments</h3>
    <ul class="commentlist">
            <?php wp_list_comments('type=comment&avatar_
size=40'); ?>
```

```
        </ul>
        <h2>Trackbacks and Pingbacks</h2>
        <ul class="commentlist">
                <?php wp_list_comments('type=pings'); ?>
        </ul>
        <div class="navigation">
            <div class="alignleft"><?php previous_comments_
    link() ?></div>
            <div class="alignright"><?php next_comments_link()
    ?></div>
        </div>
```

As you can see all the various Template Tags we were using to get the comment author, meta data, gravatars and text have been replaced by a single new Template Tag:

```
    <?php wp_list_comments(); ?>
```

This Template Tag takes a few different parameters. Here we've used the `type` parameter to split up our comments from our pingbacks. Another parameter you can use is `style` which determines whether the comments are output as nested `<ul></ul>` lists or as `<div>`'s.

Although there is more than enough HTML output to do any CSS styling that we could possibly need, there is a way to customize the HTML that WordPress uses to output the comments. To do this you need to pass a parameter called `callback` and then define a function in `functions.php` for WordPress to use. You can grab an example function from: http://codex.wordpress.org/Template_Tags/wp_list_comments.

Adding Threading / Javascript Functionality

Before going any further you should check that comment threading has been enabled in your WordPress installation. You can do this by going to *Settings* > *Discussion* and ticking the box that reads *Enable threaded (nested) comments*.

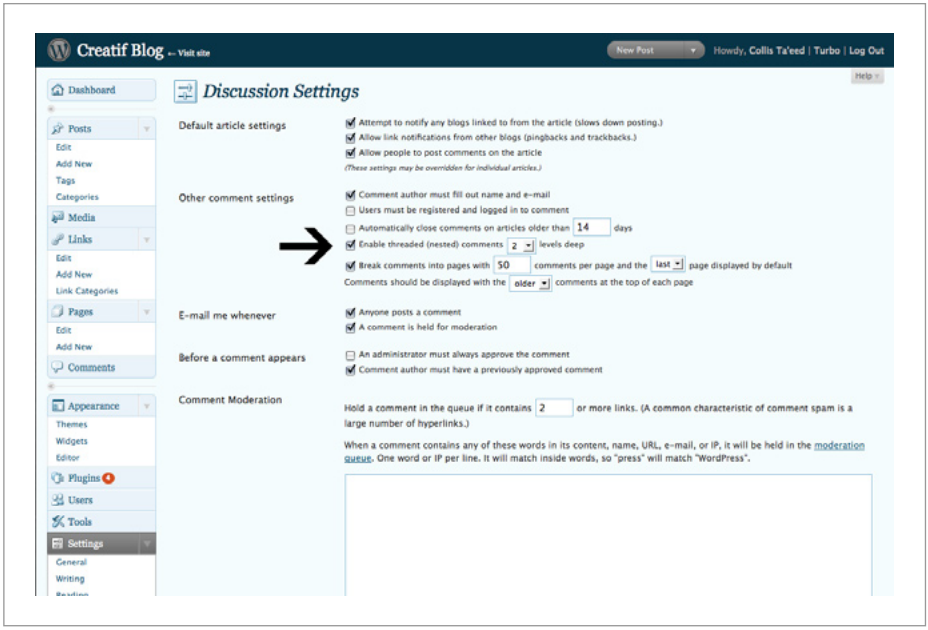


Fig 6-2 – Threading must be enabled to work on your site.

Next we need to add one more line to our `header.php` file to include the Javascript that WordPress 2.7 has introduced to enable the comment form to appear inline with comments. Add this line just before the `wp_head()` line:

```
<?php if ( is_singular() ) wp_enqueue_script( 'comment-
reply' ); ?>
```

Once this is done there are a few more adjustments to be made to the `comments.php` file as follows:

1. Wrap the leave a comment form in `<div id="respond"></div>`, this will enable the Javascript code to find the form.

2. Add `<?php comment_id_fields(); ?>` just inside the `<form>` tag. This line adds a couple of hidden input fields now needed.

3. And then because we have added the line from (2), we can remove the now redundant line:

```
<input type="hidden" name="comment_post_ID"
value="<?php echo $id; ?>" />
```

4. Finally we add a cancel link in case the commenter changes their mind:

```
<div id="cancel-comment-reply"><small><?php cancel_
comment_reply_link() ?></small></div>
```

And that's it! We've successfully upgraded the `comments.php` file. If you wish you can also simplify the first few lines that check where the file has been opened from and whether it is password protected with this slightly slimmer code:

```
<?php
if (!empty($_SERVER['SCRIPT_FILENAME']) && 'comments.php'
== basename($_SERVER['SCRIPT_FILENAME']))
    die ('Please do not load this page directly. Thanks!');
if ( post_password_required() ) {
    echo 'This post is password protected. Enter the
password to view comments.';
    return;
}
?>
```

Custom Fields

One of the most important ways that WordPress allows greater customization and extension is through the use of Custom Fields. These provide a way for WordPress users to provide extra data when creating Posts that the theme designer can then use in different ways. You can find the Custom Field section of a Post by going to the `Posts > Add New` page in the WordPress admin menu and scrolling down the page until you see the section labeled `Custom Fields`.

Each Custom Field is a Name / Value pair. So first we define a name (sometimes referred to as a key) – e.g. *"Post Image"*, *"Related Post URL"*, *"Digg Link"*. Then each time a Post is created, we have to add a value for those keys – e.g. "http://example.com/image-url", "http://example.com/related_post", "http://digg.com/link".

Because we can create any key / value pair, Custom Fields become a way to add extra functionality to the standard WordPress system. Instead of just having fields for title, content, author and so on, we can now have extra fields for just about anything. So let's take a look at some examples.

Simple Example – Featured Post Image

As you recall in our Creatif Blog theme, featured Posts have a 325px wide image image next to them. This image needed to be included somehow on a Post but it can't be in the main text area. So we used Custom Fields.

Creating the Custom Field

As discussed in Chapter 5, to add a Custom Field, simply follow these steps:

1. First in WordPress, we create a new Post, scroll down the page to find the section *Custom Fields*.

2. Under Name (or Key in older versions of WordPress) you can select a previously made Custom Field or create a new one. Since this is the first time we're using the feature, we need to create a new one. In the text box under *Name*, type in: `large_preview` and under *Value* type in the URL to your image: http://example.com/image.jpg then publish the Post.

Accessing the Custom Field Data in Our Theme

To access the Custom Fields, WordPress provides a special Template Tag:

```
<?php the_meta(); ?>
```

Unfortunately this simple tag just produces a `<ul></ul>` list of each name/value pair, which isn't very useful for our purposes. What we need is a function that takes as input the Post ID and key/name and then returns the matching value. That function is called `get_post_meta`, and it works like this:

```
<?php echo get_post_meta($post->ID, 'name', true); ?>
```

The function takes three parameters, the Post ID, the key or name we are looking up and a true/false value that tells the function if we want just one result or a whole array of results. In our case we just want one result – the image URL, and the key is `large_preview`, so the code we use is:

```
<?php if (get_post_meta($post->ID, 'large_preview', true)) { ?>
```

So in our Creatif Blog theme, open up the `featuredpost.php` file. As you will recall from the previous chapter we have a section that reads:

```
<?php if (get_post_meta($post->ID, 'large_preview', true))
{ ?>
    <div class="image_block">
            <img src="<?php echo get_post_meta($post->ID,
'large_preview', true); ?>" alt="Featured Post" />
    </div>
<?php } ?>
```

What we are doing here is first using a simple `if` statement to make sure the image is there, and if it is then we print it out as the source of the `<img>` tag. If we didn't check to make sure the image URL existed and outputted an empty image tag, we'd show a broken image in the browser. So again it's much better to degrade gracefully.

Complex Example – Showing Featured Posts in the Sidebar

Let's try another example. Imagine if we wanted to add a set of featured "Best of" posts at the end of each post, so that readers could quickly go on and browse the best content our site has to offer. What we could do is:

1. Create a Custom Field called `"feature"`. It can have any value at all, because we'll assume that if the field has been set, then the post is meant to be featured. So a simple `"Yes"` value would suffice.

2. Then in our `sidebar.php` file we'll create a custom loop using `WP_Query`. We'll make it loop through the posts checking to see if the Custom Field exists and if it does, then we show a link. Here's the code we need:

```
<li id="featured" class="widget"><h3>Best of the Blog</h3>
        <ul>
            <?php
            $bestof = new WP_Query();
            $bestof -> query('');
            while($bestof -> have_posts()) : $bestof ->
the_post(); ?>
                <?php if (get_post_meta($post->ID,
'feature', true)): ?>
                    <li><a href="<?php the_permalink(); ?>"
title="<?php the_title(); ?>"><?php the_title(); ?></a></li>
                <?php endif; ?>
            <?php endwhile; ?>
        </ul>
</li>
```

So here we run a `WP_Query`, not unlike our query to find recent posts in Chapter 5, only this time we check each time to see if the Custom Field `feature` has a value.

Adding Theme Options

Not every WordPress user is going to be familiar enough with HTML and PHP to edit theme files in order to customize a WordPress theme. Indeed it is our job to make editing the theme files as unnecessary as possible by using widgetized sidebars, by providing appropriate code hooks for plugins and where possible by adding Theme Options.

Theme Options are set through an admin screen when a theme is activated. On the admin screen you might ask the WordPress administrator to decide for example what color scheme they'd like to use, to provide a URL to their logo image or some other customization to tailor the theme to their needs.

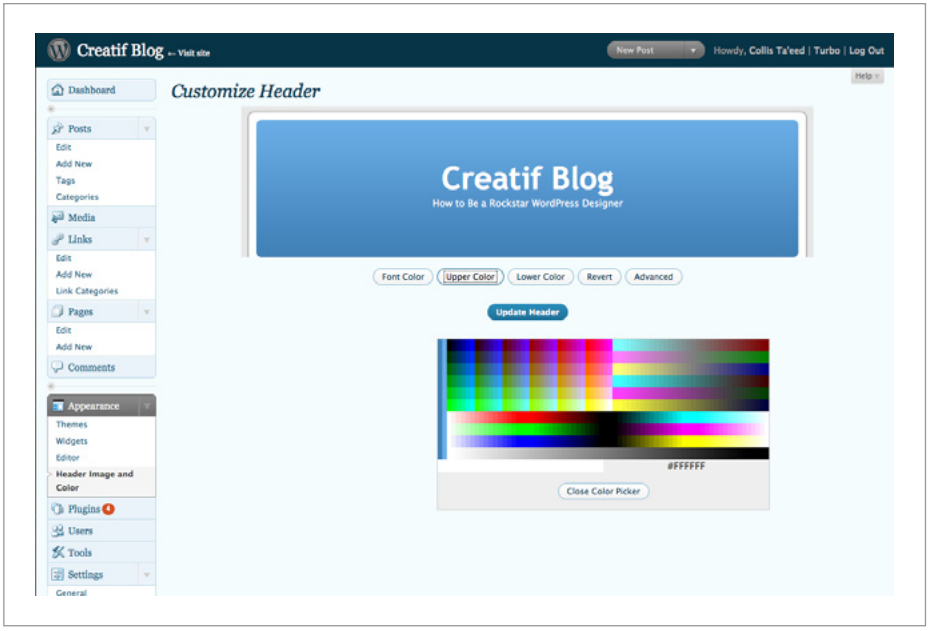


Fig 6-3 – Kubrick's Theme Options page.

An example of a theme which provides Theme Options is the default Kubrick theme. Load up the theme into your WordPress install and you will see under *Appearance* a new tab appears that reads *Header Image and Color*. On this screen you can set options for the header.

Functions.php

An admin screen can be created by using the `functions.php` file in your theme. This file is a special PHP file that behaves like a WordPress plugin and is loaded into both the admin and front-ends of the theme during WordPress initialization – if it is present in the theme directory.

In the previous chapter we used `functions.php` to register our sidebar to add and edit widgets. You can also use the file to add

general functions to do all sorts of things, including adding an admin screen. If you examine the default Kubrick theme that comes with WordPress and open up its `functions.php` file you will find a function called `kubrick_add_theme_page()` with a corresponding `add_action` line before it. These lines create the admin screen to set the options.

It's all fairly complicated though and certainly doesn't make adding Theme Options easy. Fortunately there is a much simpler way that we'll use in this book.

Using WP-Theme-Toolkit

WordPress Theme Toolkit is a PHP class developed by http://planetozh.com that makes adding Theme Options really simple. Here's how:

1. Visit: http://tinyurl.com/wp-theme-toolkit and scroll down to click Download

2. Download the two files `themetoolkit.php` and `functions.php` to your theme directory

3. If you already have functions defined in your `functions.php` file you will need to merge the two files together. Copy your existing functions and paste them at the bottom of the new file in the section labelled *"Additional Features and Functions"*.

4. You can leave `themetoolkit.php` alone, this file does all the hard work behind the scenes, instead, open up `functions.php` and find the four example options, called `setting1` through to `setting4`. You should replace these with your own Theme Options for the user to set using the same formatting.

5. Finally you can either replace the example `creditcard()` function with your own function making use of the Theme Options, or simply use the variables in your theme using this notation:

