



Community Experience Distilled

HTML5 Boilerplate Web Development

Master Web Development with a robust set of templates to get your projects done quickly and effectively

Divya Manian

www.it-ebooks.info

[PACKT] open source*
PUBLISHING community experience distilled

HTML5 Boilerplate Web Development

Master Web Development with a robust set of templates to get your projects done quickly and effectively

Divya Manian

[PACKT] open source 
PUBLISHING community experience distilled

BIRMINGHAM - MUMBAI

HTML5 Boilerplate Web Development

Copyright © 2012 Packt Publishing

All rights reserved. No part of this book may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, without the prior written permission of the publisher, except in the case of brief quotations embedded in critical articles or reviews.

Every effort has been made in the preparation of this book to ensure the accuracy of the information presented. However, the information contained in this book is sold without warranty, either express or implied. Neither the author, nor Packt Publishing, and its dealers and distributors will be held liable for any damages caused or alleged to be caused directly or indirectly by this book.

Packt Publishing has endeavored to provide trademark information about all of the companies and products mentioned in this book by the appropriate use of capitals. However, Packt Publishing cannot guarantee the accuracy of this information.

First published: November 2012

Production Reference: 1091112

Published by Packt Publishing Ltd.
Livery Place
35 Livery Street
Birmingham B3 2PB, UK..

ISBN 978-1-84951-850-5

www.packtpub.com

Cover Image by Neha Rajappan (neha.rajappan1@gmail.com)

Credits

Author

Divya Manian

Project Coordinator

Joel Goveya

Reviewers

Chad Darby

Melanie Archer

Miriam Salzer

Proofreaders

Aaron Nash

Maria Gould

Acquisition Editor

Joanna Finchen

Indexer

Hemangini Bari

Lead Technical Editor

Arun Nadar

Production Coordinators

Manu Joseph

Conidon Miranda

Technical Editor

Dominic Pereira

Cover Work

Manu Joseph

Copy Editor

Laxmi Subramanian

About the Author

Divya Manian is the co-creator of the HTML5 Boilerplate framework. She has worked on projects to benefit the web development community such as HTML5 Please, Move the Web Forward, and HTML5 Readiness. She is also a member of the W3C. Previously, she used to be an embedded C++ programmer.

I would like to thank Nicolas Gallagher, the lead developer and maintainer of HTML5 Boilerplate for all the work in keeping the project up-to-date, and Paul Irish, co-creator of HTML5 Boilerplate for the initial effort and collaboration in bringing this framework alive.

About the Reviewers

Chád Darby is an author, instructor and speaker in the Java development world. As a recognized authority on Java applications and architectures, he has presented technical sessions at software development conferences worldwide. In his 15 years as a professional software architect, he's had the opportunity to work for Blue Cross/Blue Shield, Merck, Boeing, Northrop Grumman, and a handful of startup companies.

Chád is a contributing author to several Java books, including *Professional Java E-Commerce*, Wrox Press; *Beginning Java Networking*, Wrox Press; and *XML and Web Services Unleashed*, Sams Publishing. Chád has Java certifications from Sun Microsystems and IBM. He holds a B.S. degree in Computer Science from Carnegie Mellon University.

You can read Chád's blog at <http://www.luv2code.com/> and follow him on his Twitter handle at @darbyluvs2code.

Melanie Archer is a front-end web developer living in Oakland, California, USA. Since handcoding her first web page in 1997, she's worked with design agencies and startups to bring standards-compliant delight to dozens of user interfaces.

Miriam Salzer has a background in studio art and design, but became hooked on creating websites. She is the owner of Salzer Design, which primarily designs and builds websites for visual and performing artists and for non-profit organizations. Since 2006, Miriam has worked as a software engineer for companies on products as diverse as blogging and medical applications. She lives in the San Francisco Bay Area with her family.

www.PacktPub.com

Support files, eBooks, discount offers and more

You might want to visit www.PacktPub.com for support files and downloads related to your book.

Did you know that Packt offers eBook versions of every book published, with PDF and ePub files available? You can upgrade to the eBook version at www.PacktPub.com and as a print book customer, you are entitled to a discount on the eBook copy. Get in touch with us at service@packtpub.com for more details.

At www.PacktPub.com, you can also read a collection of free technical articles, sign up for a range of free newsletters and receive exclusive discounts and offers on Packt books and eBooks.



<http://PacktLib.PacktPub.com>

Do you need instant solutions to your IT questions? PacktLib is Packt's online digital book library. Here, you can access, read and search across Packt's entire library of books.

Why Subscribe?

- Fully searchable across every book published by Packt
- Copy and paste, print and bookmark content
- On demand and accessible via web browser

Free Access for Packt account holders

If you have an account with Packt at www.PacktPub.com, you can use this to access PacktLib today and view nine entirely free books. Simply use your login credentials for immediate access.

Table of Contents

| | |
|---|-----------|
| Preface | 1 |
| Chapter 1: Before We Begin | 7 |
| Features of HTML5 Boilerplate | 7 |
| Cross-browser compatibility | 8 |
| Doctype | 8 |
| Normalize.css | 8 |
| Clearfix | 8 |
| Search box styling | 8 |
| Conditional classes | 9 |
| Modernizr | 9 |
| No console.log errors | 9 |
| Helper classes | 9 |
| Performance optimizations | 9 |
| Progressive enhancement | 10 |
| Accessible focus styles | 10 |
| Print styles | 10 |
| Tools to start with | 10 |
| Beware | 11 |
| Where to get files | 11 |
| An overview of H5BP files | 12 |
| Asking for help | 13 |
| Summary | 14 |
| Chapter 2: Starting Your Project | 15 |
| Creating your initial project folder | 15 |
| Downloading the latest version of HTML5 Boilerplate | 15 |
| Using the shell script | 16 |
| Creating our project | 17 |
| House-keeping | 18 |
| Setting the tags | 18 |

| | |
|---|-----------|
| Editing favicons | 19 |
| Adding third-party libraries | 22 |
| Using a Content Delivery Network | 22 |
| Adding Google Analytics ID | 25 |
| Updating humans.txt | 25 |
| Summary | 26 |
| Chapter 3: Creating Your Site | 27 |
| <hr/> | |
| Working on the markup | 27 |
| Creating the markup | 28 |
| Deciding which element to use | 30 |
| Writing valid markup | 30 |
| Creating the styles | 30 |
| Helpful style classes we can use | 32 |
| Image replacement | 33 |
| Hiding elements | 33 |
| Hiding elements visually | 35 |
| Hiding elements without impacting layout | 36 |
| Clearing floats | 39 |
| Writing valid stylesheets | 42 |
| Style languages to write productive stylesheets | 42 |
| Advantages | 43 |
| Disadvantages | 43 |
| Where to learn? | 44 |
| Using HTML5 Boilerplate with style languages | 44 |
| Summary | 45 |
| Chapter 4: Adding Interactivity and Completing Your Site | 47 |
| <hr/> | |
| Using jQuery | 47 |
| Using other libraries | 47 |
| Adding smooth-scroll plugin and interaction | 48 |
| Adding HTML5 features safely with Modernizr | 52 |
| When to use Modernizr.load | 55 |
| Using Modernizr to load CSS features | 55 |
| Testing our site | 56 |
| Testing on non-desktop browsers | 64 |
| Summary | 66 |
| Chapter 5: Customizing the Apache Server | 67 |
| <hr/> | |
| Server-side configurations | 67 |
| Setting up the Apache server | 67 |
| Installing Apache | 68 |
| Mac | 68 |
| Windows | 68 |
| Linux | 69 |
| Configuring Apache | 70 |

| | |
|--|-----------|
| Features available out of the box | 71 |
| Removing ETags | 71 |
| Gzip components | 72 |
| Using Expires header for better cache control | 74 |
| Custom 404 page | 76 |
| Forcing the latest IE version | 77 |
| Using UTF-8 encoding | 78 |
| Serving the right MIME types | 78 |
| Blocking access to hidden folders | 79 |
| Blocking access to backup and source files | 79 |
| Starting Rewrite engine | 80 |
| Preventing 404 errors for non-existing redirected folders | 80 |
| Additional customizations | 80 |
| Suppressing or forcing the "www." at the beginning of URLs | 80 |
| Setting cookies from iFrames | 82 |
| PHP security defaults | 83 |
| Stop advertising Apache version | 83 |
| Allowing concatenation from within specific JS and CSS files | 84 |
| Stopping screen flicker in IE on CSS rollovers | 86 |
| Preventing SSL certificate warnings | 86 |
| Cross-domain policies you should be aware of | 87 |
| Cross-domain AJAX requests | 88 |
| CORS-enabled images | 89 |
| Webfont access | 89 |
| Using other server configuration files | 90 |
| web.config | 91 |
| lighttpd.conf | 91 |
| nginx.conf | 91 |
| node.js | 91 |
| Google App Engine | 92 |
| Summary | 93 |
| Chapter 6: Making Your Site Better | 95 |
| Finding the best experience for Internet Explorer | 95 |
| Mobile-first styles for IE | 95 |
| ie.scss | 96 |
| main.scss | 96 |
| Printing with jQuery in IE6 and IE7 | 97 |
| Styling disabled form elements in Internet Explorer | 98 |
| Suppressing IE6 image toolbar | 99 |
| Writing CSS3 easier with tools | 99 |
| Sass | 100 |

| | |
|---|------------|
| Less | 100 |
| Output CSS | 100 |
| Converting HTML5 Boilerplate CSS to Sass or Less | 101 |
| HTML5 Boilerplate Compass extension | 101 |
| HTML5 Boilerplate Sass fork | 101 |
| Print considerations | 101 |
| Finding and using polyfills | 102 |
| Making your site faster | 102 |
| DNS prefetching | 102 |
| Making your site more visible on search engines | 103 |
| Directing search spiders to your site map | 103 |
| Implementing X-Robots-Tag headers | 104 |
| Trailing slash redirects | 105 |
| Option 1: Rewrite example.com/foo to example.com/foo/ | 105 |
| Option 2: Rewrite example.com/foo/ to example.com/foo | 105 |
| Handling users without JavaScript | 106 |
| Optimizing your images | 108 |
| 8-bit PNGs | 108 |
| Tools for image optimization | 108 |
| ImageAlpha | 108 |
| ImageOptim | 108 |
| Using image sprites | 109 |
| CSS sprites from within Adobe Photoshop | 111 |
| CSS sprites with Compass | 111 |
| SpriteMe | 112 |
| Augmenting Google Analytics | 112 |
| Adding more tracking settings | 112 |
| Anonymize IP addresses | 113 |
| Tracking jQuery AJAX requests in Google Analytics | 113 |
| Tracking JavaScript errors in Google Analytics | 113 |
| Summary | 114 |
| Chapter 7: Automate Deployment With the Build Script | 115 |
| The build script | 115 |
| Ant build script | 116 |
| Node build script | 116 |
| Which build script to use | 117 |
| Using the Ant build script | 117 |
| Installing the build script | 118 |
| Smaller image files | 120 |
| Smaller CSS file | 121 |
| Smaller and fewer JS files | 121 |

| | |
|--|------------|
| No comments in files | 122 |
| Build options | 122 |
| Minifying markup | 122 |
| Preventing image optimization | 122 |
| Using CSSLint | 122 |
| Using JSHint | 123 |
| Setting up the SHA filenames | 123 |
| Using with Drupal or WordPress | 124 |
| Updating build.xml | 124 |
| Setting up the project configuration properties | 124 |
| Setting the JS file delineator | 124 |
| Using the Node build script | 125 |
| Grunt | 125 |
| Installing Node build script | 125 |
| Initializing your project | 126 |
| Using the Node build script with an existing project | 127 |
| Using the Node build script to build your project | 127 |
| Text | 127 |
| Minify | 127 |
| Server | 128 |
| Connect | 129 |
| Using with Drupal or WordPress | 129 |
| Next steps | 130 |
| Summary | 130 |
| Appendix: You Are an Expert, Now What | 131 |
| <hr/> | |
| Writing unit tests for your code | 131 |
| Creating a testing environment | 132 |
| Esoteric defaults you should know about | 135 |
| Meta UTF-8 | 135 |
| The HTML Doctype | 136 |
| The details behind the clearfix solution | 136 |
| What do the print styles do | 138 |
| Print media query | 138 |
| Optimizing colors and backgrounds | 138 |
| Better links | 139 |
| Rendering all code and quotes within one page | 140 |
| Rendering tables better | 141 |
| Rendering images better | 141 |
| Margins on pages | 141 |
| Optimal settings for orphans and widows | 142 |
| Keeping headings with content | 142 |
| What are protocol-relative URLs | 142 |
| Using conditional comments | 143 |
| Browser style hacks | 143 |
| Server-side browser detection | 144 |
| Stylesheets based on conditional comments | 144 |

Table of Contents

| | |
|---|------------|
| Class names based on conditional comments | 145 |
| What is meta x-ua-compatible | 146 |
| Meta tag in your HTML page | 146 |
| HTTP header response from the server | 146 |
| Contribute | 148 |
| Reporting issues | 148 |
| Pull requests | 149 |
| Index | 151 |

Preface

Getting Started with HTML5 Boilerplate will enable you to master setting up new projects with minimal effort and deploy them to production in the most effective manner with the least time spent while also ensuring robust performance. It takes you through a step-by-step process of creating a website and teaches you to take full advantage of the defaults provided within HTML5 Boilerplate, be it styles, mark up, or code, so that you can accomplish your goals with as few cross-browser issues as possible.

What this book covers

Chapter 1, Before We Begin, covers all you need to get set up for your projects to use HTML5 Boilerplate without much effort. We also broadly look at the files that are included as part of this project and how they help you.

Chapter 2, Starting Your Project, covers how to get started with HTML5 Boilerplate with an example, single page website. In this chapter, we look at the basic essentials of configuring the default setup that works for your project.

Chapter 3, Creating Your Site, covers how to customize the styles and the markup of your website along with some tips on how to take advantage of HTML5 Boilerplate's default style options.

Chapter 4, Adding Interactivity and Completing Your Site, will help you discover how to do feature-detection, add some interactivity with JavaScript, and finalize your website implementation.

Chapter 5, Customizing the Server, looks at how you can ensure that your website gets loaded as quickly as possible by using HTML5 Boilerplate's custom configurations for the web servers that host your site.

Chapter 6, Making Your Site Better, looks at the optional features that can also be used to provide a better experience for the users of your site, which would fit well with HTML5 Boilerplate.

Chapter 7, Automate Deployment With the Build Script, helps you to make your sites ready to be deployed live by looking at the Build Script that provides tools to minify CSS, JS, HTML, and images.

Appendix, You Are an Expert, Now What? covers some basics of unit testing and provides additional research information on some of the decisions that were arrived at for the features that HTML5 Boilerplate provides.

What you need for this book

As we will be working on a website, we will need the following basic tools to get our work done:

- A text editor that you are comfortable using with; **SublimeText** is recommended with. If you do not have one yet, please download it from sublimetext.com/.
- Apache Web Server (available from httpd.apache.org) to apply HTML5 Boilerplate's server configurations.
- A browser to verify the rendering of your website on the screen. Chrome is recommended, because its developer tools are available for debugging. Download Chrome from google.com/chrome.
- Git, for making sure software is under version control; download it from git-scm.com.
- You also obviously need HTML5 Boilerplate, which you can download from html5boilerplate.com.

Who this book is for

This book is for all the authors who are familiar with creating web projects using HTML, CSS, and JavaScript. No in-depth knowledge is necessary. Some knowledge of what a web server is and how it can be configured is good to have. Also, you should not be afraid to use the command-line tool (fear not! There are links within the book that should make you less afraid). There are no expectations that you should know about HTML5 Boilerplate, except that you only try it once to see if it works for you.

Conventions



In this book, you will find a number of styles of text that distinguish between different kinds of information. Here are some examples of these styles, and an explanation of their meaning.



Code words in text are shown as follows: "However, `Normalize.css` makes sure that these default styles are consistent across all browsers."

A block of code is set as follows:

```
header h1 {
  background-image: url('/img/heading-banner.png');
  width: 800px;
  height: 300px;
}
```

New terms and **important words** are shown in bold. Words that you see on the screen, in menus or dialog boxes for example, appear in the text like this: "The following screenshot shows how the **Skip Navigation** link is instantly visible when the user switches keyboard focus to it."

 Warnings or important notes appear in a box like this. 

 Tips and tricks appear like this. 

Reader feedback

Feedback from our readers is always welcome. Let us know what you think about this book – what you liked or may have disliked. Reader feedback is important for us to develop titles that you really get the most out of.

To send us general feedback, simply send an e-mail to feedback@packtpub.com, and mention the book title via the subject of your message.

If there is a topic that you have expertise in and you are interested in either writing or contributing to a book, see our author guide on www.packtpub.com/authors.

Customer support

Now that you are the proud owner of a Packt book, we have a number of things to help you to get the most from your purchase.

Downloading the example code

You can download the example code files for all Packt books you have purchased from your account at <http://www.PacktPub.com>. If you purchased this book elsewhere, you can visit <http://www.PacktPub.com/support> and register to have the files e-mailed directly to you.

Errata

Although we have taken every care to ensure the accuracy of our content, mistakes do happen. If you find a mistake in one of our books – maybe a mistake in the text or the code – we would be grateful if you would report this to us. By doing so, you can save other readers from frustration and help us improve subsequent versions of this book. If you find any errata, please report them by visiting <http://www.packtpub.com/support>, selecting your book, clicking on the **errata submission form** link, and entering the details of your errata. Once your errata are verified, your submission will be accepted and the errata will be uploaded on our website, or added to any list of existing errata, under the Errata section of that title. Any existing errata can be viewed by selecting your title from <http://www.packtpub.com/support>.

Piracy

Piracy of copyright material on the Internet is an ongoing problem across all media. At Packt, we take the protection of our copyright and licenses very seriously. If you come across any illegal copies of our works, in any form, on the Internet, please provide us with the location address or website name immediately so that we can pursue a remedy.

Please contact us at copyright@packtpub.com with a link to the suspected pirated material.

We appreciate your help in protecting our authors, and our ability to bring you valuable content.

Questions

You can contact us at questions@packtpub.com if you are having a problem with any aspect of the book, and we will do our best to address it.

1

Before We Begin

How deliriously happy do you get when you begin a new project? Me too! The smell of a fresh new project folder is pretty exciting. Sadly, it soon devolves into a mess of folders, subfolders, and hastily written markup and before you know it, it is launch day and you realize with horror you have a page that is missing some essential metadata (uh those favicons!), some sections are unreadable in some browsers – what? It needs to look good when printed too?

HTML5 Boilerplate was born out of frustration of starting from scratch and missing out the important pieces. Having a checklist is not as useful as starting with a project that already comes with the files that your checklist demands.

HTML5 Boilerplate assembles the best tools for you to get started with your next web development project.

Features of HTML5 Boilerplate


Before we dive deep into the internals of HTML5 Boilerplate, let us look at some of its features that would help you in your next project. HTML5 Boilerplate is available for download from html5boilerplate.com and is licensed under MIT license for use in any free or commercial product. The source code is available on Github's URL, which is github.com/h5bp/html5-boilerplate/.

Cross-browser compatibility

HTML5 Boilerplate comes with a set of files that make it easy to do cross-browser development.


Doctype

The single most significant cause of cross-browser compatibilities is the use of incorrect doctype declarations. By using the HTML5 doctype declaration, you are assured that your browsers will render your site in a standard mode.

 If you are interested in learning more about doctypes, I wrote about it in detail at nimbupani.com/the-truth-about-dotypes.html.


Normalize.css

Browsers apply their default styles on elements whose properties you do not specify. The trouble is, the kind of styles that each browser applies are different. However, `Normalize.css` makes sure that these default styles are consistent across all browsers.

 Nicolas Gallagher writes in detail about the motivation behind `Normalize.css` at necolas.github.com/normalize.css/.

Clearfix

Clearfix has been a popular way of clearing floats. In HTML5 Boilerplate, this has been streamlined to use the micro-clearfix solution, a smaller set of selectors to accomplish the same goal, tested and verified to work on Opera 9 and higher, Safari 4 and higher, IE6 and higher, Firefox 3.5 and higher, and Chrome.

 Nicolas Gallagher, the inventor of the micro-clearfix solution, writes more about the choices behind the declarations used at nicolasgallagher.com/micro-clearfix-hack/.

Search box styling

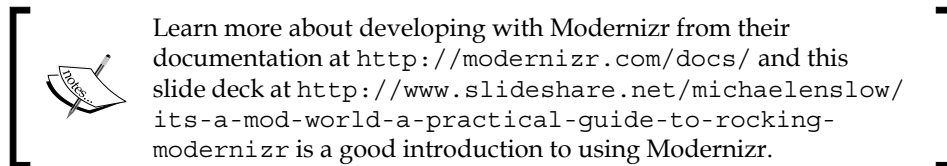
When you set the type of an input element to search, all WebKit browsers like Safari, Chrome, Mobile Safari, and so on, add UI chrome, which is difficult to style. HTML5 Boilerplate comes with a set of styles that normalize the look and feel of the search box across all browsers while also making it easy to style.

Conditional classes

The `index.html` page comes with a set of classes on the HTML element that makes it easy to tweak your styles for IE versions below 9.

Modernizr

Modernizr is a JavaScript library that tests for the existence of HTML5 technologies and outputs a set of classes on the HTML element based on their presence or absence in the browser that is loading your website. For example, if a browser lacks support for border radius, Modernizr outputs the class `no-borderradius`, while on browsers that support border radius, it will output the class `borderradius`. A custom build of Modernizr is included within Boilerplate.



No console.log errors

Oftentimes, when working in modern browsers, you tend to use the `console.log` function to debug your JavaScript code. How many times have you forgotten to remove them or comment them out in production, only to find them throwing errors in Internet Explorer or other browsers that do not support the use of this function? You can safely use the `log` function included within the `plugin.js` file to log statements only in browsers with tools that support it.

Helper classes

Ever had to hide text to show images? How about making extra text available for those who use screen readers or hide from all browsers? HTML5 Boilerplate provides classes for both and more, which have been field-tested to work across edge cases and across all browsers.

Performance optimizations

The `.htaccess` file includes the best out-of-the-box defaults for caching, which makes your pages load significantly faster when they are served by the Apache Web Server. There are also configuration files for other web servers available to provide similar functionality.

Progressive enhancement

The HTML element has a `no-js` class that can be used to provide alternative styles for browsers that do not support JavaScript. With Modernizr, this class name is replaced when used in a browser that does support JavaScript to `js`.

Accessible focus styles

All browsers provide a default focus style for links when clicked. HTML5 Boilerplate ensures that these styles are only applied when the elements are in focus while using keyboard navigation.

Print styles

A good default print stylesheet is something most of us fail to think of when we create web pages. However, HTML5 Boilerplate already does this for you by providing best-performing defaults for print styles.

Tools to start with

You can start using Boilerplate with your favorite editor. If you use Git as your version control system, we also include a `.gitignore` file that would automatically ignore files such as `.DS_STORE` or other unnecessary files from being marked for versioning. Some editors that can be used to work with HTML5 Boilerplate are as follows:

- **Aptana Studio:** HTML5 Boilerplate comes out of the box with Aptana Studio. Choose a Web Project and then select Boilerplate to start with. Robert Gravelle has a write-up explaining how to use HTML5 Boilerplate in your Aptana Studio projects, which can be found at www.htmlgoodies.com/html5/tutorials/aptana-studio-3-guided-tour-and-tutorial-create-a-web-project-using-the-html-5-boilerplate-framework.html.
- **Visual Studio:** There are two templates available for use in Visual Studio 2010. One for Web forms, which is downloadable from h5bpwebapptemplate.codeplex.com/ and the other is downloadable from www.jondavis.net/techblog/post/2011/04/24/HTML5-Boilerplate-Visual-Studio-2010-Template.aspx.

- TextMate: A year-old project, this URL hosts TextMate bundles of HTML5 Boilerplate's markup and styles at www.dontcom.com/post/1546820479/html5-boilerplate-textmate-template-bundles.

Beware

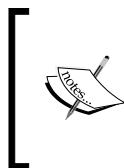
These tools are not officially maintained by the HTML5 Boilerplate project and hence are likely to be out of date. You are best off using the process outlined in the following section.

Where to get files

There are three ways to get HTML5 Boilerplate, which are as follows:

- From the website: The latest stable version of the project is available at html5boilerplate.com.
- From Initializr: Jonathan Verecchia hosts a more expansive set of modules to choose from at initializr.com. All the modules here are from the stable version that is available on the website.
- From the Github home page: HTML5 Boilerplate is hosted on Github. The latest files are available from the project's github page at github.com/h5bp/html5-boilerplate. You are safe to use these files when starting your new project and you are guaranteed to get the latest version of these files when you download from Github.

As you are just getting started with HTML5 Boilerplate, I strongly recommend you to download the files from Github, and even better to do so via Git, so you can easily update them when the master files on Github get updated.



If you are unfamiliar with Git, Roger Dudler maintains a great introduction to get you started at rogerdudler.github.com/git-guide/; if you are new to the concept of version control, there is a good explanation of what it is and why it is useful at hoth.entp.com/output/git_for_designers.html.

An overview of H5BP files

The different files and folders that are a part of HTML5 Boilerplate are explained as follows:

- `index.html`: This is the markup that we recommend you start all your HTML pages with.
- `main.css`: The styles are located in a single stylesheet known as `main.css`, found within the `css` folder.
- `normalize.css`: This file is located separately, so that you can use the latest updated version of `normalize.css` immediately. In production, ideally you should combine both `main.css` and `normalize.css` into a single file to ensure minimum number of network requests, so your pages load quicker.
- `doc`: This folder contains all the documentation necessary to understand the HTML5 Boilerplate files.
- `img`: This folder should contain all the images you will be using to create your website. This is empty to begin with, but you should include all the images you work with here.
- `js`: This is the parent folder for all your scripts. HTML5 Boilerplate comes with a set of scripts that make it easier for you to get started. This folder contains the following files and folders:
 - `vendor`: This folder contains all the script libraries. You get the latest minified and unminified versions of jQuery and a custom build of `modernizr`. Any other libraries you will be using should ideally go within this folder.
 - `plugins.js`: All the jQuery plugins that you would be using should be inlined in this file. If you are using a jQuery carousel plugin, you would copy the code to `plugins.js`.
 - `main.js`: This will be the file from where you would invoke scripts that run on your page. Taking the example of the jQuery carousel plugin, we will invoke the plugin to run on our pages from this file.
- `404.html`: If you have a page that is not found, then this page can be served. Make sure it has all the information available and uses the same look and feel as other pages in your website.
- `humans.txt`: This is a wonderful initiative that allows you to denote who worked on a website (read more about this initiative at humanstxt.org). We highly recommend you use this to indicate your work, and to inform anyone who is curious, whose work it was.

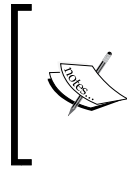
- `crossdomain.xml`: This is useful if you would like to have flash files hosted elsewhere to access assets located on the domain where your website will be hosted. You could have a flash audio player from another domain using the files hosted on your website. In this case, you need to carefully choose your cross-domain policy (we will cover this file in detail in *Chapter 5, Customizing the server*).
- `robots.txt`: Search engines use this file to understand which files to index and which not to index.
- `.htaccess`: This is an Apache server configuration file specific to your website. Loads of best practices are included by default.
- `favicon.ico`: Most browsers use the favicon when you bookmark a page on a website or next to the title of the page on a tab. By using a distinctive identifiable icon, you will be able to make your website stand out and be easy to navigate to.
- `apple-touch-icon-*.png`: iOS and Android devices allow websites to be bookmarked to the homescreen of your phone. Both of them use these touch icons to represent your site when it is added to your home screen. Boilerplate comes with a set of icons to identify all the sizes and formats you need to create your icons in.
- `readme.md`: The readme has all the license information and a list of features and where to get more information from, on using these files.

Asking for help

Now that we have seen what these files are and where to get them from, it's important that you are familiar with how to ask for help and, most importantly, where. Do remember that most of the maintainers of the HTML5 Boilerplate project work on it in their free time. The more you spend time being specific about what you want help with, the faster and better they will be able to help you. Here is how to ask for help:

- Isolate the problem: What is the exact issue? Use dabblet.com, codepen.io, jsfiddle.net, or jsbin.com to create a test case that reproduces the issue with least markup, style, and script. Most of the time the act of doing so will, in itself, have you find what the issue is.
- If you can reproduce this issue and isolate it to a problem arising because of a feature of HTML5 Boilerplate, go to github.com/h5bp/html5boilerplate.com/issues and use the Search field to check if it has already been reported. If not, create a new issue with a link to your test case.

- If this issue is not a result of HTML5 Boilerplate, but an interaction that you can't quite place, go to stackoverflow.com/questions/tagged/html5boilerplate and create a question linking to the isolated test case. Make sure you tag the question as `html5boilerplate` or `h5bp`, so one of the maintainers can catch it and answer quickly.
- If it is a small enough question to be asked on Twitter, tweet at <https://twitter.com/h5bp> with a link to the test case and the specific section you want help with.



Lea Verou has written a great article on submitting browser bug reports at coding.smashingmagazine.com/2011/09/07/help-the-community-report-browser-bugs/ and it is equally useful for asking for help with any open source web development project.

Summary

In this chapter, we have learnt about why HTML5 Boilerplate is a great toolbox for a web developer. In addition, we have seen what features are most useful for your web development projects and what each of the files in HTML5 Boilerplate does. We have also spent some time looking at where to get the files for HTML5 Boilerplate and how to ask for help. In the next chapter, we will get started with a sample project using HTML5 Boilerplate.

2

Starting Your Project

You would like to get up and running as quickly as possible with your projects, and in this chapter we will see some of the quickest ways to do so with HTML5 Boilerplate.

There are many flavors of HTML5 Boilerplate to choose from, and we will look into some of the mechanisms of creating your starting folder and take a look at the immediate tasks you can take care of once you get going.

Creating your initial project folder

HTML5 Boilerplate is available in three versions from the website, as we saw in the previous chapter. Here are two of the quickest ways to get started with the latest files:

- Download the latest version of HTML5 Boilerplate for every new project you start with
- Maintain a local, up-to-date copy of the HTML5 Boilerplate and use a script to copy files into your project

We will look at both these ways now.

Downloading the latest version of HTML5 Boilerplate

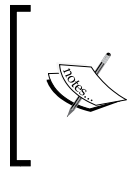
This is the easiest way to get started with the latest files on HTML5 Boilerplate. If you are conversant with Git, you can download HTML5 Boilerplate as a folder. In your command-line interface, navigate to the folder where you store your projects typically, and then enter the following command in your command-line interface:

```
git clone git://github.com/h5bp/html5-boilerplate.git
```

This will download a folder called `html5-boilerplate` to that folder. You can then rename it to your own project and get started with it.

If you are unfamiliar with the command-line interface, you can download the latest files as a ZIP file and unzip it into a folder that you can rename to the project you want to work with.

If you find these options tedious, I recommend you go with the shell script. However, it requires you to have Git set up and be familiar with the command-line interface.



If you are on Windows, be sure to download **Cygwin** at sources.redhat.com/cygwin/cygwin-ug-net/cygwin-ug-net.html and use it for typing all the command lines that I mention. There is also an illustrated guide to setting up and using Git on Windows at nathanj.github.com/gitguide/tour.html.

Using the shell script

Using this script, we will set up a local repository for HTML5 Boilerplate that can be kept up-to-date with the changes that are made in the project.

Go to a folder where you want to keep your copy of the latest HTML5 Boilerplate files that you would like to use as a reference for all your projects. In my case, I would like to keep it in a folder called `source`.

Then, use the same command-line script as mentioned in the previous section to download the latest copy of the files. The script is as follows:

```
git clone git://github.com/h5bp/html5-boilerplate.git
```

Instead of renaming the folder, we will let this folder be as it is. Next, we shall copy the `createproject.sh` shell script to this folder.

