2

Preparing a Design for our WordPress Theme

The purpose of this chapter is to help you create a working HTML5 and CSS3 based template mockup, with an eye towards having it end up being a WordPress theme. This theme will be responsive, meaning it will display content optimally on desktop browsers as well as mobile devices. All the while, we'll be staying compliant with W3C standards and following good usability practices. Our hope for this chapter is that even you design pros may discover interesting tidbits that will help you in your WordPress theme design creation.

WordPress theme design is essentially web design so throughout the chapter, we'll be focusing a bit more on thinking about semantics, standards and usability first. We'll then focus on what we want to design (keeping in mind it will end up in WordPress) using the most simple, straightforward means possible: pencil and paper, HTML and CSS, and last, our graphic editor/drawing programs. This approach will give us a more flexible, yet solid HTML and CSS structure.

While you might find this approach a little strange at first, it's by no means set in stone as the only right way to design a theme! Simply read through the chapter and, even if you already have a polished, Photoshop designed mockup, go ahead and try to set up your HTML and CSS using the steps laid out in this chapter. You may find it helps your process.

In this chapter, we're going to take a look at implementing the following strategies we learned in chapter 1 by:

* Building out our layout based on semantic content
* Adding in our content, fonts and sizing
* Setting up our layout using our CSS layout framework
* Adding in our graphic elements using CSS3 and our object-oriented approach to CSS

By the end of this chapter, we'll have a working HTML5 and CSS3 based template "comp" or "mockup" of our WordPress theme design, ready to be broken down, coded up and assembled into a fully functional WordPress theme.

Already got a design? Not a designer at all?

That's fine! This chapter covers basic, web design best practices, with an eye towards ending up with a unique and custom WordPress theme. It contains time honored and tested methods for approaching compliant, accessible and responsive HTML and CSS creation. If you're a total HTML and CSS design wizard, you can skim this chapter for any new tips and tricks that might be of use to you and then move on to Chapter 3. If you're not a designer at all and you just need to convert an existing HTML/CSS template into WordPress, we’d still recommend you skim this chapter, as it may help you better understand some of the HTML markup and CSS in your template. You can then move on to Chapter 3 to learn how to code up working HTML and CSS templates and mockups into WordPress.

# Getting Ready to Design

Design Comp is an abbreviation used in design and print. It refers to a preliminary design or sketch is a "comp," as in: "comprehensive artwork", or "composite". It is also known as: "mockup", "sample artwork", or "dummy artwork". We’ll be creating one of these in this chapter to then use to create our WordPress theme in Chapter 4.

You may already have a design process similar to the one detailed next; if so, just skim the next section and skip down to the next main heading.

## Designing in the browser

Historically, most web designers have used Photoshop or another graphics program to create a static design for a site and then either developed the site themselves or passed this to a developer to create the code.

This approach reflected the fact that web design had its background in print design, which makes good use of this sort of process. It gives you a nice static mockup that you can give to a client for approval.

But this approach isn’t so effective anymore. Now that our sites need to look good on a variety of devices, that single static design won’t apply to every screen size. Does this mean we have to prepare a full design for every conceivable screen size?

Of course not. The approach we’re moving towards is replacing designing in a graphics program with designing in the browser.

The way you approach this will depend on your own preferences and the needs of your project, but a process which works for lots of designers is:

1. Generate design concepts and ideas for the site using a mood board or similar technique, so you know what styling and graphics you’ll be using.
2. Prepare some wireframes for the site’s layout at different screen widths. These can be a rough sketch on paper or use a wireframing tool.
3. Create a static mockup of the design in the browser, using the layouts defined in your wireframes and the graphical treatments in your mood board. This gives you a working prototype of your design which is much more effective for demonstrating to clients how their site will actually work for users on different devices.
4. Turn that static design into a WordPress theme, using the HTML and CSS you’ve used for your mockup and adding WordPress goodness to it.

This is the process we’ll be using in this chapter, which will take us up to step 3 of the process. In Chapter 4 we’ll move on to the final step, turning our mockup into a WordPress theme.

Of course, you may already have a fully worked-up static design which you’ve been given by a designer, which doesn’t mean you can’t follow this chapter. Just skip to the section on Creating your design: from the sketch to the screen and instead of using wireframes and a mood board to decide how to code your design, use the design you’ve been given.

# Starting our Design

As mentioned, for the third edition of this title, we're going to stick with the magazine site. The difference is, it's been almost four years since the original design was created (eons in internet years) and it's time for an update that reflects today's newest web standards, design aesthetics and practices.

But the design of the magazine needs a "post 2010" update. This time around, we need something a little less "Martha Stewart" and a little more "Wired" magazine.

For those of you reading this title for the first time, here's a look at what the previous edition's design looked like:



img: 4224OS-02-01a-original-design.png

And here's what our final, responsive HTML5 and CSS3 based design will look like:



Img: 4224OS-02-01b-new-design.png

## Planning and sketching our design

The first step is to plan our design and layout.

Before doing this, you’ll need to think about the requirements of the site and of your client if you have one. If you’re designing a theme for release to other users, consider how they’re likely to use it.

Imagine you are someone who has come to the site for the information it contains. What do you think the user will actually do? What kind of goals might they have for coming to your site? How hard or easy will it be for them to attain those goals? How hard or easy do you want it to be for them to attain those goals?

Are you adhering to standard usability conventions? Web standards and conventions are more than what's laid out in a lengthy W3C document. A lot of them are just adhering to what we, as web users, expect. For example, if text has underlines in it and/or is a different color, we expect that text to be a link. If something looks like a button, we expect clicking on it to do something, like process the comment form we just filled out or add an item to our cart.

It's perfectly fine to get creative and break away from the norm and not use all the web conventions. But be sure to let your viewers know upfront what to expect, especially as most of us are simply expecting a web page to act like a web page!

Time for action: Planning our design

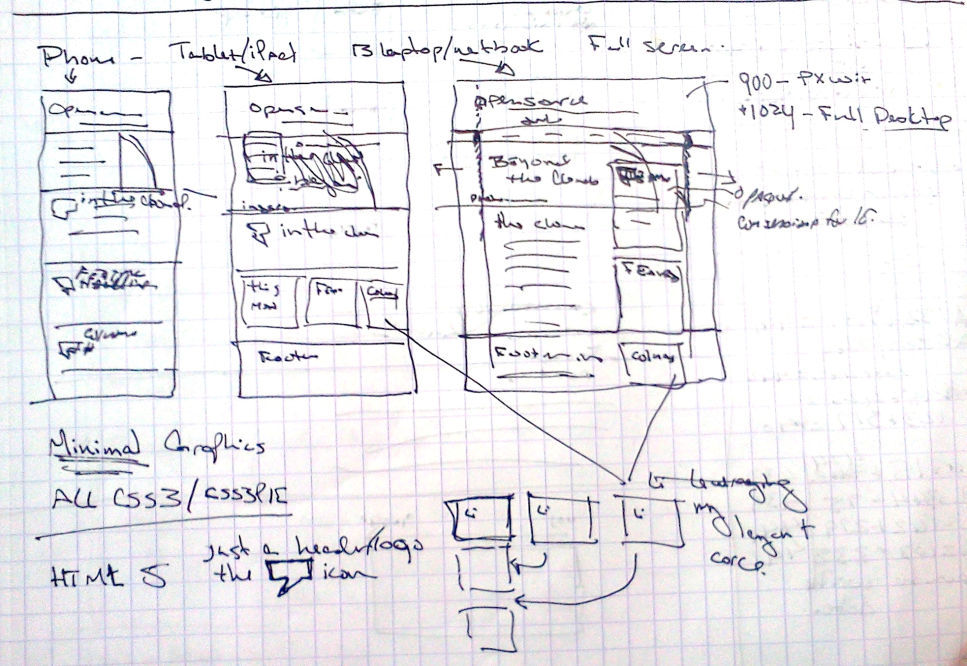
If you don’t already have a full-blown design, you’ll need to create some graphics as a starting point and some wireframes too.