```
<?php echo $mytheme->option['setting'] ?>
```

In Chapter 8 we'll go through a step by step example of using WP-Theme-Toolkit to add Theme Options to build a custom homepage for our Creatif Site theme.

Building a Basic Plugin

The two main means of extending WordPress are through theming and plugins. We've dealt with theming in some detail now, so let's take a look at plugins. Although you certainly don't need to know how to make your own, it is good to know how they work, where to find them and the sorts of things you can do with plugins.

What is a Plugin?

A plugin is a file or set of files that extend WordPress' core functionality. Plugins are installed in the `/wp-content/plugins/` directory and need to be activated through the WordPress WP-Admin to work.

The files themselves generally contain PHP functions that hook into WordPress either automatically using special API hooks called actions and filters, or manually when you call the functions in your theme like you would a Template Tag, or often a combination of the two.

Most plugins will have an instruction manual or readme file accompanying them explaining how to use the plugin. In some

cases you won't need to modify your theme at all because the plugin overrides the WordPress functionality you're already using – for example it might add a filter to `the_content`. In other cases the plugin might output some extra code in the hook lines we added in our theme, for example replacing `<?php wp_head() ?>` with some custom code in the header.

Where to Find Plugins

There are thousands of developers who create and distribute plugins to do just about everything imaginable. Generally if you need something, try Googling it or searching the WordPress Plugin Directory – http://wordpress.org/extend/plugins/.

Weblog Tools Collection – http://weblogtoolscollection.com/ – is an excellent blog for keeping up with new plugin releases.

Creating a Simple jQuery CSS Switcher Plugin

Creating plugins is an advanced topic and generally beyond the scope of this book. However in this section we're going to do some very simple plugin development to demonstrate the basics of plugin development.

Building the functionality WITHOUT the plugin

As you will recall in Chapter 3, we created two versions of our stylesheet for the Creatif themes, a light and a dark. With some simple use of the Javascript library jQuery – http://jquery.com – we should be able to switch between the two stylesheets. Our aim in this section is to later make a plugin to do this. However first let's make it work simply in our theme files so we can be sure that the underlying idea is working.

First we'll create is a sidebar element. Paste this code into the `sidebar.php` file:

```
<li id="search" class="widget"><h3>Switch Colours</h3>
        <ul id="color_switch">
            <li id="switch_light"><a>Light</a></li>
            <li id="switch_dark"><a>Dark</a></li>
        </ul>
</li>
```

Next grab both the latest jQuery file as well as the jQuery Cookie plugin from http://jQuery.com and http://www.stilbuero.de/2006/09/17/cookie-plugin-for-jquery/.

Then in the `<head></head>` area we add in some jQuery code:

```
   <script type="text/javascript" src="<?php
bloginfo('template_directory'); ?>/scripts/jquery-
1.2.6.min.js"></script>
    <script type="text/javascript" src="<?php
bloginfo('template_directory'); ?>/scripts/jquery.cookie.
js"></script>
<script type="text/javascript">
   $(function(){
              //on the document load, get the cookie
'bodyID' (our colour scheme of choice
              bodyId = $.cookie('bodyID');
              //Set it as the id. If it's dark it'll stay dark!
              $('body').attr('id', bodyId);
              $('#switch_light').click(function(){
                           $('body').attr('id', '');
                           $.cookie('bodyID', '');
                           //set the cookie to nothing if
this is clicked
              });
              $('#switch_dark').click(function(){
```

```
                                $('body').attr('id', 'dark');
                                $.cookie('bodyID', 'dark');
                                //set the cookie to dark if
this is clicked!
                });
    });
</script>
```

If you now run your theme you should see two options in the sidebar reading *Light* and *Dark*. Clicking on each will switch the background colors.

Making the plugin file

Now that we're sure the idea works, we can build it into a plugin. So first create a new directory somewhere to house the plugin. Let's call our plugin "`CSS Switcher`" and so in that directory create a file called `switcher.php` and add this code:

```
<?php
/*
Plugin Name: CSS Switcher
Plugin URI: http://rockablepress.com
Description: A *very* simple plugin to demonstrate plugin
construction. It switches our theme between two CSS files
Author: Collis Ta'eed
Version: .01
Author URI: http://rockablepress.com
*/
?>
```

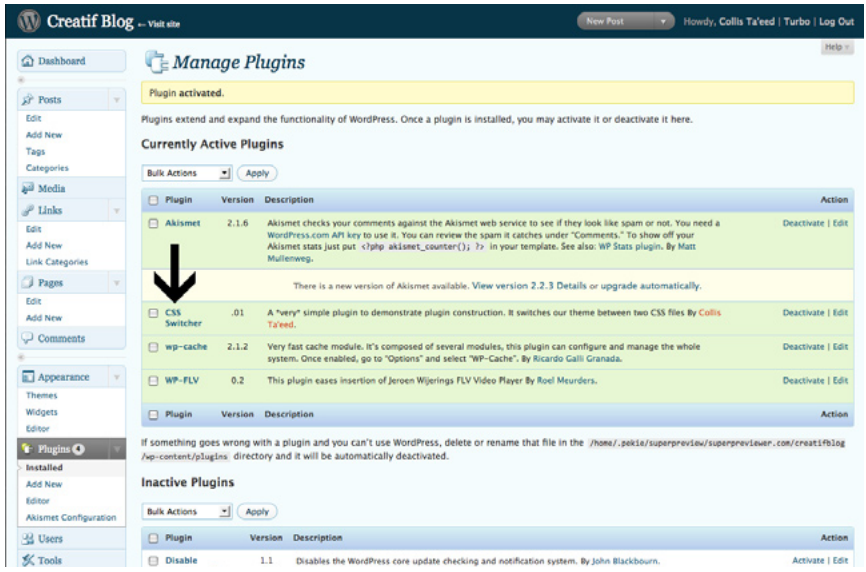


Fig 6-4 – Our simple CSS Switcher plugin.

Make sure there is no space before or after the `<?php ?>` tags or you will get errors when you run the plugin. The comment is similar to the comment we added in our theme `style.css` file in that it outputs information in the WordPress dashboard to identify the plugin. Below the comment code, we'll now add a very simple function:

```
function sidebar_switcher() {

    echo "<ul id='color_switch'>
                <li id='switch_light'><a>Light</a></li>
                <li id='switch_dark'><a>Dark</a></li>
            </ul>";

}
```

This function simply outputs the code we need in our sidebar. We can now go back to the theme files and edit `sidebar.php` to remove the lines we added earlier and replace them with:

```
<li id="switch" class="widget">
    <?php if(function_exists('sidebar_switcher')) { ?>
        <h3>Switch Colours</h3>
        <?php sidebar_switcher(); ?>
    <?php } ?>
</li>
```

As you recall from earlier in this chapter we are checking to make sure the plugin exists, if it does then we run the plugin function – `sidebar_switcher()`. So you can now test this out by uploading your plugin to `wp-content/plugins` and activating it in the WordPress admin area, then uploading the `sidebar.php` file and refreshing your site.

It should all work exactly as before. But hang on, the jQuery parts are still hardcoded into the theme! So let's move those into the plugin file as well.

Hooking into wp_head

Go back to your `header.php` file and delete the jQuery lines we added earlier and re-upload that file. Your theme is now back to normal with only the addition to the sidebar of the code to call the plugin.

Next open up the plugin `switcher.php` file and add these lines:

```
function sidebar_js() {
    echo "
    <script type='text/javascript' src='http://ajax.
googleapis.com/ajax/libs/jquery/1.2.6/jquery.min.js'></script>
```

```
    <script type='text/javascript' src='http://dev.jquery.
com/export/5918/trunk/plugins/cookie/jquery.cookie.js'></
script>

    <script type='text/javascript'>
          $(function(){
             //on the document load, get the cookie
'bodyID' (our colour scheme of choice
             bodyId = $.cookie('bodyID');
             //Set it as the id. If it's dark it'll stay
dark!
             $('body').attr('id', bodyId);
             $('#switch_light').click(function(){
                          $('body').attr('id', '');
                          $.cookie('bodyID', '');
                          //set the cookie to nothing if
this is clicked
             });
             $('#switch_dark').click(function(){
                          $('body').attr('id', 'dark');
                          $.cookie('bodyID', 'dark');
                          //set the cookie to dark if
this is clicked!
             });
       });
    </script>";
}
add_action('wp_head', 'sidebar_js');
```

Again we've created another function, only this time instead of having to call it manually with a Template Tag, we've used something called `add_action`. This hooks our new `sidebar_js` function up with the `wp_head` plugin hook that is in the `header.php` theme file. You can now re-upload the plugin and it should all work perfectly!

Other Hooks and More Information on Plugin Development

The `wp_head` action we just used is just one of many available action hooks, you can find a list of them all at: http://codex.wordpress.org/Plugin_API/Action_Reference

Actions are not the only way to hook in either, you can also use Filters which as the name suggests add Filters to regular WordPress outputs. So for example you could create a Filter to `the_title()` like this:

```
add_filter('the_title', 'some_function');
```

Where `some_function` is a reference to a function does something to the text. You can find the docs for filters at: http://codex.wordpress.org/Plugin_API/Filter_Reference

There is plenty more information on plugin topics such as creating option screens, licensing and so on in the Codex: http://codex.wordpress.org/Writing_a_Plugin

Page Templates

As you recall from Chapter 4, all WordPress operates on a Page hierarchy. That is when a certain Page is called, WordPress moves down the hierarchy of theme files to determine which file to use, with all of them defaulting to `index.php` if nothing else is available.

For Pages, you will recall we define this hierarchy as:

Page – www.example.com/page_title

```
some_template.php > page.php > index.php
```

In this case the `some_template.php` file refers to what is known as a Page Template which has been applied to that Page in WordPress' admin area. You can create multiple Page Templates and assign them to different Pages as you please. You can create a Page Template by copying the contents of our default Page Template `page.php` into a new file, let's call it `mytemplate.php` and adding this code to the top of your template file:

```
<?php
/*
Template Name: MyTemplate
*/
?>
```

Then in WordPress when you create a Page you can see an option to switch Page Templates from the default template – i.e. `page.php` – to the custom file – i.e. "MyTemplate". Since all we have done so far is copy `page.php` with no alterations this won't actually make your new Page look any different. However you can now make edits to `mytemplate.php` to alter the look or layout of all pages using this template, and your other pages will remain the same.

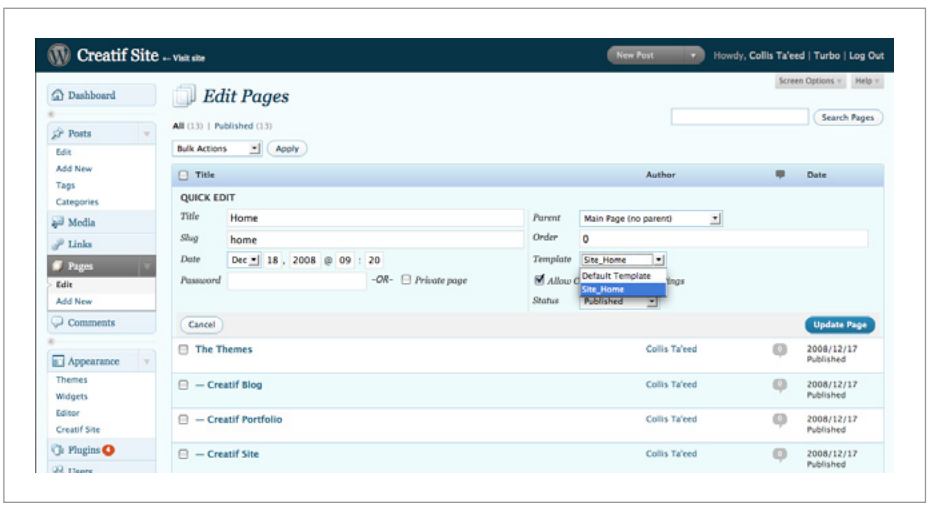


Fig 6-5 – Page Templates can be selected either when editing or quick-editing.

Example: Creating a Special Archives Page

In the last chapter we created a generic `archives.php` to display if a visitor wishes to see all Posts with a certain tag, category, author or date. Now we'll create a special Page on our site to display lists of these archives all on the one page. We'll call it the *Complete Archives*.

Start off a new file `completearchives.php` with this code:

```
<?php
/*
Template Name: Complete Archives
*/
?>
```

Below this we'll add this modification of `page.php`:

```
<?php
/*
Template Name: Complete Archives
*/
?>

<?php get_header(); ?>
<div id="block_content">
    <div id="content_area" class="block">
            <div class="block_inside">
                    <?php if(have_posts()) : while(have_
posts()) : the_post(); ?>
                    <h2><a href="<?php the_permalink(); ?>"
title="<?php the_title(); ?>"><?php the_title(); ?></a></h2>
                   <div class="separator"></div>
                           <?php the_content(); ?>
                    <?php endwhile ?><?php endif; ?>
```

```
                    <h3>Date Based Archives</h3>
                    <ul><?php wp_get_archives('type=monthly');
?> </ul>
                    <div class="separator"></div>
                    <h3>Category Based Archives</h3>
                    <ul><?php wp_list_cats(); ?></ul>
                    <div class="separator"></div>
                    <h3>Pages</h3>
                    <ul>
                        <?php wp_list_pages('title_li='); ?>
                    </ul>
                    <div class="separator"></div>
                    <h3>Tags</h3>
                    <?php wp_tag_cloud(''); ?>
                    <div class="separator"></div>
                    <h3>Still can't find what you're after?</h3>
                    <p>You can search the site:</p>
                    <?php include(TEMPLATEPATH.'/searchform.
php'); ?>
                </div>
            </div>
            <?php get_sidebar(); ?>