In your shell, navigate to the `html5 Boilerplate` folder, and download the `createproject.sh` file as shown in the following command-line script:

```
curl https://raw.githubusercontent.com/h5bp/ant-build-script/master/createproject.sh  
> createproject.sh
```

Make sure it is executable by executing the following in the shell:

```
chmod +x createproject.sh
```

The execution of these command-line scripts are shown in the following screenshot:

```

★ code git clone git://github.com/h5bp/html5-boilerplate.git
Cloning into html5-boilerplate...
remote: Counting objects: 4301, done.
remote: Compressing objects: 100% (1774/1774), done.
remote: Total 4301 (delta 2494), reused 4100 (delta 2414)
Receiving objects: 100% (4301/4301), 8.97 MiB | 2.19 MiB/s, done.
Resolving deltas: 100% (2494/2494), done.
★ code cd html5-boilerplate
★ html5-boilerplate git:(master) curl https://raw.github.com/h5bp/ant-build-script/master/createproject.sh > createproject.sh
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload  Total  Spent  Left  Speed
100 1785 100 1785    0     0  1865    0 --:--:-- --:--:-- --:--:-- 3863
★ html5-boilerplate git:(master) ✕ chmod +x createproject.sh
★ html5-boilerplate git:(master) ✕ █

```

Then execute the following script from the command line:

```
./createproject.sh <project-name>
```

This will create a folder titled with the project name in the parent folder of the `html5-boilerplate` folder. If you want your project files to be located elsewhere, you can also use the absolute path to the project folder, as shown in the following script:

```
./createproject.sh /Users/divya/projects/<project-name>
```

Creating our project

Throughout this book, we will be working on an example project to understand how to use HTML5 Boilerplate. All our project source files are available at `nimbu.in/h5bp-book/sun-shine-festival-2012/`.

Let us pretend that we need to create a website for a hypothetical sun and sand festival in Ngor and Terou Bi, Dakar, Senegal, November 12, 2012 to November 16, 2012. I would like to label this project as `sun-sand-festival-2012`.

I store all my projects in a `projects` folder, and all my frameworks and starter kits in a `source` folder.

In my `source` folder, I have the `html5-boilerplate` folder that I initially created with the following script:

```
git clone git://github.com/h5bp/html5-boilerplate.git
```

I keep it up-to-date regularly by pulling the latest changes in the master repository hosted on Github, using the following script:

```
git pull origin master
```

I also have the `createproject.sh` shell script in the same folder, which we will use to create our new project. In the shell interface, I navigate to the `html5-boilerplate` folder and enter the following script:


```
./createproject.sh ../projects/sun-sand-festival-2012
```

This creates the project folder with all the required files to get started. The files that are created are shown in the following screenshot:

```
★ sun-sand-festival-2012 ls
404.html
apple-touch-icon-114x114-precomposed.png
apple-touch-icon-57x57-precomposed.png
apple-touch-icon-72x72-precomposed.png
apple-touch-icon-precomposed.png
apple-touch-icon.png
crossdomain.xml
css
favicon.ico
humans.txt
img
index.html
js
robots.txt
★ sun-sand-festival-2012
```

House-keeping

Now we have our project ready, let us get started with some basic housekeeping that we would need to do with any project. Open the project in any text editor you are comfortable using.

 I highly recommend the cross-platform **Sublime Text** text editor, downloadable from www.sublimetext.com, if you are looking for a good text editor. The command-line masters might want to try using the **Vim** text editor, downloadable from www.vim.org.

Setting the tags

Our `index.html` page contains a few tags that we need to fill in:

```
<title></title>
<meta name="description" content="">
```

For the title tag for our project, let us enter the following:

```
<title>Home | Sun and Sand Festival 2012, Dakar</title>
```

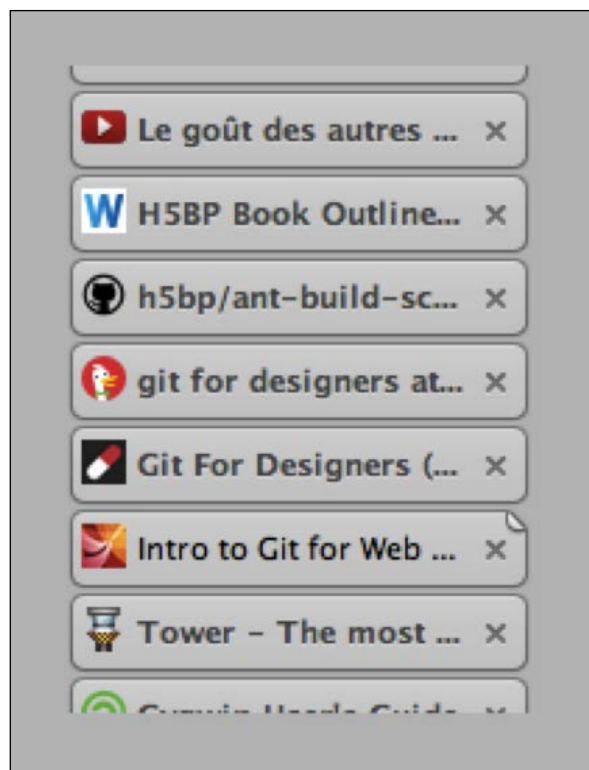
The meta tag with a name description is useful when the site is listed in search engine results. This tag would be used to render the snippet of text explaining what this page is about. Let us set this to the following:

```
<meta name="description" content="Sun and Sand Festival is occurring  
between Nov 12 to Nov 16 2012 at the Ngor and Terou Bi, Dakar  
featuring performances by top Senegal artists">
```

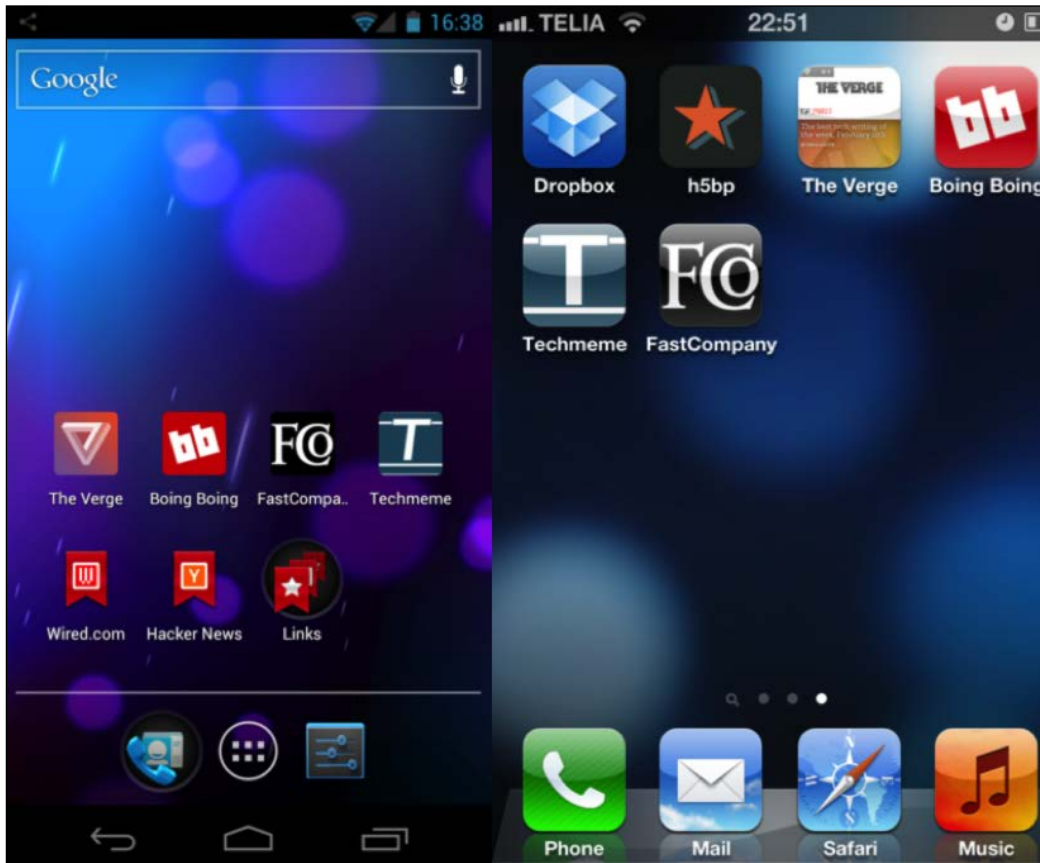
Editing favicons

Adding favicons would be the next trivial thing that most of us forget to do when we start a project. This is the next easy goal that you can reach, before you need to start thinking about the code you will be creating.

Favicons help in uniquely identifying your website. As the following screenshot shows, having a favicon makes it easy to tell which tab or bookmark you want to visit:



Touch icons are useful when your page gets added to the home screen on iOS (as shown in the following screenshot on the right-hand side) and Android devices (as shown in the following screenshot on the left-hand side):

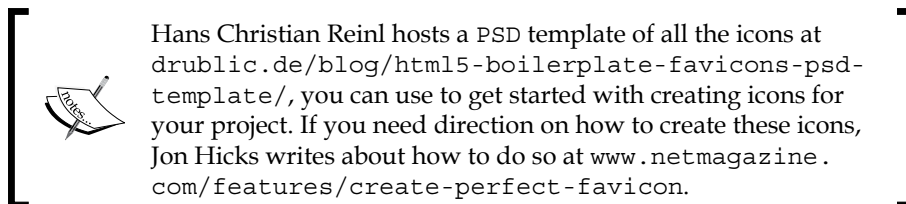


HTML5 Boilerplate comes with a set of icons in the root folder that are of the right sizes for all the required icons for both touch screen icons (used by both Android and iOS devices) and favicons. You can use them as a guide when you are working on your icons.

HTML5 Boilerplate comes with the following set of icons:

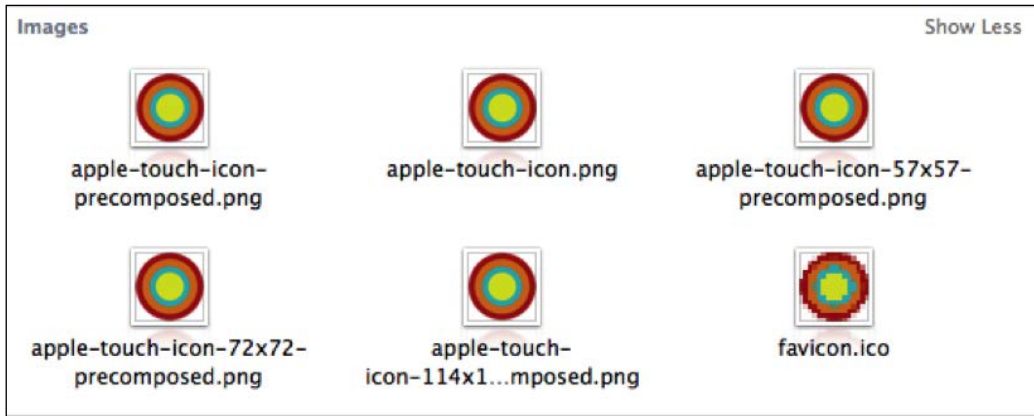
- **favicon.ico**: The default icon used by desktop browsers to render the icons on tabs or next to the title.
- **apple-touch-icon.png**: If nothing else is specified, iOS will use this icon to render on the home screen. Unfortunately, this also means iOS will add its own effects such as drop-shadow, rounded corners, and reflective shine on top of this icon. This is also a good fallback icon format if nothing else is supported, for example, iOS 1 and BlackBerry OS 6.
- **apple-touch-icon-precomposed.png**: This prevents iOS from applying any kind of effects on top of your icon and have it be presented as it is. Providing this icon will also ensure that Android 2.1 and above devices will use this as the icon when your web page is added to the home screen.
- **apple-touch-icon-57x57-precomposed.png**: This will be used by iOS devices that do not have Retina display.
- **apple-touch-icon-72x72-precomposed.png**: This will be used by iPad, which does not have a high-resolution display.
- **apple-touch-icons-114x114-precomposed.png**: This will be used by high-resolution iPhone Retina displays.
- **apple-touch-icons-144x144-precomposed.png**: This will be used by high-resolution iPad Retina displays.

The rationale for why we have so many icons has been documented by Mathias Bynens at <http://mathiasbynens.be/notes/touch-icons>.



If you have the graphic elements necessary for creating icons, you can get started with adding these icons to the root folder of the project. It is likely that you would forget to do it later when deadlines loom.

For our sun and sand festival example, we already have critical graphic elements assembled, the following screenshot shows the icons generated from the PSD template:



Adding third-party libraries

If you already have a list of libraries that you will be using, you can start adding them into the folder.

HTML5 Boilerplate comes with the latest stable version of jQuery, so you already have that. If you are inclined to use other libraries such as jQuery UI, you can copy them over to the `libs` folder.

Suppose you would like to use jQuery UI for your project, available at www.jqueryui.com, then copy the latest version of jQuery UI to the `libs` folder and then at the bottom of the markup in `index.html`, refer to it using the `script` tag.

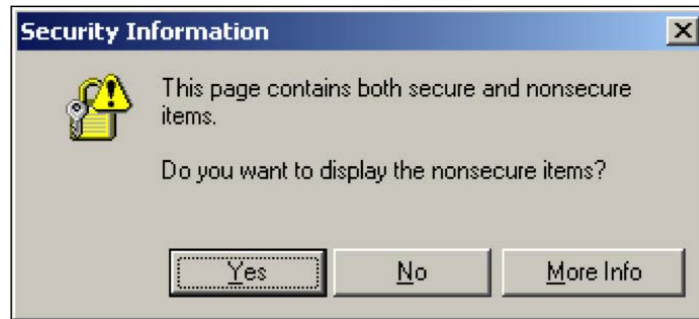
Using a Content Delivery Network

By using a **Content Delivery Network (CDN)**, we can reduce the number of resources to serve on our web servers and by referring to resources that are universally hosted by Google or Microsoft, it is more likely that the file will be cached, as a lot of other sites the user visits will also be referencing this particular resource.

If you paid close attention, you would have noticed that the source for the script that links to jQuery is different from our jQuery UI source. This is for two reasons, which are explained in the following sections.

Protocol-relative URLs

Typically most URLs that link to assets on the Web start with `http://`. However, there are occasions when the page is hosted on a server that uses encrypted communication. So, your page will be served with `https://` instead of the typical `http://`. Alas, as your script source is still referenced with the `http://` protocol, IE will throw a nasty dialog asking the following question to the visitors on your page:



You definitely do not want your visitors panicking over this. So, the easiest way to prevent this is to remove the protocol (`http:`) part of the URL completely, as follows:

```
//ajax.googleapis.com/ajax/libs/jquery/1.7.1/jquery.min.js
```

This way, the browser will use whatever protocol the page has been served with for the request. You can learn more about protocol-relative URLs in the *Appendix* section.


Of course, this means if you are testing locally, and if you view your page on the browser, the browser will use a URL that looks like `file://users/divya/projects`, and hence the browser will attempt to look for a jQuery file using the following URL:

```
file://ajax.googleapis.com/ajax/libs/jquery/1.7.1/jquery.min.js
```

This request will definitely fail, as there is no resource locally at that path. Hence, if you are using protocol-relative URLs, you need to set up a local server to test your files. This is easily done on a Mac or Unix-based OS by navigating to your project folder in your shell interface and executing the following command:

```
python -m SimpleHTTPServer
```

This will start a server and your project's `index.html` file will be available on `http://localhost:8000`.

 If you are on Windows, copy the Mongoose executable (the latest version at the time of writing was `mongoose-3.3.exe`) from `code.google.com/p/mongoose/` to your project folder and launch it. Your project's `index.html` will then be available at `http://localhost:8080`.

Google CDN hosting

Google hosts a lot of popular JavaScript libraries. A list of all libraries hosted on Google's CDN is available at `code.google.com/apis/libraries/devguide.html`.

We could take advantage of Google's CDN for jQuery UI too, as it is hosted on it. Let us convert it to use Google's CDN by changing the source of the script file from `js/libs/jqueryui-jquery-ui-1.8.17.min.js` to the following:

```
//ajax.googleapis.com/ajax/libs/jqueryui/1.8.16/jquery-ui.min.js
```

But wait! Let us take a look at how we refer to jQuery CDN in HTML5 Boilerplate. This is shown in the following code snippet:

```
<script src="//ajax.googleapis.com/ajax/libs/jquery/1.8.2/jquery.min.js"></script>
<script>window.jQuery || document.write('<script src="js/vendor/jquery-1.8.2.min.js"></script>')
</script>
```

Do you notice how we also refer to a local copy of the jQuery file? We do this just so that in the event Google's CDN fails, we still have our local copy to use. Granted this does not happen often, but it is useful to have a fallback when or if it does.

The statement `window.jQuery || document.write(...)` does two things. These are as follows:

- Check if the jQuery object exists: If it does, it means Google's CDN worked. If it exists, do nothing.
- If the `window.jQuery` object does not exist: This means Google's CDN failed; it immediately renders a `script` tag with a reference to the copy of jQuery in the project's `libs` folder. This tells the browser to immediately make a request for that resource.

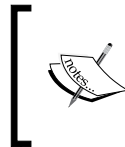
We can do something similar for jQuery UI.

All jQuery plugins are objects within the jQuery object. So, we only need to verify whether the plugin object exists and if it doesn't, load the copy of the plugin in the `libs` folder, using the following code snippet:

```
<script>window.jQuery.ui || document.write('<script src="js/libs/
jqueryui-jquery-ui-1.8.17.min.js"></script>')
</script>
```

Hence, our complete script file for referencing jQuery UI would be as shown in the following code snippet:

```
<script src="//ajax.googleapis.com/ajax/libs/jqueryui/1.8.16/jquery-
ui.min.js "></script>
<script>window.jQuery.ui || document.write('<script src="js/libs/
jqueryui-jquery-ui-1.8.16.custom.min.js"></script>')
</script>
```



There are other CDNs that host libraries too. The `cdnjs.com` URL hosts a lot of JavaScript libraries. Microsoft also hosts a few libraries on its CDN; the list is available at www.asp.net/ajaxlibrary/cdn.ashx.

Adding Google Analytics ID

This is another of those minor actions that gets forgotten when deadlines come calling. HTML5 Boilerplate already provides the ready snippet for you to use. All you need to include is the unique identifier for your website.

Note that HTML5 Boilerplate includes the snippet at the footer of the page, which means the metrics get sent only after the page is loaded. However, there are a few people who believe Analytics should occur even before the page gets loaded to measure who leaves the page even before it has completed loading. If you would like to do that, you should move the Analytics snippet to just above the closing `</head>` tag in the `index.html` page.

Updating `humans.txt`

`humans.txt` makes known the people who have worked on a website. Anyone can simply visit `example.com/humans.txt` to immediately know the names of people who have worked on that website. Add your name and those of your team members to the `humans.txt` file that comes within HTML5 Boilerplate.

For our sun and sand festival example, the following screenshot shows how our `humans.txt` will look:

```
1 /* the humans responsible & colophon */
2 /* humanstxt.org */
3
4
5 /* TEAM */
6   Web Developer: Divya Manian
7   Site: http://nimbupani.com
8   Twitter: @divya
9   Location: Seattle
10
11 /* THANKS */
12   Names (& URL):
13
14 /* SITE */
15   Standards: HTML5, CSS3
16   Components: Modernizr, jQuery
17
```

Summary

In this chapter, we looked at how to get started on a project with HTML5 Boilerplate and the first steps we should take in our new project. On the way, we learned about protocol-relative URLs and linking to libraries hosted on CDNs. We updated the `humans.txt` file and the icons to be used in our project. All the changes that we have made so far to our example project are available at nimbu.in/h5bp-book/chapter-2/. In the next chapter, we will look at writing some code for our project.

3

Creating Your Site

Now that we have done all the basic housekeeping with respect to our project, let us look at the actual task of building this site. We will first start with the markup, jump into the stylesheets, and finally add interactivity with scripts.

Working on the markup

We have a simple design in mind for our Sun and Sand festival project. The design is shown in the following screenshot:



Looking at how it is organized, the broad structure of the page is explained as follows:

- **Header:** A banner logo with a set of navigation links
- **Main content:** The meat of the page with sections that the navigation links will link to
- **Left column:** This contains the main content
- **Right column:** This contains the secondary information that would be interesting to the viewers, but not essential
- **Footer:** Sponsor logos and an audio player with music of artists, who will be participating in the festival

Creating the markup

The **HTML5 Doctor** has a list of all elements that you can use in a web page at html5doctor.com/element-index/. Comparing this to the list we made earlier, it looks like the `header` tag would be good to park our logo and navigation links in, while the sponsor logos and audio player can go inside the `footer` tag. That leaves us with the main content; it seems like the `div` tag with the role of `main` would be the best fit for it!

Here is the markup we end up with. The `index.html` page in *Chapter 2, Starting Your Project*, contains the following code as well:

```
<header>
  <a href="#main">Skip Navigation</a>

  <h1>Sun & Sand Festival 2012</h1>
  <h2>Ngor& Terou Bi, Dakar</h2>
  <nav class="site-nav">
    <a href="#tickets">Tickets</a>
    <a href="#about">About</a>
    <a href="#line-up">Line-up</a>
    <a href="#contact">Contact</a>
    <a href="#gettinghere">Getting Here</a>
  </nav>
</header>
<div role="main">
  <section id="primary">
    <article id="tickets">

    </article>
    <article id="about">
```

```
</article>
<article id="lineup">

</article>
<article id="contact">

</article>
<article id="gettinghere">
</article>
</section>

<aside id="secondary">
  <article>
    <h2>Get some sun!</h2>
    <ul>
      <li>Follow us on <a href="http://twitter.com/sunnsand">twitter</a>!</li>
      <li>Stalk us on <a href="http://facebook.com">facebook</a>!</li>
      <li>Get some sun through <a href="http://flickr.com/photos/sunnsand">flickr</a>!</li>
    </ul>
  </article>
</aside>
</div>
<footer>
  <article class="sponsors">
    <a href="#">Boca-Cola</a>
    <a href="#">Darbucks</a>
    <a href="#">Kugle</a>
    <a href="#">Pling</a>
  </article>
  <audio src="audio.webm" controls></audio>
</footer>
```



You can download the example code files for all Packt books you have purchased from your account at <http://www.PacktPub.com>. If you purchased this book elsewhere, you can visit <http://www.PacktPub.com/support> and register to have the files e-mailed directly to you.

Deciding which element to use

With HTML5, we have a glut of elements to choose from, causing choice paralysis for some of us. If there is anything in the structure of your document that seems to very obviously fit any of the new elements, go ahead and mark them so. If they don't, continue using `div` or any other element that seems to obviously fit.

In our code, we use the `section` element when we have primary content that is structurally different and the `article` element for which we have similar, but repeating sections of content. Your views on these choices may well be different; in which case, I recommend you choose what you are comfortable with.

If you wish to know more about the new HTML5 elements, I recommend you check out the chapter on semantics in *HTML5: Up & Running*, Mark Pilgrim, O'Reilly, under the Google Press imprint, at diveintohtml5.info/semantics.html.

Writing valid markup

Writing valid markup ensures your page behaves consistently across all browsers that render it. Valid markup refers to markup that adheres to the Web standards that browsers comply with. This way, you will prevent any unpredictable behavior.

The easiest way to write valid markup is to use tools that validate it instantly as and when you save your file.

In *Chapter 2, Starting Your Project*, I recommended using Sublime Text and Vim for Web development. Both of these tools have inline validation that you can use to write valid markup. Moreover, these tools also provide autocompletion of tags and elements that make writing valid markup trivial.

In the event of you not having access to these tools, I recommend using validator.w3.org/ to validate your markup.

It is essential to have these tools automated for you to make sure you reduce any issues with your site to the absolute minimum.

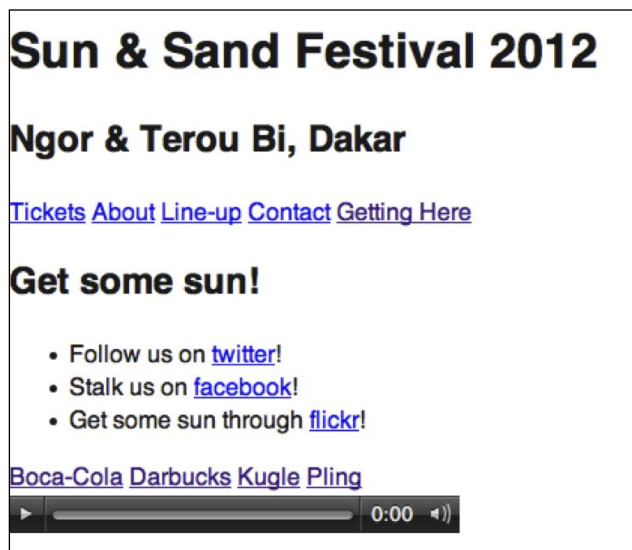
Creating the styles

Now that we have the markup ready, let us look at how we should be styling it. HTML5 Boilerplate comes with a stylesheet that has the best default styles. If you open `main.css`, you will find the following section in between the `Chrome Frame` prompt style rules and the `Helper classes` section:

```
/* =====  
Author's custom styles  
===== */
```

This is where we will compose our style rules. Later on, we will look at how you can use some of the style frameworks to make writing this easier with **Sass** (<http://sass-lang.com>) or **Less** (<http://lesscss.org/>).

Without even writing a line of CSS, you will note that our page looks like the website displayed in the following screenshot:



This default style is thanks to the normalize style rules that are available in HTML5 Boilerplate.

Why not reset.css?

For a very long time, the recommendation was to use `reset.css`, which is available at html5doctor.com/html-5-reset-stylesheet/ and resets the margin and padding of every available element to 0, in addition to making the font size of all headings the same as the body text and without a higher font weight.

HTML5 Boilerplate recommends against this policy. Browsers provide useful browser defaults, which would make your stylesheets smaller as you don't have to redeclare those styles again.

With `normalize.css`, you wouldn't be seeing the following kind of clutter in your debugging tools:

```
Inherited from body#-page-home.home.layout-
.font-sans-serif, body, input[type="text"], .home #press-container app1511104178.css:1
article time, section.posts time, .policy #container table, .blog-post-snippet .blog-
author, .blog-post-snippet .blog-date, footer#footer dd, footer#footer #copyright {
  font-family: Helvetica, "Helvetica Neue", "HelveticaNeue", Arial, sans-serif;
}

media="screen and (min-width: 481px)" /#
a, abbr, acronym, address, applet, article, aside, audio, b, big, 9603905441074.css:1
blockquote, body, canvas, caption, center, cite, code, dd, del, details, dfn, dialog,
div, dl, dt, em, embed, fieldset, figcaption, figure, font, footer, form, h1, h2, h3, h4,
h5, h6, header, hgroup, hr, html, i, iframe, img, ins, kbd, label, legend, li, mark,
menu, meter, nav, object, ol, output, p, pre, progress, q, rp, rt, ruby, s, samp,
section, small, span, strike, strong, sub, summary, sup, table, tbody, td, tfoot, th,
thead, time, tr, tt, u, ul, var, video, xmp {
  font-size: 100%;
}

Inherited from html.no-js.wf-museoslab1museoslab2-n5-active.wf-museoslab1museoslab2-n7-active.wf-proxi... /#
media="screen and (min-width: 481px)" /#
a, abbr, acronym, address, applet, article, aside, audio, b, big, 9603905441074.css:1
blockquote, body, canvas, caption, center, cite, code, dd, del, details, dfn, dialog,
div, dl, dt, em, embed, fieldset, figcaption, figure, font, footer, form, h1, h2, h3, h4,
h5, h6, header, hgroup, hr, html, i, iframe, img, ins, kbd, label, legend, li, mark,
menu, meter, nav, object, ol, output, p, pre, progress, q, rp, rt, ruby, s, samp,
section, small, span, strike, strong, sub, summary, sup, table, tbody, td, tfoot, th,
thead, time, tr, tt, u, ul, var, video, xmp {
  font-size: 100%;
}
```

Nicolas Gallagher, one of the co-creators of `normalize.css`, has written in great detail about why it is better than `reset.css` at nicolasgallagher.com/about-normalize-css/, which is a good read for those still unconvinced about the merits of normalizing CSS.

Helpful style classes we can use

In *Chapter 1, Before We Begin*, we briefly saw that HTML5 Boilerplate comes with a bunch of default classes that are useful to work with. You would have noticed that we are using some of these classes in our style rules.

All our helper classes are defined last, so they can override all your other styles when used. Make sure the properties they override are not over-specified elsewhere; you can read more about specificity at www.w3.org/community/webed/wiki/Inheritance_and_cascade#Specificity.

Image replacement

In our project, we want to have a spiffy logo for the Sun & Sand Festival 2012 heading. HTML5 Boilerplate has a handy image replacement class that can be used for this. In the markup, we will simply add a class called `ir` to the `h1` tag, as shown in the following code:

```
<h1 class="ir">Sun & Sand Festival 2012</h1>
```

What this does is apply the styles specified in HTML5 Boilerplate's image replacement class (`ir`) to hide the text. All you need to do then is add a background image to the `h1` element along with its width and height, so it displays as per your specification as shown in the following code:

```
header h1 {
  background-image: url('/img/heading-banner.png');
  width: 800px;
  height: 300px;
}
```

This will result in the heading looking similar to the following screenshot:



Hiding elements

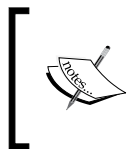
Our markup has content, which we want to show only when a user clicks. In our website, we want a Google Map to show, when the user clicks on the **Getting Here** link. It is very simple to do so by using an `iframe`, as shown in the following code snippet:

```
<iframe width="425" height="350" frameborder="0" scrolling="no"
marginheight="0" marginwidth="0" src="http://maps.google.com/maps?f=q
&source=s_q&hl=en&geocode=&q=ngor+terrou+bi,+dakar,+
senegal&aq=&sll=37.0625,-95.677068&sspn=90.404249,95.9765-
62&ie=UTF8&hq=ngor&hnear=Terrou-Bi,+Bd+Martin+Luther+Kin
g,+Gueule+Tapee,+Dakar+Region,+Guediawaye,+Dakar+221,+Senegal&t=
m&fll=14.751996,-17.513559&fspn=0.014276,0.011716&st=10
9146043351405611748&rq=1&ev=p&split=1&ll=14.711109,-
17.483921&spn=0.014276,0.011716&output=embed">
</iframe>
```

But this means, as soon as your page loads in a browser, the browser will attempt to show the map immediately and fetch assets from Google Maps. But we only want this map to show when the user clicks on the **Getting Here** link. HTML5 Boilerplate provides a class name that you can use for such purposes. We will apply a class called `hidden` to make sure these elements do not render until they are explicitly made to display. The `hidden` class is used in the following code snippet:

```
<iframe class="hidden" width="425" height="350" frameborder="0"
scrolling="no" marginheight="0" marginwidth="0" src="http://maps.
google.com/maps?f=q&source=s_q&hl=en&geocode=&q=ngor
+terrou+bi,+dakar,+senegal&aq=&sll=37.0625,-95.677068&ssp
pn=90.404249,95.976562&ie=UTF8&hq=ngor&hnear=Terrou-Bi,+Bd
+Martin+Luther+King,+Gueule+Tapee,+Dakar+Region,+Guediawaye,+Dakar+22
1,+Senegal&t=m&fll=14.751996,-17.513559&fspd=0.014276,0.0
11716&st=109146043351405611748&rq=1&ev=p&split=1&
ll=14.711109,-17.483921&spn=0.014276,0.011716&output=embed">
</iframe>
```

Do note that this makes the content disappear from screen readers and the browser displays.



Screen readers are devices used to aid in reading a web page for those who are unable to view text on the screen. Victor Tsaran has a great introduction to screen readers in a video available at www.yuiblog.com/blog/2007/05/14/video-intro-to-screenreaders/.