1. Using your preferred graphics program, create some design elements and concepts, such as buttons, color schemes, logos, backgrounds or whatever else your site needs. Our design elements are all included in the design we’ve already seen in the figure above.
2. Either on paper or using a wireframing tool, create some wireframes for the screen widths you’re targeting with your responsive design. We’ll do ours on paper for now. You may need to prepare wireframes for more than one area of the site, for example of sections of the site will have a different sidebar or no sidebar at all.
3. Now revisit the considerations for your design and your site’s users. Make any adjustments to your wireframes that you need to.

What just happened?

You’ve created your design! It may not be what you’re used to, but you’ll find it will include everything you need to create your mockup in the browser. Obviously we’ve just skimmed this process – depending on the needs of your site, this may have taken you a very long time!

Let’s have a look at the sketched wireframes for our Open Source Magazine layout:



4224OS-02-02-rough-sketch.png

Clean it up?

This might seem to defeat the purpose of rapid design comping, but if you're working within a large design team, you may need to take an hour or so to clean your sketch up into a nicer line drawing. This may help other developers on your team to understand your WordPress theme idea more clearly.

4224OS-02-03-rough-sketch-cleaned-up.png

# Creating our design: From the sketch to the screen

We're now ready to open our HTML editor and start producing our design mockup. We'll work through the layout in HTML and CSS using our sketch and then the final visual elements will be created in an image editor at the end.

Time for action: Creating our static html file

We’ll need a single HTML file for this design, so let’s create it and start setting it up.

1. Open your HTML or text editor and create a new, fresh index.html file.

Add the following to the very first line in the document:

<!DOCTYPE html>

1. Save your index.html file.

What just happened?

We created a new HTML file to contain our mockup and added the DOCTYPE declaration to it.

You must always specify the DOCTYPE in all HTML documents, so that the browser knows what type of document to expect. If you're familiar with the DOCTYPEs for HTML 4.0 or XHTML Strict or Transitional you probably remember it being a fairly lengthy line of code. The great news is that with HTML5, things just got a whole lot simpler.

Don’t forget, even though it’s such a simple declaration, you *still* need the basic DOCTYPE declaration when using HTML5.

## The semantic body

After our DOCTYPE, we can add the other essential requirements of an HTML file, the html tags, head tags, title tags and body tags.

Time for action: Adding in basic HTML structure

The next step is to give our HTML file some basic structure.

Immediately below the DOCTYPE declaration, add in the basic HTML markup structure required for any web page to work:

<html dir="ltr" lang="en-US">

<head>

<meta charset="UTF-8" />

<title> </title>

</head>

<body>

</body>

</html>

1. Save your index.html file.

What just happened?

Our page now has the core HTML structure that all site pages need. It has defining html tags, head tags where meta and other defining and included information are placed, and most importantly, body tags. HTML body tags are where everything that's seen on a web page goes.

Now we’ll add some more elements for our theme’s content.

Time for action: Adding in the semantic structure

The markup for our mockup now needs to go between those body tags we created.

Between your <body> tags in index.html, add the code for a very basic semantic overview of your theme:

<div id="container"><!--layout container-->

<header>

<em>Header:</em> background image and text elements for header will go inside this div.

</header><!--//header-->

<!-- Begin #container2 this holds the content and sidebars-->

<div id="container2">

<!-- Begin first section holds the left content columns-->

<section>

<!-- Begin content -->

<article>

<em>Main Content:</em> Post content will go here inside this div.

</article>

</section>

<!-- Second section holds the right columns-->

<section>

<!-- #left sidebar -->

<aside class="sidebar1">

<em>Left Side Bar:</em> Will contain WordPress content related links

</aside><!--//.sidebar1 -->

<!-- #right sidebar -->

<aside class="sidebar2">

<em>Right Side Bar:</em> This will include additional ads, or non-content relevant items.

</aside><!--//.sidebar2-->

</section>

</div><!--//#container2-->

<nav id="mainNav">

<em>Top Nav:</em> For reading through straight text, it's best to have links at bottom (css will place it up top, for visual ease of use)

</nav><!--//mainNav-->

</div><!--//container-->

<footer>

<em>Footer:</em> Useful information and quick links for CSS design users who've had to scroll to the bottom plus site information and copyright will go here

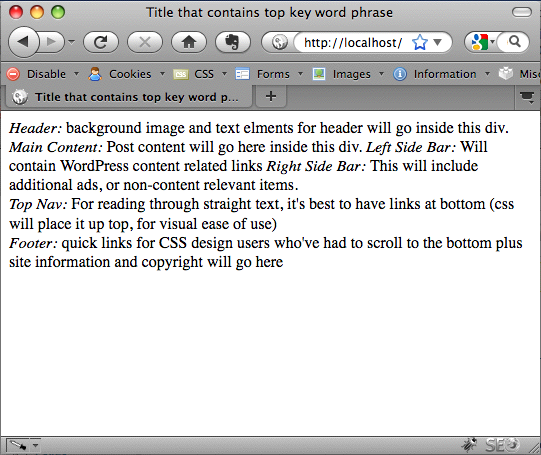
</footer>

1. Save your index.html file.

What just happened?

We added some semantic markup to our HTML file to give us an idea of what will be going where. Note that the actual text won’t be the same in our final theme, this is just to give us some direction as we complete our design.

So, what does this page look like if we open it in a browser?



4224OS-02-04-semantic-layout.png

As you can see, it’s very simple right now, but it’s a start!

If a search engine bot or someone using a text-only/text-to-speech browser or mobile device arrived and viewed our site, the following is the order they'd see things in:

* Header: Because it's good to know whose stuff you're looking at
* Main content: Get right to the point of what you're looking for
* Left column content: Under the main content, we should have the next most interesting items features list, category (sometimes referred to as columns links), and archives (sometimes called "Past Issues" links)
* Right column content: It is the secondary information such as advertisements and non-content related items
* Top page navigation: Even though in the design this will be on the top, we’ve coded it at the bottom in text-only viewing with an anchor link to it for easy access.
* Footer information: If this was a page of real content, it's nice to see whose site we're on again, especially if we've been scrolling or reading down for some time

## Attaching our CSS stylesheet

So, now that we have an HTML file set up, the next step is to create a stylesheet for our CSS.

Here's a quick refresher on how to apply the following CSS rules (if these seem unfamiliar or new to you, you may want to check out the resources in Chapter 1 and brush up on your CSS skills):

* HTML object tags (header, paragraph, list items, div tags, and so on) can just be listed as a CSS rule for example, div{...} p{...}.
* ID names that are attributes and can only be used once on a page, have a "#" hash mark in front of their CSS rule for example, #container{...} #sidebar{...}.
* Class names are attributes that can be applied multiple times on a page and combined with other classes, have a period (".") in front of the rule's name for example, .floatLeft{...}.