            <!-- a Clearing DIV to clear the DIV's because
overflow:auto doesn't work here -->
            <div style="clear:both"></div>
        </div>
    </div>
</div>
<?php get_footer(); ?>
```

As you can see we've hardcoded functions on to the page to display dates, categories and Pages as well as a tag cloud of all tags used on the site. To use this Page Template, add it to your WordPress theme directory. Then inside WP-Admin, select *Pages* > *Add New* then add a title, e.g. *"Complete Archives"*, and if you

wish some text, e.g. *"Browse our archives below"*. Then in the right sidebar set the Page Template to *Complete Archives* and hit *Publish*.

You will see the content of the Post – the title and text – now appear in an altered layout, one that shows all our archives by date, category and tags. If you switch the Page Template back to the default, the same title and text will appear on a regular Page.

We'll use Page Templates in Chapter 8 to create a custom homepage for our Creatif Site theme.

Repurposing WordPress Functionality

Although WordPress is a blogging platform, there is no reason to only use WordPress to make blogs. To take our theming to the next level, you need to break out of the idea that WordPress is for blogs only. WordPress Posts don't need to be blog posts. WordPress Post Categories don't need to be blog categories. All the functionality in WordPress can be repurposed into a variety of other uses.

In the next Chapter we'll use Posts to create portfolio items, Categories will become different parts of the portfolio, and we'll use Custom Fields to add extra information to our portfolio items to customize them appropriately.

In Chapter 8, we'll forget about Posts altogether and focus on Pages and sub-Pages to make a flexible managed site. And finally in Chapter 9, we'll look at how to repurpose WordPress to turn it into everything from a membership directory to an e-commerce store. The key is to break out of blog thinking and look at WordPress as a toolkit of ways to display and manage content.

7

Building an Advanced Theme: Creatif Portfolio

So far we've used WordPress to build a regular blog. While blogs account for the vast majority of WordPress use, with a little ingenuity, it's possible to take WordPress' flexibility and create all kinds of sites. One of the fundamental concepts to learn to do this is that a Post does not need to be a blog post, it is simply a content holder.

When you stop seeing Posts as blog posts, a whole new set of possibilities opens up. To build a portfolio we will reuse WordPress Posts as portfolio entries. So the Post title will be the portfolio item title, the Post text will be the portfolio item description, the Post categories will be the portfolio item categories, and so on.

Most important to extending and repurposing Posts are the Custom Fields options that we looked at in the last chapter. Here we'll use Custom Fields to house all our preview and additional images.

Making a Plan

Before we go diving into the HTML and PHP it makes sense to figure out a rough plan of attack. So in this chapter we are making a portfolio site, it will have categories of portfolio items and each item will have a few different images and thumbnails. In addition to our portfolio items we'll also have regular blog posts and any additional pages like an About page.

A Portfolio and a Blog

Our portfolio items will be made by repurposing regular Posts and adding Custom Fields to get our portfolio images. Since we have regular blog posts as well we are going to need two sets of categorization for our Posts. We'll create a Portfolio category and a Blog category, then anything that is in the Portfolio category or its subcategories will be templated as a portfolio item, and similarly anything in the Blog category or its subcategories will be templated as a blog post, just like we did in the previous theme!

Along with our homepage, we're going to need a listing page for all blog posts and another listing page for all portfolio items. These two (*Portfolio* and *Blog*) will go in to the menu as our main links after the home button.

One way to do this is to make a special Page Template for each and then create a Page in WordPress and assign that template. Then in each Page Template we'd create custom queries to grab Posts. This method does work but requires a few unintuitive code workarounds so we are going to use it. You can however learn about the technique at http://www.nathanrice.net/blog/creating-a-blog-page-with-paging/

Using a Clever Archives

Since the portfolio listing page is actually just the Portfolio category and the blog post listing page is actually just the Blog category, we can use a clever version of either `category.php` or `archive.php` with an if/else switcher to template the two listing pages differently. This way we don't need to worry about extra Page Templates or special `WP_Query` loops.

Hardcoding Some Category and Page Links

Because we are making both portfolio listing and blog posts out of our WordPress Posts, we're going to have to do some hardcoding. That is to say that in certain places we'll have to work out what category ID our blog category has and use that ID in the code, and again for the portfolio category.

This is generally not a great idea because different WordPress installations will have different ID numbers depending on what order the user has created those categories. So that means when we package up the theme we'll need to provide some help instructions for setting up the blog.

There are no fool-proof ways around this, even with some advanced PHP we'll still be relying on the user setting up two specific categories (*Blog* and *Portfolio*). However later in the chapter you'll see how we can at least limit the possibility of mistakes by using defined constants.

The Plan of Attack

So here's how we're going to approach the build:

1. First we'll set up our WordPress installation with categories and some sample Portfolio and Blog posts. Finally, we'll duplicate the blog theme and install it.

2. Then we'll edit our single.php file to style up the new portfolio items as well as blog posts depending on what category a Post is in.

3. Then we'll edit our homepage to match the Creatif Portfolio style.

4. Then we'll use the archive.php to cleverly make a central Portfolio and Blog listing

5. And finally we'll tidy up any remaining pages to get it all finished!

Setting up WordPress

Because we are no longer building sites the way WordPress expects us to by default, it is first necessary to set up WordPress with the relevant content and settings, then when we do our theming we'll be able to see the fruits of our labour.

So grab a fresh copy of WordPress, or wipe the Posts and categories out of a previous one, and follow these steps:

Create the Category Tree

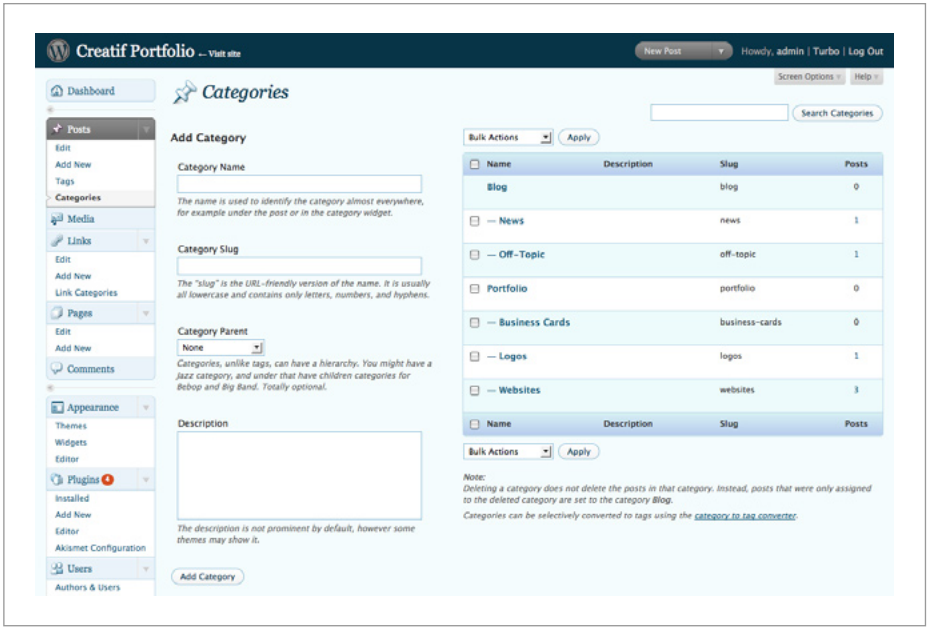


Fig 7-1 – An example category tree.

1. Create two top level categories: *Portfolio & Blog*

2. Create a set of portfolio categories and make them all children of the Portfolio category by setting *Category Parent* to Portfolio. These categories might be for example: *Websites*, *Logos*, and *Business Cards*.

3. Create a set of blog categories and make them all children of the Blog category. These categories might be for example: *News* and *Off-Topic*.

4. Finally note down the category ID numbers of the Blog and Portfolio categories. You can find the ID of any category by clicking *Edit* and looking at the URL you

are taken to. In the URL string at the very end you will see `cat_ID=1` for example.

Note: *In our example and this chapter we will assume that the Blog category has a category ID of 1 and Portfolio category has a category ID of 2.*

Add Sample Posts

1. Create a few sample blog posts. You can do this just as you would normally by clicking *Posts* > *Add New*, writing a Post and then assigning them to the relevant Blog sub-category.

2. Create a set of sample portfolio items. For each item follow these steps:

    a. Create a new Post and assign it to a Portfolio sub-category

    b. Give the portfolio item a title and description text

    c. In the *Excerpt* section give the item a 2 line summary. We'll be using this for listings.

    d. Create a set of images for the item in Photoshop. You will need:

        i. One 325px x 250px image for the homepage when the item is featured.

        ii. A set of matching thumbnail and large images for the main page. Thumbnails should be sized at 100px x 100px, and the main images should be 600px wide and any height.

e. Upload the images using WordPress' regular image uploader and note the URLs down

f. Add the image URLs in your Custom Fields section as follows:

   i. The homepage feature image should be stored as `large_preview` (just as in the blog theme)

   ii. Thumbnails should be stored as `thumbnail_1`, `thumbnail_2`, etc.

   iii. Main images should be stored as `image_1`, `image_2`, etc.

Note that thumbnails and main images should of course match up in terms of numbering.

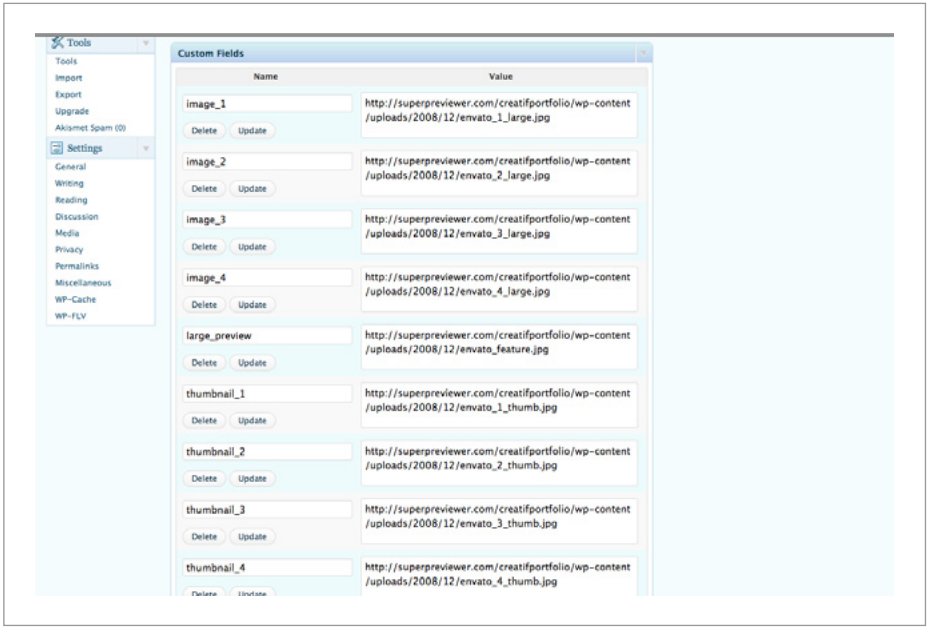


Fig 7-2 – A post with Custom Fields housing images.

Install The New Theme

Now duplicate a copy of the Creatif Blog theme from Chapter 5 and rename the new folder as *Creatif_Portfolio*. Next edit the comment at the top of `style.css` to have the new theme name as well.

Finally install the Creatif Portfolio theme files and activate the theme by going to *Appearance* > *Themes* and locating the theme and selecting it.

You should now see the site looking pretty much the same as it did in the last chapter. None of the clever images will be appearing. So now we're ready to start upgrading our theme!

Defining Constants

Before we do anything else we'll first define two constants to hold the blog and portfolio category IDs. A constant is like a variable, except it doesn't change, also in PHP constants don't need a $ sign in the name. They do however have to be defined. So open up `functions.php` and add these lines at the bottom:

```
<?php
// Set the theme categories
define('BLOG', 1);
define('PORTFOLIO', 2);
?>
```

Once this is done we'll be able to write `BLOG` or `PORTFOLIO` anywhere and the two numbers will be substituted in, in their stead. Note that the define function is by default case sensitive so you have to keep the words in all capitals, this is a good idea anyway as it helps ensure that your code is readable.

This is really important because we are going to need to reference those two numbers in about a dozen spots in the theme. So if you were to try to instruct a new user to update all the spots individually you are almost certain to have mistakes arise. Having the information in only one place means there's only spot where the user can make a mistake when setting up the theme.

Showing Both Portfolio and Blog Items

So the first page we're going to edit is `single.php`. We want to upgrade our single Post page to display either blog posts just like it always has, or our new portfolio items! How do we decide? Well we're going to need to figure out what category of item is being displayed and then we'll use an if/else statement to display the two different layouts.

The code we need to insert at the top of the page (just after the header include) is:

```
<?php
$category = get_the_category();
$parent = $category[0]->category_parent;
?>
```

To grab the category we are using the Template Tag `get_the_category()` which you can read about at: http://codex.wordpress.org/Template_Tags/get_the_category.

Because a Post can be a member of several Categories, `get_the_category()` actually returns an array of results. So the variable we are assigning the result to – `$category` – is an array. As you know from basic PHP you can access an array's contents using the square bracket notation with the first item being at `$category[0]`, the second item at `$category[1]`, and so on.

In our case we are only interested in the first category, so all we want is `$category[0]`. Now each element in the array is actually an object with a set of data representing the category. So we can actually access a variety of different data using the `->` notation:

- `cat_ID`
- `cat_name`
- `category_nicename`
- `category_description`
- `category_parent`
- `category_count`

In our case we are only interested in getting the parent category because we want to know if it is either the Blog category or the Portfolio category. So we store this value as `$parent`. What is actually being stored is the ID number of the parent category.

So next we can place a big if/else statement to decide what Post we're showing, like this:

```
<?php get_header(); ?>

<?php
$category = get_the_category();
$parent = $category[0]->category_parent;
?>
<?php if (in_category(BLOG) || $parent == BLOG): ?>
...  Code to Display Blog Posts ...
<?php else: ?>
...  Code to Display Portfolio item ...
<?php endif; ?>
<?php get_footer(); ?>
```

As you can see what we are saying here is if the Post is in the category with ID equivalent to `BLOG`, or if its parent category has ID equivalent to `BLOG`, then the Post must be a blog post.

Remember that `BLOG` is the constant we set to be the category ID of the Blog category.

Otherwise it must be a portfolio item. Note the use of `in_category` which is a *Conditional Tag* like those we covered at in Chapter 4. Also note that `||` means *or*, so that part of the `if` is executed if *either* of the two statements resolves to `true`. Another logical operator we've been using is `&&` which means and, so if we used that instead the if statement would only execute that block if *both* statements resolved to `true`.

So in the first section – the part for blog posts – we can simply use our regular code from the old Creatif Blog `single.php`. It's the other section that is going to need some new code to grab all those images and thumbnails.

Displaying Portfolio Items

Now a portfolio item is the same as a regular blog post, so we can again duplicate the same blog post code and paste it into the else section. Then make the following changes:

Delete Comments

We no longer need to have a section for comments, so we can delete the code that goes:

```
<div id="comments_template">
    <?php comments_template(); ?>
</div>
```

Note that you could of course keep comments on portfolio items if you wished!

Remove the Post Time and Author

It doesn't seem very relevant to say what time the item was added, or who the author was. So we'll change this line:

```
<small>on <?php the_time('M d'); ?> in <?php the_
category(', '); ?> tagged <?php the_tags(''); ?> by <?php
the_author_posts_link(); ?></small>
```

to:

```
<small> in <?php the_category(', '); ?> tagged <?php the_
tags(''); ?></small>
```

Add the Images to the Page

Before the heading and Post text we'll paste in all our images and thumbnails. After that we'll add in the Javascript to make them switch. So first let's add the main images:

```
<div class="portfolio_main">
... Add images here ...
</div>
```

First, as you recall the code to grab the value of the Custom Field titled `image_1` is:

```
get_post_meta($post->ID, 'image_1', true);
```

In our case however we (may) have a bunch of images to show. We're not really sure how many, but in our example we're going to assume there is less than 10. You could actually work out how many Custom Fields have been entered, but the PHP starts looking pretty confusing, so it's a lot simpler to just make an assumption that no portfolio item is every going to have bazillions of images!

So therefore we can make a loop to flick to go through and output `image_1` up to `image_10`:

```
<div class="portfolio_main">
<?php
    for ($i=1; $i<=10; $i++) {
            $image = get_post_meta($post->ID, 'image_'.$i, true);
            echo "<img src='".$image."' id='image_".$i."'>";
    }
?>
</div>
```

Here we're creating a variable called `$i` and looping through 10 times, each time substituting `$i` for the number in our Custom Field key using the . operator to stick the two bits together: `'image_' . $i`

Then we echo a regular HTML `<img>` tag. Note that we give each one an id so that we can use Javascript later to show and hide them.

But what if there is no `image_10`, or `image_9`, or any images for that matter? We certainly don't want to be outputting a whole bunch of broken images. So we'll update our code like this:

```
if ($image) {
    echo "<img src='".$image."' id='image_".$i."'>";
}
```

If you're new to PHP you might be thinking that our if statement isn't really saying anything right? Normally you would be comparing something using an == or a > or so on. But actually if statements are just checking to see if the thing in the brackets boils down to `true` or `false`. So if you have an `if(a==b)` statement, PHP is actually saying does `a==b = true`? Now in our case we saying `if($image)`, so PHP is evaluating whether `$image = true`. If

`$image` is an empty variable then it will evaluate to `false`, and if it has anything else in it, it will evaluate to `true`. So what we're really saying here is if `$image` exists!

Now there are two more important things we need to do to get ready for our Javascript to show/hide these images. First if it's not the first `<img>` we need to add a `style="display:none"` to the tag. This will ensure only the very first image is actually showing to begin with. Then we can let the user control what is shown and hidden.

The second thing we need to do is figure out exactly how many images are going to be present all together, then we can give our Javascript this value later. So our final code will look like this:

```
<div class="portfolio_main">
<?php
    $number_of_images = 0;
    for ($i=1; $i<=10; $i++) {
        $image = get_post_meta($post->ID, 'image_'.$i,
true);
        if ($image && $i == 1) {
            echo "<img src='".$image."' id='image_
".$i."'>";
            $number_of_image++;
        } elseif ($image) {
            echo "<img src='".$image."' id='image_".$i."'
style='display:none;'>";
            $number_of_images++;
        }
    }
?>
</div>
```

We are first creating a variable called `$number_of_images` and making it 0. Then we'll increment this number each time we find

ourselves an image. Next we update our if/else statement to first check if both `$image` exists and `$i` is equal to 1. If that's not true, we'll also check `elseif` just `$image` exists. Note that if we used an `else` instead of an `elseif` we would be printing the `<img>` tag whether or not `$image` existed, because except for the first cycle of the loop we'd always wind up in the `else`.

Add the Thumbnails to the Page

We can now add the thumbnail images and include on them links to show and hide the main images, like this:

```
<div class="portfolio_thumbs">
<?php
    for ($i=1; $i<=$number_of_images; $i++) {
        $image = get_post_meta($post->ID, 'thumbnail_'.$i, true);
        if ($image) {
            echo "<a href='javascript:image_
switch(".$i.",".$number_of_images.");'><img
src='".$image."' id='thumbnail_".$i."'></a>";
        }
    }
?>
</div>
```

As you can see this is a similar loop to the one we used for the main images. The only real difference is that this time we can just use `$number_of_images` as the end point of the loop, since we now have a value for it.

For each thumbnail image we are outputting the image with an attached link that activates some Javascript to show/hide the related main image. The Javascript itself can be placed into a js file called `image_switch.js`, here's the simple code we need:

```
function image_switch(active, number) {
    for (var i=1; i < number+1; i++) {
            document.getElementById('image_'+i).style.
display = 'none';
    }
    document.getElementById('image_'+active).style.display
= 'block';
}
```

And this file should be inserted into the `header.php` by adding this line:

```
<script type="text/javascript" src="<?php
bloginfo('template_directory'); ?>/scripts/image_switch.
js"></script>
```

So the Javascript is pretty simple, it just loops through all the images and first sets them all to `style="display:none"`, then finds the active one and sets it to `style="display:block"`. The two variables we need to pass it are `active` which is the image number that should be shown, and `number` which is the number of elements to loop through. Back in our PHP these are `$i` and `$number_of_images` respectively.

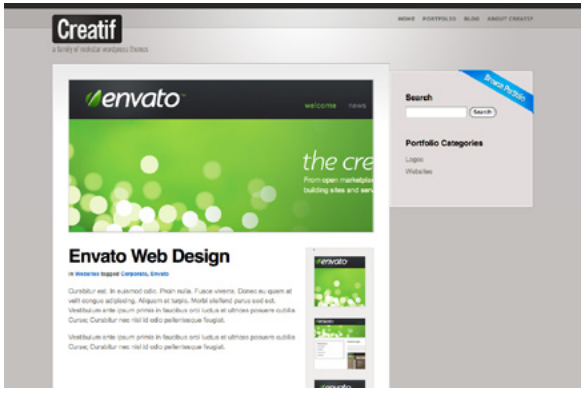


Fig 7-3 – An example portfolio item.

Making Multiple Sidebars

If you view your WordPress installation now in a browser and open up either a portfolio item or a blog post you will find there is one major problem we have yet to deal with – the sidebar is the same in both cases and the blog and portfolio categories are all mixed in together. It would make a lot more sense to have different sidebars for the blog and for the portfolio. This is easy to do, just duplicate the `sidebar.php` file and call the duplicate `sidebar-blog.php`.

We can then make use of a neat extra parameter of the `get_sidebar()` function to write `get_sidebar('blog')` and it will automatically include `sidebar-blog.php`. So in `single.php` in the first if/else block – the one for the blog, just substitute in the new command, and leave the portfolio sidebar as is. We'll make the default `sidebar.php` the portfolio sidebar since that is the main purpose of the site.

Blog Sidebar

Substitute the contents of `sidebar-blog.php` with this code:

```
<div id="sidebar">
    <img src="<?php bloginfo('template_directory'); ?>/
images/ribbon_browse.png" class="ribbon" alt="Browse
Blog"/>
    <div class="block_inside">
        <ul>
            <li id="search" class="widget"><h3>Search</h3>
                <?php include(TEMPLATEPATH.'/searchform.
php'); ?>
            </li>

            <li id="subscribe" class="widget"><h3>Subscribe
</h3>
```

```
                <ul>
                    <li><a href="<?php bloginfo('rss2_
url'); ?>">RSS Feed</a></li>
                    <li><a href="http://feedburner.google.
com/fb/a/mailverify?uri=psdtuts">Email Updates</a></li>
                    <li><a href="http://feeds.feedburner.
com/psdtuts"><img style="border:0" src="http://feeds.
feedburner.com/~fc/psdtuts?bg=a6a2a0&amp;fg=ffffff&amp;anim
=0" alt="" width="88" height="26" /></a></li>
                </ul>
            </li>
            <li id="categories" class="widget" ><h3>Blog
Categories</h3>
                <ul>
                    <?php wp_list_categories('title_
li=&orderby=name&child_of='.BLOG); ?>
                </ul>
            </li>
        </ul>
    </div>
 </div>
```

The sidebar is similar to the one used in the Creatif Blog theme, though we've removed widgetization. The one real difference here is that the categories section is now just Blog Categories. To display only those categories that are a parent of the Blog Posts category we've added the `child_of` parameter and used the constant `BLOG` we defined in `functions.php`:

```
wp_list_categories('title_li=&orderby=name&child_of='.BLOG);
```

Portfolio Sidebar

The portfolio sidebar can be virtually identical, only we need to substitute the `child_of` value for the category ID of Portfolio,

which of course is stored in the PORTFOLIO constant. We will also get rid of the subscription part since that seems a little unrelated. So we're left with:

```
<div id="sidebar">
    <img src="<?php bloginfo('template_directory');
?>/images/ribbon_browse_portfolio.png" class="ribbon"
alt="Featured Project"/>
    <div class="block_inside">
        <ul>
            <li id="search" class="widget"><h3>Search</h3>
                <?php include(TEMPLATEPATH.'/searchform.
php'); ?>
            </li>
            <li id="categories" class="widget"
style="padding-top:15px;"><h3>Portfolio Categories</h3>
                <ul>
                    <?php wp_list_categories('title_
li=&orderby=name&child_of='.PORTFOLIO); ?>
                </ul>
            </li>
        </ul>
    </div>
 </div>
```

As you can see the only other changes are to the ribbon image and the words Portfolio Categories.

Home Sidebar

There is one final sidebar we need, this one for the homepage where there is just some text content. So create a new file called `sidebar-home.php` and enter in this code:

```
<div id="text_column">
<h2 id="text_title">Creatif is a WordPress portfolio theme
for designers and creatives</h2><br />
    <p>You can use it to quickly turn WordPress into a
portfolio website.  Not familiar with WordPress? Fear
not, the theme accompanies a book called <a href="http://
rockablepress.com">How to Be a Rockstar Wordpress
Designer</a> by Rockable Press.</p>
    <p>The book teaches you to use WordPress theming to
take advantage of this flexible CMS product to create
dynamic sites.</p>
    <p>And as if that's not enough, you can see a photoshop
to HTML tutorial on designing the theme over at <a
href="http://psdtuts.com">PSDTUTS</a> and <a href="http://
nettuts.com">NETTUTS</a>.</p>
</div>
```

We'll make use of this sidebar in the next section!

Updating the Homepage

So that brings us to our `index.php` file and its includes. On this page we need to update how the Posts are displayed, add the new sidebar and clean up a few left-overs from the blog theme.

The Featured Post

Starting with our featured Post area, open up `featuredpost.php` and make the following changes.

Only Feature Portfolio Items

Currently our theme will feature the *first* item to come along. We need to make sure we're only featuring portfolio items. We can do

this by excluding the Blog category and all its children from our query like this:

```
$featured->query('showposts=1&cat=-'.BLOG);
```

As you can see we've effectively added `cat=-1` which excludes the category with ID 1, though of course we don't actually write the number in, we use that handy constant `BLOG` that we defined previously.

Update for Portfolio Items

Next we'll switch the ribbon image to `ribbon_featured.png` so that it says featured project rather than featured post. Then also we'll remove the date and author from the project to leave:

```
<small>in <?php the_category(', '); ?> tagged <?php the_
tags(''); ?></small>
```

The Index Page

Going back to `index.php` we can quickly amend the sidebar include from plain old `get_sidebar()` to `get_sidebar('home')` to switch over to the `sidebar-home.php` file that we created earlier.

Then we also need to change the area of the page that was previously just displaying our Posts to only display a select number of portfolio items.

So change the `<div id="block_content">` to read `<div id="block_portfolio">` and then delete everything in the `<div id="content_area" class="block"></div>` segment and replace in this code:

```
<div id="portfolio_items">
        <?php query_posts("cat=-".BLOG."&showposts=4"); //
First we remove the Blog Posts category ?>
        <?php $first = true; ?>
        <?php if(have_posts()) : while(have_posts()) :
the_post(); ?>
            <? if ($post->ID != $featured_ID) { ?>
                <div class="mini_portfolio_item">
                    <? if ($first) {?>
                        <img src="<?php bloginfo('template_
directory'); ?>/images/ribbon_recent.png" class="ribbon"
alt="Recent Projects"/>
                    <? }
                       $first = false;
                    ?>
                    <div class="block_inside">
                        <a href="<?php the_permalink();
?>"><img src="<?php echo get_post_meta($post->ID,
'thumbnail_1', true); ?>" class="thumbnail" alt="" /></a>
                        <h3><a href="<?php the_permalink();
?>" title="<?php the_title(); ?>"><?php the_title(); ?>
</a></h3>
                        <?php the_excerpt('<br />View
Project'); ?>
                    </div>
                </div>
            <? } ?>