The rule that makes this happen is as follows:

```
.hidden {
display: none !important;
visibility: hidden;
}
```

This ensures all screen readers (**JAWS** and **Windows-Eyes** being the most popular ones) would hide all elements that have this class name applied to them.

If you want the content to be available to those who use screen readers, you should use the next class that we will be learning about the `visuallyhidden` class.

Hiding elements visually

Sometimes, you don't want to render something to the screen, but have it available for screen readers using a **Skip Navigation** link. This would ensure that those who use screen readers can skip to the meat of the content immediately rather than listen to a list of navigation links. So, let's add this class to our **Skip Navigation** link that we have in the header, as shown in the following code:

```
<a class="visuallyhidden" href="#main">Skip Navigation</a>
```

This makes the link disappear from our screen, but is it available to screen readers. The web page displayed in the following screenshot does not show the **Skip Navigation** link:



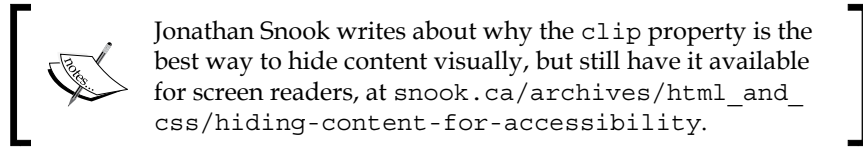
The CSS rule that makes this happen is as follows:

```
.visuallyhidden {  
border: 0;  
clip: rect(0 0 0 0);  
height: 1px;  
margin: -1px;  
overflow: hidden;  
padding: 0;  
position: absolute;  
width: 1px;  
}
```

A typical solution used to involve having them positioned absolutely with a height of 0px, but this would prevent Apple's VoiceOver screen reader from reading the content.

Another solution involves using the `text-indent` property to position the text off-screen, but then care needs to be taken when content is written in Right-To-Left language, where this solution would fail.

Using the `clip` property would avoid all of these problems, while having the content readable across all screen readers.



Those who extensively use keyboard navigation would also want to skip navigation. But, because it is visually hidden, they would not know that this option exists. For this case, you want this to be available when this element is in focus. Let us add an additional class called `focusable` to make this available for our **Skip Navigation** link that would make this option visible when they make this link active via keyboard navigation.

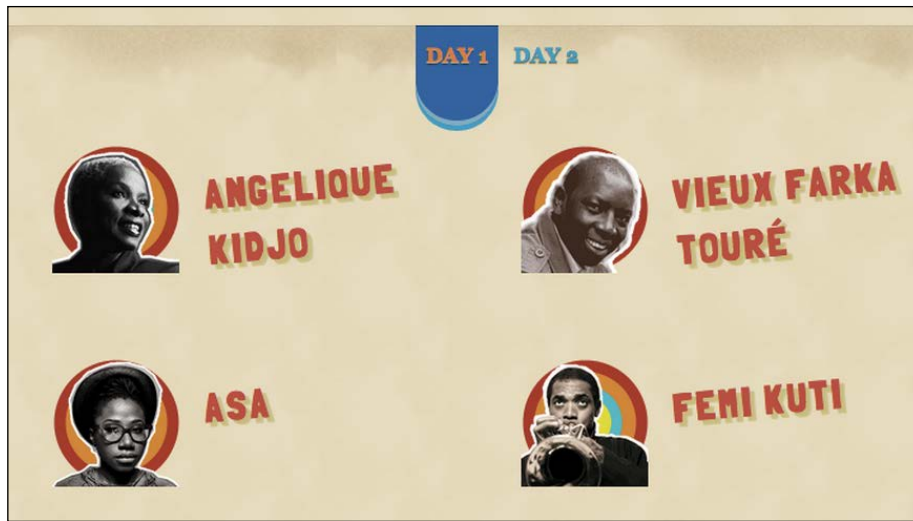
```
<a class="visuallyhidden focusable" href="#main">Skip Navigation</a>
```

The following screenshot shows how the **Skip Navigation** link is instantly visible when the user switches keyboard focus to it:



Hiding elements without impacting layout

In our website, we want a tabbed display of line up over several days, as shown in the following screenshot:



The following is a simplified view of the markup:

```
<article class="t-tabs t-section" id="lineup">
<nav class="t-tab__nav">
<a class="t-tab__navitem--active t-tab__navitem" href="#day-1">Day 1</a>
<a class="t-tab__navitem" href="#day-2">Day 2</a>
</nav>
<ul id="day-1" class="t-tab__body t-grid t-before-1-6 t-after-1-6">
<li class="t-grid__cell t-unit-1-2">
<a class="t-media--row" href="#">

<b class="t-media__body t-title-tabartist t-artist__name">Angelique
Kidjo</b>
</a>
</li>
</ul>
<ul id="day-2" class="t-tab__body t-grid t-before-1-6 t-after-1-6">
<li class="t-grid__cell t-unit-1-2">
<a class="t-media--row" href="#">

<b class="t-media__body t-title-tabartist t-artist__name">Oumou
Sangre</b>
</a>
</li>
</ul>
</article>
```

The simplest way to do this would be to show only **Day 1** and use the hidden class to hide the rest of the days, as shown in the following code snippet:

```
<article class="t-tabs t-section" id="lineup">
<nav class="t-tab__nav">
<a class="t-tab__navitem--active t-tab__navitem" href="#day-1">Day 1</a>
<a class="t-tab__navitem" href="#day-2">Day 2</a>
</nav>
<ul id="day-1" class="t-tab__body t-grid t-before-1-6 t-after-1-6">
<!--list content below -->
</ul>
<ul id="day-2" class="t-tab__body t-grid t-before-1-6 t-after-1-6
hidden">
<!--list content below -->

</ul>
</article>
```

By hiding the elements, we make the dimensions that they occupy vanish to 0. This means the area previously occupied by that content collapses.

As the user clicks on one or the other navigation links for each day's line up, the content for each day will frequently be hidden and shown, which will look jarring, as shown in the following screenshot:



In such a case, we can use the helper class `invisible` to make the element not render, but maintain its dimensions; it will not be visible on the screen or be available to screen readers. As you can see in the following screenshot, the **TICKETS** section does not change its position depending on which tab is active:



Clearing floats

We are positioning the image elements on the left-hand side of the artists' names. We do this by floating the images to the left. Luckily for us, we do not have any content that follows the container with floated elements. If we did, then that content would be overlaid on top of the floated element. You can prevent this from occurring by setting a class called `clearfix` on the parent container of the floated elements. In our case, to ensure our floated elements never trigger this behavior, we shall add the `clearfix` class to the parent element of the artist image element:

```
<a class="t-media--row clearfix" href="#">
```

To learn more about how the `clearfix` class works, read about it in *Appendix, You Are an Expert, Now What?*

Now that we have taken care of the basic essentials, let us apply styles to spruce up the page itself to look more like the design we had in mind. The following code snippet shows how to add styles to our page:

```
html {
  background: url('/img/waves-bg.png') repeat-x,
  url(/img/heading-banner-back.png) 50% 100px no-repeat,
```

```
url(/img/bg-active.png) 50% 72px repeat-x,
url('/img/bg.png') #e7dcbb;
box-sizing: border-box;
margin: 0 1em;
font: 100%/1.5 georgia, serif;
}

body {
max-width: 80%;
margin: 0 auto;
text-align: center;
}

.t-tabs {
min-height: 400px;
position: relative;
}

.t-tab__body {
position: absolute;
left: 0;
right: 0;
}

.t-tab__navitem--active {
position: relative;
}

.t-tab__navitem--active::after{
position: absolute;
bottom: -2em;
left: 0;
height: 2em;
width: 100%;
content: "";
border-radius: 0 0 20em 20em;
background: #305da1;
box-shadow: 0 -0.3em 0 0 #77aec3 inset, 0 0.3em 0 0 #1A9DC8;
}

/* TICKETS */
.t-tickets__currency {
font-family: georgia, serif;
text-align: center;
}
```

```
position: absolute;
transform-origin: 100% 100%;
transform: rotate(-90deg) translate(0, -2.1em);
}

/* MEDIA OBJECT */
.t-media,
.t-media--column,
.t-media--row,
.t-media__body {
text-align: left;
list-style: none;
}
.t-media--row .t-media__aside {
float:left;
margin-right: 16px;
}

/* Image replaced social media links */
.t-links__item--twitter,
.t-links__item--facebook,
.t-links__item--flickr {
padding: 0.25rem 1rem;
display: inline-block;
}

.ir.t-links__item--twitter,
.ir.t-links__item--facebook,
.ir.t-links__item--flickr {
background-size: contain;
background-repeat: no-repeat;
width: 1rem;
height: 1rem;
background-position: center center;
display: inline-block;
}

.ir.t-links__item--twitter {
background-image: url(/img/logo-twitter.svg);
}

.t-title--h1,
.t-title--h2,
.t-title--navsite,
```

```
.t-title-tabartist {  
font-family: FolkSolidRegular, sans-serif;  
text-transform: uppercase;  
color: #E4773A;  
text-shadow: 3px 3px 1px #C84134,  
             4px 4px 1px #C84134;  
letter-spacing: 2px;  
}
```

Writing valid stylesheets

When we went through that, you might have noticed that the styles have no typos whatsoever. The kind copy editors have no doubt done a wonderful job, but I realize you have no such assistants when you write your stylesheets! An errant typo could cause us untold trauma as we hunt why a particular style does not get applied. This is why it is important to also automate validation of your styles and use autocompletion to automate as much of your style declarations as possible.

Sublime Text and Vim both have autocompletion of CSS properties available, and you can automate the insertion of the semicolon at the end too! If you have no access to these tools, you can use the online CSS validator at jigsaw.w3.org/css-validator/ to test your CSS.

There is another way to automate writing valid and productive style rules – by using an alternative style language that compiles into CSS. We shall be looking into some of these languages next.

Style languages to write productive stylesheets

For a very long time, the only way to write stylesheets was to use the syntax that W3C provided for within the specifications that it produced. However, there are a lot of productivity benefits to be gained by using some programming logic to write stylesheets. But browsers could only understand syntaxes that are mandated by the W3C specifications. This means, any style language that uses additional programmable features should be converted to a browser-understandable typical stylesheet (this is called compilation).

One of the earliest style languages designed for this is called Sass. Now, we have a few more, the most popular ones being Sass, Less, and Stylus. In both Sass and Less, valid CSS is automatically valid Sass and Less code. This makes it trivial to port it from CSS to these languages.

Typically, you would be writing your style rules in files named as `main.scss` (if you are using Sass), `main.less` (if you are using Less), or `main.styl` (if you are using Stylus). Using the compilers that come with each of these languages, these files will respectively be compiled to `styles.css`.

Advantages

Using style languages has many merits, such as the following:

- These languages enable you to always write syntactically valid stylesheets as they all throw an error if you use any invalid syntax.
- All of these languages provide some of the sought-after features in CSS, such as variables, ability to re-use style rules in other classes without repeating yourself several times, arithmetic calculations, color functions, and more.
- You can choose to output expanded readable styles when developing, and then output a compact performance-optimized, whitespace-stripped stylesheet when you are using it in production.

Disadvantages

However, using style languages also has some disadvantages, as explained in the following points:

- While it is easy to convert to Sass or Less, it is not possible to make modifications in the resulting stylesheet and have those changes be ported over to their original Sass/Less/Style files. So, you need to be careful to make sure nobody edits the resulting CSS files.
- Working in a team requires the whole team to co-opt to use one of these languages. Without that, it is impossible to maintain two forks of the stylesheets and keep them in sync.
- When debugging, if you are inspecting an element, most debuggers only reveal the line numbers in a stylesheet and not in the original language files. This might make it difficult to find out where in your original files the particular rule would be found.

Where to learn?

If you are interested in learning more about these languages, read on for some good places to get started.

Sass

The official website is sass-lang.com. Chris Coyier has a good introduction video on Sass at css-tricks.com/video-screencasts/88-intro-to-compass-sass/.

Less

The official website is lesscss.org. A video overview of Less is available at net.tutsplus.com/tutorials/html-css-techniques/quick-tip-you-need-to-check-out-less-js/

Stylus

The official website is at learnboost.github.com/stylus. A video overview of Stylus is available at thechangelog.com/post/3036532096/stylus-expressive-robust-feature-rich-css-language.

Using HTML5 Boilerplate with style languages

If you are fairly confident in navigating your way with any of these languages, then you could use any of the available ports that we will look at next, to start your projects:

Sass

There is a fairly up-to-date port of HTML5 Boilerplate to Sass requiring **Compass**, which is a framework on top of Sass at github.com/sporkd/compass-html5-boilerplate.

Less

A less frequently updated port of HTML5 Boilerplate to Less exists at github.com/m6tt/less-boilerplate.

Stylus

There is no fully functional port of HTML5 Boilerplate available for Stylus, although using the command-line to convert it to stylus seems to be the easiest way. More information about using this method can be found at learnboost.github.com/stylus/docs/executable.html.

Summary

Woah! That was an intense coding session. In this chapter, we almost created a whole site based on HTML5 Boilerplate. We looked at how to write markup, styles, and scripts. In addition, we explored some tools to make writing valid markup and styles easier.

All the changes that we have made so far to our example project are available at nimbu.in/h5bp-book/chapter-3/.

In the next chapter, we will look at adding some interactivity to this pretty static page with jQuery and make it easier to navigate around the site.

4

Adding Interactivity and Completing Your Site

We have created the first cut of how the site will look. While the site looks pretty readable and navigable, making the interaction smoother would make it a significantly better experience.

Using jQuery

As we saw in *Chapter 2, Starting Your Project*, HTML5 Boilerplate provides a handy and safe way to load jQuery. With jQuery, it is vastly simple to work on writing scripts to access elements.

If you are writing custom jQuery script either to kick off a plugin you are using or to do some small interaction, put it in the `main.js` file in the `js` folder.

Using other libraries

If you are more comfortable using other libraries, you can also load and use them in a similar way to jQuery.

The following is how we load jQuery:

```
<script src="//ajax.googleapis.com/ajax/libs/jquery/1.8.2/jquery.min.js"></script>
<script>window.jQuery || document.write('<script src="js/vendor/jquery-1.8.2.min.js"></script>')
</script>
```

Let us say, you want to use another library (like MooTools), then look up the Google Libraries API to see if that library is available at `developers.google.com/speed/libraries/`. If it is available, just replace the reference with the appropriate reference from the site. For example, if we want to replace our jQuery link with a link to MooTools, we would simply replace the following code:

```
<script src="//ajax.googleapis.com/ajax/libs/jquery/1.7.2/jquery.min.js">
</script>
```

With the following line of code:

```
<script src="ajax.googleapis.com/ajax/libs/mootools/1.4.5/mootools-yui-compressed.js">
</script>
```

We will also download MooTools' minified file to the `js/vendor` folder locally and replace the following code:

```
<script>window.jQuery||document.write('<script src="js/vendor/jquery-1.7.2.min.js"></script>')
</script>
```

With the following line of code:

```
<script>window.jQuery||document.write('<script src="js/vendor/mootools-core-1.4.5-full-compat-yc.js"></script>')
</script>
```

For more information on why we use local copies of the code, please check *Chapter 2, Starting Your Project*. But we are pretty happy with our default choice of jQuery, so let us proceed with it.

Adding smooth-scroll plugin and interaction

If you have not noticed it already, the website we are building is a single page site! All content that is required is found on the same page. The way our site is currently designed, it would mean clicking on one of the site navigation links would scroll roughly to the section that the navigation link refers to. We would like this interaction to be smooth. Let us use jQuery's smooth-scroll plugin to provide this.

Let us download the plugin file from the Github repository, hosted on `github.com/kswedberg/jquery-smooth-scroll`.

In it, we find a minimized version of the plugin (`jquery.smooth-scroll.min.js`) that we shall open in our text editor.

Then copy all the code and paste it within the `plugins.js` file.

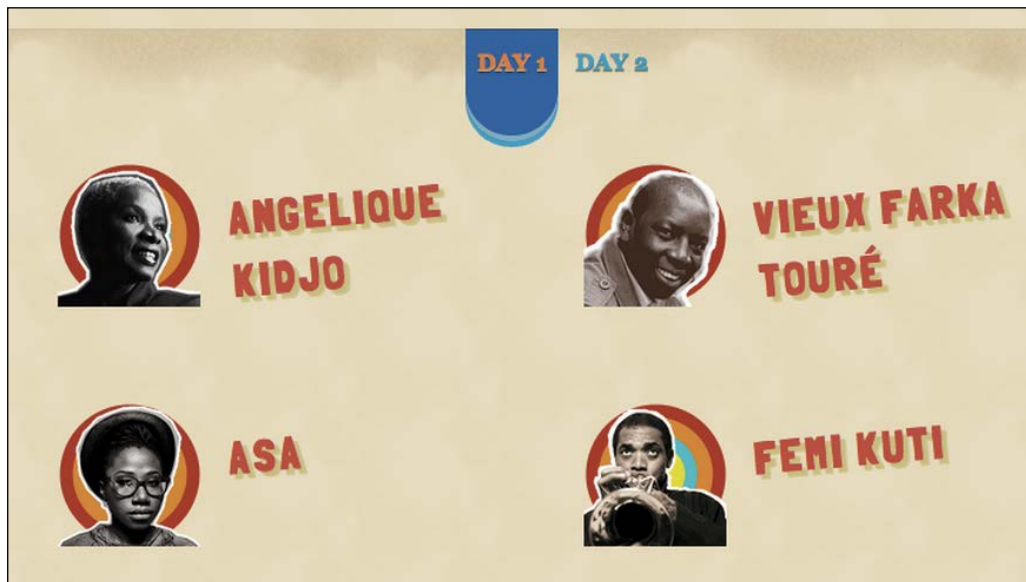
Let us add a class name `js-scrollitem` to let us distinguish that this element has a script that will be used on those elements. This way, there will be a lesser chance of accidentally deleting class names that are required for interactions prompted via JavaScript.

Now, we shall write the code to invoke this plugin in the `main.js` file. Open the `main.js` file in your text editor and type:

```
$('.js-scrollitem').smoothScroll();
```

This will make all the clickable links that link to sections on the same page within the parent container with class `js-scrollitem` scroll smoothly with the help of the plugin. If we have used our HTML5 Boilerplate defaults correctly, adding this will be more than sufficient to get started with smooth scrolling.

Next, we would like the navigation links in the line up section to open the right-hand side line up depending on which day was clicked on. Right now, in the following screenshot, it simply shows the line up for the first day, and does not do anything else:



Let us continue editing the `main.js` file and add in the code that would enable this.

First, let's add the class names that we will use to control the styling, and the hiding/showing behavior within our code. The code for this functionality is as follows:

```
<nav class="t-tab__nav">
  <a class="t-tab__navitem--active t-tab__navitemjs-tabitem" href="#day-1">Day 1</a>
  <a class="t-tab__navitemjs-tabitem" href="#day-2">Day 2</a>
</nav>
```

Now, we shall write the code that will show the element we clicked on. This code is as follows:

```
var $navlinks = $('#lineup .js-tabitem');
var $tabs = $('.t-tab__body');

var hiddenClass = 'hidden';

var activeClass = 't-tab__navitem--active';

$navlinks.click(function() {
  // our code for showing or hiding the current day's line up
  $(this.hash).removeClass(hiddenClass);
});
```

By checking how we have done so far, we notice it keeps each day's line up always visible and does not hide them once done! Let us add that too, as shown in the following code snippet:

```
var $navlinks = $('#lineup .js-tabitem');
var $tabs = $('.t-tab__body');

var hiddenClass = 'hidden';

var activeClass = 't-tab__navitem--active';

var $lastactivetab = null;

$navlinks.click(function() {
  var $this = $(this);
  //take note of what was the immediately previous tab and tab nav
  that was active
  $lastactivetab = $lastactivetab || $tabs.not('.' + hiddenClass);
  // our code for showing or hiding the current day's line up
  $lastactivetab.addClass(hiddenClass);
  $(this.hash).removeClass(hiddenClass);
```

```
$lastactivetab = $(this.hash);  
return false;  
}
```

You would notice that the active tab navigation item still seems to suggest it is **Day 1!** Let us fix that by changing our code to do something similar with the tabbed navigation anchors, as shown in the following code snippet:

```
var $navlinks = $('#lineup .js-tabitem');  
var $tabs = $('.t-tab__body');  
  
var hiddenClass = 'hidden';  
  
var activeClass = 't-tab__navitem--active';  
  
var $lastactivetab = null;  
var $lastactivenav = null;  
  
$navlinks.click(function() {  
var $this = $(this);  
//take note of what was the immediately previous tab and tab nav that  
was active  
$lastactivetab = $lastactivetab || $tabs.not('.' + hiddenClass);  
$lastactivenav = $lastactivenav || $navlinks.filter('.' +  
activeClass);  
  
    // our code for showing or hiding the current day's line up  
$lastactivetab.addClass(hiddenClass);  
$(this.hash).removeClass(hiddenClass);  
$lastactivetab = $(this.hash);  
  
    // change active navigation item  
$lastactivenav.removeClass(activeClass);  
$this.addClass(activeClass);  
$lastactivenav = $this;  
  
return false;  
});
```

Bingo! We have our day-by-day line up ready. We now need to ensure our Google Maps `iframe` renders when users click on the **Locate on a map** link. We also want to use the same link to hide the map if the users want to do so.

First, we add some identifiable features to the anchor element used to trigger the showing/hiding of map and the `iframe` for the maps, as shown in the following code snippet:

```
<p>The festival will be held on the beautiful beaches of NgorTerrou
Bi in Dakar.
<a href="#" class="js-map-link">Locate it on a map</a>
</p>

<iframe id="venue-map" class="hidden" width="425"
height="350" frameborder="0" scrolling="no" marginheight="0"
marginwidth="0" src="http://maps.google.com/maps?f=q&source=s
_q&hl=en&geocode=&q=ngor+terrou+bi,+dakar,+senegal&
;aq=&sll=37.0625,-95.677068&sspn=90.404249,95.976562&i
e=UTF8&hq=ngor&hnear=Terrou-Bi,+Bd+Martin+Luther+King,+Gue
ule+Tapee,+Dakar+Region,+Guediawaye,+Dakar+221,+Senegal&t=m&am
p;fll=14.751996,-17.513559&fspn=0.014276,0.011716&st=1091
46043351405611748&rq=1&ev=p&split=1&ll=14.711109,-
17.483921&spn=0.014276,0.011716&output=embed">
</iframe>
```

Then we use the following JavaScript to trigger the link:

```
$maplink = $(' .js-map-link');
$maplinkText = $maplink.text();

$maplink.toggle(function() {
  $('#venue-map').removeClass(hiddenClass);
  $maplink.text('Hide Map');
}, function() {
  $('#venue-map').addClass(hiddenClass);
  $maplink.text($maplinkText);
});
```


Now, let us look at how we can make our audio player work on all browsers.

Adding HTML5 features safely with Modernizr

We looked at Modernizr briefly in *Chapter 1, Before We Begin*, but we haven't used it for anything much yet. It is highly recommended that we create a custom build of Modernizr. HTML5 Boilerplate comes with a custom build of Modernizr that includes every option available in the custom builder (modernizr.com/download/) including extras such as HTML5Shiv, resource loader (modernizr.load), media queries test, and the addition of CSS class names to the `html` tag based on the test results from Modernizr.

The custom build of Modernizr enables HTML5 elements in IE (read more about it at paulirish.com/2011/the-history-of-the-html5-shiv/). But, now, with our audio player, we have the opportunity to use the other Modernizr function that is available as an extra, that is, `modernizr.load`.

Audio support in browsers is not as simple as we would expect it to be. Different browsers expect different formats because of licensing restrictions. Some browsers do not even support HTML5 audio. It would be perfect to use a framework that abstracts away all these for us. Looking at html5please.com, we see that the recommended suggestion is to use a framework called `mediaelement.js` to help us deal with these issues.

 `html5please.com` is a site that tells you which of these new features are available for use and how they should be used on browsers that do not support them.

Let us use this framework for our audio player only when audio support is not detected.

First, we download the framework from mediaelementjs.com and copy all the files from the build folder into `js/vendor/mediaelement/`. Then, we shall add the cross-browser friendly audio markup for our player in `index.html`, as shown in the following code snippet:

```
<article class="t-audio">
  <audio controls preload="none" autobuffer>
    <source src="festival.mp3" />
    <source src="festival.ogg" />
  </audio>
</article>
```

Note that we need to specify the stylesheet in the `head` element to make sure it works perfectly on all browsers (instead of loading it just in time), as shown in the following code:

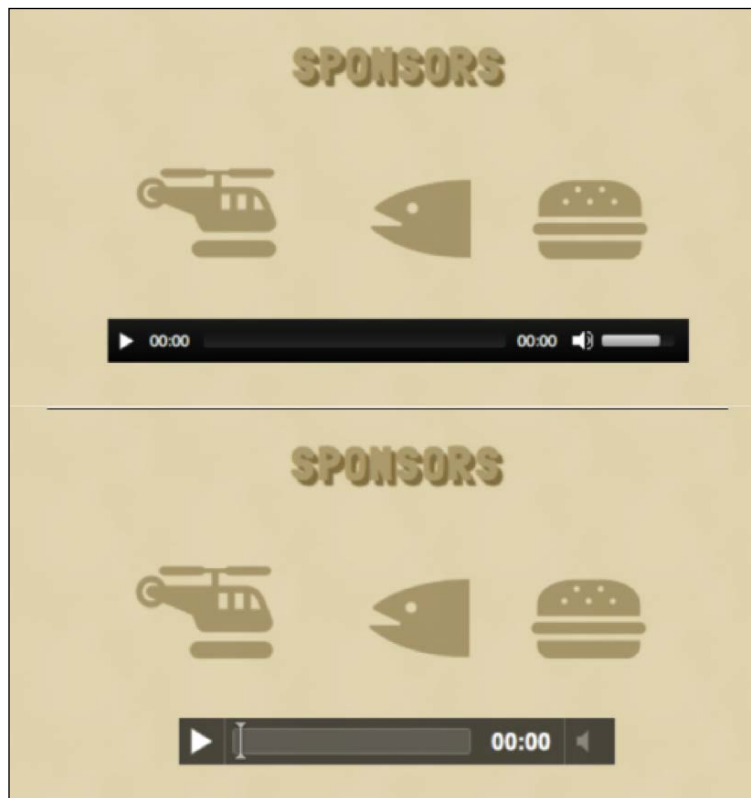
```
<link rel="stylesheet" href="js/vendor/mediaelement/
mediaelementplayer.css">
```

We then load the `mediaelement.js` only when audio support is missing by using Modernizr in our `main.js` file, as shown in the following code:

```
Modernizr.load({
  test: Modernizr.audio,
  nope: {
```

```
'mediaelementjs': 'js/vendor/mediaelement/mediaelement-and-player.min.js',  
},  
  
callback: {  
  'mediaelementjs': function() {  
    $('audio').mediaelementplayer();  
  }  
}  
});
```

This code first tests if audio is supported with Modernizr. If it is not supported, then we load the necessary resources to make the audio work using our `mediaelement.js` framework. Once `mediaelement.js` is loaded, we call it, so that it runs and converts our audio files to a format that browsers which lack audio support will understand.



The previous screenshot shows our page rendering on a browser without support for HTML5 audio (falling back to Flash with `mediaelement.js`) and in a browser with support for HTML5 audio (using native controls provided by the browser).

When to use Modernizr.load?

`Modernizr.load` is a great utility when you have multiple files you want to load conditionally like in our audio player.

Sometimes, you want something to happen only when the user clicks on a link or an element. Instead of loading all the required assets beforehand and making the browser render the page slowly, you can load these assets just in time after the user has clicked on the element.

Using Modernizr to load CSS features

Modernizr also outputs the results of its tests for various HTML5/CSS3 features on the `html` tag of your page, as shown in the following screenshot:

A screenshot of a browser's developer console showing the output of Modernizr tests. The text is: `<html lang="en-us" class=" js flexbox canvas canvastext webgl no-touch geolocation postmessage websqldatabase indexeddb hashchange history draganddrop websockets rgba hsla multiplebgs backgroundsize borderimage borderradius boxshadow textshadow opacity cssanimations csscolumns cssgradients cssreflections csstransforms csstransforms3d csstransitions fontface generatedcontent video audio localstorage sessionstorage webworkers applicationcache svg inlinesvg smil svgclippaths">`

This is very useful if you would like to style experiences based on the kind of features available. For example, we notice the class name called `no-touch` in the `html` element. This means the browser this page was loaded in, did not support touch interfaces. If touch was supported, then we could make all links with slightly more padding to account for large fingers trying to click on them. Let us add styles to our `css/style.css` file to do this, as follows:

```
.touch a {
padding: 0.25em;
background: #CEC3A1;
border-radius: 0.5em;
display: inline-block;
}
```

Here is how our site looks on a browser that supports touch events (on the left-hand side) and one that does not (on the right-hand side):



Testing our site

Whew! That was a lot to get by! But wait, we are not done yet! We have written all the code, but how about some testing? There are so many variants of browsers out there and it is impossible to test on each and every one of them. Fortunately, it is pretty simple to test on most major versions of browsers.

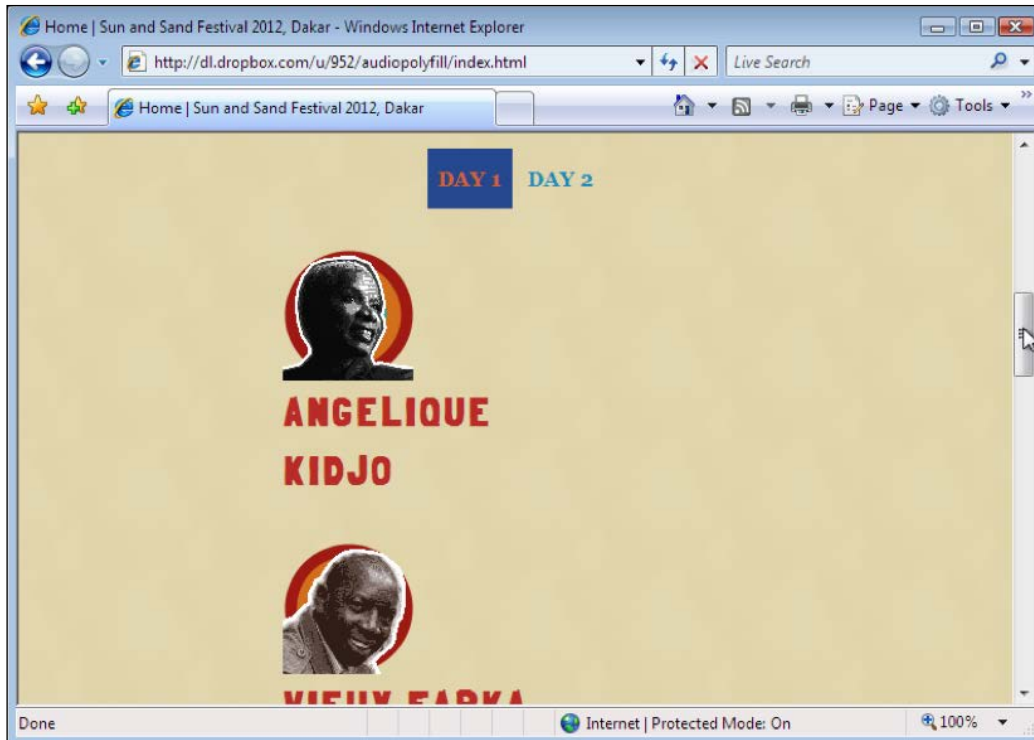
If you are on Windows, I recommend you install the latest versions of Opera, Opera Next, Safari, Chrome, Chrome Canary, Firefox, Firefox Nightly, IE8, and IE10.

If you are on Mac, get every browser listed above, except IE. If you are able to afford it, buy a Windows Operating System and install it as a virtual image on Virtual Box (www.virtualbox.org/). Microsoft provides older IEs as virtual images for testing, which you could also install on Virtual Box using [ievms \(github.com/xdissent/ievms\)](https://github.com/xdissent/ievms).