Time for Action: Create and include a style.css shell into your index.php page

Let’s create our stylesheet.

1. In your text editor, create a new file and name it style.css. Make sure it's in the same directory as your index.php file.
2. Open your index.html file, and inside the <head> tags, just under the <title> tags, add the following link to your style.css file, :

<link rel="stylesheet" type="text/css" media="all" **href="style.css"** />

What just happened?

We created a new stylesheet called style.css and attached it to our index.html file with a line of code inside the <head> section of our HTML file.

For now our stylesheet is still empty, but we’ll change that shortly.

## Prepping for responsiveness: Viewport and apple-mobile metatags

Our theme is going to be responsive – its layout will adapt to the width of the device it’s being viewed on. In order for this to work, and for mobile devices to display the site at the correct width, we need to add some more code to our <head> section in our index.html file.

Time for action: Adding in the viewport and apple-mobile metatags:

Adding the tags is very simple.

1. In your index.html head tags, place the following meta tags:

<meta name="viewport" content="width=**device-width**">

<meta name="apple-mobile-web-app-capable" content="**yes**">

<meta name="apple-mobile-web-app-status-bar-style" content="**black**">

What just happened?

We added three meta tags to our file to ensure the theme behaves as it should on mobile devices. Let’s have a look at each:

* The viewport metatag is a Webkit requirement and not (yet) a W3C's standard. The width property controls the viewport, as we'll be styling our layout based on the screen size, we'll want this set to: device-width.
* By setting the apple-mobile-web-app-capable meta tag to yes, the site will run in full screen mode (when selected by the users), meaning it will bump the address bar up out of the screen view once the page has loaded. Even though this meta-tag has the word “apple” in it, it actually affects all Webkit based browsers so even Android devices should benefit from this tag.
* On iOS devices the apple-mobile-web-app-status-bar-style specifies the status bar settings. This setting changes the color of the bar and then moves it out of the way or allows it to stay up at the top. If we set it to "default" it will stay its normal "iOS gradient-grey", but move up out of the way. We’ve changed to black (the only color you're allowed to change it to).
* If we were to change it to black-translucent it would stay permanently over the top of the HTML content and be slightly transparent. If your users will need access to the status bar while browsing the site, you might want to consider making this setting black-translucent.

## Adding in content

We're now ready to add some text-based content. Even if you're designing a very visual theme, text is the most common element of a site, so you should be prepared to put a fair amount of thought into how it will be displayed.

### Starting with the text

We’ll start by adding some dummy text to our site. As we go along, we’ll create elements for that content to go into, which will use up to date, semantic HTML5 elements.

Time for action: Adding sample text to our semantic sections

If you're adding dummy text, you can either use lorem ipsum text (search for it on Google, you’ll find plenty of examples), or you could add some more descriptive text. The second approach often helps when building sites for clients as it helps them see what sort of content will go where. For our purposes we’ll just use lorem ipsum.

1. Still in index.html, delete any HTML you may have already added in between the body tags.
2. Add your semantic elements with their content. A section of the code used in our mockup is below (to save space here we’ve only included a sample, but you can download the full code pack for this chapter from Packt's site):

<div id="container"><!--container goes here-->

<header>

<hgroup class="screen-text">

<h1>OpenSource</h2>

<h2>Online Magazine</h2>

<p><em>Using Open Source for work and play</em></p>

</hgroup>

<div id="date">Current Month and Year</div>

</header><!--//header-->

<!-- Begin #container2 this holds the content and sidebars-->

<div id="container2">

<!-- Begin #container3 keeps the left col and body positioned-->

<section class="">

<h2 class="thisMonth">This Month</h2>

<!-- Begin #content -->

<article class="post">

<h2><a href="#">Really Long Article Title Name The More Text The Better Cause You Never Know</a></h2>

<em>Main Content:</em> Post content will go here inside this div.

<p>by Author Name for <a href="#">Column Type</a></p>

<div class="entry-content"><!--//post-->

<p>Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Sed a eros nec orci volutpat vestibulum. Ut pellentesque sagittis metus. In euismod tellus id ante. Ut lectus. Nunc adipiscing. Praesent luctus, massa quis vulputate rhoncus, justo turpis mollis dolor, nec blandit nisl mauris et pede.</p>

<p>Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Sed a eros nec orci volutpat vestibulum. Ut pellentesque sagittis metus.</p>

<p><a href="#">Read the rest of this entry &raquo;</a></p>

</div><!--//.entry-content-->

1. Save your index.html file.

What just happened?

We added some semantic elements to our site with content.

Let’s look at some of the elements we’ve included:

* A #container div to contain our page content – useful for styling
* The header element – containing the site title and description, inside an hgroup with a class of .screen-text so we can make it invisible to browsers with CSS turned on.
* A #container2 div to hold the post or page content and sidebars (in the default theme this div has an ID of #main).
* A section to contain all of the post or page content
* An article element with a class of .post – this mirrors the class WordPress will eventually assign to this element, as we’ll see in Chapter 4.
* An h2 element for the title of the post.
* A p element with author information.
* A div with the class .entry-content to hold the post content itself – again this mirrors the class WordPress will assign once we set that up.
* More p elements to contain the content itself – these will eventually be automatically populated by WordPress from the site’s database but for now we’re using dummy text.

The trick: Start with *a lot* of text

Here's a tip: Use a lot of sample text. It’s tempting to create a nice mockup that's got clean, little two-word headers, followed by trim and tight, one or two-sentence paragraphs (which are also easier to handle if you did the entire mockup in Photoshop, right?).

In this optimally minimalist sample, the design looks beautiful. However, the client then dumps all their content into WordPress and your theme, which includes long, boring, two-sentence headlines and reams of unscannable text. Your beautiful theme design now seems dumpy and all of a sudden the client isn't so happy, and they are full of suggestions they want you to incorporate in order to compensate for their text-heavy site.

Just design for lots of text upfront. If the site ends up having less text than what's in your comp, that's perfectly fine; less text will always look better. Getting lots of it to look good after the fact is what's hard.

Now let’s see what our page looks like when viewed in the browser:



4224OS-02-05-basic-text.png

You can see it has a basic structure in place, but it now needs some styling. After all, this isn’t 1992!

## Styling our fonts

So, now we’re going to add some styling for our text – starting with the fonts, or font-families used to display it.

### Styling font families

By assigning font-families to our CSS rules, we can set up backup font choices. This means if someone doesn't have our preferred font, then they’ll probably have the backup we specify, and if they don't have that? Well, at the very least we can rely on their browser's built-in "generic" assigned font. Just specify serif, sans-serif, or mono-space.

When specifying font families, it’s a good idea to include one or more of the fonts which are both commonly held on PCS, Macs and mobile devices, and which look good on screen (as against on paper). Fonts designed for screens include Verdana and Georgia, and other fonts commonly available on your users’ systems will be Arial and Times New Roman.

Our headers will be Helvetica with Arial as a fallback, and the body content of our text will be Trebuchet with Helvetica and even Verdana as a fallback.