        <?php endwhile ?>
        <?php else : ?>
            <h2 class="center">Not Found</h2>
            <p class="center">Sorry, but you are looking
for something that isn't here.</p>
            <?php include (TEMPLATEPATH . '/searchform.
php'); ?>
        <?php endif; ?>
</div>
```

So here we've made a few changes to the old code:

1. We've changed the query to exclude blog posts and only display four items. By amending `query_posts()` to `query_posts("cat=-".BLOG."&showposts=4");` we remove all items from category ID `BLOG` and we limit the query to only four items.

2. We're only showing the ribbon image on the first item. By creating a variable called `$first` and setting it to `true`, we can use an if statement to figure out if this is the first portfolio item or not. This is important because otherwise we'd be repeating the ribbon image which would look strange.

3. Next we've substituted in the amended HTML code from our `portfolio.html` file from back in Chapter 3. This code requires a thumbnail image so we grab the item's first thumbnail image using the same old Custom Field code as we've been using on the `single.php` page.

4. Finally because we only want a one or two line description we've used `the_excerpt()` Template Tag instead of `the_content()`. This forces WordPress to grab those short excerpts we added when creating the Posts.

Creating Listing Pages

When a person arrives at our site we need a page where they can find all the portfolio items and another page where they can find all the blog posts. Then we can make our main menu go something like this: `Home | Portfolio | Blog | Other Pages`.

We could create a special Page Template and use `WP_Query` to grab specific types of Posts, but in our case we can actually simplify our theme a lot and just use the `archive.php` file to do everything!

As you recall `archive.php` is the template file used for category archives, author archives, tag archives and date archives. In our case it is the category archives that we are interested in. If we can make an archive that displays the Blog category and the Portfolio categories differently, we can use the respective category lists as our listing pages. So the list of everything in the Blog category is the blog, and similarly with the portfolio.

Figuring Out What Category is Displaying

As we did with our `single.php` file before, the first step is to figure out what category of Posts we're dealing with. The code we need is:

```
<?php
    $catid = get_query_var('cat');
    $cat = &get_category($catid);
    $parent = $cat->category_parent;
?>
```

So here we are grabbing the variable `cat` out of the URL string, this corresponds to the category ID number and from there we can work backwards to get the parent ID just as we did in `single.php`.

Then all we need is another big if/else statement, just as we had in `single.php`:

```
<?php get_header(); ?>
<?php
$catid = get_query_var('cat');
$cat = &get_category($catid);
```

```
$parent = $cat->category_parent;
?>
<?php if (is_category(BLOG) || $parent == BLOG):?>
... Format for blog posts
<?php else: ?>
... Format for portfolio posts
<?php endif;?>
<?php get_footer(); ?>
```

The only difference that we have is that instead of `in_category()` which was necessary for our single Post, we have `is_category()`.

Listing Posts

To format and list the two types of Posts is easy. For blog posts we can use our standard blog post formatting:

```
<div id="block_content">
        <div id="content_area" class="block">
            <div class="block_inside">
                    <?php if(have_posts()) : while(have_
posts()) : the_post(); ?>
                        <h2><a href="<?php the_permalink();
?>" title="<?php the_title(); ?>"><?php the_title(); ?></
a></h2>
                        <small>on <?php the_time('M d'); ?>
in <?php the_category(', '); ?></small>
                        <?php the_excerpt(); ?>
                    <div class="separator"></div>
                    <?php endwhile ?>
                        <div id="posts_navigation">
                            <?php previous_posts_link(); ?>
                            <?php next_posts_link(); ?>
                        </div>
                    <?php else : ?>
```

```
                        <h2 class="center">Not Found</h2>
                        <p class="center">Sorry, but you
are looking for something that isn't here.</p>
                        <?php include (TEMPLATEPATH . '/
searchform.php'); ?>
                    <?php endif; ?>
            </div>
         </div>
        <?php get_sidebar('blog'); ?>
         <!-- a Clearing DIV to clear the DIV's because
overflow:auto doesn't work here -->
         <div style="clear:both"></div>
        </div>
    </div>
</div>
```

And for portfolio items we'll use the same sort of formatting we have on the homepage:

```
<div id="block_portfolio">
        <div id="portfolio_items">
                <?php if(have_posts()) : while(have_
posts()) : the_post(); ?>
                        <div class="mini_portfolio_item">
                            <div class="block_inside">
                                <a href="<?php the_
permalink(); ?>"><img src="<?php echo get_post_meta($post-
>ID, 'thumbnail_1', true); ?>" class="thumbnail" alt=""
/></a>
                                <h3><a href="<?php the_
permalink(); ?>" title="<?php the_title(); ?>"><?php the_
title(); ?></a></h3>
                                <?php the_excerpt('<br
/>View Project'); ?>
                            </div>
                        </div>
```

```
                    <?php endwhile ?>
                        <div id="posts_navigation">
                            <?php previous_posts_link(); ?>
                            <?php next_posts_link(); ?>
                        </div>
                    <?php else : ?>
                        <h2 class="center">Not Found</h2>
                        <p class="center">Sorry, but you are
looking for something that isn't here.</p>
                        <?php include (TEMPLATEPATH . '/
searchform.php'); ?>
                    <?php endif; ?>
            </div>
            <?php get_sidebar(); ?>
             <!-- a Clearing DIV to clear the DIV's because
overflow:auto doesn't work here -->
             <div style="clear:both"></div>
            </div>
        </div>
</div>
```

Adding a Breadcrumb Trail

The only problem left is that it's not clear what page we are actually showing because there's no heading. What we'll do instead is manufacture a little breadcrumb trail that will let the user see what category they are browsing, and if it's a subcategory of Blog or Portfolio to list the parent category too. We insert this code just before the main if/else statement:

```
<div style="padding-bottom:5px;">
<?php if($parent == BLOG): ?>
            <em><a href="<?php bloginfo('url'); ?>">Home</a> >
<a href="<?php bloginfo('url'); ?>/category/blog">Blog</a>
> <?php wp_title(' ', true, ''); ?></em>
```

```
<? elseif($parent == PORTFOLIO): ?>
        <em><a href="<?php bloginfo('url'); ?>">Home</
a> > <a href="<?php bloginfo('url'); ?>/category/
portfolio">Portfolio</a> > <?php wp_title(' ', true, '');
?></em>
<? else: ?>
        <em><a href="<?php bloginfo('url'); ?>">Home</a> >
<?php wp_title(' ', true, ''); ?></em>
<? endif; ?>
</div>
```

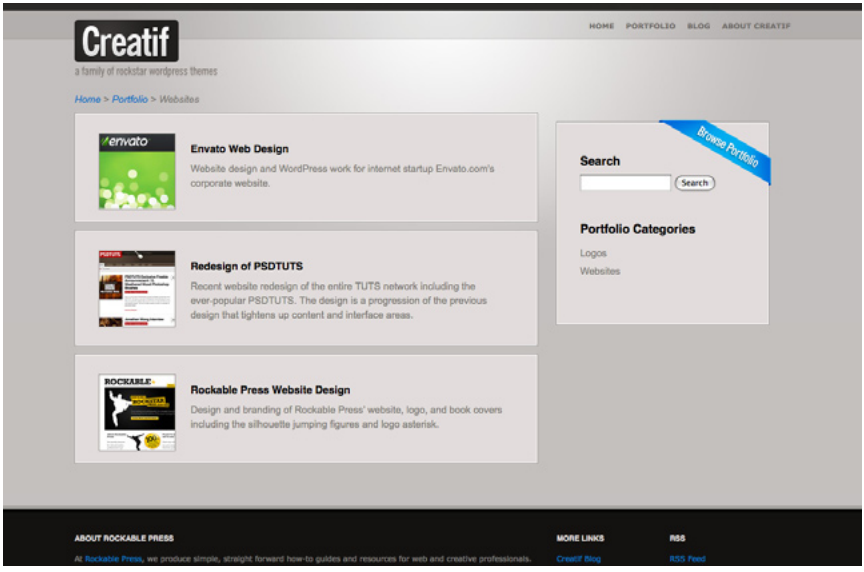


Fig 7-4 – The portfolio listing page with breadcrumb trail.

Here we are first checking if the parent category is `BLOG`. If that's the case we'll output `Home > Blog > Title` – using `wp_title` for the final part. Note that we are using a permalink to the Blog category. If permalinks had not been switched on we'd need to use `<?php bloginfo('url'); ?>/?cat=1` which goes to the same place.

The next part of the if statement checks if the parent category is `PORTFOLIO`, in which case we do something similar: `Home > Portfolio > Title`. Finally if neither is true we just display `Home > Title`.

Adding Links to the Menu

The last thing we need to do for our listing pages is add them manually to the menu. To do this open up `header.php` and edit the menu section to become:

```
<ul id="menu">
    <li><a href="<?php bloginfo('url'); ?>"
title="Home">Home</a></li>
    <li><a href="<?php bloginfo('url'); ?>/category/
portfolio/">Portfolio</a></li>
    <li><a href="<?php bloginfo('url'); ?>/category/blog/
">Blog</a></li>
    <?php wp_list_pages('title_li='); ?>
</ul>
```

As you can see we've added two permalinks in after Home, then followed it up with any Pages the user may have created.

Tying Loose Ends

With the homepage, single Page and archives all sorted, we're pretty much finished with our theme. A few loose ends that need to be tied are:

1. **Clean Up Wording**
   You may wish to tidy up a few pages to swap blog references in the text for more portfolio-centric references, for example instead of "`posts`" you might like to say "`items`" and so on.

2. **Switch RSS feeds**
   The default RSS feed is for the whole site (including portfolio items) however if you only want RSS to cover the blog posts, this is easily done because you can create an RSS feed for any category. So if the blog sits at: http://example.com/category/blog/ then the feed will be at: http://example.com/category/blog/feed.

3. **Remove Unused Theme Files**
   There are a few files from Creatif Blog that we don't need anymore, specifically: `author.php`, `category.php` and `completearchives.php`.

4. **Update the Screenshot!**

Wrap Up of Creatif Portfolio

In this chapter we have gone through a concrete example of repurposing WordPress functionality to produce a different type of site. And importantly we did it while keeping a blog present. In Chapter 9 you'll find lots of other ideas for repurposing WordPress that all rely on the same basic principles – rethink the Post and use Custom Fields for extra information.

In this chapter we used category templates to make our listing pages. As mentioned previously another method to do the same thing is to use Page Templates – though this requires some extra code hacking. There is in fact yet a third method using one of WordPress' Settings to change what template is used for the homepage. We'll cover this method in the next chapter.

We'll also look at another way of extending WordPress using not Posts, but Pages and sub-Pages. This format suits many client and business applications where content is mostly static but still needs to be content managed.

8

Building a Site Theme: Creatif Site

Up until now we have focused heavily on WordPress' Post functionality. In this chapter we'll build a theme that puts the emphasis on Pages and sub-Pages to build a more traditional content managed site, similar to what many business clients ask for. In this context we can still make use of Posts to add time-dependent content such as news, press releases or of course a blog, but the focus will be on the Page structure. In this way, with a bit of training WordPress can be used as a content management system for a huge array of client projects!

Making a Plan

Again before we get started with the build, we'll plan out how the site should be constructed. In this case we want to have:

1. A custom homepage with a featured section and three subsections.

2. A set of top level Pages, each of which can have unlimited sub-Pages.

3. On each of the top level Pages we'll have a special graphic header while the sub-Pages will simply have a regular text heading.

4. A "news" section with the latest three news items appearing on the homepage.

Custom Homepage

Unlike the blog and portfolio themes we covered previously the homepage of our Creatif Site theme can't just rely on the latest Post content. Instead we need to have several images and bits unrelated to either Posts or Pages, that the client can keep updated. To achieve this result we will make use of WordPress Theme Options.

We'll also have a small section on the page to show the latest three news items, this can be piped in easily using `WP_Query`.

Pages and Sub-Pages

WordPress makes it easy to create a hierarchy of Pages by assigning a Page parent to new Pages. We can then put together some code that figures out what submenu to show in the sidebar based on the Page currently showing.

Because Pages can make use of Custom Fields just like Posts, we can easily add images for our top level Pages just as we did in our Creatif Portfolio theme.

The News Section

In Chapter 7 we used a category archive template to create a blog listing page. This technique could just as easily be applied again here, but instead we'll demonstrate another method for achieving the same result. This time we will use WordPress' Settings to switch what page is used for blog listing and what page is used for the homepage, enabling us to use our regular `index.php` file for the news page.

Setting up WordPress

Again the first step is to setup our WordPress installation with the right Pages, Posts and Theme Options. Then we can theme around our content to get it all working as we want.

So grab a fresh copy of WordPress, or wipe the posts and categories out of a previous one, and follow these steps:

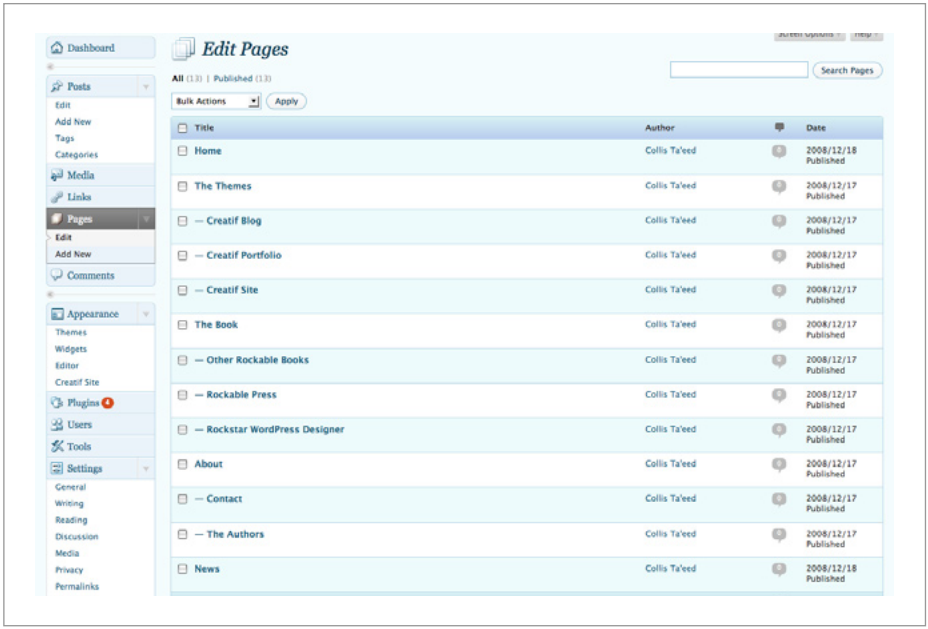


Fig 8-1 – An example Page listing.

Create a Set of Pages

1. First create a set of top level Pages, in our example we have *The Themes*, *The Book* & *About*.

2. Next for each top level Page, add sub-Pages by creating a new Page and setting its parent to the relevant top level Page.

3. Then for each top level Page add a background header image and a text blurb as Custom Fields with the keys `header_image` and `header_text`.

4. Next create a Page called *News*. We will use this Page for the Post listings of our news section.

5. Next create a Page called *Home*. Later we will give this Page a special Page Template to turn it into the custom homepage.

6. You may also wish to go through and order the top level Pages so that they will appear in the correct order in the menu. To do this edit each Page and give it an *Order* value starting at 1 for the first item (Home) and going up from there.

Setup the Homepage and Listing Page

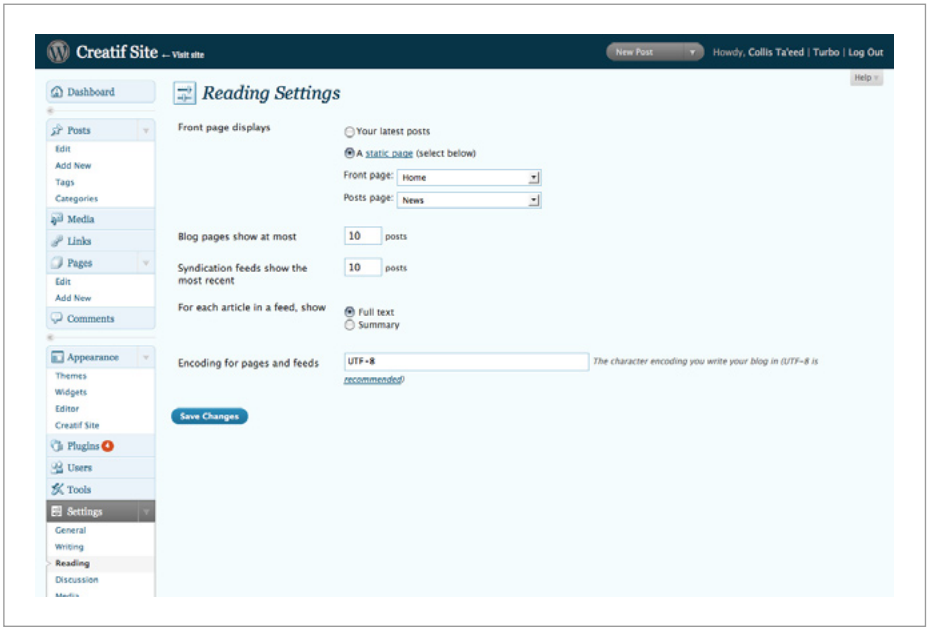


Fig 8-2 – The Settings > Reading screen.

With all our Pages created we can now setup the special homepage and listing page setup we will be using. To do this go to *Settings* > *Reading* in WordPress and in the section that says *Front page displays* change the value from *Your latest posts* to *A static page*.

Then set the *Front page* to Home and *Posts page* to News. These are the two pages we created earlier for this very purpose. When we begin our theming we will create a special Page Template for Home and because we changed this setting the News page will automatically use `index.php`.

Create Some News Categories and Posts

1. Next create some Categories, in our example we have *WordPress News* and *Updates*

2. Then create a few sample Posts and assign them to each category.

3. For each Post, add just a title, text and short excerpt to display on the homepage.

Install The New Theme

Before we can add our Theme Options we need to actually install the theme. So next duplicate a copy of the Creatif Blog theme from Chapter 5 and rename the new folder as *Creatif_Site*. Next edit the comment at the top of `style.css` to have the new theme name as well.

Install the Creatif Site theme files and activate the theme by going to *Appearance* > *Themes* and locating the theme and selecting it.

Add Theme Options using Theme Toolkit

In Chapter 6 we looked at using the third party Theme Toolkit class available from http://tinyurl.com/wp-theme-toolkit to add Theme Options. We'll now make use of this technique in a concrete example.

1. Download the `functions.php` and `themetoolkit.php` files from the link above and add them to the Creatif_Site theme (overwriting the old `functions.php` file)

2. All the settings you need to edit are inside `functions.php` so open this file up and replace the example settings with this code:

```
'main_title' => 'Main Title ## The title of the main area
on the homepage.',
'main_image' => 'Main Image ## URL of the image on the
homepage, should be 400px wide x 250px high.',
'main_text' => 'Main Text ## The main text to go on the
homepage.',

'feature_1_title' => 'Featured Area 1 - Title ## The title
of this featured area on the homepage',
'feature_1_image' => 'Featured Area 1 - Image ## URL of the
image for this featured area on the homepage, should be
100px wide x 100px high.',
'feature_1_text' => 'Featured Area 1 - Text ## The text for
this featured area on the homepage.',

'feature_2_title' => 'Featured Area 2 - Title ## The title
of this featured area on the homepage',
'feature_2_image' => 'Featured Area 2 - Image ## URL of the
image for this featured area on the homepage, should be
100px wide x 100px high.',
'feature_2_text' => 'Featured Area 2 - Text ## The text for
this featured area on the homepage.',

'feature_3_title' => 'Featured Area 3 - Title ## The title
of this featured area on the homepage',
'feature_3_image' => 'Featured Area 3 - Image ## URL of the
image for this featured area on the homepage, should be
100px wide x 100px high.',
```

```
    'feature_3_text' => 'Featured Area 3 - Text ## The text for
    this featured area on the homepage.'
```

What we are doing here is creating a set of 12 fields that the user will be able to edit to customize the homepage. We have an image / title / text combo for the main area and three feature spots.

Add Values to the Theme Options

With our `functions.php` file saved, you can now open up WordPress and with the theme loaded you will have a new option now available in the *Appearance* submenu, that reads *Creatif Site*. If you click this option you will be shown a page listing all the Theme Options you created in the previous step.

Note: *Since WordPress 2.7 Theme Toolkit has lost some of its formatting and looks a bit chaotic. You can make this page a little more user friendly by editing the* `themetoolkit.php` *file. As a start, find line 255 and add* `style='width:600px'` *to the* `<input type='text'` *HTML. This will widen the input fields. You can apply other HTML formatting, but be careful about what you edit and always keep a backup.*

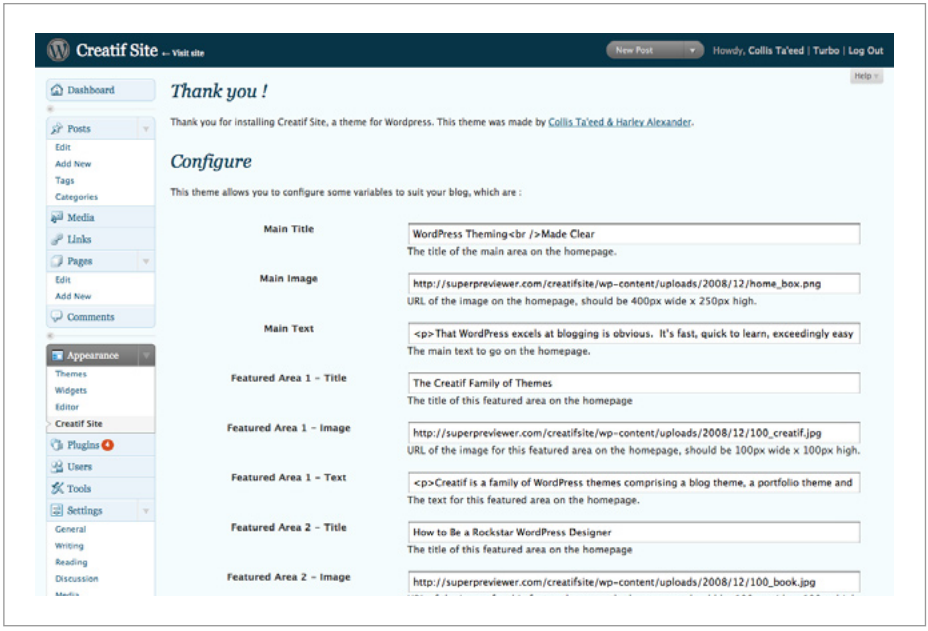


Fig 8-3 – The ThemeToolkit Options page.

You can now go through and add sample data to the different fields. The sample data we've used in our example theme can be found by checking the homepage of the demo for this book at http://superpreviewer.com/creatifsite.

Setting up the Menu

With WordPress suitably setup with plenty of content we can now start molding our theme into shape. We are again starting with Chapter 5's Creatif Blog theme and working from there.

So the first step is to open up `header.php` and modify the menu code to:

```
<ul id="menu">
    <?php wp_list_pages('title_li=&depth=1'); ?>
</ul>
```

There are two changes here, firstly we've gotten rid of the link to the homepage. This is no longer needed since we have literally created a Page called Home. The second change is the addition of `depth=1`. This parameter limits `wp_list_pages()` to only show top level pages. By default the Template Tag will produce a nested, unordered list including submenus.

Create a Home Template

Next we need to setup a Page Template for the homepage. To do this, simply duplicate `index.php` and call the new file `site_home.php`. Then at the top of the file add the code below to turn it into a Page Template named `Site_Home`.

```
<?php
/*
Template Name: Site_Home
*/
?>
```

Note: *It's important you don't name the file* `home.php` *as that is a special template file name, similar to* `index.php`, `sidebar.php` *and so on. The file* `home.php` *would normally display on the homepage, but because we have switched the homepage it will instead appear in our news area confusing things.*

You can now go back into the WordPress WP-Admin and edit the Page called Home to set its Page Template to `Site_Home`. Once that is done clicking on Home will be showing us the contents of `site_home.php`, and clicking on News will be showing us the contents of `index.php`. Of course at the moment those two pages are identical, but we'll change that shortly!

Showing Submenus and Page Titles

Next we'll turn our attention to the Pages themselves. Here we need to display the relevant submenu, and amend the Page's title to either use the image we attached with our Custom Field or a simple text title.

Sidebar Submenus

Once a user clicks through to either a Page or sub-Page, we are going to want the sidebar to display the top level menu item and submenu for that section. To do so we need to first work out what the top level or parent item is. If we are on the top level this is easy, it's simple the current Page. And if we are on a sub-Page then we can use `post_parent` to find the top level Page ID:

```
<?php
    if ($post->post_parent != 0) {
        $parent = $post->post_parent;
    } else {
        $parent = $post->ID;
    }
    $parent_title = get_the_title($parent);
    $parent_link = get_permalink($parent);
?>
```

In this code segment we first check if the Page has a parent. If one exists then the field `$post->post_parent` would be something other than `0`, in which case we record this value in the variable `$parent`. Else, we can assume that the Page currently displaying is the parent and we'll just grab the current Page's ID. Finally we use the Template Tags `get_the_title` and `get_permalink` to work backwards from the ID to grab the title and link for the parent Page.

We can now use our favorite Page listing Template Tag, making use of the `child_of` parameter to do the rest of the work:

```
<li id="subpages" class="widget">
    <h3>
        <?php echo ('<a href="'.$parent_link.'">' .
$parent_title . '</a>');  ?>
    </h3>
    <ul>
        <?php wp_list_pages('title_li=&child_of='.$parent); ?>
    </ul>
</li>
```

The only other thing we need in `sidebar.php` is a search box, so removing everything else our final `sidebar.php` code looks like this:

```
 <div id="sidebar">
    <img src="<?php bloginfo('template_directory'); ?>/
images/ribbon_navigation.png" class="ribbon" alt="Featured
Project"/>
    <div class="block_inside">
        <ul>
            <?php

                if ($post->post_parent != 0) {
                    $parent = $post->post_parent;
                } else {
                    $parent = $post->ID;
                }
                $parent_title = get_the_title($post->post_
parent);
            ?>
            <li id="subpages" class="widget">
                <h3>
                    <?php echo ('<a href="'.get_
```

```
permalink($post->post_parent).'">' . $parent_title . '</
a>');  ?>
                </h3>
                <ul>
                    <?php wp_list_pages('title_li=&child_
of='.$parent); ?>
                </ul>
            </li>

            <li id="search" class="widget"><h3>Search</h3>
                <?php include(TEMPLATEPATH.'/searchform.
php'); ?>
            </li>
        </ul>
    </div>
 </div>
```

Page Titles

Next open up `page.php` and find the line that outputs our heading text:

```
<h2><a href="<?php the_permalink(); ?>" title="<?php the_
title(); ?>"><?php the_title(); ?></a></h2>
```

We are going to run a combination of if/else statements to change the heading in a selection of different ways:

1. If the Page is a sub-Page, then we will output the top level menu item just above the heading.

2. If the Page is a top level Page, then we will see if there is a header image and text set in our Custom Fields. If they are there we'll use those as the heading.

3. If there is no header image or text, then we'll revert back to the basic heading text.

So to achieve this we need two if/else statements nested inside each other. Here's the first of them:

```
<?php if ($post->post_parent != 0) {

        $parent = $post->post_parent;
        $parent_title = get_the_title($parent);
        $parent_link = get_permalink($parent);

        echo ('<h4><a href="'. $parent_link .'">' .
$parent_title . '</a></h4>'); ?>
        <h2><a href="<?php the_permalink(); ?>"
title="<?php the_title(); ?>"><?php the_title(); ?></a></
h2>