For a far easier but less rigorous testing option—say when you have not yet finalized your website—try www.browserstack.com or browserling.com.

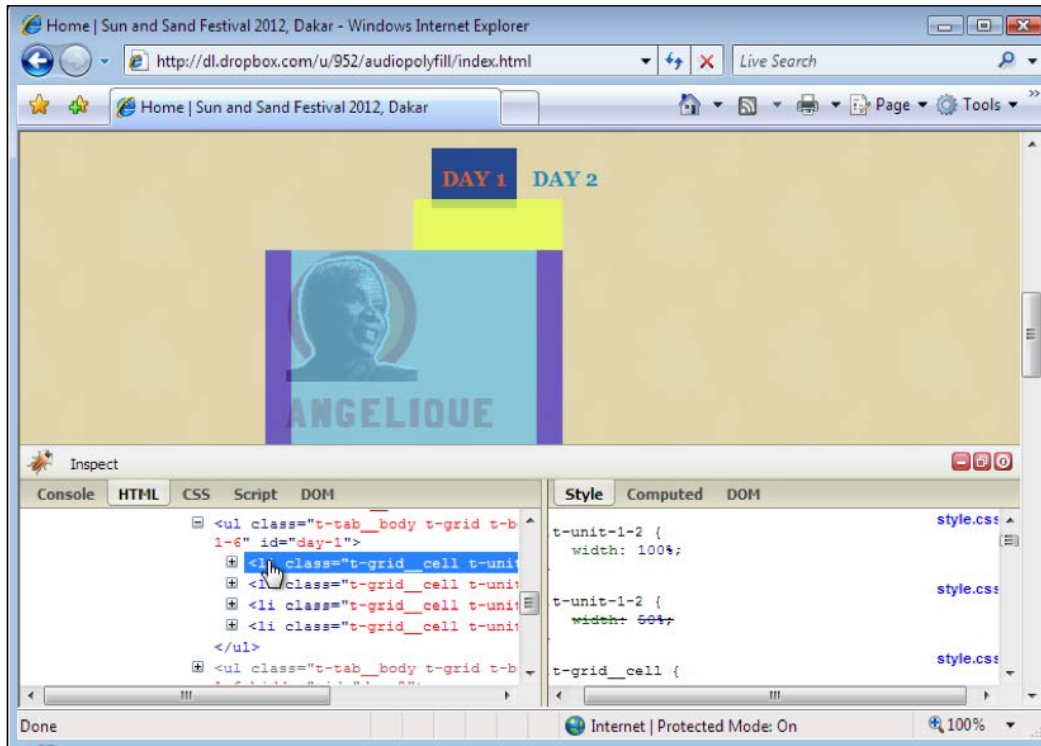
All of these browsers have developer tools that make it very easy to detect when a page is not rendered as expected.

Let us test our Sun and Sand Festival website in Internet Explorer 7. At first glance, everything appears to work as expected. But looking at the tabs, it seems like everything has gone haywire! The following screenshot displays our page on the Internet Explorer browser:



To debug this, let us use Firebug Lite to check what styles are being applied on these elements. You can install Firebug Lite as a bookmarklet on IE7 (<http://getfirebug.com/firebuglite>). Clicking on that bookmarklet would enable us to use a constrained version of Firebug on IE7.

Using Firebug, we see a debugging window, as shown in the following screenshot:



Checking into our `main.css`, it seems like our media query-based styles are all being parsed and interpreted by IE7, irrespective of the conditionals within! For example:

```
.t-unit-1-2 {  
  width: 100%;  
}
```

The previous style was declared within the media query `@media only screen and (max-width: 750px)`, which is supposed to override the existing rule `(.t-unit-1-2 { width: 50%; })` only if the query is satisfied. But IE7 simply ignores the features mentioned and blindly applies all the style rules it finds.

Thanks to conditional CSS class names, we can fix this trivially by adding an additional style rule to the original CSS declaration to prevent this override in IE6 to IE8. The *Appendix, You Are an Expert, Now What?* covers conditional CSS class names in greater detail.

HTML5 Boilerplate gives you three class names to use for such cases, described as follows:

- `.lt-ie7`: Targets all IE versions that are lower than IE7 with this class name. This would apply styles to IE 6 and below.
- `.lt-ie8`: Targets all IE versions that are lower than IE8 with this class name. This would apply styles to IE6 and IE7.
- `.lt-ie9`: Targets all IE versions lower than IE9. This would apply styles to all IE versions 8 and below.

Thanks to this, we can now apply rules that target IE8 and below, which do not understand conditions in media queries by applying style rules as follows:

```
.lt-ie9 .t-unit-1-2 {  
width: 45%;  
}
```

As IE8 and below also do not support the `box-sizing` property (Mozilla Developer Network describes the effects of this property at developer.mozilla.org/En/CSS/Box-sizing), this means the widths of these boxes will expand as we add padding. Let us remove the margins on the parent element to prevent the boxes from stacking up, as shown in the following code snippet:

```
.lt-ie9 .t-before-1-6,  
.lt-ie9 .t-after-1-6 {  
margin-left: 0;  
margin-right: 0;  
}
```

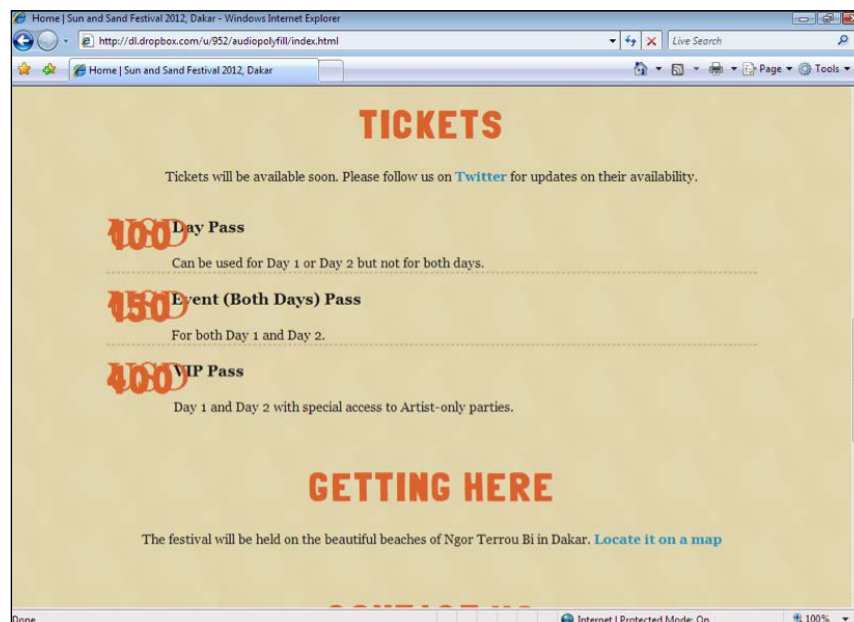
However, that doesn't quite solve our problem. Then, looking further up, we notice that our grid cells, that is, the elements with the class `t-grid__cell`, have the `display` property set to `inline-block`. Knowing that IE7 does not apply this to any element other than those with natural inline property, we would have to add an additional declaration to make this work, as shown in the following code snippet:

```
.lt-ie9 .t-grid__cell {  
display: inline;  
}
```


Finally, now this works as we wanted it to!



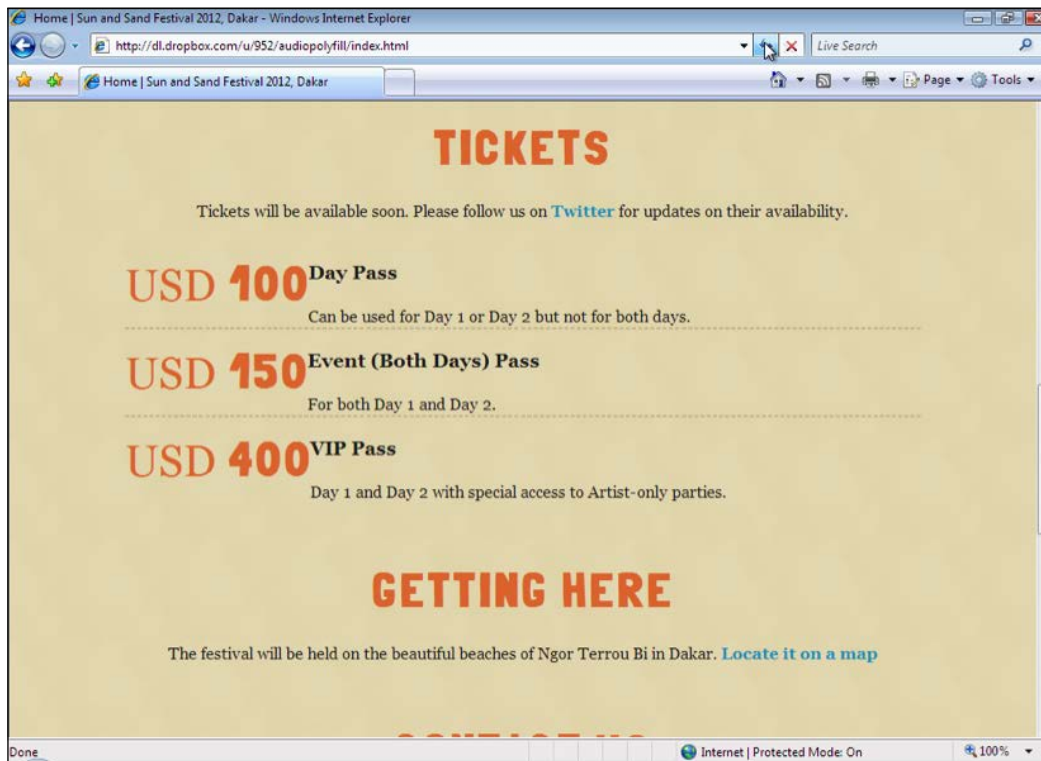
Let us scroll to the bottom of the page. We notice the prices are all scrambled because of a lack of CSS3 transforms support in IE7, as shown in the following screenshot:



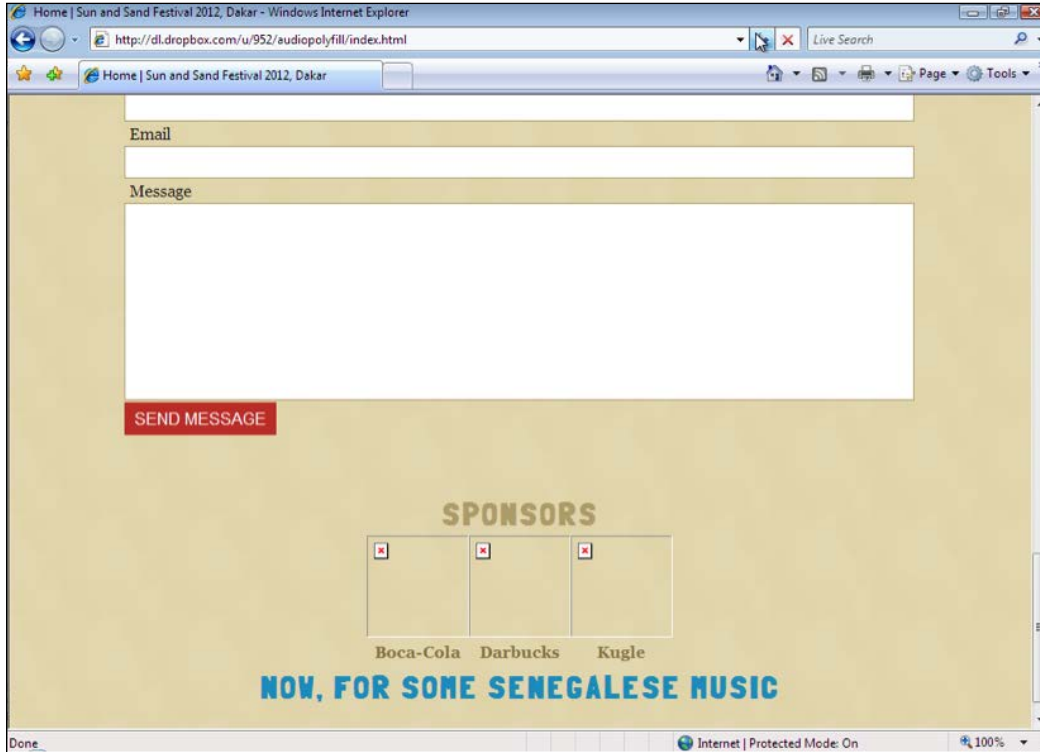
With Modernizr, all we need to do is to add this rule to our stylesheet:

```
.no-csstransforms .t-tickets__currency {  
  position: static;  
}
```

This would make it more readable for any browser that does not support CSS transforms, as shown in the following screenshot:




Scrolling further down, we notice our SVG icons are missing as IE8 and below do not recognize SVG files, as shown in the following screenshot:

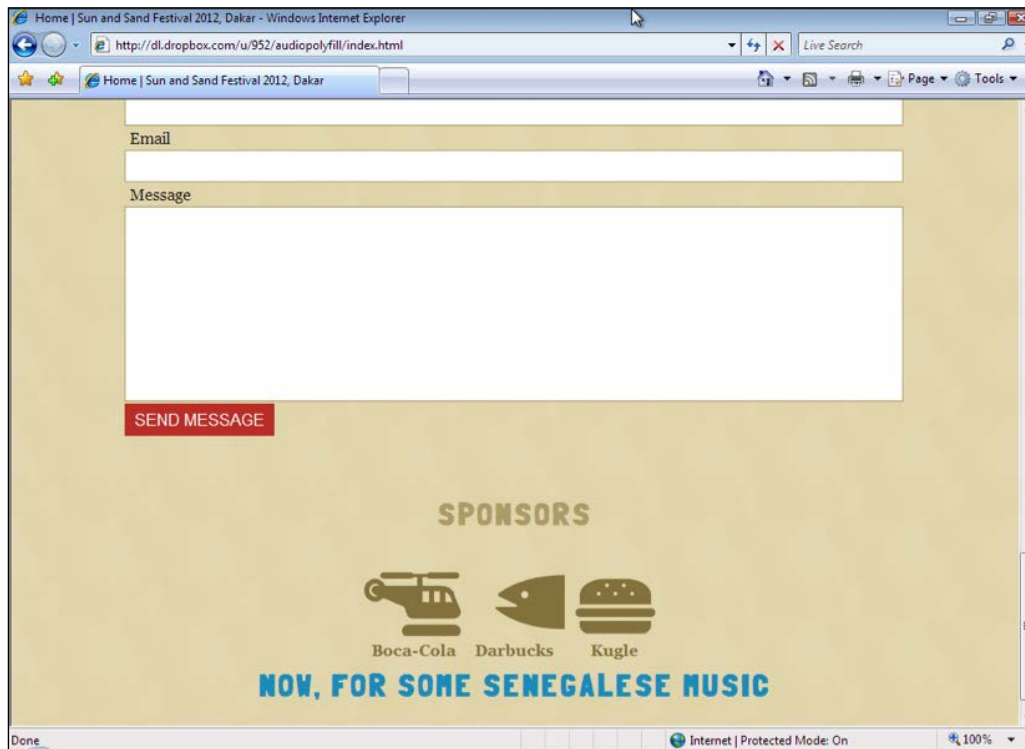


Again Modernizr comes to our rescue! In our `main.js` file, we will check the outcome of the SVG test in Modernizr and then replace all the SVG images with their equivalent PNG ones. Do note that this means you need a PNG equivalent for every SVG file you use in your HTML page. The code to replace SVG with PNG files is as follows:

```
if(Modernizr.svg == false) {
  $('img[src$=".svg"]').each(function() {
    this.src = /(.*).svg$.exec(this.src)[1] + '.png';
  });
}
```

 **Why use SVG?** We are using SVG icons as these can scale as per our needs of a responsive website, as SVG is a vector image format. Moreover, they are extremely lightweight compared to typical PNG files and can load significantly faster than PNG formats.

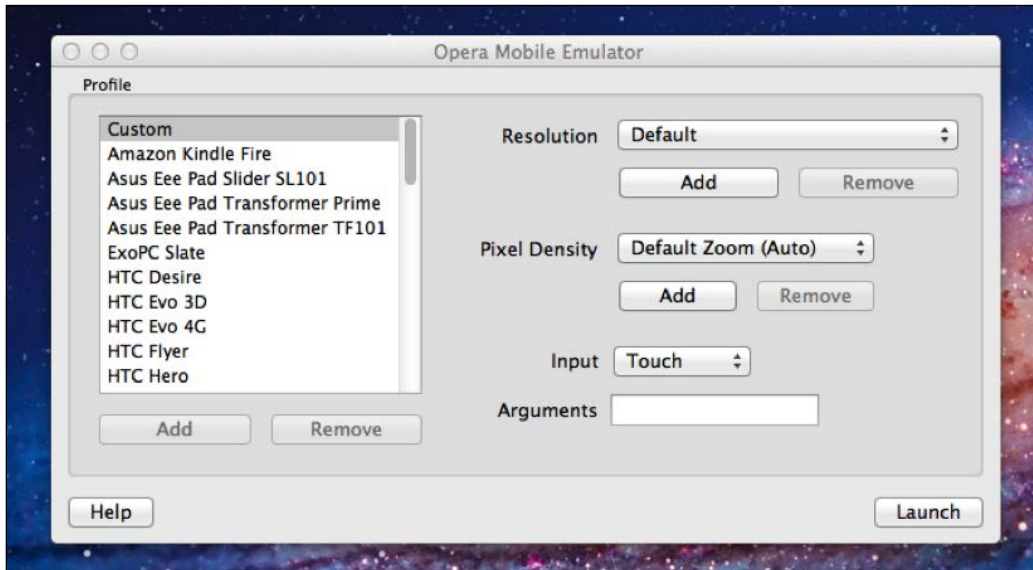
The following screenshot shows how IE7 renders the icons in PNG format thanks to Modernizr:



As you get into web development, you should spend more time using browser developer tools; Andi Smith wrote a good post outlining some of the features of each of them at andismith.com/blog/2011/11/25-dev-tool-secrets/.

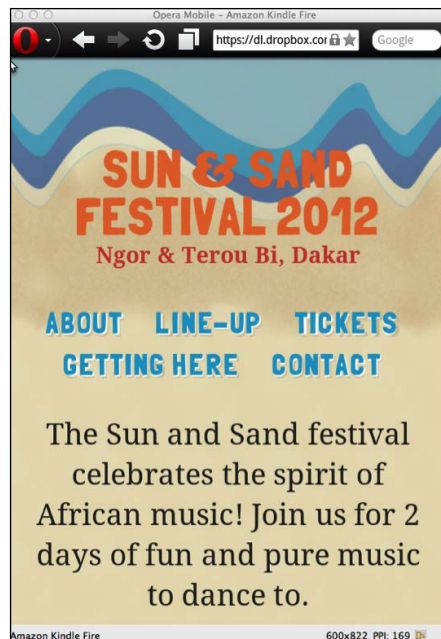
Testing on non-desktop browsers

Let us look at how the site looks on smaller-scale devices. The quickest and easiest way to do this would be to download **Opera Mobile Emulator** from www.opera.com/developer/tools/mobile/ and use one of the several available options to load our page. This emulator is shown in the following screenshot:



Choose one of the options on the left-hand side of the emulator and click on the **Launch** button to open an Opera browser instance that emulates how it would appear on the device you have selected.


For example, the following screenshot shows how our page renders on an instance of **Opera Mobile Emulator** for **Amazon Kindle Fire**:



The best part is that the **Opera Mobile** browser is one of the most modern mobile browsers available, which makes it a very good browser to test on when you are actively developing your website. It is also available on a wide variety of devices, which makes it easy to use **Opera Mobile Emulator** for testing various device widths if you are using media queries to style the page to adapt to different device dimensions.

If you also possess an iPhone running iOS 6, it is fairly easy to use **Remote Debugging with Safari 6** and inspect the code using Safari developer tools (Max Firtman has more information on how to enable this at <http://www.mobilexweb.com/blog/iphone-5-ios-6-html5-developers>).

If you have an Android device, you can enable debugging with Chrome for Android browser, but you need to install Android developer tools to do so. More help on how to do this is found in this guide to remote debugging on Chrome for Android at <https://developers.google.com/chrome/mobile/docs/debugging>.

 If you have multiple mobile devices that run different browsers available at your fingertips, you can also use **Adobe Edge Inspect** from html.adobe.com/edge/inspect/ to test how these pages look in tandem across all of these devices.

Summary

In this chapter, we looked at adding some interaction to the site using jQuery plugins. We also looked at how to use `Modernizr.load` to load scripts to make it easy to conditionally detect support for HTML5 audio and load resources for browsers that lack support and render the audio correctly. We also looked at some of the ways we can debug our site using browser developer tools and verify how the page appears on various browsers.

In the next chapter, we will look at how to optimize our site server-side on Apache and other web servers.

5

Customizing the Apache Server

Our Sun and Sand festival site is more or less done! But before we deploy it to production, let us make sure we have optimized the configuration of the server where the page and the associated files will be served from, so that the end users can load the page as quickly as possible, while we check against security vulnerabilities that might cause our site to get hacked.

Server-side configurations

Before we go further, let us briefly look at what a server does. The server understands a browser's request for a page of your site and then looks for the file the URL requests. The server then sends the file back to the browser with additional information called HTTP headers. **Apache** is the most popular server software for websites, and HTML5 Boilerplate comes with a configuration file for Apache called `.htaccess`.

Setting up the Apache server

Before we check out the various features of the Apache configuration file provided by HTML5 Boilerplate, let us set up a local Apache server, so that we can see these features in action.

Installing Apache

We will look at the installations of Apache on Mac, Windows, and Linux.

Mac

You do not have to do anything special; Apache is already installed. But to use it for this project, ensure you copy all the files to the website's folder in your home folder (`~/<username>`). Edit the `/etc/apache2/httpd.conf` file to change the following highlighted code:

```
<Directory /usr/share/web>
  AllowOverride None
    Options MultiViewsFollowSymlinks
    Order allow,deny
    Allow from all
    Header Set Cache-Control no-cache
</Directory>
```

To the following:

```
<Directory /usr/share/web>
  AllowOverrideAll
    Options MultiViewsFollowSymlinks
    Order allow,deny
    Allow from all
    Header Set Cache-Control no-cache
</Directory>
```

You will also need to change this entry in `/etc/apache2/<username>.conf` the same way.

Windows

You need to download and install Apache on Windows; it can be downloaded from httpd.apache.org/docs/2.2/platform/windows.html. Note that you need to add the following code snippet to `conf/httpd.conf`, located within the folder where the Apache application is found:

```
<Directory "/apache/htdocs/">
  AllowOverride All
  Options None
  Order deny, allow
</Directory>
```

Linux

If you are using Ubuntu, there is a friendly documentation available at <https://help.ubuntu.com/8.04/serverguide/C/httpd.html>. To enable `.htaccess` files, used to configure your Apache server, you need to edit `/etc/apache2/sites-available/default` from the following code snippet:

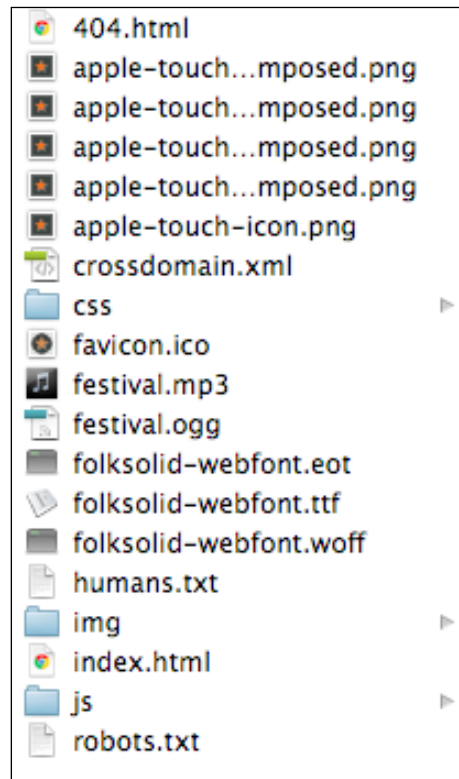
```
<Directory /var/www/>
Options Indexes FollowSymLinksMultiViews
AllowOverride None
    Order allow,deny
allow from all
    # Uncomment this directive is you want to see apache2's
    # default start page (in /apache2-default) when you go to /
    #RedirectMatch ^/$ /apache2-default/
</Directory>
```

To the following code snippet:

```
<Directory /var/www/>
Options Indexes FollowSymLinksMultiViews
AllowOverrideAll
    Order allow,deny
allow from all
    # Uncomment this directive is you want to see apache2's
    # default start page (in /apache2-default) when you go to /
    #RedirectMatch ^/$ /apache2-default/
</Directory>
```

Configuring Apache

Our folder for HTML5 Boilerplate contains a file called `.htaccess`. As the filename starts with a `.`, it is likely that `.htaccess` won't show up when you list your files in Finder/Windows Explorer or other file manager utilities, as shown in the following screenshot:



But if you enable the hidden files to appear on your OS, you will be able to see this file.

All that is required now is to move our site files (including the `.htaccess` file) to the server we just set up. Apache looks for a `.htaccess` file on all folders (unless told not to by a configuration setting) and so having our `.htaccess` file in the parent folder of our site is just fine.

Using a `.htaccess` file for testing is not a bad idea in general. However, if you want to make your site really zippy, it is best to put the configuration directly on the Apache server's main configuration file (`httpd.conf`). Unfortunately, not all hosting providers allow this.

If you do have access to the Apache server's main configuration file (`httpd.conf`), you should copy the configurations from HTML5 Boilerplate's `.htaccess` file and put them within `httpd.conf` inside a `Directory` tag, as shown in the following code snippet:

```
<Directory /path/to/website/root>
[htaccess rules]
</Directory>
```

You should then remove the `.htaccess` file as the directives are already on the server's main configuration file.

Features available out of the box

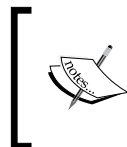
Most of the advantages that HTML5 Boilerplate's `.htaccess` file provides are not immediately obvious. If your site receives low traffic and does not make too many network requests, you may not notice any significant difference using HTML5 Boilerplate's `.htaccess` file. However, when you do have spikes of high activity (not uncommon!) or suddenly have a lot of network requests for images and videos that your site requires, HTML5 Boilerplate's `.htaccess` comes to your rescue automatically.

All of these features are available to you as soon as you either put a `.htaccess` file in the project folder or if you set up Apache's main configuration file as indicated earlier.

Removing ETags

Entity Tags (ETags) validate whether components, that is, images, files, and so on, in a browser's cache match components on the server. Unfortunately, ETags do more harm than good. Most servers have ETags available by default, which is why HTML5 Boilerplate's server configuration file prevents a server from serving them, as shown in the following code snippet:

```
<IfModule mod_headers.c>
  Header unset ETag
</IfModule>
FileETag None
```



Steve Souders writes in depth on why ETags fail to solve the problem they were designed for and why you should remove them, at developer.yahoo.com/blogs/ydn/posts/2007/07/high_performanc_11/.

Gzip components

Gzip is the most popular compression method. By compressing your files with Gzip, you can make sure your files get transferred more quickly, even with low bandwidth connections. Sometimes the savings are as much as 70 percent of file size, making this a great performance configuration default.

Let us look at how big our files are without our `.htaccess` Gzip feature in place. To do this, we simply comment out that section, as shown in the following code snippet:








```
#<IfModule mod_deflate.c>
#
# # Force deflate for mangled headers developer.yahoo.com/blogs/ydn/
# posts/2010/12/pushing-beyond-gzipping/
# <IfModule mod_setenvif.c>
#   <IfModule mod_headers.c>
#     SetEnvIfNoCase ^(\Accept-EncodXng|X-cept-
# Encoding|X{15}|~{15}|~{15})$ ^((gzip|deflate)\s*,?\s*)+|[X~-]{4,13}$
# HAVE_Accept-Encoding
#     RequestHeader append Accept-Encoding "gzip,deflate" env=HAVE_
# Accept-Encoding
#   </IfModule>
# </IfModule>
#
# # HTML, TXT, CSS, JavaScript, JSON, XML, HTC:
# <IfModule filter_module>
#   FilterDeclare    COMPRESS
#   FilterProvider   COMPRESS  DEFLATE resp=Content-Type $text/html
#   FilterProvider   COMPRESS  DEFLATE resp=Content-Type $text/css
#   FilterProvider   COMPRESS  DEFLATE resp=Content-Type $text/plain
#   FilterProvider   COMPRESS  DEFLATE resp=Content-Type $text/xml
#   FilterProvider   COMPRESS  DEFLATE resp=Content-Type $text/x-
# component
#   FilterProvider   COMPRESS  DEFLATE resp=Content-Type $application/
# javascript
#   FilterProvider   COMPRESS  DEFLATE resp=Content-Type $application/
# json
#   FilterProvider   COMPRESS  DEFLATE resp=Content-Type $application/
# xml
#   FilterProvider   COMPRESS  DEFLATE resp=Content-Type $application/
# xhtml+xml
#   FilterProvider   COMPRESS  DEFLATE resp=Content-Type $application/
# rss+xml
#   FilterProvider   COMPRESS  DEFLATE resp=Content-Type $application/
# atom+xml
```

```








#   FilterProvider  COMPRESS  DEFLATE  resp=Content-Type $application/
vnd.ms-fontobject
#   FilterProvider  COMPRESS  DEFLATE  resp=Content-Type $image/
svg+xml
#   FilterProvider  COMPRESS  DEFLATE  resp=Content-Type $image/x-icon
#   FilterProvider  COMPRESS  DEFLATE  resp=Content-Type $application/
x-font-ttf
#   FilterProvider  COMPRESS  DEFLATE  resp=Content-Type $font/
opentype
#   FilterChain      COMPRESS
#   FilterProtocol  COMPRESS  DEFLATE  change=yes;byteranges=no
# </IfModule>
#
# <IfModule !mod_filter.c>
#   # Legacy versions of Apache
#   AddOutputFilterByType DEFLATE text/html text/plain text/css
application/json
#   AddOutputFilterByType DEFLATE application/javascript
#   AddOutputFilterByType DEFLATE text/xml application/xml text/x-
component
#   AddOutputFilterByType DEFLATE application/xhtml+xml application/
rss+xml application/atom+xml
#   AddOutputFilterByType DEFLATE image/x-icon image/svg+xml
application/vnd.ms-fontobject application/x-font-ttf #font/opentype
# </IfModule>
#
#</IfModule>

```

Now, let us look at the sizes of files that get delivered to our browser through the network tools available in browser developer tools (in this case, Chrome Developer Tools):

| Name Path | Method | Status Text | Type | Initiator | Size Content | Time Latency |
|---|--------|-------------|-----------------|------------------------|--------------------|----------------|
|  /~manian/ /~manian | GET | 200 OK | text/html | Other | 11.56KB 11.18KB | 6ms 5ms |
|  style.css /~manian/css | GET | 200 OK | text/css | /~manian/:23 Parser | 20.88KB 20.54KB | 7ms 6ms |
|  mediaelementplayer.css /~manian/js/vendor/mediaelement | GET | 200 OK | text/css | /~manian/:24 Parser | 20.45KB 20.10KB | 17ms 7ms |
|  jquery.min.js ajax.googleapis.com/ajax/libs/jquery/1.7.2 | GET | 200 OK | text/javascr... | /~manian/:31 Parser | 33.27KB 92.62KB | 350ms 266ms |
|  modernizr-2.5.2.min.js /~manian/js/vendor | GET | 200 OK | application/... | /~manian/:31 Parser | 15.37KB 15.01KB | 387ms 386ms |
|  plugins.js /~manian/js | GET | 200 OK | application/... | /~manian/:31 Parser | 3.97KB 3.61KB | 1.39s 1.39s |
|  main.js /~manian/js | GET | 200 OK | application/... | /~manian/:31 Parser | 1.70KB 1.34KB | 1.39s 1.39s |

Now, let's enable Gzip by enabling the appropriate rules in `.htaccess` by removing `#` from the beginning of the lines. Notice the difference, as shown in the following screenshot:

| Name Path | Method | Status Text | Type | Initiator | Size Content | Time Latency |
|---|--------|----------------|-----------------|------------------------|--------------------|-----------------|
|  /~manian/ /~manian | GET | 200 OK | text/html | Other | 3.66KB 11.18KB | 7ms 6ms |
|  style.css /~manian/css | GET | 200 OK | text/css | /~manian/:23 Parser | 5.89KB 20.54KB | 19ms 17ms |
|  mediaelementplayer.css /~manian/js/vendor/mediaelement | GET | 200 OK | text/css | /~manian/:24 Parser | 3.57KB 20.10KB | 14ms 12ms |
|  jquery.min.js ajax.googleapis.com/ajax/libs/jquery/1.7.2 | GET | 200 OK | text/javascr... | /~manian/:31 Parser | 33.27KB 92.62KB | 112ms 68ms |
|  modernizr-2.5.2.min.js /~manian/js/vendor | GET | 200 OK | application/... | /~manian/:31 Parser | 6.36KB 15.01KB | 878ms 877ms |
|  plugins.js /~manian/js | GET | 200 OK | application/... | /~manian/:31 Parser | 2.08KB 3.61KB | 1.88s 1.88s |
|  main.js /~manian/js | GET | 200 OK | application/... | /~manian/:31 Parser | 1.02KB 1.34KB | 1.88s 1.88s |

If you would like to learn more about Gzip, *Chapter 4, Smaller Components, Book of Speed*, Stoyan Stefanov, found at www.bookofspeed.com/chapter4.html, would be a good place to start from.