Time for Action: assigning your font families

Let’s add some font families to our stylesheet.

1. In your style.css sheet, under the TYPOGRAPHY comment, add the following code:.

/\*------------------TYPOGRAPHY -------------------\*/

/\*

Set font stacks here

Assign default colors only. Otherwise color is handled at BOTTOM of sheet.

\*/

body

{font-family: 'Trebuchet MS', Helvetica, Arial, Verdana, sans-serif;

}

h1, h3, h5{

font-family: Helvetica, Arial, sans-serif;

}

h2, h4, h6{

font-family: 'Helvetica Neue', Helvetica, 'Arial Condensed', Arial, sans-serif;

}

a {

font-family: Helvetica, Arial, sans-serif;

}

pre, code{

font-family: Courier, monospace;

}

1. Save your stylesheet.

What just happened?

We set the font families that our theme will use, including:

* The default font for the body element – which will apply to anything we don’t specific an alternative font family for, because of CSS inheritance
* Font families for our headings and links
* Finally, a font family for pre and code element, in other words for the display of code in our theme.

#### @font-face techniques

You’ll be pleased to know we can take advantage of a much wider world of typography than just what’s hopefully installed on other people's computers. Using @font-face and other techniques you can serve up font's of your choosing to your site's users. We'll go over these techniques in detail in *Chapter 7.*

Warning: Most fonts are licensed! You must not violate the terms and licensing of fonts. As most were expecting to be used with print, many have licenses which will be violated if you use them on the web with the @font-face, Cufon (or sIFR) techniques. Your best bet is to use Open Source fonts. A great repository is http://www.fontsquirrel.com/. We'll also be using Google's new font repository: http://www.google.com/webfonts

The next step is to specify sizing for our fonts.

### Styling font sizes

We have our font families in place, but we need to tell browsers at what size to display the text in our theme.

Time for action: Sizing your fonts

Let’s add some styling for font sizes to the CSS declarations we’ve set up for our font families.

1. In style.css, edit the font styling so it reads as follows:

/\*------------------TYPOGRAPHY -------------------\*/

/\*

Set font stacks here

Assign default colors only. Otherwise color is handled at BOTTOM of sheet.

\*/

body{

font-family: 'Trebuchet MS', Helvetica, Arial, Verdana, sans-serif;

font-size: 0.9em;

color: #333;

}

h1, h3, h5{

font-family: Helvetica, Arial, sans-serif;

font-weight: 100;

line-height: 120%;

color: #666;

margin: 0.5em 0 0.3em 0;

margin-top: 20px;

}

h2, h4, h6{

font-family: 'Helvetica Neue', Helvetica, 'Arial Condensed', Arial, sans-serif;

font-weight: 100;

line-height: 110%;

color: #999;

margin: 0.5em 0 0.3em 0;

margin-top: 20px;

}

a {

font-family: Helvetica, Arial, sans-serif;

font-size: 100%;

color: #666;

font-weight: 100;

text-decoration: none;

}

pre, code{

font-family: Courier, monospace;

font-size: 100%;

margin-bottom:10px;

}

1. Now add some more specific height styling for some other elements. Add the following code below the code you’ve just added:

h1 {

font-size: 280%;

font-weight: 600;

}

h2 {

font-size: 220%;

border-bottom: 1px solid #ccc;

padding-bottom: 10px;

}

h3 {

font-size: 180%;

}

h4{

font-size: 200%;

color: #999

}

h5{

font-size: 115%;

}

h6{

font-size: 100%;

}

p {

**line-height: 150%;**

margin-bottom: 170%;

}

1. Save your stylesheet.

What just happened?

We added some additional styling for font sizing. You’ll notice that we’ve also included some styling for margins as well, to give our text some extra space where it’s needed.

As you can see, in the code examples above, the only em size we used was in the body rule. The rest of our header, paragraph and other typography based rules rely on upping or downing the font size based on percentages. 100% = 0.9em and so 90% would size down the font a tad while 280% sizes the font up considerably. Now, if our client asks to just "bump down" (or up) the size on everything "a little" all we have to do is change the main em size in the body rule. Everything else will size up or down, relatively based on the percentage we assigned it. Easy!

We then moved on to using percentages to help us with the line-height property and also made sure our a href links stand out with a different font-family, yet still have the familiar underline appear on :hover.

Using ems and percentages in this way is also far better for accessibility than using pixels, as it means that if a user has set their browser to resize text this will be applied across our theme, and not be overridden by any pixel-based text styling.

The final stage in styling our text is to deal with text we want to hide from browsers with CSS turned on, while making them visible to screen readers and search engine bots.

Time for action: Handling search engine bots/screen reader text:

You'll note in the HTML5 markup, we have several headers and hgroups assigned a class called .screen-text. This is text that users viewing the styled site in a browser won’t see but makes things clear for text screen readers and may have some SEO benefits. Let’s add the styling for it.

1. In the TYPOGRAPHY section of your stylesheet, add the following:

/\* Text meant only for screen readers \*/

.screen-text{

**position: absolute;**

**left: -5000em;**

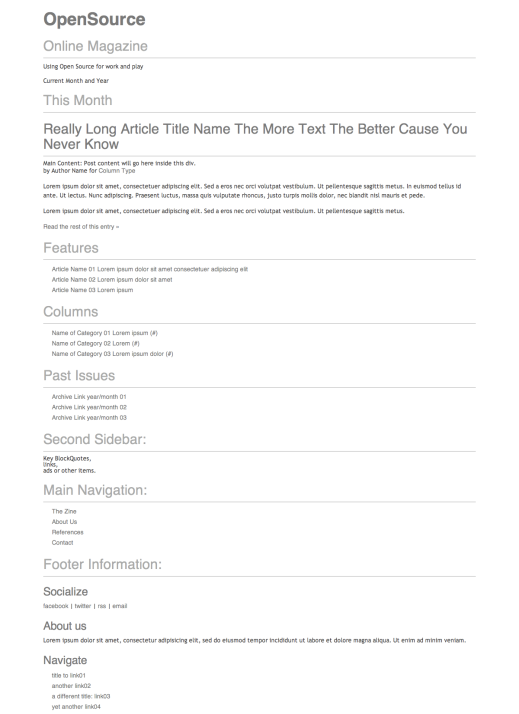
}

1. Save your style.css file.

What just happened?

We added some property settings to move any text that has the .screen-text class assigned to it, 5000 em units to the left. Assuming most people have a screen that's smaller than 5000ems, this will, be hidden from view.

Let's take a look at our basic text styling so far. The following image shows our mockup starting to take shape:



4224OS-02-06-basic-text-formatted.png

If you don't like how your text looks here, then a bunch of graphics, columns, and layout adjustments won't help! Take your time getting the text to look nice and read well now. You'll have less edits and tweaks to make later.

## Setting up our layout with CSS

Now that we've got our initial typography set up, let's start making this stuff look like our sketch! First up, we’ll add a call to the layout-core stylesheet to give our theme some basic layout styling.

Time for action – referencing our layout core to set up or positions

Let’s make sure our style sheet references the layout core.

1. At the very top of your stylesheet, add the following code:

/\*

---------------------------------------------------

|NOTE: This style sheet leverages: layout-core.css. |

---------------------------------------------------

\*/

@import url(layout-core.css);

1. Make sure you have a copy of layout-core.css in the same directory as your style.css and index.html files – you can find a copy in the code bundle for this chapter.
2. Save your stylesheet.

