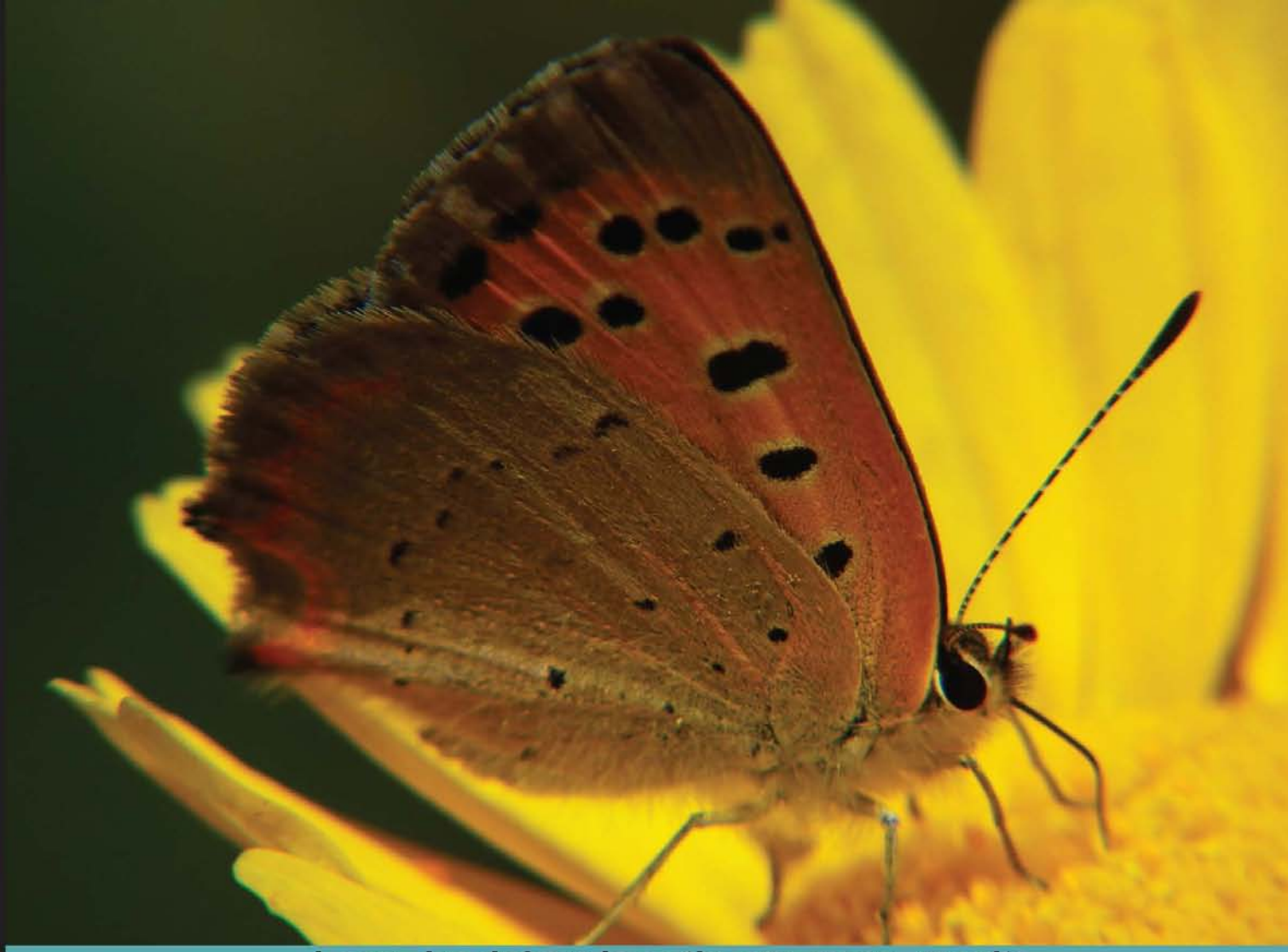


Learn by doing: less theory, more results

Drupal 7 Mobile Web Development

Transform your existing Drupal site into one that is completely compatible with mobile and tablet devices