Using Expires header for better cache control

Servers can indicate to browsers how long they can keep files in the cache. This is pretty useful for static files that don't change frequently, and will reduce your page-load time. HTML5 Boilerplate's `.htaccess` file has a set of defaults for most static files, as shown in the following code snippet:

```
<IfModule mod_expires.c>
ExpiresActive on

# Perhaps better to whitelist expires rules? Perhaps.
ExpiresDefault          "access plus 1 month"

# cache.appcache needs re-requests in FF 3.6 (thanks Remy ~Introducing
HTML5)
ExpiresByType text/cache-manifest "access plus 0 seconds"

# Your document html
ExpiresByType text/html   "access plus 0 seconds"

# Data
```

```

ExpiresByType text/xml                "access plus 0 seconds"
ExpiresByType application/xml          "access plus 0 seconds"
ExpiresByType application/json         "access plus 0 seconds"

# Feed
ExpiresByType application/rss+xml      "access plus 1 hour"
ExpiresByType application/atom+xml     "access plus 1 hour"

# Favicon (cannot be renamed)
ExpiresByType image/x-icon             "access plus 1 week"

# Media: images, video, audio
ExpiresByType image/gif                "access plus 1 month"
ExpiresByType image/png                "access plus 1 month"
ExpiresByType image/jpg                "access plus 1 month"
ExpiresByType image/jpeg              "access plus 1 month"
ExpiresByType video/ogg                "access plus 1 month"
ExpiresByType audio/ogg                "access plus 1 month"
ExpiresByType video/mp4                "access plus 1 month"
ExpiresByType video/webm               "access plus 1 month"

# HTC files (css3pie)
ExpiresByType text/x-component         "access plus 1 month"

# Webfonts
ExpiresByType application/x-font-ttf   "access plus 1 month"
ExpiresByType font/opentype            "access plus 1 month"
ExpiresByType application/x-font-woff  "access plus 1 month"
ExpiresByType image/svg+xml           "access plus 1 month"
ExpiresByType application/vnd.ms-fontobject "access plus 1 month"

# CSS and JavaScript
ExpiresByType text/css                  "access plus 1 year"
ExpiresByType application/javascript    "access plus 1 year"
</IfModule>

```


This tells the server to cache requests for files of each type as soon as it is accessed freshly for the period specified by the text "access plus...". For example, consider the following code snippet:

```

# CSS and JavaScript
ExpiresByType text/css                "access plus 1 year"
ExpiresByType application/javascript   "access plus 1 year"

```


This fragment makes the server tell the browser, requesting CSS and JavaScript files, to cache each of these files for at least a year since the first time it was accessed, unless the user deliberately clears their cache.

 Yahoo's best practices for speeding up your site has a detailed explanation of what the Expires header does at developer.yahoo.com/performance/rules.html#expires.

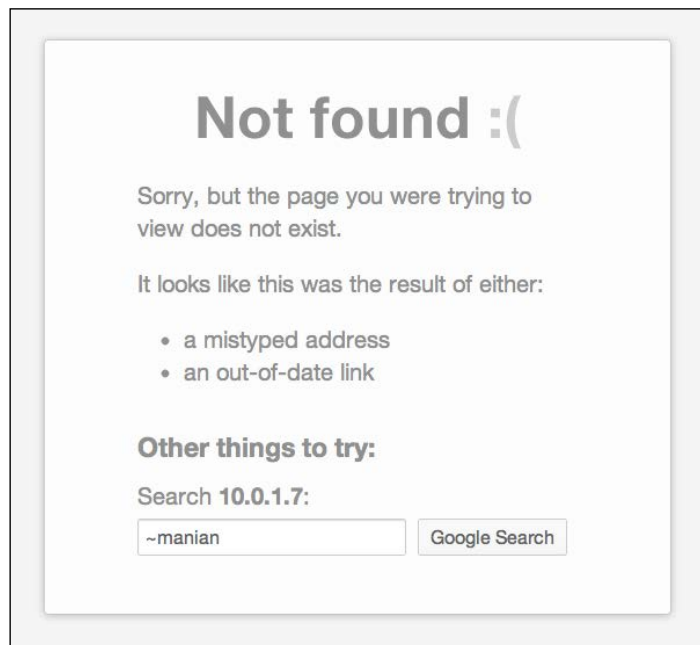
Custom 404 page

HTML5 Boilerplate provides a custom 404 page called `404.html`. But, this will never be used, unless the server knows to serve this file every time a resource is not found. HTML5 Boilerplate's `.htaccess` file has a configuration that tells the server to use this file as follows:

```
ErrorDocument 404 /404.html
```

Make sure to refer to the `404.html` using the full path. For example, on a Mac, the full path would be `/~<username>/404.html`, if you are hosting it in the website's folder under your `<username>` folder.

The following screenshot shows how a browser renders the default HTML5 Boilerplate 404 page, when HTML5 Boilerplate's `.htaccess` file is used:



Forcing the latest IE version

Internet Explorer utilizes a `meta` tag to decide whether it should render a site in compatibility mode or use the latest rendering engine to render it.

Google Chrome has released a plugin named **Chrome Frame**, downloadable from <https://developers.google.com/chrome/chrome-frame/> that, if installed on a user's computer, will provide the experience of a modern browser when the user uses older versions of Internet Explorer. Your site can opt-in to using this plugin on a user's computer, when your page is being viewed on older versions of Internet Explorer. To opt-in to using this plugin automatically, append `, chrome=1` to the content attribute value for the `http-equiv` meta tag.

This tag can be set within the HTML file itself, which is what HTML5 Boilerplate does, as shown in the following code snippet:

```
<meta http-equiv="X-UA-Compatible" content="IE=edge,chrome=1">
```

However, as HTML5 Boilerplate uses conditional comments around the `html` tag, IE will render the HTML in **Compatibility View**, not with Chrome Frame. Hence, using the `meta` tag with conditional comments around the `html` tag would not work. HTML5 Boilerplate's `.htaccess` file sets this as an HTTP header instead, as shown in the following code snippet:

```
<IfModule mod_headers.c>
  Header set X-UA-Compatible "IE=Edge,chrome=1"
  # mod_headers can't match by content-type, but we don't want to send
  this header on *everything*...
  <FilesMatch "\.(js|css|gif|png|jpe?g|pdf|xml|oga|ogg|m4a|ogv|mp4|m4v|w
  ebm|svg|svgz|eot|ttf|otf|woff|ico|webp|appcache|manifest|htc|crx|oex|x
  pi|safariextz|vcf)$" >
    Header unset X-UA-Compatible
  </FilesMatch>
</IfModule>
```

This forces IE to respect the HTTP header that is sent, and use the latest rendering engine irrespective of what the `meta` tag states. You can also set IE to use whatever rendering engine you like. We discuss this feature in depth in *Appendix, You Are an Expert, Now What?*, under the section *What is meta x-ua-compatible?*.

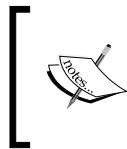


There is a great level of detailed testing and comments that informed our decision to recommend this method for setting IE Compatibility mode, which is available from the **Issue Tracker** on Github at github.com/h5bp/html5-boilerplate/issues/378.

Using UTF-8 encoding

Character encoding is a way to represent your text data in byte sequences. There have been different standards available for different scripts, for example, Greek, Japanese, and so on, but the standards body that creates HTML specifications, W3C, strongly endorses the use of **UTF-8** as the de-facto encoding scheme for all text served on the Web to ensure all browsers can render your text data correctly. The `.htaccess` file sets it in the following manner:

```
# Use UTF-8 encoding for anything served text/plain or text/html
AddDefaultCharset utf-8
# Force UTF-8 for a number of file formats
AddCharset utf-8 .css .js .xml .json .rss .atom
```



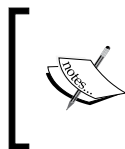
Edward Z. Yang wrote an informative post on why UTF-8 is the best choice for character encoding at htmlpurifier.org/docs/enduser-utf8.html#whyutf8; it is worth reading if you are interested in this topic.

Serving the right MIME types

A **Multipurpose Internet Mail Extensions (MIME)** type sent as a HTTP header helps the browser decide how to process the content that is sent. For example, a browser needs to know when a file is a stylesheet and when it is a downloadable text document. This information is provided by the MIME type HTTP header that the server returns when sending that resource. HTML5 Boilerplate's `.htaccess` file ensures your server provides the right MIME type when serving content.

For example, in our Senegal music festival website, we need our Web fonts to be understood by the browser to be a font file and not garbled text. In our HTML5 Boilerplate `.htaccess` file, the following lines make sure the server returns the correct MIME type so browsers can do that:

```
AddType application/vnd.ms-fontobject
AddType application/x-font-ttf ttf
AddType font/opentype otf
AddType application/x-font-woff woff
```



More information on MIME types can be found on the **Mozilla Developer Network** at developer.mozilla.org/en/Properly_Configuring_Server_MIME_Types#What_are_MIME_types.3F.

Blocking access to hidden folders

If you use a **Version Control System (VCS)** to manage your website's code, the hidden folders used to manage versioning (`.git` or `.svn`) are likely to exist in your production servers too. You do not want anyone to access these files and find any information that could be used to hack your website. HTML5 Boilerplate prevents the server from providing content requested of these folders within the `.htaccess` file, as shown in the following code snippet:

```
# Block access to "hidden" directories whose names begin with a
# period. This
# includes directories used by version control systems such as
# Subversion or Git.
<IfModule mod_rewrite.c>
RewriteCond %{SCRIPT_FILENAME} -d
RewriteCond %{SCRIPT_FILENAME} -f
RewriteRule "^(|/)\." - [F]
</IfModule>
```

Blocking access to backup and source files

If you have your databases backed up on the server, for example, `database.sql.bak`, you do not want anyone to access that either, nor logfiles or any of your source files, such as Photoshop files for logos – we know it happens! The following code in the `.htaccess` file prevents access to these files:

```
# Block access to backup and source files
# This files may be left by some text/html editors and
# pose a great security danger, when someone can access them
<FilesMatch "(\\. (bak|config|sql|fla|psd|ini|log|sh|inc|swp|dist) |~)$">
  Order allow,deny
  Deny from all
  Satisfy All
</FilesMatch>
```

This tells the server to look for files that end with any of these extensions: `<filename>.bak`, `<filename>.config`, and so on, and if so, deny processing requests for such files. It will return a 403 Forbidden error instead.

Starting Rewrite engine

The Apache server requires you to start the rewrite engine before you do any URL rewriting. The HTML5 Boilerplate `.htaccess` file enables this as shown in the following code snippet:

```
<IfModule mod_rewrite.c>
  Options +FollowSymlinks
  RewriteEngine On
  # RewriteBase /
</IfModule>
```

If your site is in a subfolder, remove the `#` from the `RewriteBase` line and set it to the full path to the subfolder from the root.

Preventing 404 errors for non-existing redirected folders

In Apache, you need to disable `MultiViews` if you want to redirect URLs requested from paths, that do not exist, to another path.

For example, if you have an incoming request to `http://example.com/beaches/10` and you want it to internally redirect to `http://example.com/index.php?q=10` and the folder `beaches` does not exist in the root folder of your website, Apache would throw an error. The HTML5 Boilerplate's `.htaccess` file prevents this from occurring by using the following code statement:

```
Options -MultiViews
```

Additional customizations

Many additional customizations are provided but are commented out, as they require careful consideration and sometimes may have unwanted repercussions.

Suppressing or forcing the "www." at the beginning of URLs

Most of us do not realize that `http://example.com` and `http://www.example.com` are treated as two separate sites by search engines. You can either force rewriting of URLs to `www` or the non-`www` ones. I prefer the non-`www` URL, because it is three characters shorter!

HTML5 Boilerplate's `.htaccess` file provides you with choices for doing either of them. By default, the configuration forces the server to rewrite requests for `http://www.example.com` to `http://example.com`. If you prefer the other way around, have the server rewrite requests for `http://example.com` to `http://www.example.com`, as described in the following steps:

1. Comment out the default option shown in the following code snippet:

```
# Option 1:
# Rewrite "www.example.com -> example.com"
<IfModule mod_rewrite.c>
  RewriteCond %{HTTPS} !=on
  RewriteCond %{HTTP_HOST} ^www\.(.+)$ [NC]
  RewriteRule ^ http://%1%{REQUEST_URI} [R=301,L]
</IfModule>
```

2. The commented-out default section should now look like the following code snippet:

```
# Option 1:
# Rewrite "www.example.com -> example.com"

# <IfModule mod_rewrite.c>
# RewriteCond %{HTTPS} !=on
# RewriteCond %{HTTP_HOST} ^www\.(.+)$ [NC]
# RewriteRule ^ http://%1%{REQUEST_URI} [R=301,L]
#</IfModule>
```

As you might notice, all we did was add a `#` character and a space before each of those lines.

3. Now, we shall enable the second option by uncommenting it. Change the following code snippet by uncommenting it:

```
# Option 2:
# To rewrite "example.com -> www.example.com" uncomment the
following lines.
# Be aware that the following rule might not be a good idea if you
# use "real" subdomains for certain parts of your website.

# <IfModule mod_rewrite.c>
# RewriteCond %{HTTPS} !=on
# RewriteCond %{HTTP_HOST} !^www\..+$ [NC]
# RewriteRule ^ http://www.%{HTTP_HOST}%{REQUEST_URI} [R=301,L]
# </IfModule>
```

4. The uncommented code section should look like the following code snippet:

```
# Option 2:
# To rewrite "example.com -> www.example.com" uncomment the
# following lines.
# Be aware that the following rule might not be a good idea if you
# use "real" subdomains for certain parts of your website.

<IfModule mod_rewrite.c>
RewriteCond %{HTTPS} !=on
RewriteCond %{HTTP_HOST} !^www\..+$ [NC]
RewriteRule ^ http://www.%{HTTP_HOST}%{REQUEST_URI} [R=301,L]
</IfModule>
```

All we did was remove the # character and a space in front of the lines starting with `<IfModule mod_rewrite.c>` and ending with `</IfModule>`.

Whichever option you want to use, make sure you don't have both these options enabled at the same time, as that would prevent Apache from serving your page.

Setting cookies from iFrames

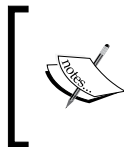
IE usually blocks cookies set from within an IFrame. If you require such cookies to be set, especially if you have advertisements or social networking plugins, you need to send a **Platform for Privacy Preferences Project (P3P)** header.

Look for the comment in the `.htaccess` file with the same text as the title of this section and change the following lines:

```
# <IfModule mod_headers.c>
#   Header set P3P "policyref=\"/w3c/p3p.xml\"", CP=\"IDC DSP COR ADM
DEVI TAIi PSA PSD IVAi IVDi CONi HIS OUR IND CNT\"
# </IfModule>
```

To the following code snippet:

```
<IfModule mod_headers.c>
  Header set P3P "policyref=\"/w3c/p3p.xml\"", CP=\"IDC DSP COR ADM
DEVI TAIi PSA PSD IVAi IVDi CONi HIS OUR IND CNT\"
</IfModule>
```



Eric Law wrote about IE's cookie policies in detail, which makes for good reading at blogs.msdn.com/b/ieinternals/archive/2009/08/20/wininet-ie-cookie-internals-faq.aspx.

PHP security defaults

If you are serving PHP, there are a lot of configuration options in the HTML5 Boilerplate's `.htaccess` file that could make your PHP installation more secure. If you are using PHP, you can turn them on using the same procedure as the one outlined in the section titled *Suppress or force the "www." at the beginning of URLs*.


Given we aren't using PHP in our website, we do not need to turn them on.

Stop advertising Apache version

You can prevent Apache from advertising its version to mitigate chances of malicious programmers exploiting vulnerabilities in a particular version. Here is how the Apache version is advertised:

```
▼ Response Headers view source
Accept-Ranges: bytes
Cache-Control: max-age=2592000
Connection: Keep-Alive
Content-Length: 4866
Content-Type: image/png
Date: Sun, 24 Jun 2012 00:31:17 GMT
Expires: Tue, 24 Jul 2012 00:31:17 GMT
Keep-Alive: timeout=15, max=96
Last-Modified: Tue, 29 May 2012 00:29:04 GMT
Server: Apache/2.2.21 (Unix) DAV/2
```

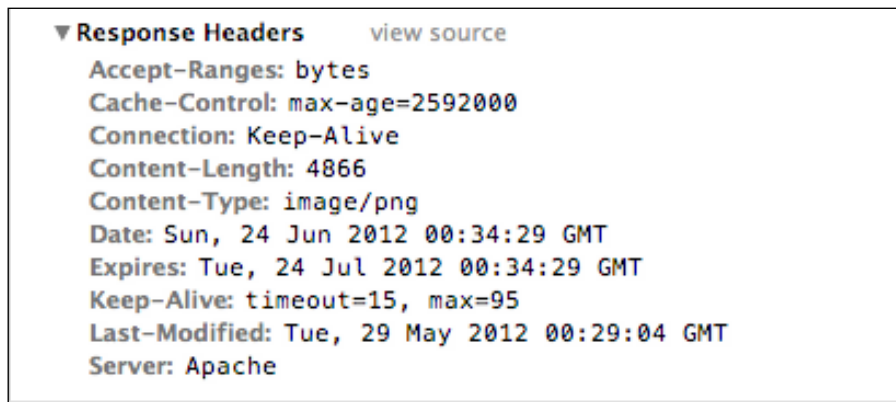
This previous screenshot shows the Apache version number sent as a HTTP header to the browser.

 You can use the developer tools that come with most browsers to verify HTTP headers. In this case, we are using Chrome's Developer Tools **Resources** tab. More information on how to use this tool is available on Chrome's help center at developers.google.com/chrome-developer-tools/docs/network.

This needs to be configured from within the server's main configuration file, and we cannot do this from within the `.htaccess` file. So, let's remove the following directive from the HTML5 Boilerplate's `.htaccess` file and replace the one found in `/etc/apache2/httpd.conf` (the path to this file will be different if you are using Windows or Linux):

```
ServerTokens Prod
```

The following screenshot shows the version-less HTTP header sent by Apache after applying the configuration value to the Apache server's main configuration file:



Allowing concatenation from within specific JS and CSS files

Sometimes, you may want the server to combine multiple script or stylesheet files into one response when a request is made. Note that doing so does not make it any faster for your page to load, as the server takes its own time to stitch these files together.

This is an option I recommend you consider last, when every other solution has failed. Ideally, you should never be doing this.

To do this, first uncomment the following lines in the `.htaccess` file from:

```
#<FilesMatch "\.combined\.js$">  
# Options +Includes  
# AddOutputFilterByType INCLUDES application/javascript application/  
json
```

```
# SetOutputFilter INCLUDES
#</FilesMatch>
#<FilesMatch "\.combined\.css$">
# Options +Includes
# AddOutputFilterByType INCLUDES text/css
# SetOutputFilter INCLUDES
#</FilesMatch>
```

To the following code snippet:

```
<FilesMatch "\.combined\.js$">
  Options +Includes
AddOutputFilterByType INCLUDES application/javascript application/json
SetOutputFilter INCLUDES
</FilesMatch>
<FilesMatch "\.combined\.css$">
  Options +Includes
AddOutputFilterByType INCLUDES text/css
SetOutputFilter INCLUDES
</FilesMatch>
```

Then, in the `js` folder, create a file called `script.combined.js`.

Open the `script.combined.js` file in your text editor, and use the following syntax with all the files that should be combined and output in the `script.combined.js` file:

```
# <!--#include file="<path/to/file.js"> -->
# <!--#include file="<path/to/another-file.js"> -->
```

If you are looking to combine stylesheets on the fly, you can do the same. Create a file in the `css` folder called `style.combined.css`.

Open the `style.combined.css` file in your text editor, and use the following syntax with all the files that should be combined and output in the `style.combined.css` file:

```
# <!--#include file="<path/to/file.css " -->
# <!--#include file="<path/to/another-file.css"> -->
```

As I mentioned earlier, doing this will make Apache slower to respond to these requests. You should be using a build script to concatenate files (we will look into the build script in *Chapter 7, Automate Deployment with the Build Script*). So uncomment this setting only if you have no other choice.

Stopping screen flicker in IE on CSS rollovers

When you use background images that change on hovering over a link, you will see a flicker in IE when this occurs. You can prevent this by changing the following lines in the `.htaccess` file from:

```
# BrowserMatch "MSIE" brokenvary=1
# BrowserMatch "Mozilla/4.[0-9]{2}" brokenvary=1
# BrowserMatch "Opera" !brokenvary
# SetEnvIfbrokenvary 1 force-no-vary
```

To the following code snippet:

```
BrowserMatch "MSIE" brokenvary=1
BrowserMatch "Mozilla/4.[0-9]{2}" brokenvary=1
BrowserMatch "Opera" !brokenvary
SetEnvIfbrokenvary 1 force-no-vary
```

Preventing SSL certificate warnings

If you want to serve your site only on a secured connection, you need to obtain a **Secure Sockets Layer (SSL)** certificate that browsers will use to identify your website. If the domain on the certificate does not match the domain on the incoming request —, for example, you had a SSL certificate for `https://secure.example.com`, and the assets that are being loaded on the page hosted on that domain are served from `https://example.com`, but all of the files are hosted on the same Apache server; — then browsers will throw warnings and inform the user that the authenticity of the Web page is unverifiable.

You can make sure the requests to domain, which does not have the SSL certificate, is rewritten to the one for which you do have an SSL certificate. If you require this, you can uncomment the following snippet from:

```
# <IfModule mod_rewrite.c>
# RewriteCond %{SERVER_PORT} !^443
# RewriteRule ^ https://example-domain-please-change-me.
com%{REQUEST_URI} [R=301,L]
# </IfModule>
```

To the following code snippet:

```
<IfModule mod_rewrite.c>
RewriteCond %{SERVER_PORT} !^443
RewriteRule ^ https://example-domain-please-change-me.com%{REQUEST_
URI} [R=301,L]
</IfModule>
```

Note that the `https://example-domain-please-change-me.com` URL needs to point to the domain you have an SSL certificate for.



More details on the SSL and SSL certifications are given in the Linux documentation project at tldp.org/HOWTO/SSL-Certificates-HOWTO/x64.html.

That covers all the optional features that HTML5 Boilerplate's `.htaccess` file provides. Let us take a look at cross-domain policies and how to set them.

Cross-domain policies you should be aware of

An HTTP request is called a **cross-domain** request when a page served from one domain, for example, `http://example.com`, requires data from another, say `http://foo.com`. By default, most browsers do not allow cross-domain requests of data—be it data or flash assets—to prevent malicious access.

However, you can set a cross-domain policy file on the server (in the previous example, the server where `http://foo.com` is served from), which allows browsers to access these resources.

Flash requires this policy file to be specified in a file called `crossdomain.xml`, where you can specify which domains can make request of assets from the server.

This file is provided within HTML5 Boilerplate, and by default, the most restrictive policy is enabled. If you do want the least restrictive policy, you can uncomment that option and comment away the most restrictive one.



Do realize that you need to fully understand the implications of allowing cross-domain requests for access to assets before you make it less restrictive.

You can also make cross-domain AJAX requests, or restrict access to images or fonts, by setting an HTTP header. This is known as the **Cross Origin Resource Sharing (CORS)** policy.

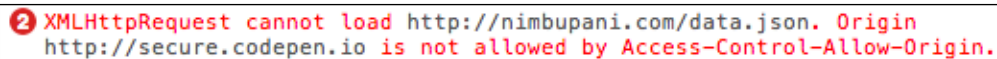
Cross-domain AJAX requests

AJAX requests can only be made if the requesting page is on the same domain as the URL it is requesting data from. CORS is a new HTML5 feature that will allow you to make AJAX requests from any domain, provided permission has been given to the requesting domain. By setting an HTTP header on the server from which you are requesting data using an AJAX request, you can overcome this limitation. Let us look at how this can be done.

The following is an example of a cross-domain request that you could make:

```
var CORSRequest = new XMLHttpRequest();
CORSRequest.onload = function(e){
    // Process returned data
}
CORSRequest.open('GET', 'http://nimbupani.com/data.json');
CORSRequest.send( null );
```

We note that the browser throws an error saying such access is forbidden, as shown in the following screenshot:



XMLHttpRequest cannot load http://nimbupani.com/data.json. Origin http://secure.codepen.io is not allowed by Access-Control-Allow-Origin.

Now, in our `.htaccess` file hosted on `http://nimbupani.com`, we will uncomment the following directive:

```
# <IfModule mod_headers.c>
#   Header set Access-Control-Allow-Origin "*"
# </IfModule>
```

Let us try our code again. Aha! Now it works!

This is the least restrictive setting, which can allow any domain to make an AJAX request on your server. It is fairly trivial to make a very high volume of requests because of this and also to pretend it's your site and fool the visitors, and so on. Use this setting with care.

CORS-enabled images

Typically, browsers allow all images to be linked from any other domain. This is called **hotlinking**. Read more about it at en.wikipedia.org/wiki/Inline_linking. If a high-traffic website links to assets that are hosted on your server, your hosting provider might even fine you for excessive use of bandwidth (or your site might go down!). If you want to prevent this, for example, if you do not want `http://example.com` to use an `img` element with an `src` attribute pointing to an image on your server `http://foo.com/image.jpg`, you can enable a more restrictive policy that only allows certain domains to access your image files by changing the following line in the `.htaccess` file from:

```
Header set Access-Control-Allow-Origin "*" env=IS_CORS
```

To the following line:

```
Header set Access-Control-Allow-Origin "http://example.com" env=IS_CORS
```

Where you replace `http://example.com` with the domain name that is only allowed access to that image. The server will then prevent any other domain from accessing images on your domain.

If you want your images to be accessed by multiple domains, you will have to write a convoluted regex comparison for the origin, as shown in the following code snippet:

```
SetEnvIf Origin »
    "^http(s)?://(.\+.)?(example-1\.com|example-2\.com)$" origin_is=$0
Header always set Access-Control-Allow-Origin %{origin_is}
eenv=origin_is
```

In this case, replace `example-1\.com` with your domain (take care to place the forward slash before the `.com`), and likewise for `example-2\.com`.

Webfont access


Most of the time, you will be hosting fonts on the same domain where you will be using them. If you do host fonts in a separate domain, Firefox will not request them without the right HTTP header. This directive is already enabled by default in `.htaccess` file. In case you want to restrict access, you need to change these lines from:

```
<IfModule mod_headers.c>
  <FilesMatch "\.(ttf|ttc|otf|eot|woff|font.css)$">
    Header set Access-Control-Allow-Origin "*"
  </FilesMatch>
</IfModule>
```

To the following code snippet:

```
<IfModule mod_headers.c>
  <FilesMatch "\.(ttf|ttc|otf|eot|woff|font.css)$">
    Header set Access-Control-Allow-Origin "http://example.com"
  </FilesMatch>
</IfModule>
```

Replace `http://example.com` with the domain name you would like to specifically allow access to the Webfonts.

 If you would like to get a good overview of how CORS-enabled image, Webfont, and AJAX requests work and differ from `crossdomain.xml`, you should read the HTML5security project wiki page at code.google.com/p/html5security/wiki/CrossOriginRequestSecurity.

Using other server configuration files

We have seen how to use the features available in the Apache `.htaccess` file that HTML5 Boilerplate comes with. But there is a repository of configuration files for other kinds of servers such as Nginx, Node, Google App Engine, IIS, and Lighttpd. The following table contains the configuration filenames and their corresponding server software:

| Config filename | Server software |
|---|--|
| <code>.htaccess</code> | Apache Web server at httpd.apache.org/docs/2.2/howto/htaccess.html . |
| <code>Web.config</code> | IIS Web server from learn.iis.net/page.aspx/376/delegating-configuration-to-webconfig-files/ . |
| <code>Node.js</code> | Node Web server from nodejs.org . |
| <code>Nginx.conf</code> | Nginx server at wiki.nginx.org/Configuration . |
| <code>Lighttpd.conf</code> | Lighttpd server at redmine.lighttpd.net/projects/lighttpd/wiki/TutorialConfiguration . |
| <code>App.yaml</code> and <code>gae.py</code> | Google App Engine at code.google.com/appengine/docs/python/config/appconfig.html . |

The configuration files for these servers are available at github.com/h5bp/server-configs.

web.config

HTML5 Boilerplate's `web.config` file is used to configure options for your site running on an IIS7 server or higher.

As with the `.htaccess` file, merely having it in the root folder of your website allows it to be recognized and used to configure an IIS7 server.

lighttpd.conf

As with the other configuration files, place it in the root folder for the Lighttpd server to configure the server.

nginx.conf

Nginx is a lightweight server popular with websites using the Ruby on Rails framework.

As with the `.htaccess` file, place this file in the root folder of your website. In addition, make sure `nginx-mime.types` is also in the root folder. This file is required for Nginx to make sure it sends each file with the right MIME type.

node.js

With the `node.js` configuration file, the usage is different. The configuration file assumes you are using the Express/Connect framework for managing resource requests for your app. In your server-side app code, you can use the following to start the server:

```
var h5bp = require('h5bp');
var app = express.createServer();
app.use(h5bp.server());
app.listen(3000);
```

This requires you to install the `h5bp` package using **Node Package Manager (NPM)** and the `express` package using the same. The `h5bp` package has a slew of configurations that will be used when the server is started. If you wish to use only some specific configurations, you can pass them in as options to the server function, as shown in the following code snippet:

```
app.use(h5bp.server({
  server: true,
  setContentType: true,
  removeEtag: true
}));
```


Google App Engine

Some websites are also served from Google App Engine (<http://code.google.com/appengine/>), which requires your website's backend to be written in Java, Python, or Go.

You need to ensure that the `app.yaml` file is in the root folder of your website.