What just happened?

We added the @import directive to call an external stylesheet, meaning we can make use of the layout styling already set up in layout-core.css and won’t have to add it all in manually.

Layout core uses a few classes to help us achieve our layout – to use them, you’ll simply assign whether a div, section, article or aside tag should float, left or right and then assign an additional class of full, half, two-thirds, three-quarters, third, or quarter to set the width of that HTML element. We’ll come to this later in this chapter.

CSS Resets

Of course, you might be building your own stylesheet from scratch and won’t want to include layout-core.css. If so, it’s a good idea to include a CSS Reset at the beginning of your stylesheet. This resets any browser-specific CSS to we can start with a clean sheet regardless of what browser the user is viewing our site in.

Our layout-core.css file includes a rest so if you’re importing that, you don’t need to add another one.

For more on CSS resets, and an example of a great one to use, see http://meyerweb.com/eric/tools/css/reset/.

Now we have our @import directive set up, we’ll move on to adding some media queries so our theme can be responsive.

Media queries sit at the end of a stylesheet and they specify styling to apply depending on the width of the screen the site’s being viewed on. For much more on building responsive themes and leveraging media queries, see WordPress Mobile Web Development: Beginner’s Guide, written by Rachel and published by Packt.

Time for Action: Adding our media queries

Follow the steps below to add your media queries.

1. Below your typography section in your style.css stylesheet (or at the bottom of the stylesheet if you have other code below your typography section), add the following media queries:

@media (min-width: 1220px) {

}

@media (max-width: 1024px) {

}

@media (min-width: 480px) and (max-width: 800px) {

}

@media (max-width: 480px) {

}

@media only screen and (min-width: 320px) and (max-width: 480px) {

}

1. Save your stylesheet.

What just happened?

We added some media queries to target the screen sizes most commonly used. Let’s have a look at how they work:

* @media (min-width: 1220px) targets very large screens.
* @media (max-width: 1024px) targets small desktop screens and larger tablet screens.
* @media (min-width: 480px) and (max-width: 800px) targets small tablet screens or larger tablet screens in portrait.
* @media (max-width: 480px) targets small screens including phones in landscape.
* @media only screen and (min-width: 320px) and (max-width: 480px) targets phones in portrait .

These media queries work for the vast majority of devices available at the time of writing. But as more and more devices are released with different screen widths, you may find these media queries don’t continue to target the devices you expect them to. When working on your theme, you may find it helps to tweak these media queries so they target widths at which the design needs to be altered for it to look good, rather than focusing on specific devices.

## Setting up the desktop view:

We'll first start with out desktop browser view. We'll use layout-core.css to help us set our columns up.

Time for action: Standard settings

Now we’ll set up the default styling for the desktop view.

1. Below the typography section in your stylesheet and above the media queries, add the layout styling like so:

/\*------------------ STANDARD STYLING -------------------\*/

header{

height: 110px;

}

#mainNav{

position:absolute;

top: 110px;

width: 100%;

}

#mainNav li a{

display: block;

padding: 10px 15px 13px 15px;

line-height: 100%;

-size: 120%;

border: none;

color: #036;

}

#across{

margin: 0;

width:100%;

}/\*for a stretched bottom only\*/

h2.thisMonth{

font-size: 260%;

}

.content{

margin-top: 250px;

}

.sidebar{

margin-top: 150px;

}

.sidebar div{

margin-top: 30px;

padding-bottom: 10px;

}

.sidebar h2{

margin-left:20px;

}

1. Save your stylesheet.

What just happened?

We added styling for our layout on standard desktop screens. We don’t need to examine all of the CSS in details but some points to note are as follows:

* Our #mainNav navigation element has been positioned absolute so it could be brought up to the top of our layout.
* We added some sizing for our .thisMonth h2 title
* Our #across div is stretched all the way out to the full width of the screen
* We added margins and padding for our .content and .sidebar divs, in particular margin-top to push them down (this is because we'll be adding in a background image here in a bit).

Time for action: Checking in on larger desktops

We’ve styled our standard, default view, but we'll also want to handle larger desktop browsers. Let's add some CSS to our first media query, to widen that #mainNav nav element out to match our #container div if we detect a larger screen.

1. Inside your first media query, add the following code:

@media (min-width: 1220px) {

#mainNav{

position:absolute;

top: 110px;

width: 1100px;

}

}

1. Save your stylesheet.

What just happened?

Our layout-core.css stylesheet sets our div widths as percentages, so they'll expand to whatever size the #container div is set to. It also sets the #container div to 1100 pixels. Our #mainNav nav will now match and not extend over the #container div on larger screens.

Time for action: Making sure smaller screens are handled

When our layout-core.css file snaps the #container div in on smaller screens to 950 pixels, we'll want #mainNav to match that as well.

1. In the next media query, add the following code:

@media (max-width: 1024px) {

/\*for netbook/tablet screens\*/

#mainNav{

position:absolute;

top: 110px;

width: 950px;}

}

1. Save your stylesheet.

What just happened?

Our #mainNav will now snap in to the width of our #container div if the media query for 1024px or less is called. Let’s see how it looks on the relevant size screen:



4224OS-02-07-layout-laptop-desktop.png

## Setting up the tablet view

We're now ready to focus on our tablet and media player views. There are lots of devices out there in this range: the following two media queries help catch the majority of these devices.

Time for action: Adjusting the standard layout for tablets

We'll start with devices that range from 480 pixels wide up to 800 pixels. This range includes the iPad when it's held in portrait mode.

1. Add the following code to your media query:

@media (min-width: 480px) and (max-width: 800px) {

header{

height: 100px;

}

#mainNav{

top:100px;

width: 300px;

}

#mainNav li{

float:none;

clear:both;

}

#container2{

background-position: 70% -90px;

}

.home article.post h2{

font-size: 150%;

margin-bottom: 10px;

}

.home article.post .entry-content,article.post .entry-meta, article.post a.more{

display:none;

}

.content.left.two-thirds, .sidebar.right.third{

float:none;

clear:both;

margin: 0 auto;

width:98%;

}

.sidebar div{

width: 30%;

margin: 1.2%;

float:left;

}

.home .content.left.two-thirds{

margin-top:180px;

}

}

1. Save your stylesheet.

What just happened?

We added some styling to adjust the layout on tablet devices. Specifically:

* We removed the “tab” floats of our #mainNav list items li and set them to list vertically.
* We changed some font sizes of our titles
* We hid our article content, only displaying the titles
* We turned off our .sidebar element's right float.
* We set each div in the .sidebar to float: left of each other, creating a three-up box spread under our main article headlines.

The end result looks like the following:



4224OS-02-08-layout-portrait-tablet.png

## Setting up the small screen view

We're now ready to set our small screen view up, which will target smartphones and other smaller devices in portrait orientation.

Time for action: Setting up our small screen layout

Here our media query will be a little different, only applying to screens (the other queries will affect print layout as well).