Beginner's Guide

Tom Stovall

[PACKT] PUBLISHING open source* community experience distilled

Drupal 7 Mobile Web Development

Beginner's Guide

Transform your existing Drupal site into one that is completely compatible with mobile and tablet devices

Tom Stovall

[PACKT] PUBLISHING open source community experience distilled

BIRMINGHAM - MUMBAI

Drupal 7 Mobile Web Development
Beginner's Guide

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: March 2012

Production Reference: 1020312

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

ISBN 978-1-84951-562-7

www.packtpub.com

Cover Image by Charwak A (charwak86@gmail.com)

Credits

Author

Tom Stovall

Reviewers

Sumit Kataria

Trevor James

Michael Peacock

Acquisition Editor

Sarah Cullington

Lead Technical Editor

Hithesh Uchil

Technical Editors

Vrinda Amberkar

Mehreen Shaikh

Copy Editors

Leonard D'Silva

Neha Shetty

Project Coordinator

Kushal Bhardwaj

Proofreaders

Bernadette Watkins

Ting Baker

Indexers

Rekha Nair

Monica Ajmera Mehta

Hemangini Bari

Graphics

Manu Joseph

Production Coordinator

Melwyn D'sa

Cover Work

Melwyn D'sa

About the Author

Tom Stovall got a Timex Sinclair 1000 in 1982 from his mom for his birthday and the first night he slept with it under his pillow. Both school teachers, his mom and dad always made sure he had access to computers and today's programming chops owe their origins to those lazy summers spent in front of whatever hardware he could beg, borrow, or use when no one was looking.

Tom started doing websites in 1995, then with PERL, later with PHP. He was the principal front-end developer on Performance.gov, a cost-tracking, Drupal-based website for REI Systems, Inc and the President's Office of Management and Budget during it's year-long development cycle. He now works for Apigee, Inc in Palo Alto, CA developing Drupal sites in support of their enterprise API product and is the maintainer on several Drupal contrib modules.

About the Reviewers

Sumit Kataria is a software engineer and technology enthusiast who trusts in open source software. He possesses both a deep knowledge of Drupal programming and magical mobile application development skills that allow him to make Drupal and mobile sing together in beautiful harmony.

Sumit has done more than 15 iPhone/iPad/Android apps using Drupal as a base system. He has worked with Lullabot and CivicActions building several Drupal-integrated mobile applications, including the Drupalize.Me app and Do It With Drupal conference app.

Sumit is very passionate about everything he does and tries to bring the same enthusiasm to his projects. He has been a presenter at four DrupalCons and several DrupalCamps, advocating Drupal as a mobile application cloud-based backend. He was a Google Summer of Code student for Drupal in 2008, and in 2011, he managed the whole Drupal Summer of Code program.

When Sumit is not working on Drupal/mobile, he enjoys traveling and exploring new places. Sumit lives in New Delhi, India. Find him on LinkedIn or Twitter as `@sumitk` or on IRC as `sumitk`.

Trevor James is a Drupal developer based in Middletown, MD, USA. Trevor has been designing websites for 15 years using a combination of PHP, HTML, XHTML, CSS and ColdFusion, and has been using Drupal intensively for 5 years. Trevor's focus is on building web applications and portals for education, non-profit, medical systems, and small business environments.

He is interested in best methods of integrating web services with Drupal sites; optimizing Drupal site performance, and using Drupal content types, views, panels, and other contributed modules to develop front end interfaces that support data intensive websites. He loves teaching people about Drupal and how to use this excellent open source content management framework. He is also an active member of the `drupal.org` community of developers and frequently supports other Drupal users and developers via the `drupal.org` forums.

Trevor authored the Packt book, *Drupal Web Services*, published in November 2010. For more on this title, visit `http://www.packtpub.com/drupal-web-services/book`.

Trevor co-authored the Packt title, *Drupal 6 Performance Tips*, published in February 2010. For more on this title, visit `https://www.packtpub.com/drupal-6-performance-tips-to-maximize-and-optimize-your-framework/book`.

Much thanks to the Packt editorial team for giving Tom Stovall the green light for this book, a much anticipated addition to the ever-growing Drupal library. Tom is a devoted and passionate Drupal developer and this text shows all of his enthusiasm and knowledge of the Drupal project. We stand to learn a lot, not only about mobile development, but also about Drupal from Tom's contributions here. Thanks Tom for taking the time to write this excellent resource.