<? } else { ?>

  ... Code for top level menu item

<? } ?>
```

As we did earlier we are checking if a parent exists, if it does then we'll grab the details and output them wrapped in an `<h4>` tag. The `else` part is as follows:

```
 <? } else {
    $header_image =  get_post_meta($post->ID, 'header_
image', true);
    $header_text =  get_post_meta($post->ID, 'header_text',
true);
    if ($header_image && $header_text) { ?>
        <div style="background-image:url(<? echo $header_
image ?>);" class="header_image">
```

```
                <? echo $header_text ?>
            </div>
        <?php } else { ?>

            <h2><a href="<?php the_permalink(); ?>"
    title="<?php the_title(); ?>"><?php the_title(); ?></a></h2>
        <?php } ?>
    <? } ?>
```

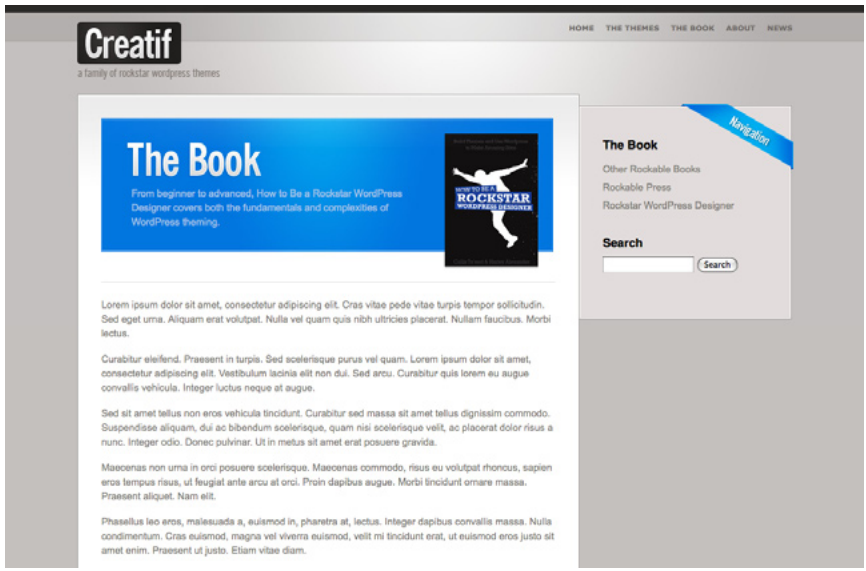


Fig 8-4 – Custom header image and sidebar menu in action.

This time we grab the two Custom Fields we set up earlier – `header_image` and `header_text` and assign them to variables. Then we check that both are present, and if so we display a `<div>` that uses the image as a background and places the text on top. We've also added a new CSS class `header_image` to our `style.css` file:

```
#content_area .header_image {
    background-repeat:no-repeat;
    padding:90px 200px 50px 40px;
    color:#9dc5e9;
}
```

This CSS code simply sets the text color to a light blue to match the example header image being used, and adds padding to place the text in the right spot.

If either Custom Field is not present then we fall back on the regular `<h2>` title. And with that our `page.php` is complete!

Creating the News Section

With our Pages sorted, we'll turn our attention to the news section. This is basically a little blog and as you will recall we have set things up so that when a user clicks News in the menu they will be served up content packaged in the `index.php` template file. So our first stop is to edit that file to display a simple blog listing.

Updating Index.php

The main update we need to make is to remove the featured Post section that has been ported over from Creatif Blog. To do this, simply remove these bits of code:

```
<?php if(is_home()){ include(TEMPLATEPATH.'/
featuredpost.php'); } ?>
```

`<? if ($post->ID != $featured_ID) { ?>` and `<? } ?>` (but not what's in between!)

Also because we want a simpler looking news section, we'll remove the following code from the Post byline:

```
tagged <?php the_tags('');?> by <?php the_author_
posts_link();?>
```

Finally we need a new sidebar that caters to the blog. So replace the `get_sidebar()` line with:

```
<?php get_sidebar('blog');?>
```

Adding a Blog Sidebar

Next we duplicate the `sidebar.php` file and name the new copy `sidebar-blog.php`. As you recall from Chapter 7, the `get_sidebar()` function takes a special parameter that lets you include multiple sidebar files using the parameter as the second half of the filename.

In our new sidebar we can revert back to a simple variation of the original sidebar from Creatif Blog:

```
<div id="sidebar">
    <img src="<?php bloginfo('template_directory');?>/
images/ribbon_browse.png" class="ribbon" alt="Featured
Project"/>
    <div class="block_inside">
        <ul>
            <li id="search" class="widget"><h3>Search</h3>
                <?php include(TEMPLATEPATH.'/searchform.
php');?>
            </li>
            <li id="subscribe" class="widget"><h3>Subscribe
</h3>
                <ul>
```

```
                    <li><a href="<?php bloginfo('rss2_
url'); ?>">RSS Feed</a></li>
                    <li><a href="http://feedburner.google.
com/fb/a/mailverify?uri=creatif">Email Updates</a></li>
                    <li><a href="http://feeds.feedburner.
com/creatif"><img style="border:0" src="http://feeds.
feedburner.com/~fc/creatif?bg=a6a2a0&amp;fg=ffffff&amp;anim
=0" alt="" width="88" height="26" /></a></li>
                </ul>
            </li>
                <li id="recent_posts" class="widget"><h3>Re
cent Posts</h3>
                    <ul>
                        <?php
                            $recent = new WP_Query();
                            $recent ->
query('showposts=7');
                            while($recent -> have_posts())
: $recent -> the_post();
                        ?>
                            <li><a href="<?php the_
permalink(); ?>" title="<?php the_title(); ?>"><?php the_
title(); ?></a></li>
                        <?php endwhile; ?>
                    </ul>
                </li>
                <li id="archives" class="widget"><h3>Archives
</h3>
                    <ul><?php wp_get_archives('type=monthly
&limit=7'); ?></ul>
                </li>
                <li id="categories" class="widget"><h3>Cate
gories</h3>
                    <ul>
                        <?php wp_list_categories('title_
li=&orderby=name'); ?>
```

```
                    </ul>
                </li>
            </ul>
        </div>
    </div>
```

Then there is one other place to update the sidebar. When a single post is displayed we need the `single.php` file to also use the blog version of our sidebar. So we'll need to open up `single.php` and replace `get_sidebar()` with the same `get_sidebar('blog')` code.

The Homepage

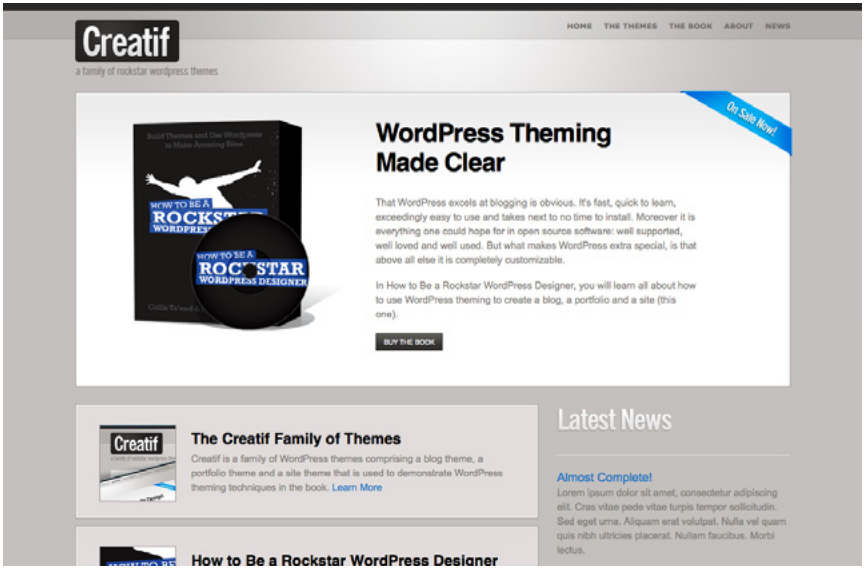


Fig 8-5 – The Creatif Site homepage, based on the Creatif Portfolio styles.

All that is now left is the homepage, for which we are going to use the Theme Options we setup earlier. Open up the `site_home.php` template file, and take the following steps:

1. Replace the Featured Post include with the actual code from the file `featuredpost.php`. Then delete the old include. Having the code together will keep things simpler in this theme.

2. Replace the `get_sidebar()` command with `get_sidebar('home')`. Then create an empty file called `sidebar-home.php` which we'll fill in a moment.

3. Replace the main Post area with three of the portfolio blocks from Chapter 7. We'll use those portfolio styles for our site homepage.

4. Finally replace all the images, titles and text with our Theme Option values. The code to grab a particular Theme Option is this:

```
<?php echo $mytheme->option['main_image'] ?>
```

   Simply substitute `main_image` for the variable name set in `functions.php`.

You should be left with this:

```
<?php
/*
Template Name: Site_Home
*/
?>
<?php get_header(); ?>
        <div id="block_featuredblog" class="block">
            <img src="<?php bloginfo('template_directory'); ?>/images/ribbon_salenow.png" class="ribbon" alt="On Sale Now"/>
            <div class="block_inside">
                <div class="image_block">
```

```
                    <img src="<?php echo $mytheme-
>option['main_image'] ?>" alt="Featured Post" />
                </div>
                <div class="text_block">
                        <h2><?php echo $mytheme-
>option['main_title'] ?></h2><br />
                        <?php echo $mytheme->option['main_
text'] ?>
                    </div>
            </div>
        </div>
        <div id="block_portfolio">
                  <div id="portfolio_items">
                    <div class="mini_portfolio_item">
                        <div class="block_inside">
                            <img src="<?php echo $mytheme-
>option['feature_1_image'] ?>" class="thumbnail"/>
                            <h3><?php echo $mytheme-
>option['feature_1_title'] ?></h3>
                            <?php echo $mytheme-
>option['feature_1_text'] ?>
                        </div>
                    </div>
                    <div class="mini_portfolio_item">
                        <div class="block_inside">
                            <img src="<?php echo $mytheme-
>option['feature_2_image'] ?>" class="thumbnail"/>
                            <h3><?php echo $mytheme-
>option['feature_2_title'] ?></h3>
                            <?php echo $mytheme-
>option['feature_2_text'] ?>
                        </div>
                    </div>
                    <div class="mini_portfolio_item">
                        <div class="block_inside">
                            <img src="<?php echo $mytheme-
```

```
>option['feature_3_image'] ?>" class="thumbnail"/>
                                <h3><?php echo $mytheme-
>option['feature_3_title'] ?></h3>
                                <?php echo $mytheme-
>option['feature_3_text'] ?>
                            </div>
                        </div>
                    </div>
                    <?php get_sidebar('home'); ?>
                     <!-- a Clearing DIV to clear the DIV's
because overflow:auto doesn't work here -->
                    <div style="clear:both"></div>
                      </div>
               </div>
          </div>
<?php get_footer(); ?>
```

The Home Sidebar

Finally we need to setup a special sidebar that grabs the latest news Posts. We can again use the styles from Creatif Portfolio and a simple `WP_Query()` to loop through the three latest Posts. Here's the code to place inside `sidebar-home.php`:

```
<div id="text_column">
<h2 id="text_title">Latest News</h2>
    <div class="separator"></div>
    <?php
        $recent = new WP_Query();
        $recent -> query('showposts=3');
        while($recent -> have_posts()) : $recent -> the_
post();
    ?>
          <big><a href="<?php the_permalink(); ?>"
title="<?php the_title(); ?>"><?php the_title(); ?></a></big>
```