1. In your final media query add the following CSS rules and changes:

@media only screen and (min-width: 320px) and (max-width: 480px) {

header{

height: 70px;

}

#mainNav{

top:70px;

width: 220px;

}

#mainNav li{

float:none;

clear:both;

}

#mainNav li a{

font-size: 100%;

padding: 10px

}

.home article.post h2{

font-size: 120%;

margin-bottom: 10px;

padding-left: 50px;

}

.home article.post .entry-content,article.post .entry-meta, article.post a.more{

display:none;

}

.home article.post a.comments{

position:absolute;

margin-top: -55px;

}

.content.left.two-thirds{

margin-top:150px;

}

.soc {

text-indent: -5000em;

}

}

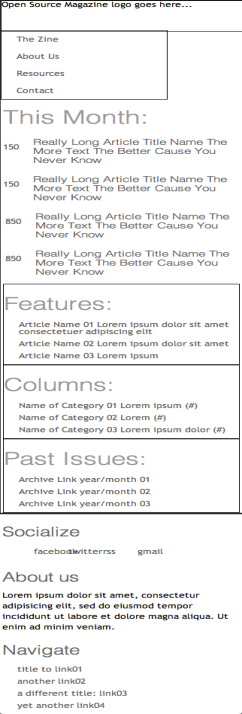
1. Save your stylesheet.

What just happened?

We’ve added some styling to improve the layout on small screens. In particular:

* We shortened up our header even more and tightened up the width of our #mainNav.
* We reduced font sizes in our #mainNav and titles.
* We reduced the margin-top of our .content div.

The most notable thing that happens is actually handled by default in our layout-core.css. All left and right floats are turned off, cleared on both sides and all percentage widths are set to 100%. This makes every div laid out in our phone view push edge to edge. The result looks like the image below:



4224OS-02-09-layout-phone.png

# Adding in design treatments

Finally! Now we have our mockup's responsive layout set up. Let's polish it off!

We'll be using CSS3 techniques, even in IE

We'll be taking advantage of many CSS3 techniques in our design, most notably, simple gradients, rounded corners and box shadows. Trouble is, IE7 and 8 don't really support any of those CSS3 features yet.

Not to worry, there's a wonderful library called CSS3PIE which will be using to create polyfill fallbacks for IE 7 and 8.

The most recent (as of this writing) version is included in this chapter's code pack but you can also go and pick up the most recent version of the library from here: http://css3pie.com/

We’ll talk about how to properly implement it for our mockup as well as how to get it working in our WordPress theme in *Chapter 3*.

Time for action: Setting up our graphic treatments in the style sheet

Now we need to set up a section in our stylesheet for design treatment rules.

1. The first step is to upload our graphics files. Create a folder called images inside the folder containing your other files. Upload your image files to it.
2. In style.css, below the STANDARD STYLING section, but above our media queries, add in our color scheme for background colors:

/\*------------------REUSABLE GRAPHIC TREATMENTS -------------------\*/

/\*main background colorscheme\*/

.bg-main{

background-color: #222;

}

.bg-secondary{

background-color: #666;

}

.bg-tertiary{

background-color: #999;

}

.bg-light1{

background-color: #eee;

}

.bg-light2{

background-color: #ddd;

}

.bg-dark1{

background-color: #000;

}

.bg-dark2{

background-color: #444;

}

1. Next, add some gradient schemes below. You’ll need to include the browser-prefixed versions of the CSS as well – we’ve left it out here to save space.

.grd-vt-main{

background: linear-gradient(top, #333, #000);

}

.grd-vt-secondary{

background: linear-gradient(top, #555, #222);

}

.grd-vt-tertiary{

background: linear-gradient(top, #ddd, #999);

}

1. Beneath this, add some rules for handling borders:

/\*borders\*/

.bdr{

border: 1px solid;

}

/\*apply thickness\*/

.bdr-2px{

border: 2px solid;

}

/\*pick a side\*/

.bdr-top{

border-left:none;

border-right: none;

border-bottom: none;

}

.bdr-left{

border-top:none;

border-right: none;

border-bottom: none;

}

.bdr-right{

border-left:none;

border-top: none;

border-bottom: none;

}

.bdr-bottom{

border-left:none;

border-right: none;

border-top: none;

}

/\*leverage selectors for border colors

-be sure to apply your rules in this order\*/

.bg-main.bdr{

border-color: #aaa;

}

.bg-secondary.bdr{

border-color: #999;

}

.bg-tertiary.bdr{

border-color: #eee;

}

1. Next, set up the rounded corners. As with gradients, we’ve omitted the browser-prefixed code to save space, but you’ll find it in the code pack.

/\*rounded corners\*/

.rnd

border-radius: 5px;

}

/\*only two corners\*/

.rnd-top{

border-radius: 5px 5px 0 0;

}

.rnd-left{

border-radius: 5px 0 0 5px;

}

.rnd-right{

border-radius: 0 5px 5px 0;

}

.rnd-bottom{

border-radius: 0 0 5px 5px;

}

1. Finally, set up some box shadows (again, browser-prefixed code has been omitted).

...

/\*box-shadows\*/

.shdw-centered{

box-shadow: 0 0 15px rgba(0, 0, 0, 0.5);

}

.shdw-offset{

box-shadow: 2px 2px 10px rgba(0, 0, 0, 0.5);

}

1. Now save your stylesheet again.

What just happened?

We added styling for borders, gradients, rounded corners and box shadows.

Keeping in mind our “object oriented” CSS strategy, the above rules have been added, not because we'll definitely use them but in case we need to use them. Just as the layout-core.css sheet freed us to open up our index.html file and simply apply class rules to our HTML elements to get our layout going, we can now go in and start applying our color scheme and graphic embellishments to our layout. And of course, if we want to change our mind about colors or gradients or borders, it's easy to update in one or two places in the stylesheet and our entire site will evenly update.

Keeping WordPress in mind

In the next chapter, we'll get into “WordPress-ifying” this layout. Keep in mind that WordPress can spit out quite a few classes of its own, and it likes to take advantage of applying multiple classes to HTML objects as well. The good news is, the majority of these classes are so that objects can be identified and offered special styling. We’ll come back to this as we work through creating our theme in the next two chapters.

# Adding graphics and background images

Having added all of the CSS-generated styling, we need to think about any graphics we’ll be using that can’t be generated by CSS.

The beauty of CSS3 is that it reduces the need for these graphics, as we no longer need to create background images for gradients, rounded corners or shadows. But there are some elements of our design that can’t be handled by CSS.

We already have a number of images ready to import into our theme – you’ll find them in the file bundle that goes with this book.

Our theme makes extensive use of background images to avoid any problems with inline images conflicting with any other content that our theme’s users may add in future. You may prefer to use inline images in your markup, which has the advantage of being better for accessibility and SEO but the disadvantage that if a future user of your theme edits the template files, he or she may accidentally delete images that are required for the design. Our images are for design only and not part of the content, so we’re using background images.

## Setting up our background images in our style sheet

We've exported our logos out to one single image. Using the background-position property will be displaying the different sized logos depending on which screen size triggers our media query.

Wellstyled has an excellent tutorial on how to use a single image technique (also referred to as "CSS sprites") to handle image background rollovers with CSS: http://wellstyled.com/css-nopreload-rollovers.html.

You can also check out CSS Tricks, and their article CSS Sprites: What They Are, Why They're Cool, and How To Use Them at http://css- tricks.com/css-sprites/.

Remember: To see the full and final CSS mockup style.css and index.html page, please refer to the code download section in the preface.