As with my own book authoring, the reviewing process takes a great deal of time, so I'd like to thank my family for allowing me the time out from other daily family obligations to devote to the review process. Thanks to my wife, Veronica, and our lovely twin daughters, Francesca and Clare.

Michael Peacock is a web developer and Zend Certified Engineer from Newcastle, UK with a degree in Software Engineering from the University of Durham.

After working as Managing Director and Lead Developer and overseeing the development team at the web agency he co-founded almost five years ago, Michael stepped back from the business and now acts as Senior/Lead Web Developer on the telemetry project for Smith Electric Vehicles.

Michael loves working on web-related projects and is currently incubating a number of ideas for launch through his latest venture, Central Apps Limited (`www.centralapps.co.uk`). When he isn't developing or writing, Michael can often be found at user groups and conferences talking about web-related technologies, including large-scale data problems, innovative technologies, continuous integration, and automated deployment.

He is the author of *Drupal 7 Social Networking*, *PHP 5 Social Networking*, *PHP 5 E-Commerce Development*, *Drupal 6 Social Networking*, *Selling online with Drupal e-Commerce,* and *Building Websites with TYPO3*. Other publications Michael has been involved with include *Mobile Web Development* and *Drupal for Education and E-Learning*. He was a technical reviewer for both of these books.

You can follow Michael on Twitter at `www.twitter.com/michaelpeacock` or find out more about him through his blog, `www.michaelpeacock.co.uk`.

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.

I'd like to dedicate this book to my mom, Sheila Stovall, who took me to my first computer show at the University of South Florida when I was in grade school and triggered a life-long obsession with technology that continues to this day.

- Tom Stovall

Table of Contents

Preface 1

Chapter 1: When is a Phone Not a Phone? 7

"Dumb" phones 8
"Smart-er" phones 9
Smart phones 10
Tablets 11
WebKit 12
Mobile-ize me! 12
What is mobile? 13
The "One Design" myth 13
Mobile simulators 14
Time for action – installing an Android development package 14
iOS 16
Time for action – installing the Mac OS developer's package 17
Summary 20

Chapter 2: Setting up a Local Development Environment 21

Drupallo's Pizza Kitchen 22
Averting disaster 23
A word about platforms 24
Time for action – downloading the SCM client for Mac 25
Cygwin and GIT for Microsoft Windows 28
Time for action – installing a development environment using Windows and Cygwin 28
Text editors for Mac OS X and Microsoft Windows 33
Serving web pages on your computer 33
MAMP 34
Time for action – configuring our first virtual host 35
Time for action – changing the default configuration for MAMP 39

WAMP (Windows, Apache, MySQL, and PHP) **41**
Time for action – installing WAMP **41**
Drush and Drush Make **45**
Time for action – installing Drush and Drush Make for Mac OS X **45**
Time for action – installing Drush for Windows **46**
Building a Drupal website with Drush Make **47**
Time for action – building a Drupal install from a make file **50**
The local database **51**
Time for action – creating a database **51**
Summary **58**

Chapter 3: Selecting the Right Domain for your Mobile Site **59**
Once upon a website… **60**
One ring to rule them all **62**
Domain Access versus Multisite 63
Time for action – installing Domain Access module **63**
Domain management 64
Time for action – configuring Apache **64**
Bootstrapping the domain **67**
Time for action – bootstrapping the Domain Access module **67**
Introduction to the Features module **70**
Time for action – installing and creating your first feature **70**
Time for action – updating the feature with new settings **74**
Deployment—best practices **77**
Time for action – code check-in and deployment **78**
Pushing out features 81
Time for action – check in your features module **81**
Summary **83**

Chapter 4: Introduction to a Theme **85**
Progressive Enhancement **86**
HTML5 and the simplified DOCTYPE 87
New HTML5 semantic elements 89
Drupal 6 versus Drupal 7 theming **89**
Time for action – installing the default mobile theme **93**
The simple life **95**
Media queries **96**
Time for action – personalizing the mobile theme **97**
Redirecting mobile clients **101**
Time for action – writing JavaScript redirection for our theme **102**
Give them what you think they need until they tell you what they want **105**