```
            <?php the_excerpt(); ?>
            <div class="separator"></div>
        <?php endwhile; ?>
    </div>
```

The `WP_Query` code is similar to code we've used in the Creatif Blog sidebar. To make the styles work we need a new image for the `#text_title` style, and then we also need a revised height variable.

Wrap up of Creatif Site

In this chapter we've seen how Pages and sub-Pages can be used to manage a static site, how Theme Options can be used to create a custom homepage and another method for adding a non-home blog listing page using the WP-Admin Reading settings.

Theme Options

In this book we've made us of *ThemeToolkit* to set our Theme Options. This has the advantage that it makes the coding very simple, but the options page itself leaves a little to be desired. If you are interested in developing a custom options page you can find a great tutorial at: http://blog.themeforest.net/wordpress/create-an-options-page-for-your-wordpress-theme/

It requires some extra coding, but the end result is much more flexible and you can customize a page that your client will have an easier time with.

Plugins for CMS Use

Using WordPress in this way to develop static sites is a common business task, and as such there are a variety of plugins that you'll want to take a look at:

1. **Adding Forms**
   There are many form plugins for WordPress. One of the best-known and supported is *CForms* – http://wordpress.org/extend/plugins/cforms/

2. **Adding Newsletters**
   Clients love newsletters, and *Subscribe2* – http://wordpress.org/extend/plugins/subscribe2/ – is a popular plugin for adding this functionality into WordPress.

3. **Managing Multiple Clients**
   If you need to manage what users can do what in WordPress – either because there are multiple client users, or because you want to prevent your client from touching certain settings, you'll want *Role Manager* – http://redalt.com/Resources/Plugins/Role+Manager

9

Innovative Ways to Use WordPress

In this book we've walked through building a blog theme, using WordPress to go out of the box and create a portfolio and how to use WordPress as a general site content management system.

Thanks to its flexible plugin and theming architecture, WordPress is capable of much, much more. In this final chapter, we'll look at how you can use WordPress to create a variety of different types of sites and functionality. As you'll see with a bit of cleverness, the sky is the limit!

1. WordPress as a Membership Directory

A membership directory is a site that allows people to join, add their details and then create some sort of membership listing. So for example you might use WordPress to create a directory of tourist activities and accommodation in a certain location. Then small businesses could sign up, add their business details and images and create a listing.

WordPress' out of the box user management is fairly sophisticated and includes a solid login and registration system. To create the membership pages you need to theme up a special author.php layout (http://codex.wordpress.org/Author_Templates). You can also add in special user fields using the WP-User-Manager plugin (http://www.dealsway.net/2007/11/05/wp-user-manager/)

A great tutorial on building a membership directory, including information on how to deal with privileges that authors have can be found here: http://www.wpdesigner.com/2008/03/01/how-to-use-wordpress-as-a-membership-directory/

2. WordPress as an E-Commerce Store

Add a shopping cart, integrate with PayPal or Authorize.net and sell products using WordPress? Not as hard as you might think. Here are three options for setting WordPress up as a store:

Use a Plugin:

The most popular e-commerce plugin is definitely: http://wordpress.org/extend/plugins/wp-e-commerce/ which is not only free but includes PayPal integration and an AJAX shopping cart.

Use a Theme:

MarketTheme – http://www.markettheme.com – is a premium theme (so you need to pay) that includes a large featureset for creating an e-commerce store. Like all themes it's customizable with a bit of theming knowledge and all of the hard work of sales integration is taken care of for you.

Use a Third Party Service:

E-Junkie – http://e-junkie.com – is a service that lets you sell items from anywhere by pasting in a few lines of javascript. Create a WordPress theme that turns Posts into product listings, Post categories into product types and then add E-Junkie code to each Post to let visitors buy products.

3. WordPress as a Premium Membership Site

A premium membership site is one where visitors pay either a subscription or one-off fee to access special members-only content. WordPress can be used to build a premium membership site in one of two ways:

Use a Plugin

There are several PayPal-integration plugins for WordPress that allow you to add paid-for membership functionality, including:

http://wp-member.com/
http://tinyurl.com/easypaypal

These work by boosting the WordPress membership system to let you create member's only sections which can only be accessed after a new member has paid for their account.

Use a Third Party Membership System

Many popular membership systems have plugins to work with WordPress. The best example is aMember (http://amember.com) which can be easily integrated using a standard add-on (http://www.amember.com/integration.php). The aMember software then protects either the entire WordPress install or certain parts. When a visitor joins aMember they automatically become a user in the WordPress installation, however you have access to a host more membership management features through the aMember console. This means you can run voucher promotions, give out free memberships, integrate with other systems and more.

4. WordPress as a Social Media Feed Aggregator

With all the many social services out there, there are two reasons why you might want to make a feed aggregator. You might want to build a popurls type service (http://popurls.com/) to display content from popular feeds or alternatively you may simply want to aggregate all your own feeds onto a single page as a type of homepage.

Arguably the best PHP RSS aggregator plugin is SimplePie (http://simplepie.org) and fortunately SimplePie have a freely available WordPress plugin to help you get going: http://simplepie.org/wiki/plugins/wordpress/simplepie_plugin_for_wordpress

Using SimplyPie you can create a theme that pipes through content from any site that produces an RSS feed – in other words virtually any modern web service.

While SimplePie is perfect for processing any type of feed, there are also specialized plugins for specific types of feeds, for example: FlickrRSS (http://eightface.com/wordpress/flickrrss/) will port in photos, Smart YouTube (http://wordpress.org/extend/plugins/smart-youtube/) will bring in videos, and Twitter for WordPress (http://wordpress.org/extend/plugins/twitter-for-wordpress/) will display your twitter feed. Search the plugins library (http://wordpress.org/extend/plugins) for a specific service and chances are there's a plugin made specially for it.

If that seems like a lot of work, you could just customize ericulous' OneNews WP Theme (http://ericulous.com/2007/06/11/popurls-clone-using-wordpress/) that is a popurls clone right out of the box.

5. WordPress as a Musician/Band Website

There are several great plugins for turning WordPress into a musician's site. First grab Discography – http://wordpress.org/extend/plugins/discography/ – which automatically creates a page for every track and album you enter where fans can comment or listen. Second add in a Gig calendar with http://wordpress.org/extend/plugins/events-calendar/ and you can automate upcoming gigs the band is playing and keep the fans informed. Using plugins like FlickRSS (http://eightface.com/wordpress/flickrrss/) or Smart

YouTube (http://wordpress.org/extend/plugins/smart-youtube/) you can port in images and videos to provide some visual goodness. And of course with WordPress' regular blogging and content management abilities you have all the makings of a great band site.

6. WordPress as a Design Gallery

Some very well known CSS galleries including BestWebGallery (http://bestwebgallery.com/) are powered by WordPress. The basic principle behind building a theme that acts as a design gallery is to make your Posts into the gallery entries. So the Post title becomes the entry name, the Post body becomes the entry's description. To add an image you use Custom Fields as we did in the Portfolio theme. Using Custom Fields you can add all sorts of extra data about the gallery entries and with a rating plugin or javascript add-on like JS-Kit's Rating service (http://js-kit.com/ratings/) you can add user scores.

There are also plenty of Design Gallery themes out there that you can grab and modify including: CSS-Gallery-Theme http://themeforest.net/item/css-gallery-theme/19920 and Design Showcase http://themeforest.net/item/design-showcase/19791

7. WordPress as a Podcasting Site

WordPress is a great platform for creating a podcasting site because all you really need is a way to post up your latest podcasts. At its most basic you can simply place a link to each podcast in a Post and be done with it. WordPress' page on Podcasting – http://codex.wordpress.org/Podcasting – explains about setting up feeds so iTunes users can subscribe in a single click. You might also want to look at using http://feedburner.com services – be sure to click track enclosures so you can measure your podcast downloads.

The best known podcasting plugin is http://wordpress.org/extend/plugins/podpress/ which adds a player to your page amongst other things. But at the end of the day building a podcasting site is no different to building a regular blog, you just need to add the actual podcast files in.

8. WordPress as a Review Site

Review sites can be tailored to virtually any niche from restaurant reviews to electronics reviews. Creating a WordPress theme for such a site is mainly a case of clever design (see the section on Magazine themes below). Regular Posts can be used as the editorial reviews and the addition of rating plugins like http://wordpress.org/extend/plugins/gd-star-rating/ or JS-Kit's Review http://js-kit.com/reviews/ and Navigator http://js-kit.com/navigator/ widgets allows user interaction in the review process.

If you're looking for a paid product to create a review site, go no further than WPReviewSite – http://www.wpreviewsite.com/ which adds some clever extra functionality like post sorting based on rating.

9. WordPress as a Social Network

Using WPMU

If you really want to build a social network using WordPress you should use WordPress Multi-User (WPMU) which is a fork of the WordPress codebase and is what Automattic uses to create WordPress.com. You can grab WPMU from http://mu.wordpress.org.

WPMU allows people to sign up and get their own WordPress blog complete with themes and posts and everything else you know and love about WordPress. Then there is one central account that lets

you control various settings for your users. Where's the social part of this you ask? Well, add in a forum like BBPress – http://bbpress.org and a special set of WPMU plugins called BuddyPress – http://buddypress.org/ and you have a social network. BuddyPress was recently acquired by Automattic and gives WPMU a ton of new functionality like messaging, profiles and more.

Also be sure to check out WPMU Dev Premium – http://premium.wpmudev.org/ – and WPMU Dev – http://wpmudev.org/ – for lots of plugins, themes and WPMU setup help.

Using Plain Ol' WordPress

So what if you want to just use a regular WordPress install? Well your options are a little more limited, but you can still do some neat stuff. You'll first want to boost up membership pages so that people can join the site and get a profile page. For information on how to do this, check out the section on building Membership Directories at the start of this chapter.

Once you've got profile pages going, you'll also want some sort of communal forums. Automattic also produce BBPress which is reasonably easy to integrate – http://bbpress.org/blog/2006/09/simpler-integration-with-wordpress/ – but the forum itself isn't that developed yet, so you may have to hunt around on how to integrate your favorite forum software with WordPress.

You might also want to check out Aleph – http://wordpress.org/extend/plugins/el-aleph/ – a set of social network plugins that isn't very well supported, but looks promising.

10. WordPress as a Job Board

Building a job board with WordPress from scratch is a reasonably difficult task because you need to integrate with a payment system like PayPal. Fortunately the work is done with a premium theme from DailyWP called JobPress – http://www.dailywp.com/jobpress-wordpress-theme/

JobPress is a decent looking theme with some solid payment functionality, so if you're not too keen on the regular look, just re-theme it keeping the underlying PayPal integration and Post structure.

11. WordPress as a Community News Site

Blogs naturally build up a following of users who are often quite willing to participate in the site by submitting news and posts. Building a Community News engine into WordPress can be done in one of two ways.

Community Posting

Allowing anyone to submit a post isn't too difficult. You'll need to set the default user type to Contributor rather than the not-able-to-post Subscriber role, and then open up registrations. Contributors can create Posts but not publish them, so as editor you'll need to approve which ones go up.

To fine tune exactly what your users can do and see you'll want the Role Manager plugin – http://www.im-web-gefunden.de/wordpress-plugins/role-manager/ – which lets you specify exactly what privileges your users have.

If you're uncomfortable with letting just anyone register as a Contributor, you can leave the default user type to be Subscriber and then treat them as applications. If you approve a person you simply manually update them to Contributors.

You can then add a simple rating system using JS-Kit's Rating widget – http://js-kit.com/ratings/ – which allows you to create a Popular box using their Navigator widget – http://js-kit.com/navigator/ – thus creating a lightweight voting system.

Community Links

Another community news solution is to allow users to contribute links to posts and sites for approval by the editor. This can be done with the aid of a plugin – http://wordpress.org/extend/plugins/fv-community-news/ – or simply by hacking up WordPress' comment feed – http://nettuts.com/working-with-cmss/hack-together-a-user-contributed-link-feed-with-wordpress-comments/ – to work as a link feed. These links can then be shown in the sidebar alongside the regular site content.

12. WordPress as a Video Portal

Setting up a video portal is not difficult simply because you can use third part video sites like Vimeo and Blip.tv and then just use WordPress posts to embed the videos in using the regular embed code that such sites provide.

There are two premium theme companies that make some great WordPress Video themes: Press75 – http://www.press75.com/ – have a whole set of different video themes which not only are feature packed but just look damn nice, while Quommunication – http://quommunication.com/video/ – have just one video theme but it's so brilliant it's all they need.

13. WordPress as a Mobile Site

Mobile phones are getting a lot of attention these days and though their browsers are edging towards a conventional display there's a long way to go yet. WordPress makes a great engine for building sites aimed at mobile users thanks to MobilePress – http://mobilepress.co.za/ – a plugin that lets you develop a specifically mobile version of your theme as well as to force certain types of phone browsers (iPhone, Windows SmartPhone and Opera Mini) to render the mobile version.

So how do you actually theme for mobile devices? Just like you'd usually do – but of course with mobile constraints in mind. Here's a great article on developing for phones – http://www.webcredible.co.uk/user-friendly-resources/web-usability/mobile-guidelines.shtml

14. WordPress as a Freebie Aggregator

The web is full of great free stuff – from icons to programs to scripts to photos. There's so much that it's hard to keep up, which is why aggregator sites like Brusheezy – http://brusheezy.com or QBrushes – http://qbrushes.com are extremely popular.

Building an aggregator site is easy with WordPress thanks to the magic of Custom Fields. Simply make each item listing a Post and use Custom Fields to give any special information (download size, author URL etc) and then output them on the Post. In many ways a freebie aggregator is similar to the Design Gallery theme covered above, only instead of users visiting a design site they are downloading an item.

15. WordPress as a Twitter Clone

OK you can't *really* make a Twitter clone, but you can do some neat quasi-twitter stuff using the Prologue theme that the Automattic folks released – http://en.blog.wordpress.com/2008/01/28/introducing-prologue/

It's actually designed to act as a messaging system for a smallish group of people (think less than 20) and Automattic apparently use a password protected version for their internal team messaging. Prologue is a great example of just how flexible the WordPress engine is.

16. WordPress as a Magazine or News Site

The basics of building a magazine theme have all been covered in this book. Essentially a magazine theme is a combination of a clever design and using a lot of `WP_Query` to do custom queries. So for example if your theme has a set of featured news this would be done in much the same way as we featured a blog post in Creatif Blog. If you wanted to show the five most recent posts from a certain category, you simply run a `WP_Query` on that category in a similar manner to how we extracted archived posts in Creatif Blog.

A great tutorial on the basics of building a magazine theme can be found at

http://nettuts.com/tutorials/wordpress/build-a-newspaper-theme-with-wp_query-and-the-960-css-framework or you can simply grab a copy of a magazine theme like Mimbo – http://www.darrenhoyt.com/2007/08/05/wordpress-magazine-theme-released/ and retheme the main functionality.

Even More Ideas on Theming WordPress

The tools that we've covered in this book coupled with the astounding range of plugins available at http://wordpress.org/extend/plugins/ mean you have everything you need to take WordPress and build almost any type of site.

The defining principle behind using WordPress to build non-standard sites is to rethink Posts to be pretty much whatever data type you want. Whether it's real estate listings – http://themeforest.net/item/real-estate-theme/17730 – or address book entries – http://designintellection.com/downloads/wp-contact-manager/ – or any of the other examples we covered above.

Abstracting out the Post idea and coupling this with clever visual design means WordPress can be the back end of almost any type of data site you can think of. Add in the user functionality and plugin extensions and you've got one heck of a site engine!

Afterword

As you've learnt in this book, WordPress is not only easy to use, but it is also so flexible in its architecture that you can bend it to almost any project. With WordPress in your toolkit, you'll be able to deliver fast and effective solutions to almost any website project that comes your way.

Whatever you choose to do with WordPress we wish you success in your theming endeavors and hope this book has been an inspiration and a guide to getting you on your way!

Collis Ta'eed & Harley Alexander