Time for action: Adding background images to our design

The images we’ve created need to be added to our stylesheet as background images.

1. In your stylesheet, edit the STANDARD STYLING section to add background images and colors, with the following code. Best practice is to add each declaration within the description blocks you’ve already set up – you can see the final code in the code bundle.

header{

background: url(images/osmag-logos.png) no-repeat 0 0;

}

#mainNav li a:hover{

color:#088fff;}

#across{

border-top: 2px solid #444;

background: #000

}

#container{

background: url(images/osmag-container-bg.png) no-repeat 50% -30px;

}

#container2{

background: url(images/osmag-earth.jpg) no-repeat 50% 0;

}

h2.thisMonth{

color: #fff;

}

h2.pastIssues{

color:#222;

}

article h2{

background: url(images/pngs/highlight-border.png) repeat-x 0 bottom;

}

.comments{

background: url(images/pngs/comments-icon.png) no-repeat 0 0;

}

1. Now add the following new section below your standard styling section:

/\*------------------REUSABLE GRAPHIC TREATMENTS -------------------\*/

/\*reusable image backgrounds\*/

.img-quote-light{

background: url(images/pngs/r-quotes-light.png) no-repeat -10px -7px;

text-indent: 55px;

}

.img-quote-dark{

background-image: url(images/pngs/r-quotes-dark.png);

background-repeat: no-repeat;

background-position: -10px -7px;

text-indent: 55px;

}

.img-bottom-shadow{

background: url(images/pngs/bot-r-shadow.png) no-repeat 50% bottom;

}

.img-top-shadow{

background: url(images/pngs/top-r-shadow.png) no-repeat 50% 0;

}

.img-line-hz{

background: url(images/pngs/highlight-border.png) repeat-x left bottom;

}

1. Save your style.css file.

What just happened?

We added CSS for backgrounds and colors into our stylesheet. As you’ll notice, we made a lot of use of no-repeat to ensure our background images didn’t repeat, and used positioning of background images to place them in our design.

Now let’s add any changes needed for those backgrounds to our media queries.

Time for action: Adding background image styling to the media queries

We’ll need to edit the two media queries for the two smallest screen sizes.

1. Inside the media query targeted at small tablets and phones in landscape – i.e. @media (min-width: 480px) and (max-width: 800px) – add the following code. Again, you’ll find you can add it to your existing declarations.

header{

height: 100px;

background-position: 0 -220px;

}

#mainNav{

top:100px;

}

#mainNav li{

background: url(images/pngs/highlight-border.png) repeat-x 0 bottom;

}

#container2{

background-position: 70% -90px;

}

1. Next, in the media query targeting small screens – i.e. @media only screen and (min-width: 320px) and (max-width: 480px) – add the following:

header{

height: 70px;

background-position: 0 -425px;

}

#mainNav{

top:70px;

}

#mainNav li{

background: url(images/pngs/highlight-border.png) repeat-x 0 bottom;

}

#mainNav li a{

padding: 10px

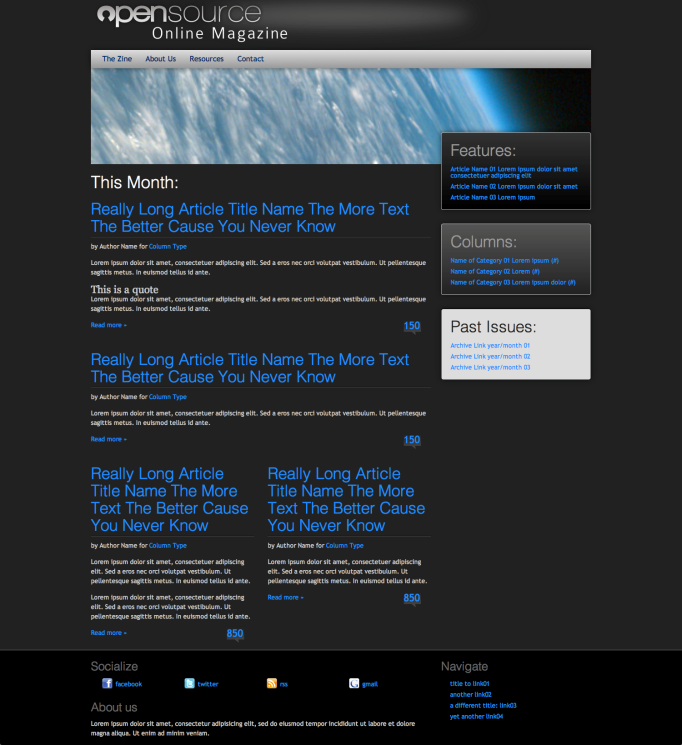
}

1. Finally, save your stylesheet.

What just happened?

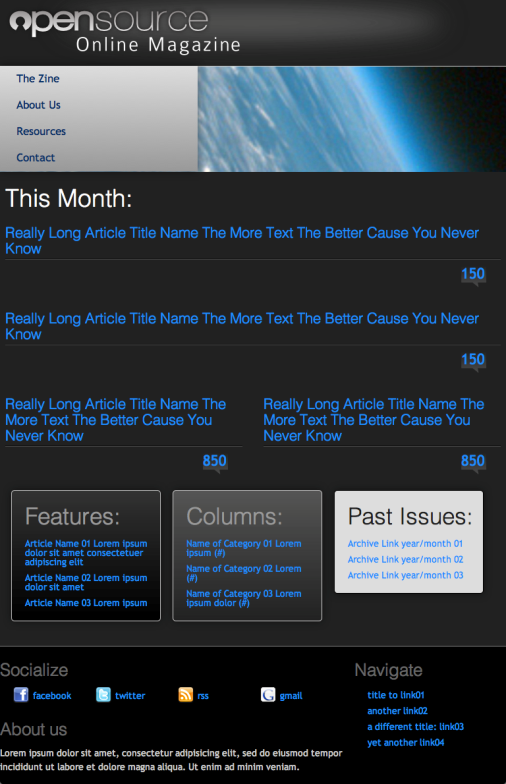
We loaded up our graphics into our STANDARD STYLING rules and then moved on to each media query modifying the height, background-position and other visual properties as needed along the way. Most notably, we changed our header height and using our CSS sprite technique with our logo loaded up the different sized logo as our screen dimension changed.