Behave yourself **106**
Drupal behaviors 107
Time for action – redirection with a cookie to remember state **108**
Summary **112**

Chapter 5: A Home with a View **115**
The Context and Display suite modules **116**
Time for action – creating a mobile-friendly home page **117**
Pushing changes from one environment to another **125**
Time for action – updating the Home Page feature **125**
The menu **129**
Time for action – creating the menu content types **130**
Bundling up the changes **136**
Time for action – bundling the changes into a package **136**
Summary **140**

Chapter 6: The Elephant in the Room: Audio, Video, and Flash Media **141**
Flash and iOS **141**
Incorporating video into your web content **142**
Time for action – embedding media files **143**
Time for action – adding content **147**
A word about encoding 150
"I did it my way" 151
Charting and graphs **152**
Time for action – graphing a view **153**
Summary **157**

Chapter 7: Location, Location, Location **159**
Geolocation **160**
The navigator.geolocation object **160**
Time for action – adding location data to nodes **161**
From address to longitude and latitude **175**
Time for action – geocoding a node's location data **176**
The close2u module **177**
Time for action – downloding and enabling the close2u module **178**
Finishing the page **190**
Time for action – finding the closest franchise the hard way **190**
Summary **197**

Chapter 8: Services with a Smile **199**
Using Drupal to power your native application **200**
Time for action – creating a REST service **201**
Time for action – testing your new REST service **203**

APIs: The future of the interactive web 208
Customized services 209
Time for action – custom REST service formatter 209
jQuery Mobile 212
Time for action – using jQM as our base theme 213
jQuery Mobile JavaScript Events 219
Time for action – the AJAX login form 219
Summary 222

Chapter 9: Putting it Together 223
Display Suite 224
Hooks, styles, and build modes 224
Time for action – retheming nodes for our jQuery mobile theme 230
Time for action – adding theming to the rendered node 236
Beyond core menu items 240
Time for action – customized menu attributes 240
Fonts 242
Time for action – adding fonts 244
Summary 251

Chapter 10: Tabula Rasa: Nurturing your Site for Tablets 253
The human touch 254
The event-driven model 254
Touch and go 256
The main event 257
Time for action – adding a swipe advance to the home page 257
The changing landscape (or portrait) 260
"Starting over" or "Everything you know about designing websites is wrong" 261
Wire framing made easy 262
Drupal 7 Commerce module 263
Time for action – the one true theme 264
Time for action – creating a product 268
A room with a viewport 273
Time for action – setting the viewport with JavaScript 273
Time for action – advanced media queries for tablets 274
Summary 278

Chapter 11: A Home in the Clouds **279**
Problems introduced by modern websites **280**
Amazon Web Services (AWS) **281**
Time for action – setting up AWS and RightScale **283**
Time for action – using an AMI to create a server **289**
Time for action – Jenkins builds our site **298**
Summary **303**
Appendix: Pop Quiz Answers **305**
Index **309**

Preface

It's not an overstatement to say that handhelds have changed the world. What was, just 10 years ago, simply a phone is now the center of your online life and, for many users, their primary Internet device. The power of the smart phone is shaking up the world from Main Street and Wall Street to Pennsylvania Avenue and Downing Street.

Drupal is the perfect platform on which to build a mobile strategy. The power of millions of developers world-wide ensures that there's no problem you face that has not already been overcome by multiple developers and solved with any one of the hundreds of thousands of Drupal contributed projects.

What this book covers

Chapter 1, *When is a Phone Not a Phone?*, explains what we mean when we say "mobile." In this chapter, we'll take a look at the mobile platforms in use today and how they behave and render today's HTML standards.

Chapter 2, *Setting up a Local Development Environment*, teaches you to work in a team environment with version control and to create a local version of our site on Windows or Mac OS with Drush, Drush Make and a make file, and our standard open source PHP *AMP stack. It outlines a team workflow of building the code locally and pushing it to the live site.

Chapter 3, *Selecting the Right Domain for your Mobile Site*, guides you through setting up the Domain Access and Drupal Behaviors modules that redirect mobile and desktop browsers to the version of the website most appropriate for their client. In this chapter, we will learn to share content across sites without resorting to a multisite install.