The following table contains the summary of all the features available in HTML5 Boilerplate server configurations:

| Feature name | Apache | Nginx | IIS | Lighttpd | Node.js | Google App Engine |
|--|--------|-------|-----|-------------------|---------|-------------------|
| ETags | Yes | Yes | Yes | Yes | No | No |
| Gzip | Yes | Yes | Yes | Yes | Yes | Yes |
| Expires header | Yes | No | No | No | Yes | No |
| Custom 404 page | Yes | Yes | Yes | No | No | No |
| Forcing the latest IE version | Yes | Yes | Yes | Yes | Yes | Yes |
| Using UTF-8 encoding | Yes | Yes | Yes | No | No | No |
| Serving the right MIME types | Yes | Yes | Yes | Yes | No | Yes |
| Blocking access to hidden folders | Yes | No | No | No | Yes | No |
| Blocking access to backup and source files | Yes | No | No | Yes (only ~&.inc) | Yes | No |
| Stop advertising server information | No | No | Yes | No | Yes | No |
| Starting Rewrite Engine | Yes | No | No | No | No | No |
| Preventing 404 errors for non-existing redirected folder | Yes | No | No | No | No | No |
| Suppressing or forcing the "www." at the beginning of URLs | Yes | No | Yes | No | Yes | No |
| Setting cookies from iFrames | Yes | No | Yes | No | No | No |
| PHP security defaults | Yes | No | Yes | No | No | No |
| Stop advertising Apache version | Yes | No | No | No | No | No |

| Feature name | Apache | Nginx | IIS | Lighttpd | Node.js | Google App Engine |
|---|--------|-------|-----|----------|---------|-------------------|
| Allowing concatenation from within JS and CSS files | Yes | No | Yes | No | No | No |
| Stopping screen flicker in IE on CSS rollovers | Yes | No | Yes | No | No | No |
| Preventing SSL certificate Warnings | Yes | No | Yes | No | No | No |
| Cross-domain AJAX requests | Yes | No | Yes | No | Yes | No |
| CORS-enabled Images | Yes | No | No | No | No | No |
| Webfont Access | Yes | No | No | No | No | No |

Summary

We dived deep into the internals of serving pages over several servers and configuration files for some of the major servers. We looked at some of the good defaults provided out of the box and some optional ones that you can enable with careful understanding.

Now that our site is almost ready to get out of the door, we shall look at some of the other ways to make it better.

6

Making Your Site Better

The nature of website design and development is such that not all optimizations and recommendations apply in all scenarios. In this chapter, we will look at the various optimization tools available and which situations they are best suited for, to make an HTML5 Boilerplate site load and render faster.

Finding the best experience for Internet Explorer

Internet Explorer versions 8 and below have had very haphazard support for standards and consistent rendering. Depending on the number of users who visit your site using Internet Explorer, you may or may not want to spend the effort optimizing for Internet Explorer.

Mobile-first styles for IE

Media Queries are CSS features that allow you to apply different sets of rules depending on the value of a particular media feature. For example, if the browser has minimum width of 500 pixels, you could make all your h1 elements turn red, as shown in the following code:

```
@media only screen and (min-width: 500px) {  
  h1 { color: red; }  
}
```

However, IE6, IE7, and IE8 do not understand media queries that are typically used to adjust widths according to different screen widths. As such, they will never render the optimized styles you create for browsers with screen widths that match a certain media query break point (`min-width: 500px` in the previous snippet). In our Sun and Sand Music Festival website, we have style rules within three different media queries, as shown in the following code snippet:

```
@media only screen and (max-width: 300px) { /*CSS rules */ }

@media only screen and (max-width: 750px) { /*CSS rules */ }

@media only screen and (max-width: 1150px) { /*CSS rules */ }
```

This means IE6, IE7, and IE8 will render the styles as though these queries did not exist! If you specify rules for device widths that are smaller at the end, it is likely that those will be overriding the rules for device widths that are larger, leading to less than optimal designs on Internet Explorer 8 and below.

Ideally, in this situation, you simply want IE to render all the styles and have the user scroll, if necessary, so that the style rules for the largest widths always apply. To do this, we can create a separate `ie.css` file that will render all the rules within `main.css` except these rules will no longer be contained in media queries.

Doing this manually is hard work, and almost impossible to maintain. However, Nicolas Gallagher writes about an elegant solution he invented, which uses Sass to import separate stylesheets for each media query breakpoint and compile them into two separate stylesheets; one without media queries (`ie.css`) and the other with media queries (`main.css`); we will look at this next.

ie.scss

The code snippet for `ie.scss` is as follows:

```
@import "300-up";
@import "750-up";
@import "1150-up" /* Make sure largest is last */
```

main.scss

The code snippet for `main.scss` is as follows:

```
@import "base";
@media (min-width:300px) {
  @import "300-up"; }
```

```
@media (min-width:750px) {
  @import "750-up"; }
@media (min-width:1150px) {
  @import "1150-up"; }
```

Do note that you need each file titled `300-up.scss`, `750-up.scss`, and `1150-up.scss` within the same parent folder as `main.scss` and `ie.scss`.

In the head tag of `index.html` page, you can now write the following code:

```
<!--[if (gt IE 8) | (IEMobile)]><!-->
<link rel="stylesheet" href="/css/style.css">
<!--<![endif]-->

<!--[if (lt IE 9) & (!IEMobile)]>
<linkrel="stylesheet" href="/css/ie.css">
<![endif]-->
```



Jake Archibald also has a far more easy-to-write solution using Sass at jakearchibald.github.com/sass-ie/. It takes advantage of newer features of Sass 3.2, and has slightly different composition for `main.scss` and `ie.scss`. It requires advanced knowledge of Sass, which is beyond the scope of this book.

Printing with jQuery in IE6 and IE7

IE6 and IE7 do not support the `:after` pseudo selector that all other browsers support. This means our print stylesheet that provides a feature for all links to be printed alongside the linked text will not work in IE6 and IE7. You can simply use jQuery code to overcome this.



Bill Beckelman has written a post about this on his blog at beckelman.net/2009/02/16/use-jquery-to-show-a-links-address-after-its-text-when-printing-in-ie6-and-ie7/. IE supports its own proprietary `onbeforeprint` and `onafterprint` events that can be used to our advantage. Based on Bill Beckelman's work, we can write our own simple jQuery code to print link URLs in IE6 and IE7.

First, we check if `window.onbeforeprint` exists, because this would indicate this code is being executed on one of the IE browsers. We also want to verify if this browser supports generated content or not, as we only need to use this code when it is not supported. The following code snippet checks for the presence of the `window.onbeforeprint` event:

```
if (Modernizr.generatedcontent == false &&window.onbeforeprint !==
undefined) {
```

Then, we set functions to execute when either `onbeforeprint` or `onafterprint` occurs, as shown in the following code:

```
window.onbeforeprint = printLinkURLs;
window.onafterprint = hideLinkURLs;
```

Then, we write the following functions:

```
functionprintLinkURLs() {
$("a[href]").each(function() {
$this = $(this);
$this.data("originalText", $this.text());
$this.append(" (" + $this.attr("href") + ")");
});
}

functionhideLinkURLs() {
$("a[href]").each (function() {
$(this).text($(this).data("originalText"));
});
}
```

Styling disabled form elements in Internet Explorer

Internet Explorer up to version 9 has no way to indicate a form field is disabled other than the color of the text used in that field. Sometimes, a field simply has an icon rather than text (or it could be an empty input textbox), in which case it is almost impossible to discern which buttons are disabled and which are not.

For Internet Explorer 7 and above, by just adding the following rules in `main.css`, you can get the disabled fields to display significantly differently from the enabled ones:

```
.lt-ie9 input[type='text'][disabled],
.lt-ie9 textarea[disabled] {
```

```
background-color: #EBEBE4;
}
```

If you have to support Internet Explorer 6, then make sure you add a class called `disabled` on the form elements that have the `disabled` attribute set and alter the previous rule to the following:

```
.lt-ie9 input.disabled,
.lt-ie9 textarea.disabled {
background-color: #EBEBE4;
}
```

Suppressing IE6 image toolbar

In IE6, all images, have a toolbar visible when hovered over. You can disable them by adding the following code within the `head` tag in the `index.html` file:

```
<metahttp-equiv="imagetoolbar" content="false">
```

Writing CSS3 easier with tools

CSS3 is at the bleeding edge. Some properties require what is known as a vendor prefix. For example, the 3D transforms property `perspective` is implemented as follows, in different browsers:

```
-webkit-perspective //Safari, Chrome
-ms-perspective // Internet Explorer
perspective // Firefox
```

Only a short while ago, Firefox implemented this property as `-moz-perspective`, but have since dropped support for the `-moz-` prefix.

As you will come to realize, it is really hard to keep track of which browser requires a prefix and which browser does not, and it is not quite feasible to keep all the sites that we create updated on a regular basis every time a browser adds or drops support for a prefix.

To make this easier, we could use abstractions without these prefixes such that a tool that has an updated index of which property requires which prefix could convert them into the required final CSS.

This is exactly what Sass (sass-lang.com) or Less (lesscss.org) provide. Sass is a language that comes with a compiler that converts code written in Sass to CSS, in Ruby. Less is a similar language, but written in JavaScript.

In both cases, the languages are extensions of the syntax used in CSS, which means you can copy your existing CSS files into Sass or Less files and have them compile into pure CSS files without any errors.

The extra features that these languages provide are the ability to use mixins, variables, functions, and more.

For Sass, **Compass** is an additional framework that provides a ready library of CSS3 mixins found at compass-style.org/reference/compass/css3. Less has many options; the most popular and frequently updated can be found within Twitter Bootstrap and are available at twitter.github.com/bootstrap/less.html#mixins. The following sections show you how to create a rule that uses CSS transforms in Sass and Less.

Sass

The code snippet for Sass is as follows:

```
.btn-arrow {
  @include transform(scale(2));
}
```

Less

The code snippet for Less is as follows:

```
.btn-arrow {
  .scale(2);
}
```

Output CSS

The output CSS would be as follows:

```
.btn-arrow {
  -webkit-transform: scale(2);
  -moz-transform: scale(2);
  -ms-transform: scale(2);
  -o-transform: scale(2);
  transform: scale(2);
}
```

Converting HTML5 Boilerplate CSS to Sass or Less

You could typically just rename the `main.css` file to `main.scss` or `main.less` and start using that as your base Sass or Less file. To compile these files to the corresponding Less or Sass files, you can either use GUI-based browser refreshing software that compiles these files automatically like **LiveReload** (livereload.com/) or **Codekit** (incident57.com/codekit).

If you are someone familiar with the command line, you can install Less or Sass and run their respective command-line interpreters to compile the files into pure CSS.

If you wish to use a pure Sass or Less file to start with (instead of the contents of the `main.css` file), there are also forks of HTML5 Boilerplate that have the stylesheet converted to Sass. We will see two of them in the following sections.

HTML5 Boilerplate Compass extension

There is a Compass extension that is available for use with Compass at github.com/sporkd/compass-html5-boilerplate. Note that it is not as frequently updated as the `main.css` file you find in HTML5 Boilerplate. This is extensively modularized and splits the `main.css` file into multiple Sass files. The CSS comments are also removed from the resulting CSS file.

HTML5 Boilerplate Sass fork

There is a Sass fork of the `main.css` that is frequently updated at github.com/grayghostvisuals/html5-boilerplate/tree/h5bp-scss that you can use, if all you want is a base Sass file to start from. This version uses Sass variables but does not split the file into individual files.

Unfortunately, there is no up-to-date Less fork of HTML5 Boilerplate. However, you can rename the `main.css` to `main.less` and then use it as a Less file.

Print considerations

If your web page is likely to be printed, you might want to consider using colors that are printable. Some browsers consider some colors too light to print and force a darker version of the color for printing; merttol.com/articles/code/too-light-for-print.html has more details on this interesting quirk.

Appendix, You Are an Expert, Now What?, covers the reasoning and rationale behind the print styles in great detail.

Finding and using polyfills

Most of HTML5 and CSS3 features have differing levels of support in different browsers, hence, either use JavaScript code to mimic these features in browsers that do not support them or provide an altering view. Such snippets of code are called **polyfills**.

I help maintain html5please.com which is an opinionated list of polyfills for some popular HTML5 and CSS3 features.

Beware of the performance penalty of using a lot of polyfills on a browser that does not support a lot of features.

When you do use your polyfills, make sure you use Modernizr's `load` function like we did for the audio polyfill for Sun and Sand Music Festival website in *Chapter 4, Adding Interactivity and Completing Your Site*. This would prevent unnecessary loading of polyfills on browsers that support the features you want to use.

A comprehensive list of all kinds of polyfills is available on the Modernizr Wiki at github.com/Modernizr/Modernizr/wiki/HTML5-Cross-browser-Polyfills.

Making your site faster

If your pages use a lot of resources such as images, then perhaps it would be wise to prefetch those resources so your page loads faster. **DNS prefetching** would be a way to do that.

DNS prefetching

DNS prefetching informs the browser of resources on other domain names that are referred to within the page early on during the page load, so it can do the DNS resolution of these domain names.

A browser has to look up a domain name on a **Domain Name Server (DNS)** to figure out where it is located on the Internet. Sometimes, it has to go through multiple layers of Domain Name Servers and it could be very slow and is not always consistent. By using DNS prefetching, even before a user clicks a link, or loads a resource, the DNS resolution for that particular domain name is done and the resource can be fetched much faster.

Google states that this saves about 200 milliseconds on a resource hosted on an external domain name.

If you host your assets on a **Content Delivery Network (CDN)** like Amazon's S3 or even if you refer to Google's API or Microsoft's API CDN, it would be faster to get these files when they are prefetched.

DNS prefetching is invoked by writing the following code within the head tag of a HTML file:

```
<link rel="dns-prefetch" href="//image.cdn.url.example.com">
```

Browsers that understand prefetching would immediately start attempting to resolve the DNS for the link within the href attribute. The following is how it would look for Amazon S3:

```
<link rel="dns-prefetch" href="//s3.amazonaws.com">
```

Currently, Firefox 3.5 and higher, Safari 5 and higher, and IE9 and higher, support DNS prefetching.

Making your site more visible on search engines

While the content of your website matters the most, making sure everything else supports better visibility of the content on search engines is important too. The following sections explain some ways you can do this.

Directing search spiders to your site map

Site maps inform search engines of the existence of pages within your site that are otherwise not discoverable; perhaps they are not linked to from other pages on your site, or from external sites.

Some CMSs provide plugins to generate site maps, listed at code.google.com/p/sitemap-generators/wiki/SitemapGenerators, or you can write one yourself using the guidelines at www.sitemaps.org/protocol.html.

Once you have written your site map, you can let search engine spiders discover it when they crawl your website if you add a link to the sitemap by using the following:

```
<linkrel="sitemap" type="application/xml" title="Sitemap" href="/sitemap.xml">
```

You can also submit the site map to individual search engines instead of linking to the site map within the HTML page, if you would like to make your page as small as possible.

Implementing X-Robots-Tag headers

You will likely sometimes have a staging server, such as `staging.example.com` for your site `example.com`. If an external site links to the files on the staging server (say you were asking a question about some feature not working on a forum and link to the staging server), it is likely to be indexed by search engines even though the domain name does not figure in the `robots.txt` file or does not hold a `robots.txt` file.

To prevent this, you can add X-Robots-Tag HTTP header tags by appending and uncommenting the following code snippet to the `.htaccess` file on the staging server:

```
# -----
# Disable URL indexing by crawlers (FOR DEVELOPMENT/STAGE)
# -----

# Avoid search engines (Google, Yahoo, etc) indexing website's content
# http://yoast.com/prevent-site-being-indexed/
# http://code.google.com/web/controlcrawlindex/docs/robots_meta_tag.html
# Matt Cutt (from Google Webmaster Central) on this topic:
# http://www.youtube.com/watch?v=KBdEwpQRD0

# IMPORTANT: serving this header is recommended only for
# development/stage websites (or for live websites that don't
# want to be indexed). This will avoid the website
# being indexed in SERPs (search engines result pages).
# This is a better approach than using robots.txt
# to disallow the SE robots crawling your website,
# because disallowing the robots doesn't exactly
# mean that your website won't get indexed (read links above).

# <IfModule mod_headers.c>
#   Header set X-Robots-Tag "noindex, nofollow, noarchive"
#   <FilesMatch "\.(doc|pdf|png|jpe?g|gif)$">
#     Header set X-Robots-Tag "noindex, noarchive, nosnippet"
#   </FilesMatch>
# </IfModule>
```

Trailing slash redirects

Search engines consider folder URLs `http://example.com/foo` and `http://example.com/foo/` as two different URLs and as such would consider the content to be duplicates of each other. To prevent this, rewrite the URLs either to change `http://example.com/foo` to `http://example.com/foo/` or `http://example.com/foo/` to `http://example.com/foo`.

The way we do this is to edit the `.htaccess` file for Apache server and add the following rewrite rules (see *Chapter 5, Customizing the Apache Server*, for details on how we edit `.htaccess` files).

Option 1: Rewrite `example.com/foo` to `example.com/foo/`

The following code snippet helps us to rewrite `example.com/foo` to `example.com/foo/`:

```
RewriteCond %{REQUEST_FILENAME} !-f
RewriteCond %{REQUEST_URI} !(\.[a-zA-Z0-9]{1,5}|/|#(.*))$
RewriteRule ^(.*)$ $1/ [R=301,L]
```

Option 2: Rewrite `example.com/foo/` to `example.com/foo`

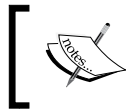
The following code snippet helps us to rewrite `example.com/foo/` to `example.com/foo`:

```
RewriteRule ^(.*)/$ $1 [R=301,L]
```

If you have existing rewrite rules, perform the following steps to make sure you set up your rewrite rules correctly. Not doing so can cause incorrect redirects and 404 errors.

- Keep a backup: Back up the `.htaccess` file you are going to add redirects to, before you start adding them. This way you can quickly go back to the backup file if you are unable to access your site because of an error in the `.htaccess` file.
- Do not append or replace existing rewrite rules: Instead of appending or replacing existing rules from CMSes you are using, merge them within.

- Watch the order of rewrite rules: Make sure you add the slash first and then your existing rules, which might rewrite the end paths.
- Confirm the RewriteBase path: If your website is in a subfolder, ensure you have set the right RewriteBase path for your rewrite rules. If you have a working RewriteBase path, do not remove it.



Finally, consider implementing guidelines from Google's *SEO Starter Guide* at <http://googlewebmastercentral.blogspot.com/2008/11/googles-seo-starter-guide.html>.

Handling users without JavaScript

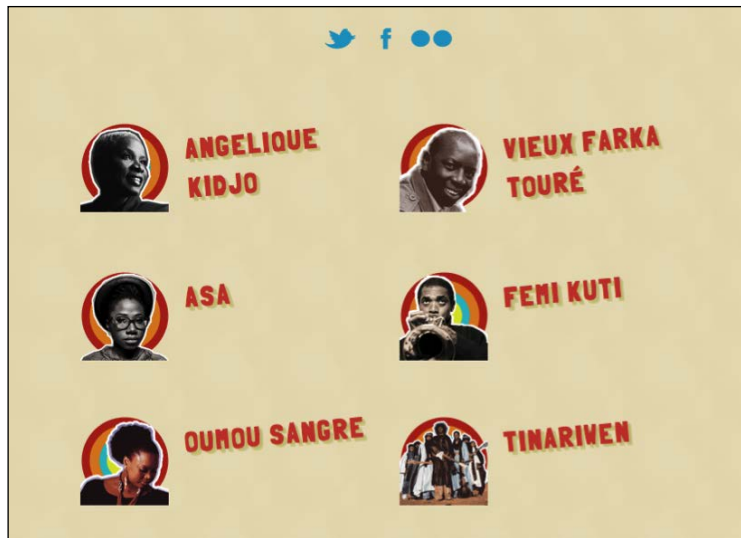
HTML5 Boilerplate provides a class called `no-js` that gets replaced with a class called `js`, when JavaScript is detected by Modernizr on the `html` tag. Using this class name, you can craft the style for how the website should look when JavaScript is disabled.

In our Sun and Sand Festival website, when JavaScript is not enabled, clicking on the **Day 2** link produces nothing.

You can view how the site works when JavaScript is disabled on various browsers, in the following ways:

- **Firefox:** Go to **Preferences**, click on **Content**, and then uncheck the **EnableJavaScript** checkbox.
- **Chrome:** Download the **Chrome Web Developer** extension and disable JavaScript from within the extension.
- **Safari:** Click the **Disable JavaScript** menu item on the **Develop** menu. You can see the **Develop** menu when you check the **Show Develop** toolbar on the **Advanced** tab in the Safari **Preferences** pane.
- **Internet Explorer:** Click the **Internet Options** in the **Settings** menu, and click **Custom Level** and check the **Disable in the Active scripting** menu.
- **Opera:** Click **Quick Preferences** and unselect the **Enable JavaScript** option.

Let us make sure that the tabs do not render when JavaScript is not available, and make sure the whole listing is displayed at the same time, as shown in the following screenshot:



We can do this by editing `main.css` to take advantage of the `no-js` class. First, we need to remove the tabbed navigation, as shown in the following code:

```
.no-js .t-tab__nav {
  display: none;
}
```

Then, we need to make the two lists be positioned statically instead of being absolutely positioned one below the other, as shown in the following code:

```
.no-js .t-tab__body {
  position: static;
}
```

We need to make sure the `hidden` class does not get applied in this specific instance for **Day 2**, so we can see all the artists at once, as shown in the following code:

```
.no-js .t-tab__body.hidden {
  display: block !important;
  visibility: visible;
}
```

Now, when you enable JavaScript again, you will notice that the tabbed navigation appears and everything functions as you expected.

Optimizing your images

Every resource you add to your page is an extra request to the server and extra network trip for the browser before it declares the page complete. Network requests are typically the slowest components of a page load. This is especially obvious on mobile devices when surfing websites on 3G or even lower connections. The smaller your files are, the faster they will reach the browser.

If you can avoid using a large image, you are better off not using them.

8-bit PNGs

If you are considering using a GIF format for your images, always use PNG. PNG formats for images are far less heavy and much smaller. Even further, 8-bit PNGs are significantly smaller in size.

If you are using PNGs, you should use PNG-8 with a full alpha channel that gives you compatibility all the way back to IE6. Ensure you verify the final output to ensure they are not too grainy or pixelated.

Tools for image optimization

There are build tools in HTML5 Boilerplate that will optimize images, which we will look into in the next chapter. There are also standalone tools that are worth looking at, when you want to compress a bunch of images in one go. If you wish to upload your images and optimize them, you can do so at smushit.com/ysmush.it/.

ImageAlpha

If you have 24-bit PNG images, you can convert them into 8-bit PNGs that have a full alpha channel with this tool that you can download from pngmini.com. It is only for Mac OS X.

GUIs and command-line tools that work on other Operating Systems are outlined at pngquant.org.

ImageOptim

If you would like to optimize images of various formats in one go, **ImageOptim** would be your best tool of choice. You can download this from imageoptim.com. This is also for Mac OS X only, and takes advantage of several tools to perform these optimizations.

If you would like to use something similar on other systems, you can download the specific tool you need for each image format. The following table lists tools for some of the popular image formats:

| Format | Tool |
|--------------|--|
| Animated GIF | Gifsicle www.lcdf.org/gifsicle/ |
| JPEG | Jpegtran jpegclub.org/ |
| PNG | Pngcrush pmt.sourceforge.net/pngcrush/ |
| | Imageworsener entropy.mine.com/imageworsener/ |
| | Optipng optipng.sourceforge.net/ |
| | PNGOUT advsys.net/ken/utills.htm |

If you would like to learn more about using these tools for optimization, read more on the slides by Stoyan Stefanov about image optimization for the Web at www.slideshare.net/stoyan/image-optimization-for-the-web-at-phpworks-presentation. There are even more clever optimizations that could be done for PNG and JPEG image formats, detailed over on *Smashing Magazine* at www.smashingmagazine.com/2009/07/15/clever-png-optimization-techniques/ and <http://www.smashingmagazine.com/2009/07/01/clever-jpeg-optimization-techniques/> respectively.

Using image sprites

Network requests take a long time to make for each resource. To make these smaller, you could combine multiple image files into one single image file that needs to be requested once and be cached for a very long time so that the page loads significantly faster. This is especially useful if your page is going to be viewed on devices with very low bandwidth connections to the Internet.

This means, you combine multiple images in one large image and use the CSS background property on all selectors where these images would be used. Let us convert all our artist images into a big sprite and replace the image elements into background images.

The following is our final sprite:



Let us replace our image elements in `index.html`, like the following one:

```

```

With the following:

```
<i class="t-artist__image artist-tinariwen"></i>
```

We do this for each of the artists. Then, in our `style.css`, we add the following code snippet:

```
.t-artist__image {
background: url(../img/artists-image.png) top left no-repeat,
url(../img/bg-artist.png) no-repeat center center;
float: left;
display: block;
}
.artist-asa { background-position: -0px -0px, 0 0; }
.artist-kidjo { background-position: -0px -100px, 0 0; }
.artist-kuti { background-position: -100px -0px, 0 0; }
.artist-sangre { background-position: -100px -100px, 0 0; }
.artist-tinariwen { background-position: -200px -0px, 0 0; }
.artist-toure { background-position: -200px -100px, 0 0; }
```

Nothing has changed in the final page, except we have now reduced the number of network requests to 1 instead of 6 for these images. By optimizing the final sprite, we can make this request even faster.

Generating a sprite would seem like a lot of work, but there are many tools to help with this.

CSS sprites from within Adobe Photoshop

Using the instructions documented at arnaumarch.com/en/sprites.html, you can use a script file from Photoshop to select a folder of images and also generate the associated CSS file using images within these files positioned and corrected as a background image.

There are some things to note when using this tool, as explained in the following points:

- Make sure the folder only contains the images you want to add to a sprite
- The resulting CSS file is generated within the folder used to create the sprite
- The generated sprite is opened in Adobe Photoshop and you are required to crop it before you save it out as an image at the location of your choice

CSS sprites with Compass

Compass – the framework on top of Sass – can stitch your images together at compile time and have the images referenced in your Sass file, turned into a sprite in the resulting CSS file.

All you need to do is to make sure you set up a folder within your images folder, such that you have the right names for each of the images, as described in the following list (taken from Compass documentation):

- `images/my-icons/new.png`
- `images/my-icons/edit.png`
- `images/my-icons/save.png`
- `images/my-icons/delete.png`

The name `my-icons` can be any name you prefer. Then in the Sass file, use the following code:

```
@import "my-icons/*.png";
@include all-my-icons-sprites;
```

Use the same name you used instead of `my-icons` in the previous step. Presto!
You are done! Compass generates a CSS file that has the following code:

```
.my-icons-sprite,  
.my-icons-delete,  
.my-icons-edit,  
.my-icons-new,  
.my-icons-save { background: url('/images/my-icons-s34fe0604ab.png')  
no-repeat; }  
  
.my-icons-delete { background-position: 0 0; }  
.my-icons-edit { background-position: 0 -32px; }  
.my-icons-new { background-position: 0 -64px; }  
.my-icons-save { background-position: 0 -96px; }
```

Now, use the appropriate class name in your markup to add the appropriate image to your element.

SpriteMe

SpriteME, available at spriteme.org/, is a bookmarklet that analyses images used on a page and creates sprites out of them. If you have an existing site to convert to using sprites, this would be a great place to start from.

Augmenting Google Analytics

Google Analytics can track several kinds of data and here are some easy, obvious augments you can make to your Analytics data.

Adding more tracking settings

Google Analytics gives you a number of optional settings to track, which you need not use the `.push()` method on; instead you can directly append to the initial array. Instead of the following:

```
var _gaq = _gaq || [];  
_gaq.push(['_setAccount', 'UA-XXXXX-X']);  
_gaq.push(['_trackPageview']);
```

You can do the following:

```
var _gaq = [['_setAccount', 'UA-XXXXX-X'],['_trackPageview']];
```

Anonymize IP addresses

In some countries, no personal data may be transferred outside jurisdictions that do not have similarly strict laws (that is, from Germany to outside the EU). Thus, a webmaster using the Google Analytics script may have to ensure that no personal (trackable) data is transferred to the U.S. You can do that with the `_gat.anonymizeIp` option. In use it looks like the following:

```
var _gaq = [['_setAccount', 'UA-XXXXX-X'], ['_gat._anonymizeIp'], ['_trackPageview']];
```

Tracking jQuery AJAX requests in Google Analytics

Steve Schwartz writes about a simple script you can use in the `plugins.js` that will allow you to track jQuery AJAX requests at www.alfajango.com/blog/track-jquery-ajax-requests-in-google-analytics. The following code snippet shows that script:

```
/*
 * Log all jQuery AJAX requests to Google Analytics
 * See: http://www.alfajango.com/blog/track-jquery-ajax-requests
-in-google-analytics/
 */
if (typeof _gaq !== "undefined" && _gaq !== null) {
  $(document).ajaxSend(function(event, xhr, settings){
    _gaq.push(['_trackPageview', settings.url]);
  });
}
```

Tracking JavaScript errors in Google Analytics

If you want to track JavaScript errors on your page using Google Analytics, it is possible to do so with the following script, which you need to add after the Google Analytics variable `_gaq` has been defined in the `index.html` page:

```
(function(window){
var undefined,
link = function (href) {
var a = window.document.createElement('a');
```

```
a.href = href;
return a;
};
window.onerror = function (message, file, row) {
var host = link(file).hostname;
_gaq.push([
    '_trackEvent',
    (host == window.location.hostname || host == undefined || host
== '' ? '' : 'external ') + 'error',
message, file + ' LINE: ' + row, undefined, undefined, true
]);
};
}(window));
```

Summary

In this chapter, we looked at how we can provide a better experience for users of Internet Explorer. We also considered very briefly some tools that can help us write more efficient and robust stylesheets that are easier to maintain in the light of cutting edge developments in CSS. We looked at how to use polyfills and write pages that are faster to load and more secure in general. We looked in detail at how to render the Sun and Sand website when JavaScript is disabled and also stitched the artists' images together into a sprite and saved on several network requests.

In the next chapter, we will look at automating deployment of our site using the build script that HTML5 Boilerplate provides.

7

Automate Deployment With the Build Script

We are ready to deploy our site! But before we do that, we should ensure we minimize all our scripts and optimize the images, so that these pages load as quickly as possible anywhere in the world. We can automate these tasks by executing a script on the command line. Let us look at the options we have.

The build script

Once you are done with your project, you would like to generate files that strip comments and are optimized for loading quickly. There are software build systems that are typically used in software projects with similar goals. HTML5 Boilerplate's build scripts provide tasks scoped to what a typical web development project would need.

The script should be used only after you have confirmed your project is ready for deployment and it has been well tested. The build script merely automates the process of removing comments, optimizing files, and making sure the files are production ready.

There are currently two kinds of build scripts that are actively maintained by HTML5 Boilerplate contributors; these are explored in the following section.

Ant build script

The Ant build script is a set of files that work on top of the Apache Ant build system (ant.apache.org/) that has been available since the early days of HTML5 Boilerplate. It offers a variety of options, described as follows:

- Publishes files to test, development, and production environments
- Checks syntax and code quality of your script files with **JSHint** or **JSLint**, or your stylesheets with **CSSLint**
- Concatenates and minifies all your JavaScript files into a single file and updates the HTML pages with reference to this new file
- Cleans up and tidies HTML markup by removing comments, whitespaces, and compressing inline styles and scripts
- Concatenates and minifies all your stylesheets and updates the HTML pages with reference to the new file instead of multiple CSS files
- Compiles style preprocessor files such as Less or Sass into the resulting CSS stylesheets and updates references in HTML pages
- Optimizes PNG and JPEG images within the `img` folder using OptiPNG from optipng.sourceforge.net/ and JPEGTran from jpegclub.org/jpegtran/ respectively
- Builds documentation from your scripts using JSDoc3 from github.com/jsdoc3/jsdoc

Node build script

A new build script that builds on top of Node, found at nodejs.org/, is under active development. While it is not out for production use yet, it offers a lot of tasks that are similar to the Ant build script with some new features described as follows:

- Concatenates and minifies all your JavaScript files into a single file and updates the HTML pages with reference to this new file
- Concatenates and minifies all your stylesheets and update the HTML pages with reference to the new file instead of multiple CSS files
- Cleans up and tidy HTML markup by removing comments, whitespaces, and compressing inline styles and scripts

- Optimizes PNG and JPEG images within the `img` folder using OptiPNG and JPEGTran respectively

Watch the project files for changes, and automatically run the build script and reload open pages in browsers when they do.