Our final desktop layout now looks like this:



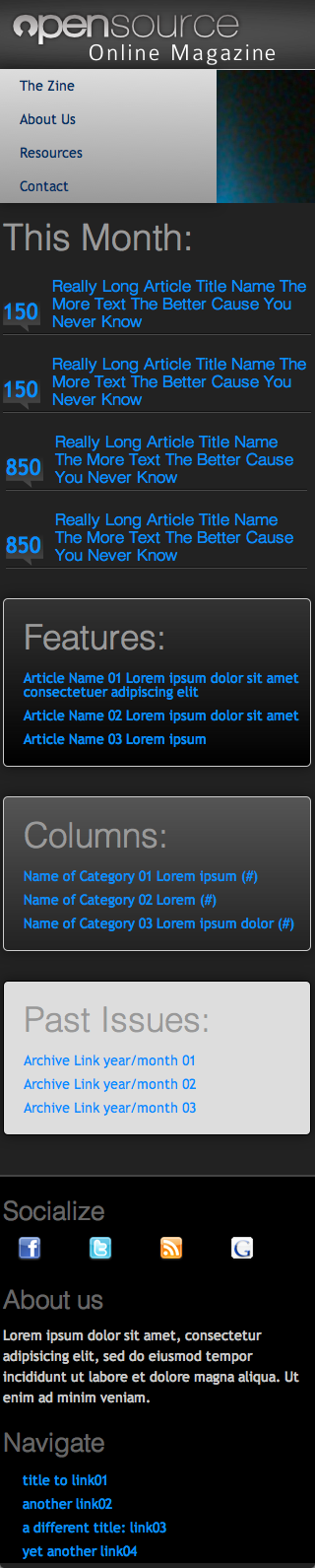
4224OS-02-13-final-largedesktop.png

Our final portrait table layout now looks like this:



4224OS-02-14-final-portrait-tablet.png

And last, our final phone layout now looks like this:



4224OS-02-15-final-phone.png

# Don't forget the Favicon and Touch icon!

You certainly don't need a favicon or touch icon, but it does add a finishing touch. Let’s work through the steps to do it.

## Adding a favicon

Favicons are those little 16 x 16px icons that appear next to the URL in the address bar of a web browser. They also show up on the tabs (if you're using a tabbed browser), in your bookmarks, as well as on shortcuts on your desktop or other folders in Windows XP and Vista.

The easiest (and quickest) way to create a favicon is to take your site's logo, or key graphic (in this case, the opened O in Open Source), and size it down to 16 x 16 pixels; then save it as a .gif or .png file.

For advice on creating a favicon, see the page on the WordPress codex at http://codex.wordpress.org/Creating\_a\_Favicon.

Time for action: adding the favicon you just created

Having created a favicon, we need to upload it to our theme.

1. Once you have your favicon.ico, place the file in the same folder as your stylesheet and index.html file
2. Open index.html and add this code inside the <head> section:

<link rel="shortcut icon" href="favicon.ico" type="image/x-icon" />

What just happened?

We added a favicon, uploaded it to the correct place, and added a line of code in our index.html file referencing it.

If you refresh your browser you’ll see it in your navigation bar.

Can’t see your favicon?

Be sure to name your file favicon.ico correctly! For some reason, even though you call the file by name in the link tag within your header tags, it just won't work if it's not named favicon.ico.

You may also find you need to clear your cache and reload several times before you see your new favicon. Be sure to actually clear your cache through your browser's preference panel. The keyboard shortcut Shift+F2(Refresh) sometimes only clears the web page cache. Some browsers cache favicons in a separate directory.

Have a go hero: Making your favicon high-res!

A little known fact about the .ico format is that it can contain multiple versions of itself at different color depths and resolutions. This is how your operating system is able to display those "smooth icons" that seem to be the right resolution no matter how large or small they're displayed. You may have noticed that some favicons if saved as shortcuts to your desktop, look great and others look jaggy and terrible. The ones that look great take advantage of this feature.

The three main sizes that Windows will display a favicon in are: 16x16, 32x32, and 48x48. Sometimes favicons go all the way up to 128 x128. It's up to you; just remember, the more resolutions, color depths, and transparencies you add, the larger your favicon file is and longer it will take to load.

You'd basically use the same steps listed above to create your favicon, just starting with 48 x 48 pixels, then save it (so as to not overwrite your original file) down to 32 x 32 and last 16 x 16. It helps to save each icon initially in PNG format, especially if you want the background to be transparent.

This article on the Egressive site is a great reference for putting a multi resolution, transparent favicon together: http://egressive.com/creating-a-multi-resolution-favicon-microsoft- windows-icon-file-including-transparency-with-the-gimp

## Touch icons

Since we just took the time to add a great, multi-resolution favicon, we might as well go all out and add in a nice touch icon. Touch icons are used by iOS devices and Android devices newer than 2.1.+. While you can technically just create one higher, res image (the 114 x 114, for example) the other devices will size it down, but then why waste bandwidth time loading in a larger image if you don't have to? It’s better to create the three required sizes.

Time for action: Adding a Touch icon

Once you have a Touch icon (which you should have saved as a png or a set of them), you’ll need to add them to your theme.

1. Upload your Touch icon to the same folder as your index.html file.
2. In index.html, add the following code inside the <head> section:

<link rel="apple-touch-icon" sizes="57x57" href="/apple-touch-icon-57x57.png"/>

<link rel="apple-touch-icon" sizes="72x72" href="/apple-touch-icon-72x72.png"/>

<link rel="apple-touch-icon" sizes="114x114" href="/apple-touch-icon-114x114.png"/>

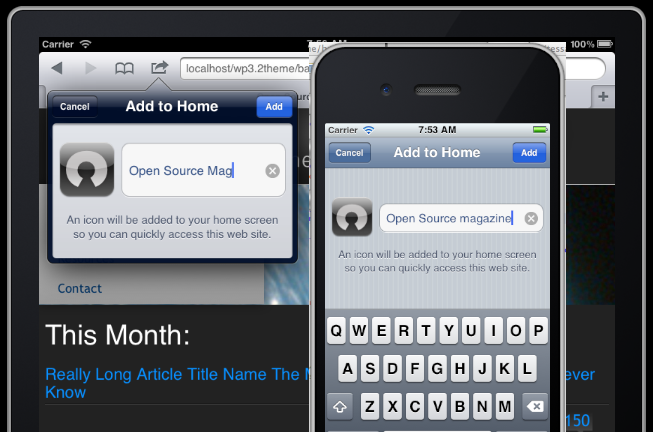
1. Save your index.html file.

What just happened?

We added a few lines of code to fetch our Touch icon to index.php. You’ll notice that we included three file sizes:

* The 57 x 57 pixel icon is what older iOS devices and Android devices will load in.
* The 114 x 114 pixel icon is for the high-res Retina displays.
* The 72 x 72 pixel icon is for iPads.

The icons become available when you save pages as web-clips to the home screen like so:



img: 4224OS-02-19-touchicons-live.png

Pop Quiz: A few questions about Chapter 2

1. What three things should you take into consideration when planning your theme?
2. What are the main steps involved with rapid design comping?
3. What does sketching your theme design accomplish?
4. Why separate out text and layout into CSS first and from the graphics created in an image editor/drawing tool?
5. How much sample text should you start with?

Answers:

1. 1)What type of site or blog your theme will be for. 2) How many layouts or “views” your theme will have and 3) what plugins or widgets will it support.
2. 1)Sketch it. 2) Select Typography. 3) Set up Layout. 4) Apply Color. 5) Create Graphics.
3. It allows you to quickly see your design and start considering usability.
4. Because it's easier to manage web typography and layout with CSS, a standard made to handle those elements well and let graphic editing tools do what they do best: make great graphics.
5. Lots. The more, the better. Less text always looks better, it's hard to make more look good after the fact.

# Summary

You have now learned the key theme design considerations to make when planning a WordPress theme. We've walked through the basics of creating a functional mockup of our theme design in the browser, with the following features:

* Use of HTML5 semantic elements
* CSS3 for gradients, shadows etc. to save on loading images
* Font styling and sizing
* A color scheme
* Some graphic image treatments using background images
* A favicon and Touch icon.

Now that we can see and even get a sense of the user experience of our mockup, let's dive right in to coding it up into a fully working WordPress theme!