Chapter 4, *Introduction to a Theme*, introduces the idea of progressive enhancement with CSS. In this chapter, we'll create a very simple HTML5 theme that will serve mobile clients with CSS Media Queries until a highly customized one can be devised.

Chapter 5, *A Home with a View*, demonstates the use of Context and Image Styles to create a customized view for the home page. In this chapter, we'll create a mobile-friendly menu and bundle it up into a feature that can push the new content to your live site in one fell swoop.

Chapter 6, *The Elephant in the Room: Audio, Video, and Flash Media*, teaches you to create a compelling audio and video experience without using Flash. It teaches you to create data visualization using data we've pulled from a View and the HighCharts JavaScript library.

Chapter 7, *Location, Location, Location* helps you to set up location services and cover some common use cases, as well as some uncommon ones using GMap, Location, Open Layers and Map Box.

Chapter 8, *Services with a Smile*, explores the Services module which serves up pieces of node content from a REST and/or SOAP API. In this chapter, we will leverage this module to add some interesting interactivity to our example site.

Chapter 9, *Putting it Together*, guides you in addding some advanced theming to your site and making the site more responsive to the various devices that will be accessing it.

Chapter 10, *Tabula Rasa: Nurturing your site for tablets,* explores the emerging tablet market and covers special design considerations and conventions for designing for tablet use.

Chapter 11, *A Home in the Clouds*, explores team deployment solutions such as Hudson/ Jenkins, Features integration hooks and breaks down the go-live process to something that's repeatable and, with any luck at all, scriptable.

Appendix, *Pop Quiz Answers*, contains the answers to all the pop quiz questions for all the chapters.

What you need for this book

You'll need a Mac or PC to develop your website. Optionally, you might want to get an Amazon AWS account for the chapter on deployment.

Who this book is for

This book is for the aspiring website developer as well as more experienced developers.

Conventions

In this book, you will find several headings appearing frequently. To give clear instructions of how to complete a procedure or task, we use:

Time for action – heading

1. Action 1
2. Action 2
3. Action 3

Instructions often need some extra explanation so that they make sense, so they are followed with:

What just happened?

This heading explains the working of tasks or instructions that you have just completed.

You will also find some other learning aids in the book, including:

Pop quiz – heading

These are short multiple choice questions intended to help you test your own understanding.

Have a go hero – heading

These set practical challenges and give you ideas for experimenting with what you have learned.

You will also 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: "Go to `http://developer.java.com` and search for the JDK developer download. It will be a `.exe` file. Run the installer "

A block of code is set as follows:

```
<VirtualHost *:80>
  ServerName dpk.local
  DocumentRoot "C:\cygwin\home\[YOUR USER NAME]\sites\dpk"
  <Directory "C:\cygwin\home\[YOUR USER NAME]\sites\dpk">
    Options Includes FollowSymLinks
    AllowOverride All
    Order allow,deny
    Allow from all
  </Directory>
</VirtualHost>
```

When we wish to draw your attention to a particular part of a code block, the relevant lines or items are set in bold:

```
core = "7.x"
dependencies[] = "context"
dependencies[] = "views"
description = "Home page view"
features[context][] = "Home Page"
features[ctools][] = "context:context:3"
features[ctools][] = "views:views_default:3.0"
features[views_view][] = "home"
name = "Drupallos Homepage"
```

Any command-line input or output is written as follows:

```
cd ~/Sites
git clone git://github.com/drupal4mobile/dpk.git
```

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: "We then need to create an Android Virtual Device. Click on the **Virtual devices** tab and then click on the **New** button".

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 book that you need and would like to see us publish, please send us a note in the **SUGGEST A TITLE** form on `www.packtpub.com` or e-mail `suggest@packtpub.com`.

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

The code for this book is stored on GitHub at `http://github.com/drupal4mobile/dpk` and we'll use the GIT version control system throughout the book. I've organized the code for each chapter into a branch on GitHub. As this code is progressive from beginning to end, you can obtain the code for each chapter by checking out the GIT branch for that chapter. Feel free to fork any code on the repository and use it as you see fit. Any code in this book should be considered open source and released under the same license as Drupal.