Which build script to use?

Depending on what platforms you are comfortable with, you can choose one over the other. Both the build scripts are stable enough to use to deploy your production files, so your choice is down to what you are most comfortable using.

If you already have Ant installed, the Ant build script might be an obvious choice. If you find yourself using Node frequently or using it in your projects, then the Node build script could be a good start. In this chapter, we will look at using both, so you can become comfortable with either of them.

Using the Ant build script

First, confirm you have Ant installed on your system by entering the following in your command-line tool:

```
ant-version
```

If you do not have Ant, install it first before proceeding to the next step.



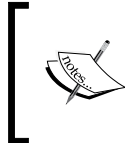
Ant is installed by default on Macs, while it is available as a package to install on most Linux platforms. For Windows, installing Ant is slightly more complicated. You need to install the Java SDK from www.oracle.com/technetwork/java/javase/downloads/index.html and then download WinAntcode.google.com/p/winant/ and point the installer to `Program Files/Java/jre6/bin/`.

Next, you need to install **ant-contrib**, a utility that makes a lot of functionality available for Ant that HTML5 build script uses. **WinAnt** does this automatically, when you use it to install Ant on Windows. However, for Linux users, you can use **yum** to install it as a package. On Mac, you can install MacPorts (www.macports.org/install.php) and then enter the following in your command-line tool (typically Terminal):

```
sudo port install ant-contrib
```

Finally, ensure that the image optimization tools are installed. For Mac users, you need to make sure you have **jpegtran** (www.ijg.org/) and **optipng** (optipng.sourceforge.net/) installed and on your path. You can install these two files by entering the following in your command-line terminal:

```
sudoport install jpeg optipng
```



The PATH is an environmental variable that holds a list of folders that your command-line interface searches through when you enter a command. You can learn about how to add folders to the path from www.cs.purdue.edu/homes/cs348/unix_path.html.

If you are on Windows, the Ant build script project contains the required binaries for these image tools for you to install.

Installing the build script

In Terminal (or your command-line tool), we will navigate to our project folder and install the build script using Git, as shown in the following screenshot:

```
chapter-7 — bash — 80x24
manian at manian-MacBookAir in ~/Documents/projects/chapter-7
$ git clone git://github.com/h5bp/ant-build-script.git
```

We now have to rename the build script folder from ant-build-script to build before we continue. This is done by using the following command:

```
mv ant-build-script build
```

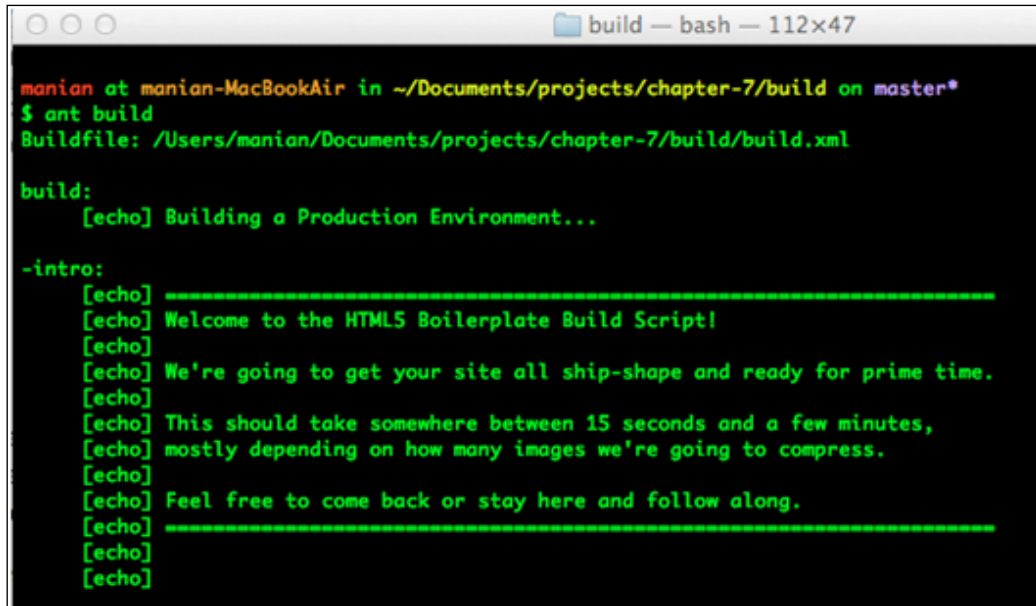
Once that is done, let us navigate to the build script folder by using the following command:

```
cd build
```

Now, we shall execute the build script! Go to your command-line tool and enter the following:

```
ant build
```

If you set up your build script folder correctly, then you should get the following screen:

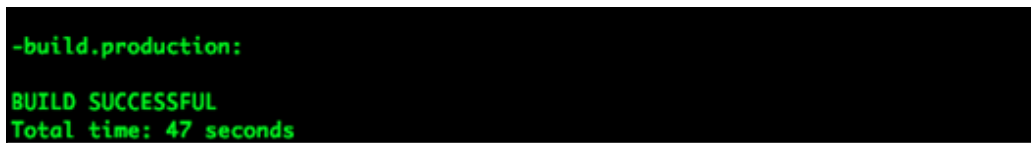


```
manian at manian-MacBookAir in ~/Documents/projects/chapter-7/build on master*
$ ant build
Buildfile: /Users/manian/Documents/projects/chapter-7/build/build.xml

build:
  [echo] Building a Production Environment...

-intro:
  [echo] -----
  [echo] Welcome to the HTML5 Boilerplate Build Script!
  [echo]
  [echo] We're going to get your site all ship-shape and ready for prime time.
  [echo]
  [echo] This should take somewhere between 15 seconds and a few minutes,
  [echo] mostly depending on how many images we're going to compress.
  [echo]
  [echo] Feel free to come back or stay here and follow along.
  [echo] -----
  [echo]
  [echo]
```

Then, after the tasks are executed, you should get the following output:



```
-build.production:
BUILD SUCCESSFUL
Total time: 47 seconds
```








Now, you have a brand new `publish` folder where the optimized files are stored. Let us look at what all have been optimized, by opening the `index.html` page from the `publish` folder, in Chrome browser and using the Chrome Developer Tools' **Network** tab to observe the files that are loaded and their associated sizes.








Please note that you must load the page with the **Network** tab open to log the files being requested.

Smaller image files

The **Network** tab records all images that are fetched for use on `index.html`. We can see that the images that are fetched for the `index.html` page in the **publish** folder are noticeably smaller in size.

In the following screenshot, the bottom section of the screenshot shows the list of images in the **publish** folder, which are noticeably smaller than the images used in our original project (listed in the top section of the screenshot):

| Name | Size |
|--|----------|
|  <code>bg.png</code> | 4.07KB |
|  <code>bg-active.png</code> | 186.20KB |
|  <code>heading-banner-back.png</code> | 15.57KB |
|  <code>waves-bg.png</code> | 16.57KB |
|  <code>heading-banner.png</code> | 8.78KB |
|  <code>bg-artist.png</code> | 5.40KB |
|  <code>artists-image.png</code> | 83.22KB |

| Name | Size |
|--|----------|
|  <code>heading-banner.png</code> | 6.37KB |
|  <code>waves-bg.png</code> | 10.55KB |
|  <code>heading-banner-back.png</code> | 7.91KB |
|  <code>bg-active.png</code> | 180.29KB |
|  <code>bg.png</code> | 3.88KB |
|  <code>bg-artist.png</code> | 3.36KB |
|  <code>artists-image.png</code> | 83.07KB |

Smaller CSS file

We note that before we used the build script, our CSS file was called `main.css` and was approximately 21 KB, but after using the build script, the file has been renamed and is now almost half the original size, as shown in the following screenshot:

| Name | Size |
|--------------------------|---------|
| <code>main.css</code> | 21.38KB |
| Name | Size |
| <code>48346cf.css</code> | 12.79KB |

Smaller and fewer JS files

After executing the build script, you will notice that `main.js` and `plugin.js` have been combined into one. Not only have they been combined together, but they have also been minified, leading to a smaller file size of the final script file.

The `index.html` page from the `publish` folder generated via the build script invokes only four JavaScript files as shown in the bottom section of the following screenshot, compared to five JavaScript files originally placed in the folder (top section):

| Name | Size |
|-------------------------------------|---------|
| <code>modernizr-2.5.2.min.js</code> | 15.22KB |
| <code>main.js</code> | 2.08KB |
| <code>plugins.js</code> | 3.82KB |
| <code>ga.js</code> | 14.71KB |
| <code>jquery.min.js</code> | 33.27KB |
| Name | Size |
| <code>modernizr-2.5.2.min.js</code> | 15.22KB |
| <code>b611f1a.js</code> | 4.13KB |
| <code>jquery.min.js</code> | 33.27KB |
| <code>ga.js</code> | 14.71KB |

No comments in files

The HTML, CSS, and JS files in the `publish` folder have no comments that HTML5 Boilerplate files contain.

Build options

The Ant build script has a few tasks that are not executed by default, but are available to you if you need them. The following sections explain what these tasks allow you to do.

Minifying markup

By default, the Ant build script does not remove whitespaces from the `index.html` page when optimizing; if you want to also remove whitespaces in the HTML file and minify it, you can execute the following:

```
ant minify
```

Preventing image optimization

When executing the build script, you will notice that the script takes the longest time to optimize images. If you are executing the build script to merely test the final production-ready files, then you would not have to optimize images. In this case, you should execute the following command:

```
ant text
```

Using CSSLint

CSS Lint (csslint.net) is an open source CSS code-quality tool that performs a static analysis of your code and flags style rules that are invalid or may be the cause of problems. To use CSS Lint on your project's CSS files, just enter the following:

```
ant csslint
```

Typically, you will observe a bunch of warnings. CSS Lint has a lot of options you can set. To do this, open the `project.properties` file within the `config` folder in build. Uncomment this line by removing the `#`, by using the following command:

```
#tool.csslint.opts =
```

Enter all the options you want to use with CSS Lint after the `=` sign and save. The various options that you can use are mentioned at github.com/stubbornella/csslint/tree/master/src/rules.

Using JSHint

JSHint (jshint.com) is a community-driven tool to detect errors and potential problems with your JavaScript code and enforce your team's coding conventions. To execute JSHint on your JavaScript files, go to your project and execute the following:

```
ant jshint
```


Once executed, we see a bunch of errors being listed for our `main.js`. The corrected file is included within the code for this chapter. Once corrected, you will also notice a whole slew of errors being thrown for the code in `plugin.js`. This is because we used the minified code of the smooth-scroll plugin. Let us replace it with the unminified code from the project repository at github.com/kswedberg/jquery-smooth-scroll/blob/master/jquery.smooth-scroll.js.

Now, we get a bunch of errors all telling us we need to use the stricter comparison operator. Let us turn this off for our current project. We can do this by opening the `project.properties` file within the `config` folder under our `build` folder and uncomment the following line that allows you to use your own options for JSHint:

```
#tool.jshint.opts
```

Change it to the following snippet:

```
tool.jshint.opts = maxerr=25,eqeqeq=false
```


 More options are listed on the JSHint website at jshint.com.

Our errors have disappeared!

Setting up the SHA filenames

The concatenated and minified CSS and JS filenames are set to uniquely generated strings, which ensures a cached local copy of these files never get loaded when a new production build is deployed to the server. By default, the number of characters used in the filename is 7. You can set it to a smaller or larger number by changing the following line in `project.properties` within the `config` folder inside the `build` folder:

```
#hash.length = 7
```

Uncomment the previous line and then alter the number 7 to the number of characters you prefer, using the following syntax:

```
hash.length = <number of characters you prefer>
```


Using with Drupal or WordPress

Minor changes need to be made to make sure that these Ant build scripts work as intended for Drupal. Do note that there is not much help in minifying the HTML pages as a significant portion of the markup will be generated by Drupal or WordPress.

Updating build.xml

There is a minor change you need to make to `build.xml` to make it work with the file structure of Drupal or WordPress.

Look for `<echo message="Minifying any unconcatenatedcss files..." />` within the file. Just after that line of code, change the following:

```
<filesetdir="${dir.source}/${dir.css}/" excludes="${concat-files}"
includes="**/*.css"/>
```

To the following:

```
<filesetdir="${dir.source}/${dir.css}/" excludes="${concat-files},
${dir.build.tools}/**/*.css, ${dir.intermediate}/**/*.css, ${dir.
publish}/**/*.css" includes="**/*.css"/>
```

Setting up the project configuration properties

In the `project.properties` file within the `config` folder of the `build` folder, add the following code:

```
dir.css = .
dir.images = images
file.root.stylesheet = style.css
```

Setting the JS file delineator

WordPress or Drupal themes require you to split your markup into separate files (for example, `footer.php` for WordPress or `footer.tpl.php` for Drupal). You need to know in which of these files the following code is located:

```
<!-- scripts concatenated and minified via build script -->
<scriptsrc="js/plugins.js"></script>
<scriptsrc="js/main.js"></script>
<!-- end scripts -->
```

Use that filename (for example, `footer.php`) to set the `file.root.page` property in the `project.properties` file by using the following code:

```
file.root.page = <name of file>
```

A sample Drupal and WordPress theme that contains the modified build script are provided in the code for this chapter.

Using the Node build script

The Node build script is different from the Ant build script in the following two ways:

- It installs universally and does not need to be copied from one project to the other.
- All projects should be initialized with the Node build script. It is significantly more troublesome to add it on to a project that is already underway.

The Node build script requires Node, so verify you have Node installed by entering the following command:

```
node -v
```

If you do not have Node already, you can install it from nodejs.org/ (or by using **package manager** from github.com/joyent/node/wiki/Installing-Node.js-via-package-manager).

Grunt

Grunt (gruntjs.com/) is a Node-based command-line build tool on which this Node build script is based. The Node build script provides HTML5 Boilerplate optimized tasks that plug into Grunt.

This requires using a `package.json` file along with a `grunt.js` file within your project folder, which can be set up when you initialize your project.

Installing Node build script

In your command-line tool, first install the Node build script package, by entering the following command:

```
npm install https://github.com/h5bp/node-build-script/tarball/master -g
```

The Node build script can also be used as a part of a bigger build setup. If you are inclined to using it differently, take a look at all the possible ways of doing so here at github.com/h5bp/node-build-script/wiki/install.

Once installed, you can create your HTML5 Boilerplate project folder by initializing it.

Initializing your project

You can choose from various options to set up a project folder for yourself. Let us use this to set up a temporary project, to learn how to use this script to start your HTML5 Boilerplate project.

Create a folder where your HTML5 Boilerplate project should be. Navigate to it within your command-line tool and enter the following command:

```
h5bpinit
```

This will start setting up a whole set of command-line interactions for you to choose from. It is mostly used to set up information for package management that will be used by Grunt.

Once you have done that, you have three options to choose from for setting up the files you want to start with; these options are as follows:

- `[D]efault`: Standard set of files for HTML5 Boilerplate.
- `[C]ustom`: Get all the standard files with the option of choosing to rename `js/`, `css/`, or `img/` folders. You would typically want to do this if your files are going to be used as templates for other systems such as Drupal or WordPress.
- `[S]illy`: Prompts to rename every folder/file in HTML5 Boilerplate. You are least likely to use this option, unless you are a semantic perfectionist.

After you choose the type of installation you want to do, there are also more questions that are asked. Note that if you press *Enter*, the default value as indicated within parenthesis will be set.

This will then download the latest version of HTML5 Boilerplate from the Github repository for you to start as your base.

Using the Node build script with an existing project

It is not impossible to use the script with an existing project, but it's just a bit more tedious. There is work underway in the project to make this happen at github.com/h5bp/node-build-script/issues/55, but until then, the following is how we can use it with our Sun and Sand website:

1. First, create a temporary folder, and execute the Node build script to initialize an empty project from the command line as described in the earlier section.
2. Then, copy only `package.json` and `grunt.js` into your project folder.

You can see the code in the `nimbu.in/h5bp-book/chapter-7-node-init/` folder to see this in action.

Using the Node build script to build your project

Navigate to the Sun and Sand project folder (that you initialized it in the previous section) in your command-line tool and enter the following command:

```
h5bpbuild:default
```

This will combine the files and the results are published in the `publish` folder just like the Ant build script. You can also use these other build options like the Ant build script.

Text

If you would like to leave out compressing images when building your project, then use the following command:

```
h5bpbuild:text
```

Minify

If you would like to additionally minify HTML files, then use the following command:

```
h5bpbuild:minify
```

The results are similar to what you would find with Ant build script; the following screenshot shows the result of the minification process:

| Name | Size |
|-------------------------|----------|
| bg.png | 4.07KB |
| heading-banner.png | 8.78KB |
| waves-bg.png | 16.57KB |
| heading-banner-back.png | 15.57KB |
| bg-active.png | 186.20KB |
| bg-artist.png | 5.40KB |
| artists-image.png | 83.22KB |

| Name | Size |
|-------------------------|----------|
| heading-banner.png | 6.43KB |
| waves-bg.png | 10.60KB |
| heading-banner-back.png | 7.97KB |
| bg-active.png | 184.75KB |
| bg.png | 3.94KB |
| artists-image.png | 83.22KB |
| bg-artist.png | 3.41KB |

There are some additional options available that you don't get with Ant build script.

Server

This will open a local server instance you can immediately preview your website on. This is useful when you want to test the pages where protocol-relative URLs are used to link to files. To do this, simply navigate to your project folder in the command-line tool and enter the following command:

```
h5bp server
```

You will see the server being started for both the `publish` folder and the `intermediate` folder, as shown in the following screenshot:

```
Running "server" task
>> Started static web server in intermediate/ on port 3000...
>> Started static web server in publish/ on port 3001...
```

Then, open <http://localhost:3001> to view the published site.

Connect

Using this command, you can see your page reload on browsers it is open in as soon as you have made changes to any asset within the project. This saves you from refreshing the page manually to see the change. To do this, just navigate to your project folder in the command-line tool and enter the following command:

```
h5bp connect
```

Using with Drupal or WordPress

It is fairly trivial to initialize an HTML5 Boilerplate project with Node build script and then convert it into a template you are building for Drupal or WordPress. First, make sure you choose the `Custom` option when executing `h5bp init`. Then, when setting the folders, set `inc` as the folder where your stylesheet will be, and `images` as the name of the folder that would contain your template images. When you are prompted again, enter in the same values and the project framework will be generated for you. Make sure you replace `index.html` with your template files.

Once you have done this, open the `grunt.js` file in your project folder and confirm that the stylesheet's folder is set to the parent folder by using the following code:

```
css: {
  'style.css': ['*.css']
},
```

Make sure that only JavaScript files and stylesheets get prefixed with the SHA filenames, by editing or removing `images` from being renamed. This is done by using the following code:

```
rev: {
  js: 'js/**/*.js',
  css: '*.css',
},
```

The script also needs to know the new location of the `images` folder. We can do this by setting the source and destination folders for images, as shown in the following code snippet:

```
img: {
  dist: {
    src: 'images',
    dest: 'images'
  }
},
```

Next steps

Once we are satisfied with our production files in the `publish` folder, then we can move it to our hosting provider to replace the files that make our website.

Ideally, you would be using a Version Control System to do this, so you can quickly roll back an update in the unlikely event of this update making some page unavailable.

If you are only creating a template for Drupal or WordPress, then it may help to move this to within the WordPress folder on the server that is under a Version Control System.

Alternatively, you can compress your project and then copy the files to the server where they can be decompressed and used. The Node build script provides an option to do this. Go to your project folder in your command-line tool and enter the following command:

```
h5bptar --input publish --output <project-name>.tgz
```

Use the name that best describes your project instead of `<project-name>`. Then, copy the `<project-name>.tgz` to your server and expand it into the folder you would want the files in.

Summary

In this chapter, we learned how to use two kinds of build scripts that are available from the HTML5 Boilerplate team. We also looked at how we can use them both with Drupal or WordPress templates. We also looked at what we can do once the files have been built.

In the next chapter, we will look at some advanced tasks you can take on, now that you know how to create and deploy projects using HTML5 Boilerplate.

You Are an Expert, Now What?

We are all set with our website. We have learned how to write its code, build it with the build scripts and deploy it to the production code, so that it goes live without any hiccups. You are effectively done learning HTML5 Boilerplate. If you are curious in becoming a better web developer, you could spend time understanding some other relevant and useful parts of the Web! Let us explore a few of them.

Writing unit tests for your code

We wrote some JavaScript for our website. While the browsers let us know if the code is written incorrectly, there is no way to tell if the code works as intended. Perhaps there are edge cases that we failed to account for. The code should be as robust as possible and works around all expected use cases and is capable of handling most error conditions. You can ensure this is possible by writing tests to test every function your code calls.

A unit can be considered as the smallest testable part of your code. When you write unit tests, you ensure every section of the code behaves correctly. The easiest way to get started with unit tests is to use a test suite.

`QUnit.js` is a popular browser-based test suite that tests your code in the browser. Let us use this on our code that we wrote for the Sun and Sand Festival website.

Creating a testing environment

Let us create a `tests` folder within our project.

Then, we download `qUnit.js` from code.jquery.com/qunit/qunit-1.9.0.js and the associated CSS file `qunit.css` from code.jquery.com/qunit/qunit-1.9.0.css. The latest versions of these files can be found at github.com/jquery/qunit.

We now create a testing environment by creating a `tests.html` page within the `tests` folder, and have the following code:

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8">
<title>Tests for Sun n' Sand Festival Code</title>
<link rel="stylesheet" href="qunit-1.9.0.css">
</head>
<body>
<div id="qunit"></div>
<div id="qunit-fixture"></div>
<script src="//ajax.googleapis.com/ajax/libs/jquery/1.7.2/jquery.min.js"></script>
<script>window.jQuery || document.write('<script src="js/vendor/jquery-1.7.2.min.js"></script>')</script>
<script src="qunit-1.9.0.js"></script>
<script src="../js/main.js"></script>
<script src="test.js"></script>
</body>
</html>
```

In this code, we have included our `main.js` file that we are using on our website. We will be testing the code we wrote for the tabs used to display the line up.

Now, we will create the `test.js` file, where we will write all our tests for our code.

As our test depends on the markup that is used for the tabs, let us copy the markup without the content from `index.html` to `tests.html`.

If we execute this test as it is, we will get an error claiming global failure. If you open up the console of your browser's developer tools, you should see the following error:

```
Uncaught TypeError: Object [object Object] has no method
'smoothScroll'
```

This is because we invoke the plugins from `main.js` but we have not included those plugins here because we are not testing them. We can corral all of our plugin-dependent code and invoke them only if QUnit is not used, by testing for the existence of QUnit first before invoking plugins and frameworks like in the following snippet of code:

```
if(window.QUnit == undefined) {
  $(' .js-scrollitem').smoothScroll();
  if(Modernizr.svg === false) {
    $('img[src$=".svg"]').each(function() {
      this.src = /(.*)\.svg$/ .exec(this.src)[1] + '.png';
    });
  }

  if (Modernizr.generatedcontent === false && window.onbeforeprint !==
undefined) {
    window.onbeforeprint = printLinkURLs;
    window.onafterprint = hideLinkURLs;
  }

  Modernizr.load({
    test: Modernizr.audio,
    nope: {
      'mediaelementjs': 'js/vendor/mediaelement/mediaelement-and-
player.min.js'
    },
    callback: {
      'mediaelementjs': function() {
        $('audio').mediaelementplayer();
      }
    }
  });
}
```

Make sure you remove the condition—`if(window.QUnit == undefined)`—in the production code.

Now, let us write a test to confirm that when a navigation tab is clicked the correct class is applied to itself by using the following code snippet:

```
$(' .js-tabitem').each(function() {
  var $this = $(this);
  $this.trigger('click');
```

```
test( "navigation tabs", function() {
  ok($this.hasClass('t-tab__navitem--active'),
    'The clicked navigation item has the correct active class
    applied');
});
```

The `test()` function is a function available from the QUnit test suite. The first argument is the title of the text, and the second is the actual test function you want to execute.

We also use `ok()`, which is one of the assertions from the QUnit test suite to confirm the class does apply. An assertion is an essential element of unit testing, where you test if the result of the execution of your code returns the expected value. QUnit has different kinds of assertions that are all documented at api.qunitjs.com/category/assert/.

In `ok()`, the first argument we pass to this function is an expression that evaluates to true or false. The second argument is the message you want to display when the assertion is executed.

Now, let us test that the inactive navigation items do not contain the class name that makes a navigation item appear active, by using the following code snippet:

```
$('.js-tabitem').not(this).each(function() {
  ok(!$this.hasClass('t-tab__navitem--active'),
    'Inactive item does not have active class');
});
```

Let us now execute these tests! Open the `tests.html` page in your browser. You should see something like the following screenshot:



You can execute more complicated tests too! Learn more about QUnit from their online cookbook at qunitjs.com/cookbook/.

Esoteric defaults you should know about

There is a lot of research that was done to arrive at the defaults that are part of HTML5 Boilerplate. It is really fun to understand how different browsers behave and what drove us to choose the defaults as they are.

Meta UTF-8

The `meta` element represents any metadata information for the page. Setting `<meta charset="utf-8">` in the `<head>` element will ensure browsers parse the page with UTF-8 encoding in the absence of any other information about the encoding of the page.

The interesting thing to note is that most browsers look for character encoding metadata only within the first 512 bytes of the page. Hence, you need to ensure that if you have a lot of data in your `<head>` element, this meta element occurs before everything else.

In the absence of `charset` encoding information, browsers have to guess which `charset` encoding to apply. The HTML5 spec outlines the sniffing algorithm that all browsers must implement at www.whatwg.org/specs/web-apps/current-work/multipage/parsing.html#encoding-sniffing-algorithm. Unfortunately, older browsers had their own mechanisms for guessing character encoding.

In the case of Internet Explorer 7 and below, the default character encoding preference is typically set to `Auto Select`. This means the browser scans the content of the page to detect what character encoding would best apply. In case of Internet Explorer, if it finds a UTF-7 string within 4096 characters of the page, it would assume the page uses UTF-7 encoding and your page will become vulnerable to cross-site scripting attacks using UTF-7 encoding. Hence, the `meta` element declaration and right on top of the page in the `index.html` page.

Note that if your server sends an HTTP header that is of a different encoding, then that would take precedence. Make sure your server is set up to serve the right `charset` encoding as an HTTP header.

The HTML Doctype

Before the standardization of HTML and CSS, most markup and styles did not render consistently in any browser. But when we had standards about how markup should be written and more and more developers started adopting these standards, browsers then had to face the problem of detecting which of the pages on the Internet conformed to these standards and which weren't.

The Doctype was invented so that developers could inform the browser to render the page using the newer standards mode. Without a Doctype declaration, browsers would render the page in what is known as **Quirks Mode** (the way browsers used to render the pages before standards became an acceptable practice). In IE6, having a comment or an XML namespace declaration above the Doctype would render the page in Quirks Mode too. In the early 2000s when using the XHTML Doctype with an XML namespace declaration was recommended, this would be the cause of significant issues in Internet Explorer.

Not all Doctype declarations render in standards mode. The easiest way to use standards mode is to use the smallest recommended Doctype, `<!doctype html>`. You can use any mix of upper or lowercases in the Doctype declaration (for example, `<!DoCtYpE hTmL>`).

The details behind the clearfix solution

The `clearfix` CSS class is used to make sure floated elements fit in their parent container. The very first exploration of this idea occurred in 2002, and is elaborated further in the article at www.positioniseverything.net/easyclearing.html.

The `clearfix` selector works in the following manner:

```
.clearfix:after {
  content: ".";
  display: block;
  height: 0;
  clear: both;
  visibility: hidden;
}
.clearfix { zoom: 1; } /* IE 5.5/6/7 */
```

The biggest problem with this method is that margins do not collapse consistently across all browsers. Thierry Koblentz writes more about it at www.tjkdesign.com/lab/clearfix/new-clearfix.html.

Thierry Koblentz updated this method in 2010 introducing the use of both `:before` and `:after` pseudo-elements in the post at www.yuiblog.com/blog/2010/09/27/clearfix-reloaded-overflowhidden-demystified/ to be so. Both the pseudo-elements are used in the following code snippet:

```
.clearfix:before,  
.clearfix:after {  
  content: ".";  
  display: block;  
  height: 0;  
  overflow: hidden;  
}  
.clearfix:after {clear: both;}  
.clearfix {zoom: 1;} /* IE < 8 */
```

Using both pseudo-elements prevents the problem of inconsistent margin collapsing while using the `clearfix` class.

Nicolas Gallagher, in 2011, found an alternative way that reduces the lines of code necessary for the `clearfix` class if our target browsers are IE6 and higher and other modern browsers, as he explains in his article at nicolasgallagher.com/micro-clearfix-hack/. Nicolas' code is given in the following code snippet:

```
.cf:before,  
.cf:after {  
  content: " ";  
  display: table;  
}  
  
.cf:after {  
  clear: both;  
}  
  
/**  
 * For IE 6/7 only  
 * Include this rule to trigger hasLayout and contain floats.  
 */  
.cf {  
  *zoom: 1;  
}
```

In this method, using `display: table` would create an anonymous table cell (more information on what this means is available in the specification at www.w3.org/TR/CSS2/tables.html#anonymous-boxes) within the pseudo-element, which prevents the collapsing of top margins. The `content` property does not require any content within to work, but this method uses a space character to overcome an Opera bug when used on elements that are editable.

This is how the `clearfix` class evolved! As you can see, a great deal of research and development went into crafting the best `clearfix` class possible that would work across dominant browser platforms.

What do the print styles do?

The HTML5 Boilerplate stylesheet comes with a set of styles that are useful defaults when a user prints your page. Styling how a page would appear in print is something that most of us do not consider while designing a web page, and HTML5 Boilerplate gives you a set of good defaults, so you do not have to consider it most of the time (however, it would be good practice to do so).

Print media query

We have inlined all our print styles within a CSS media query called "print". This media query is matched whenever a user selects a page for printing, and these style rules will be applied in that case. We declare all of our rules within the `@media print` query as shown in the following code snippet:

```
@media print {  
  a, a:visited { text-decoration: underline; }  
  /* More Styles below */  
}
```

Optimizing colors and backgrounds

We then make sure we optimize the page to appear most readable when printed and also ensure we are not wasting too much printing ink printing superfluous images, colors, and text. This means we make sure we remove all background images or images, which are just of a slightly different shade of white or transparent for all elements. We also make sure all of the colors are black as it means the printer does not have to mix any ink and hence can print faster. We also remove shadows, as that would make the text less readable.

The final rule we have for these updates is as follows:

```
* {
    background: transparent !important;
    color: #000 !important; /* Black prints faster: h5bp.com/s */
    box-shadow:none !important;
    text-shadow: none !important;
}
```

Better links

Not many designers now use `text-decoration: underline` to style links on pages. Typically, people use colors to indicate something is a link. However, underlines are easier to discern in case of print, especially when you have no control over the printer and colors used to render them. Hence, we have all links (active or visited) to be styled with a line below the text by using the following code snippet:

```
a,
a:visited {
    text-decoration: underline;
}
```

It would also be helpful to have a reference to the actual link in print as there is no way for the user to navigate to that link if they are reading from a printed page and would like to visit the link. We do this by using the `attr()` function in CSS. `attr()` returns the value of an attribute of the element that the current rule will be applied to. In this case, as we are applying it on links, we can use `attr()` to obtain the value of the `href` attribute of links and print them. A space character is used to concatenate strings together when they are used as a value in the `content` property. We also want to make sure that if a link has a title, we print that too as a title is only visible on hovering on a link. All of this expressed in CSS looks like the following code snippet:

```
a[href]:after {
    content: " (" attr(href) )";
}

abbr[title]:after {
    content: " (" attr(title) )";
}
```


But, this means even links that are just linking to another location in the same page or are used for JavaScript actions (with the `javascript:` prefix) would render the same way! So, we would need to make sure we do not do this for these links.

For this, we use the attribute selector that allows us to select elements that have properties that begin, end, or contain certain values. By using the selector `a[href^="javascript:"]:after`, we ensure we are only selecting the `:after` pseudo-elements of links that have the attribute `href`, whose value starts with the string `javascript:`.

Similarly, we also select all links which have the `href` attribute that begin with the `#` character, as that means such links are inline links linking to another location within the same page.

We then make sure we render no content for pseudo-elements within these links. The rule then looks like the following code snippet:

```
.ir a:after,
a[href^="javascript:"]:after,
a[href^="#"]:after {
    content: "";
}
```

Do note that these rules are not available for IE6 and if it is highly necessary to offer this functionality in IE6, you would like to use JavaScript that provides this.

Rendering all code and quotes within one page

It sometimes happens that your printed page would contain quotes or code, and as a reader, it is annoying to keep referring back to a previous page when the code (or the quote) could all have been within one page without any break. For this, we can use the CSS `page-break-inside` property that allows you to tell the browser if you prefer these elements to break over two pages or stay within the same page. The code for this is shown in the following code snippet:

```
pre,
blockquote {
    border: 1px solid #999;
    page-break-inside: avoid;
}
```

Do note that `page-break-inside` is not supported in Firefox, but is available in all other browsers.

Rendering tables better

By default, putting headings within the `thead` tag would ensure that the headings get repeated every time a table breaks across two pages. However, only Firefox and Opera have support for this at the moment. In IE, you can do this but you would have to explicitly state it, as stated in the following code snippet:

```
thead {
    display: table-header-group; /* h5bp.com/t */
}
```

Rendering images better

Ideally, we want to prevent table rows and images from breaking across pages, so we use the now familiar `page-break-inside` property to tell the browser of our preference, as shown in the following code snippet:

```
tr,
img {
    page-break-inside: avoid;
}
```

It also does not appear too well, when images run off beyond the page or print cropped while appearing in full on the website. Hence, we restrict the maximum width to be as wide as the page itself and no more, as shown in the following code snippet:

```
img {
    max-width: 100% !important;
}
```

Margins on pages

The `@page` rule allows you to modify the properties of a page when printing. All browsers except Firefox support this rule. This rule sets the margins to be 0.5 cm per page, as shown in the following code snippet:

```
@page {
    margin: 0.5cm;
}
```

Optimal settings for orphans and widows

Orphans are the lines of text that appear at the bottom of the page. **Widows** are those that appear at the top of the page. We make sure that lines do not break in a manner that leaves fewer lines at the bottom or top than desired. This will create a more readable experience. The following code snippet is used for that purpose:

```
p,  
h2,  
h3 {  
    orphans: 3;  
    widows: 3;  
}
```

Keeping headings with content

It is not readable to have the headings appear at the bottom of one page and the content that the heading is for, appearing on the next. To tell the browsers to avoid doing this, we can use the `page-break-after` setting, as shown in the following code snippet:

```
h2,  
h3 {  
    page-break-after: avoid;  
}
```

What are protocol-relative URLs?

In HTML5 Boilerplate, when we refer to jQuery, we refer to it as follows:

```
<script src="//ajax.googleapis.com/ajax/libs/jquery/1.8.1/jquery.min.  
js">  
</script>
```

Note that we do not have either `http` or `https` in front of the URL; instead, it starts with `//`. These are called protocol-relative URLs and are useful when you want to use a protocol-agnostic resource in a HTTP or HTTPS environment.

When you serve pages using HTTPS, browsers will throw warnings and errors when the page loads assets and resources that use HTTP protocol. To prevent this, you need to ensure you use the HTTPS protocol for all the assets you are requesting. This is typically not a problem if you are using relative URLs to refer to assets within the parent folder of your page. However, if you are referring to external URLs like the CDN URL for jQuery (shown previously), then you need to ensure you use `https` when the page is being served with the HTTPS protocol and the `http` prefix when the page is being served with HTTP protocol.

Instead of using JavaScript to do that determination, simply omitting the protocol ensures browsers use the currently used protocol when requesting that external URL. In this case, if this page gets served on HTTPS as `https://example.com`, then the URL requested will be `https://ajax.googleapis.com/ajax/libs/jquery/1.8.1/jquery.min.js`.

You can learn more about this at paulirish.com/2010/the-protocol-relative-url/.

Using conditional comments

Historically, IE6, IE7, and IE8 have been browsers with the most bugs and inconsistent rendering of styles. There are many ways to serve styles to IE versions 8 and below, here are a few.

Browser style hacks

The most prevalent technique is to use hacks in CSS style rules that target only one browser.

For IE6 and below, use the following code snippet:

```
* html #uno { color: red }
```

For IE7, use the following code snippet:

```
*:first-child+html #dos { color: red }
```

For IE8, use the following code snippet:

```
@media \0screen {  
  #tres { color: red }  
}
```

There are more hacks that target two or more browsers (or exclude two or more browsers) all listed in the post at paulirish.com/2009/browser-specific-css-hacks/.

The problem with these hacks is that first they exploit holes in the browser's parsing technology. If browsers fix these parsing errors then they may not work. Luckily, we do not have to fear about this for older browsers such as IE6 and IE7.

These hacks are also not readable and without comments it is impossible to understand which browsers they target.

The advantage of these methods is that you can keep your style rules together, and you do not have to serve a separate stylesheet for the browsers requiring hacks.

Server-side browser detection

When they make a request to a web server, browsers send a User Agent String along with the request. Servers can serve different resources based on their interpretation of the User Agent String. For example, if a browser identifies itself as IE6 with the following User Agent String:

```
Mozilla/4.0 (compatible; MSIE 6.0; Windows XP)
```

Then, the server can send back a different stylesheet to IE6. While this may seem like a simple, easy solution, the problem occurs when browsers lie. Historically, browsers have never exactly claimed to be which browser they are, and hence, it is likely that you may send the wrong stylesheet to a browser.

It also involves a little overhead server-side to process the request according to the browser's User Agent setting, and hence is not an ideal way to serve different stylesheets to IE8 and below.

Stylesheets based on conditional comments

Conditional comments are HTML comments with special syntax that are understood by IE9 and below. The following is a sample conditional comment:

```
<!--[if lt IE 9]>  
<p>HTML Markup here</p>  
<!--<![endif]-->
```

All browsers except Internet Explorer 9 and below ignore content within these conditional comments. IE9 and below try to interpret the `if` condition within these comments and selectively render the content if the version number of the IE browser matches the one within the `if` condition.

The previous example will render the `p` tag on all 8, 7, 6, and below versions of IE.

Conditional comments are perfect to target older versions of IE and this is what HTML5 Boilerplate uses. There are two ways of using them. The first is to output a separate stylesheet based on matching a conditional comment, as shown in the following code snippet:

```
<!--[if lt IE 9]>
<link rel="stylesheet" href="/css/legacy.css">
<![endif]-->
```

This will make IE8 and below use `legacy.css` and other browsers will ignore this snippet of code.

The problem with a standalone stylesheet is that while you develop your styles you have two different stylesheets to target, and occasionally IE-specific stylesheets could be forgotten.

Some people provide only a very basic experience for IE8 and below, as shown in the following code snippet:

```
<!--[if ! lte IE 6]><!-->
/* Stylesheets for browsers other than Internet Explorer 6 */
<!--<![endif]-->
<!--[if lte IE 6]>
<link rel="stylesheet" href="http://universal-ie6-css.googlecode.com/
files/ie6.1.1.css" media="screen, projection">
<![endif]-->
```

But HTML5 Boilerplate prefers a more readable and targeted approach that provides the best possible styles to all browsers using class names, which we will look at next.

Class names based on conditional comments

An iteration of the previous conditional comments method would be to append class names on the root element based on conditional comments, as shown in the following code snippet:

```
<!--[if IE 8]>
<html class="no-js lt-ie9">
<![endif]-->
```

Then in your stylesheet, you can use it to set styles in IE8 and below as follows:

```
.lt-ie9 h1 { color: red }
```

You can read more about this solution at paulirish.com/2008/conditional-stylesheets-vs-css-hacks-answer-neither/.

This solution does not require separate stylesheets, but allows you to write readable class names that indicate why that style rule exists in the stylesheet. This is the solution we have adopted in HTML5 Boilerplate, and recommend.

What is meta x-ua-compatible?

`x-ua-compatible` is a header that defines how Internet Explorer renders your pages. It declares which mode Internet Explorer should use to render your page. This is primarily targeted towards older websites that break in Internet Explorer 9 onwards because of better support for standards. It can be set in two ways.

Meta tag in your HTML page

In this case, we merely add a meta tag between the `<head></head>` tag in your HTML pages as follows:

```
<head>
<meta http-equiv="X-UA-Compatible" content="IE=EmulateIE7" >
</head>
```

HTTP header response from the server

In Apache, in the `.htaccess` file, writing the following would make the server send the `X-UA-Compatible` HTTP header as a response to any request on that parent folder:

```
LoadModule headers_module modules/mod_headers.so
Header set X-UA-Compatible "IE=EmulateIE7"
```

We recommend this method of setting its value because HTTP header values override any value set via the meta tag. Moreover, using the meta tag with IE conditional comments on the `html` element causes this meta tag to be ignored. The `X-UA-Compatible` header can have the following values.

Edge

This would use the latest mode of rendering available. For example, within Internet Explorer 10, it would be IE10. We would want to always use the latest rendering mode available, as this means we have access to the latest and most standards-compliant version of the browser. This is why it is our default option in HTML5 Boilerplate.

IE9

This would use only IE9 mode to render the page. For example, when you use this mode and this page is viewed in Internet Explorer 10, it would use the IE9 mode to render the page.

IE8

This would render the page as though it is being viewed on Internet Explorer 8.

IE7

This mode renders content, as it would display if Internet Explorer 7 rendered it in standards mode.

Emulate IE9

This mode tells Internet Explorer to use the `<!DOCTYPE>` directive to determine how to render content. Standards mode directives are displayed in IE9 mode and quirks mode directives are displayed in IE5 mode. All Emulate modes, unlike the previous modes, respect the `<!DOCTYPE>` directive.

Emulate IE8

This mode tells Internet Explorer to use the `<!DOCTYPE>` directive to determine how to render content. Standards mode directives are displayed in IE8 mode and quirks mode directives are displayed in IE5 mode. Unlike IE8 mode, Emulate IE8 mode respects the `<!DOCTYPE>` directive.

Emulate IE7

This mode tells Internet Explorer to use the `<!DOCTYPE>` directive to determine how to render content. Standards mode directives are displayed in Internet Explorer 7 standards mode and quirks mode directives are displayed in IE5 mode. Unlike IE7 mode, Emulate IE7 mode respects the `<!DOCTYPE>` directive. For many websites, this is the preferred compatibility mode.

IE5

This mode renders content as if Internet Explorer 7 displayed it in quirks mode. You can learn more about these modes on MSDN documentation at [msdn.microsoft.com/en-us/library/cc288325\(v=VS.85\).aspx](http://msdn.microsoft.com/en-us/library/cc288325(v=VS.85).aspx).

Contribute

If you like what you have seen so far of the project, you might want to contribute! Contributing to HTML5 Boilerplate is rewarding in all the learning and understanding that you get out of making even the smallest of changes. There are two ways to contribute; these are as follows:

- Reporting issues
- Submitting pull requests

Reporting issues

If you find something that is a mistake or is incorrect in one of the files in HTML5 Boilerplate, then you can file an issue so any of the contributors can take a look at it and see if it can be resolved.

The trick is to find out if something is an issue on HTML5 Boilerplate or something is caused by the code your project uses. You can verify if this is a problem with HTML5 Boilerplate by starting a clean install of HTML5 Boilerplate and verifying if the error still occurs.

If it is an issue with HTML5 Boilerplate, before you file an issue, make sure it has not been already reported. The GitHub Issues page at github.com/h5bp/html5-boilerplate/issues lists all open issues. Use the **Search** bar on top to search for the issue you are facing. It is likely that it may have been fixed, but the fix has not yet been pushed to stable branch.

If the issue is brand new, then make sure you isolate the problem in a way that is obvious through a reduced test case (Chris Coyier writes about what a reduced test case is in css-tricks.com/reduced-test-cases/). When you file a bug report, make sure it is easy to understand, so we can find a speedy solution. Ideally your bug report should contain the following:

- A short and descriptive title
- A summary of the issue and the browser/Operating Systems where this bug occurs
- If it is possible, steps to reproduce the bug
- A URL to the reduced test case (you can host one on jsfiddle.net or codepen.io)
- Any other information that would be relevant to the bug, including lines of code that might be the cause of the bug, and potential solutions

Ideally, a bug report should be self-contained, so contributors do not have to follow up with you again to find out more about the bug and can instead focus on resolving it.

Following this process to file a bug report is a learning experience in itself in how to find out what is wrong with the markup, style, or script that you wrote.

Pull requests

If you have ideas on how to improve HTML5 Boilerplate, patches to fix some existing issues, improvements or new features, you would submit what is known as a **pull request**. A pull request is a set of changes you can submit for review to the HTML5 Boilerplate GitHub repository, so it can be reviewed by the core contributors and merged into HTML5 Boilerplate if found to be useful.

A good way to start contributing would be to find a small issue that you think you can fix, fork the GitHub project (learn more on what this means at help.github.com/articles/fork-a-repo), work on your changes and submit a pull request.

If your contribution changes a lot of lines of code and alters the nature of the project drastically, consider opening an issue on the GitHub project first.

The following are the steps to get started with creating a pull request:

- Fork the project.
- Clone your fork (in your terminal, enter `git clone https://github.com/<your-username>/html5-boilerplate.git` and press *Enter*).
- Add an upstream remote (in your terminal enter `git remote add upstream https://github.com/h5bp/html5-boilerplate.git` and press *Enter*).
- Get the latest changes from upstream (for example, by entering `git pull upstream master` and pressing *Enter* in your terminal).
- Create a new topic branch to contain your feature, change, or fix (`git checkout -b <topic-branch-name>`).
- Make sure that your changes adhere to the current coding conventions used throughout the project; that is, indentation, accurate comments, and so on.
- Commit your changes in logical chunks; use Git's interactive rebase feature (more about this feature at help.github.com/articles/interactive-rebase) to tidy up your commits before making them public. Please adhere to these Git commit message guidelines at tbaggery.com/2008/04/19/a-note-about-git-commit-messages.html or your pull request is unlikely be merged into the main project.

- Locally merge (or rebase) the upstream branch into your topic branch.
- Push your topic branch up to your fork (`git push origin <topic-branch-name>`).
- Open a pull request with a clear title and description. Please mention which browsers you tested in.

This may seem like a lot of work, but it makes your pull requests significantly easier to understand and faster to merge. Moreover, your code becomes the documentation of the work you have done and anyone who wants to know why that section looks the way it does can go back to your commits and understand exactly why it is the case.

Working on HTML5 Boilerplate would get you started with best practices of collaborative development that you can take back to your workplace or any other collaborative work you do.

Index

Symbols

- 8-bit PNGs
 - using 108
- 404.html 12
- .htaccess file 9, 13
- .lt-ie7 class 59
- .lt-ie8 class 59
- .lt-ie9 class 59

A

- Adobe Edge Inspect
 - URL 66
- Animated GIF 109
- Ant build script
 - about 116
 - ant-contrib, installing 117
 - build options 122
 - build script, installing 118, 119
 - CSS file 121
 - Drupal or WordPress, using with 124
 - features 116
 - image files 120
 - installing 117
 - JS files 121
 - using 117, 118
- ant-contrib
 - installing 117
- Apache
 - about 67
 - configuring 70, 71
 - installing 68
 - installing, on Linux 69
 - installing, on Mac 68

- installing, on Windows 68
- Apache Server
 - customizing 67
 - server configuration files 90
 - server-side configurations 67
 - setting up 67
- Apache Server customizations
 - .htaccess file features 71
- apple-touch-icon-*.png 13
- Aptana Studio 10

B

- borderradius class 9
- build options, Ant build script
 - about 122
 - CSSLint, using 122
 - image optimization, preventing 122
 - JSHint, using 123
 - markup, minifying 122
 - SHA filenames, setting up 123
- build options, Node build script
 - about 127
 - connect 129
 - minify 127
 - server 128
 - text 127
- build script
 - about 115
 - Ant build script 116
 - Node build script 116
 - selecting 117
- build.xml
 - updating 124

C

CDN

- about 22, 103
- Google CDN hosting 24, 25
- Protocol-relative URLs 23
- using 22

Chrome 8

Chrome Frame 77

Clearfix 8

clearfix class

- about 39
- working 39

clearfix CSS class 136, 137

code and quotes

- rendering 140

Codekit

- URL 101

colors and backgrounds

- optimizing 138

Compass 44, 100

Compass extension 101

conditional comments

- browser style hacks 143, 144
- class names 145, 146
- server-side browser detection 144
- stylesheets 144
- using 143

console.log function 9

Content Delivery Network. *See* CDN

cross-browser compatibility

- about 8
- Clearfix 8
- conditional classes 9
- doctype 8
- helper class 9
- Modernizr 9
- no console.log errors 9
- normalize.css 8
- search box styling 8

crossdomain.xml 13

CSS3

- about 99
- output CSS 100
- writing, easier with tools 99

CSSLint

- about 122

- URL 122

- using 122

CSS sprites, from within Adobe Photoshop 111

CSS sprites, with Compass 111

CSS validator 42

custom 404 page 76

D

disabled form elements

- styling, in IE 98

DNS prefetching 102

doc 12

doctype declarations 8

Domain Name Server (DNS) 102

Drupal or WordPress, using with Ant build script

- about 124
- build.xml, updating 124
- JS file delineator, setting up 124
- project configuration properties, setting up 124

Drupal or WordPress, using with Node build script 129

E

Entity Tags. *See* ETags

esoteric defaults 135

ETags 71

example.com/foo

- rewriting, to example.com/foo/ 105

example.com/foo/

- rewriting, to example.com/foo 105, 106

example project

- creating 17, 18
- housekeeping 18

Expires header

- using 74, 75

F

favicon.ico 13

favicons

- editing 19, 20

features, HTML5 Boilerplate

- accessible focus styles 10

- cross-browser compatibility 8
- performance optimizations 9
- print styles 10
- progressive enhancement 10

Firebug Lite
installing 57
using 57, 58

Firefox 3.5 8

focus styles 10

G

Google Analytics
about 112
augmenting 112
IP addresses, anonymizing 113
JavaScript errors, tracking 113
jQuery AJAX requests, tracking 113
tracking settings, adding 112

Google Analytics ID
adding 25

Google CDN hosting 24, 25

Google Libraries API
using 48

Grunt
about 125
URL 125

Gzip
about 72
enabling 74
used, for compressing files 72, 73

H

H5BP files. See HTML5 Boilerplate headings
keeping, with content 142

helper class invisible
using 39

help, HTML5 Boilerplate 13, 14

hidden class
about 33, 34
using 36, 38

housekeeping, example project
about 18
favicons, editing 19-21
Google Analytics ID, adding 25
humans.txt, updating 25

tags, setting 18, 19

third-party libraries, adding 22

HTML5 Boilerplate
about 7
backup and source files access, blocking
with .htaccess file 79
build script 115
contributing 148
downloading 15, 16
example project 17
features 7, 71
Google Libraries API, using 48
Gzip components 72
help 13, 14
hidden folder access, blocking with .htaccess file 79
icons 21
issues, reporting 148, 149
jQuery, using 47
MooTools, using 48
pull request 149, 150
shell script, using 16, 17
site, creating 27
tools 10
using, with style languages 44

HTML5 Boilerplate Compass extension 101

HTML5 Boilerplate CSS
converting, to Sass or Less 101

HTML5 Boilerplate files
404.html 12
.htaccess 13
apple-touch-icon-*.png 13
crossdomain.xml 13
doc 12
downloading 11
downloading, from Github 11
downloading, from Initializr 11
favicon.ico 13
humans.txt 12
img 12
index.html 12
js 12
main.css 12
main.js 12
normalize.css 12
overview 12, 13
plugins.js 12

- readme.md 13
- robots.txt 13
- vendor 12
- HTML5 Boilerplate Sass fork 101**
- HTML5 Boilerplate site**
 - CSS3, writing 99
 - Google Analytics, augmenting 112
 - images, optimizing 108
 - performance, improving 102
 - Polyfills, using 102
 - print considerations 101
 - smooth-scroll plugin, adding 48
 - testing 56-63
 - testing, on non-desktop browsers 64-66
 - users, handling without JavaScript 106
 - visibility, improving 103
 - visiting, using IE 95
- HTML5 Boilerplate site, optimizing for IE**
 - disabled form elements, styling 98
 - IE6 image toolbar, suppressing 99
 - mobile-first styles 95, 96
 - printing, jQuery used 97
- HTML5 Doctor 28**
- HTML5 feature**
 - adding, with Modernizr 52
- Html5please.com 102**
- HTML Doctype 136**
- HTTP header response, from server**
 - about 146
 - edge 146
 - Emulate IE7 147
 - Emulate IE8 147
 - Emulate IE9 147
 - IE5 147
 - IE7 147
 - IE8 147
 - IE9 147
- humans.txt**
 - about 12, 25
 - updating 25

I

- icons, HTML5 Boilerplate**
 - apple-touch-icon-57x57-precomposed.png 21
 - apple-touch-icon-72x72-precomposed.png 21
 - apple-touch-icon.png 21
 - apple-touch-icon-precomposed.png 21
 - apple-touch-icons-114x114-precomposed.png 21
 - apple-touch-icons-144x144-precomposed.png 21
 - favicon.ico 21
- IE6 8**
- IE6 image toolbar**
 - suppressing 99
- ie.css 96**
- iframe 52**
- ImageAlpha 108**
- ImageOptim 108**
- image optimization**
 - 8-bit PNGs 108
 - image sprites, using 109-111
 - tools 108
- image replacement class 33**
- images**
 - rendering 141
- image sprites**
 - CSS sprites, from within Adobe Photoshop 111
 - CSS sprites, with Compass 111
 - SpriteMe 112
 - using 109-111
- img 12**
- index.html 12**
- Internet Explorer**
 - about 77
 - setting 77
 - setting, for rendering site 77
- IP addresses**
 - anonymizing, in Google Analytics 113
- issues**
 - reporting 148, 149

J

- JavaScript**
 - disabling, on Chrome 106
 - disabling, on Firefox 106
 - disabling, on IE 106

- disabling, on Opera 106
- disabling, on Safari 106
- JavaScript errors**
 - tracking, in Google Analytics 113
- JPEG 109**
- jpegtran**
 - URL 118
- jQuery**
 - used, for printing with IE6 and IE7 97
 - using 47
- jQuery AJAX requests**
 - tracking, in Google Analytics 113
- js 12**
- JS file delineator**
 - setting up 124
- JSHint**
 - about 123
 - URL 123
 - using 123

L

learning resources, style languages

- Less 44
- Sass 44
- Stylus 44

Less

- about 99
- code snippet 100
- URL 31, 44

Linux

- Apache, installing 69

LiveReload

- URL 101

M

Mac

- Apache, installing 68

MacPorts

- URL 117

main.css 12

main.js 12

main.scss 96

markup

- creating 28

- section element, using 30
- valid markup, writing 30

Media Queries

- about 95, 96
- ie.css 96
- main.css 96

Meta UTF-8 135

meta x-ua-compatible

- HTTP header response, from server 146
- meta tag 146

micro-clearfix solution 8

MIME types 78

Modernizr

- about 9, 53, 54
- custom build 53
- used, for loading CSS features 55

modernizr.load

- about 52-54
- using 55

MooTools

- using 48

N

no-borderradius class 9

Node build script

- build options 127
- Drupal or WordPress, using with 129
- features 116
- Grunt 125
- installing 125
- project, initializing 126
- used, for building project 127
- using 125
- using, with existing project 127

normalize.css 12, 32

Normalize.css 8

O

Opera 9 8

Opera Mobile browser 65

Opera Mobile Emulator

- URL 64

optipng

- URL 118

orphans 142

P

plugin.js file 9

plugins.js 12

PNG 109

Polyfills

about 102

finding 102

using 102

print media query 138

print styles

about 10, 138

code and quotes, rendering within page 140

colors and backgrounds, optimizing 138

functionalities 138

headings, keeping with content 142

images, rendering 141

margins, setting 141

optimal settings 142

print media query 138

style links 139, 140

tables, rendering 141

progressive enhancement 10

project configuration properties

setting up 124

protocol-relative URLs 23, 24, 142

pull request

about 149, 150

creating 149

Q

QUnit.js 131

R

readme.md 13

Remote Debugging with Safari 6 65

reset.css 31

RewriteBase path 106

robots.txt 13

S

Safari 4 8

Sass

about 99

code snippet 100

URL 31, 44

Sass fork 101

search box styling 8

section element

using, in markup 30

server configuration files, Apache 90

SHA filenames

setting up 123

site, creating

markup, working on 27

styles, creating 30

site performance

improving 102

site visibility, improving

about 103

search spiders, directing to site map 103

slash redirects, trailing 105

X-Robots-Tag headers, implementing 104

slash redirects, trailing

example.com/foo, rewriting to example.

com/foo/ 105

example.com/foo/, rewriting to example.

com/foo 105

smooth-scroll plugin

adding, to site 48

class names, adding 50

js-scrollitem class 49

navigation links 49

plugin file, downloading 48

using 48-52

SpriteMe

about 112

URL 112

style classes

about 32

clearfix class 39

hidden class 33, 34

image replacement class 33

visuallyhidden class 35, 36

style languages

about 42

advantages 43

disadvantages 43

HTML5 Boilerplate, using with 44

learning resources 44

style links 139, 140

- styles**
 - creating 30, 31
 - reset.css, using 31
 - style classes 32
 - style languages 42
 - valid stylesheets, writing 42
- Stylus**
 - URL 44
- T**
- tables**
 - rendering 141
- test() function** 134
- testing environment**
 - creating 132-134
- TextMate** 11
- third-party libraries**
 - adding 22
 - Content Delivery Network, using 22
- tools**
 - Aptana Studio 10
 - TextMate 11
 - Visual Studio 10
- tools, for image optimization**
 - about 108
 - ImageAlpha 108
 - ImageOptim 108
- Twitter Bootstrap** 100
- U**
- unit tests**
 - clearfix CSS class 136, 137
 - conditional comments, using 143
 - HTML Doctype 136
 - Meta UTF-8 135
 - meta x-ua-compatible 146
 - print styles 138
 - testing environment, creating 132-135
 - writing 131
- up-to-date port, HTML5 Boilerplate**
 - Less 44
 - Sass 44
 - Stylus 44
- users**
 - handling, without JavaScript 106, 107
- UTF-8** 78
- UTF-8 encoding**
 - using 78
- V**
- valid markup**
 - writing 30
- valid stylesheets**
 - writing 42
- vendor** 12
- Virtual Box**
 - URL 56
- visuallyhidden class** 35
- Visual Studio** 10
- W**
- WebKit browsers** 8
- widows** 142
- WinAnt** 117
- Windows**
 - Apache, installing 68
- X**
- x-ua-compatible** 146
- Y**
- yum** 117



Thank you for buying HTML5 Boilerplate Web Development

About Packt Publishing

Packt, pronounced 'packed', published its first book "*Mastering phpMyAdmin for Effective MySQL Management*" in April 2004 and subsequently continued to specialize in publishing highly focused books on specific technologies and solutions.

Our books and publications share the experiences of your fellow IT professionals in adapting and customizing today's systems, applications, and frameworks. Our solution based books give you the knowledge and power to customize the software and technologies you're using to get the job done. Packt books are more specific and less general than the IT books you have seen in the past. Our unique business model allows us to bring you more focused information, giving you more of what you need to know, and less of what you don't.

Packt is a modern, yet unique publishing company, which focuses on producing quality, cutting-edge books for communities of developers, administrators, and newbies alike. For more information, please visit our website: www.packtpub.com.

About Packt Open Source

In 2010, Packt launched two new brands, Packt Open Source and Packt Enterprise, in order to continue its focus on specialization. This book is part of the Packt Open Source brand, home to books published on software built around Open Source licences, and offering information to anybody from advanced developers to budding web designers. The Open Source brand also runs Packt's Open Source Royalty Scheme, by which Packt gives a royalty to each Open Source project about whose software a book is sold.

Writing for Packt

We welcome all inquiries from people who are interested in authoring. Book proposals should be sent to author@packtpub.com. If your book idea is still at an early stage and you would like to discuss it first before writing a formal book proposal, contact us; one of our commissioning editors will get in touch with you.

We're not just looking for published authors; if you have strong technical skills but no writing experience, our experienced editors can help you develop a writing career, or simply get some additional reward for your expertise.

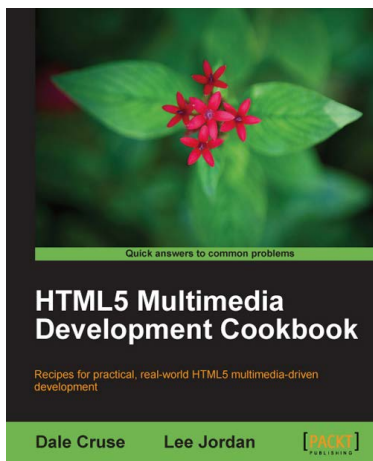


HTML5 Mobile Development Cookbook

ISBN: 978-1-849691-96-3 Paperback: 254 pages

Over 60 recipes for building fast, responsive HTML5 mobile websites for iPhone 5, Android, Windows Phone, and Blackberry

1. Solve your cross platform development issues by implementing device and content adaptation recipes.
2. Maximum action, minimum theory allowing you to dive straight into HTML5 mobile web development.
3. Incorporate HTML5-rich media and geo-location into your mobile websites.



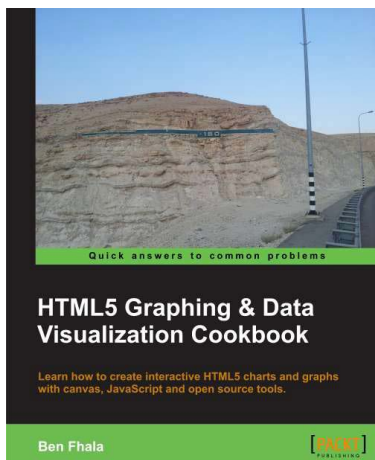
HTML5 Multimedia Development Cookbook

ISBN: 978-1-849691-04-8 Paperback: 288 pages

Recipes for practical, real-world HTML5 multimedia-driven development

1. Use HTML5 to enhance JavaScript functionality. Display videos dynamically and create movable ads using JQuery.
2. Set up the canvas environment, process shapes dynamically and create interactive visualizations.
3. Enhance accessibility by testing browser support, providing alternative site views and displaying alternate content for non supported browsers.

Please check www.PacktPub.com for information on our titles



HTML5 Graphics & Data Visualization Cookbook

ISBN: 978-1-849693-70-7 Paperback: 396 pages

Learn how to create interactive HTML5 charts and graphs with canvas, JavaScript and open source tools.

1. Build interactive visualizations of data from scratch with integrated animations and events
2. Draw with canvas and other html5 elements that improve your ability to draw directly in the browser
3. Work and improve existing 3rd party charting solutions such as Google Maps



HTML5 Video How-to

ISBN: 978-1-849693-64-6 Paperback: 82 pages

Over 20 practical, hands-on recipes to encode and display videos in the HTML5 video standard

1. Encode and embed videos into web pages using the HTML5 video standard
2. Publish videos to popular sites, such as YouTube or VideoBin
3. Provide cross-browser support for HTML5 videos and create your own custom video player using jQuery

Please check www.PacktPub.com for information on our titles