After this book's publication I will attempt to take some of the custom modules used in the book to Drupal `contrib` module status. If I am successful, I will note such a change in the `dpk.make` file in the root of the install directory.

Also, 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

When is a Phone Not a Phone?

On January 9, 2007, Steve Jobs addressed a group of adoring Mac heads at the annual MacWorld conference. He talked about how Nokia had pioneered the mobile phone industry and created some of the best mobile phones in the market. He talked about how Blackberry had changed the world by creating devices that excelled at getting e-mails anywhere in the world securely and quickly, allowing you to respond to them instantly. He talked about how desktop computers were fantastic for browsing the Web, but no one had quite captured that desktop browsing experience in a handheld. Steve then showed the world the first iPhone—the world's first phone, e-mail, and mobile Internet device. The moment I saw it, I knew I had to have one.

I am, and have been an unabashed Apple fan. Both my parents were teachers and were able to check out Apple computers from the school library during the summer. This meant that I spent almost every summer of grade school in front of a monitor with the Apple logo on it. The iPhone was the culmination of all of those summers' enthusiasm times one thousand.

I went to the Tampa Apple Store after work on the day of the launch and I was the first one on my block to have a new iPhone. The data speed was excruciatingly slow and the phone dropped calls more than a freshman football player drops passes, but the device revolutionized what we thought a handheld could be. And with each release, the iPhone continued to get better. As it had done three decades earlier with the personal computer industry and two decades earlier with the GUI, Apple had ignited innovation in the handheld space that the world would never be the same.

Some five years later, it's impossible to imagine a Friday night without texting your friends and getting information on which restaurant in the area serves the best Persian Kebob, and then sharing the location with them all, so they meet you there without a single phone call exchanged between you. We live in a handheld world and many a pundit has stated, "If you don't have a mobile strategy, you don't have a strategy".

So, what is your mobile strategy? Whether you sell cupcakes on Main Street to 12-years-old girls, or servers to wall street traders, Drupal gives you a framework to create a first-class mobile website.

Drupal is an open source piece of software installed on servers that allows you to manage content for a website through a web-based interface. You can pair your content with one or more themes, which will wrap the content in the HTML design of your choice.

There are tons of better CMSs out there, both paid and open source. What makes Drupal what it is today are two basic things—one is the hundreds of thousands of developers contributing modules to the Drupal framework. Second is Drupal's approach. Drupal approaches web development as a series of Lego™ blocks, each one building on the other until the entity becomes greater than the sum of the parts. This book will attempt to demystify the process.

So what is it that you hope to accomplish with your mobile site? The chances are that it's one of the following three general goals:

- **Broadcast message**: Includes either news or information. Allows you to share information about your business or non-profit with current or future customers.
- **Interaction**: Allows your customers to find you, contact you, or in other ways, interact with you.
- **Location portal**: For example, find the best Kebob in Arlington, Virginia.

What you hope to accomplish will give you a better idea of who your audience is and what type of hardware they might have.

"Dumb" phones

In the beginning mobile telephones were large bricks that needed constant charging and had a very limited range. By the year 2000, Nokia and Motorola had most of the US Phone market selling what we now call "dumb" phones. Dumb phones were dubbed, thus, as a reaction to newer "smart" phones. They usually have a single color screen and a 12-key standard phone keypad with a power on and off button and maybe one or two other function keys. You can accomplish text input with a combination of presses on the keypad. The phone's secondary features such as storing phone numbers and texting are usually accomplished through a series of menus. The browsers on these phones use a protocol called **Wireless Application Protocol** (**WAP**). WAP was evolved by necessity on these "dumb" phones in the early 2000s. WAP breaks up the content into a series of menus and text cards, and you can use the menus to navigate through the text cards to achieve something approaching a wireless Internet experience. Unless you know for sure that your audience

is using WAP-only phones, you can probably choose to disregard it and author your site in HTML only. This book will only touch briefly on the WAP protocol and will turn most of its attention to full HTML sites, as the vast majority of mobile web traffic in the US and Europe are smartphones using WebKit-based full-HTML browsers.

However, there are some places where "dumb" phones in the US excel and are readily accepted. These include the following:

- Government jobs that require that your cell phone does not have a camera
- Many Health and Human Services buildings that deal with minors and much of the court system has security in place that doesn't allow cameras in the building

If you have a cell phone with a camera, it's confiscated by the security guards and given back to you when you exit. After having my cell phone confiscated at the door on a trip to traffic court recently, I noticed many of the lawyers in the building were carrying phones that look like they were from another era. As long as security measures such as these are in effect, there will always be a market for the circa 2001-era "dumb" phone with a T9 keyboard and a WAP web browser.

"Smart-er" phones

So between the "dumb" phone and today's high-end smart phones are what the industry calls "Smart-er" phones. They usually have color screens and a keyboard that enables faster SMS communication, but don't have the processing power or storage capability of a fully-fledged smart phone. Recently, cheap Android-based smart phones have supplanted these phones, but there are still a few of them in the market. They mostly use the WAP protocol that the dumb phones use, but with a few exceptions.

So, as with dumb phones, unless you know for sure that your audience is using these devices, you can eschew WAP and turn your attention toward building sites with standard HTML.

Smart phones

About a year after the iPhone was released, word leaked out onto some blogger sites that Google was creating an open source operating system for phones. Based on Linux, the new OS would be made available free of charge to handset manufacturers and would also offer some of the advanced functions available to iPhone owners. Google would also allow developers to submit their applications to a common market where users of the OS could purchase and download these. The industry was instantly a buzz. Android adoption started out slow, but as the OS improved, more and more manufacturers started releasing phones with their tweaked version of the Android OS. The touch screens and features on a par with the iPhone at a fraction of the cost and on the carrier of their choice drove Android to success and made millions for Google in search revenue. More than that, it gave the iPhone some competition—and competition almost always drives innovation.

Android's greatest strength was also its greatest weakness. Anyone could alter the OS and put it on their handset, which means that programmers had to contend with changes in the configuration from a multitude of hardware sources. When writing custom applications for the handheld, testing, and vetting each of the hardware configurations was, and continues to be, a nightmare for Android developers. And the submission process for applications to the Android App Store was not nearly as streamlined as for Apple's App Store. But creating a truly cross-hardware version of your application needn't be a lesson in patience.

Tablets

Just as earth-shattering as the iPhone announcement was Apple's release of the iPad. Tablet computing has been around for a number of years, but with a stripped down version of Windows as its OS. Many users rejected tablets as they considered them reductive—"less" of a computer than their desktop—and they never really caught on.

The iPad was all about more. It was like an iPhone, but more than that. Like a laptop, but without a clumsy fold-out keyboard and the battery lasts for 10 hours.

Since then a myriad of Android tablet-style devices have been released with a varying amount of sales success, but as of the date of this book's printing, the iPad remains king of the tablet world and will for a number of years in the future. There's some marketing research to suggest consumers are buying tablets in situations where they normally would have bought a laptop for $500, and with the emergence of cloud computing, it makes sense to forgo a bulky laptop in favor of a lightweight tablet.

Now, I have replaced my laptop with a 3G iPad that is always connected wherever I go. It's possible for me to travel for days without touching my laptop and do almost everything I could do with it, including administering servers, writing code, and authoring this book.

The emergence of the tablet is one of those decade-definer events along with the social networking, the Internet, and the personal computer before it, that would change the landscape of computing in ways that no one could predict.

WebKit

Although the Android platform and iPhone differ greatly, there's one very important similarity—their web browser. Both OSs have their web browsers based on the open source WebKit project.

In 2001, Apple needed a small, lightweight web browser for it's new Mac OS X operating system. They chose a little-known open source project called **KHTML** for its clean code, support of web standards, and minuscule (for a web browser) codebase. KHTML was basically a set of libraries you could use to render HTML documents in Windows. From the KHTML project, Apple spun off the WebKit and created their Safari web browser, and a few years later, Google created their Chrome browser from the same set of libraries. Because of its modern compact rendering engine, Safari was able to leapfrog Firefox and Internet Explorer in its support for emerging standards. It became the logical choice for both Apple and Google's handheld operating systems. When Blackberry updated its operating system to enable touch screens, they also chose the WebKit rendering engine, so when you're developing for mobile browsers, with a few exceptions, you're basically developing for WebKit, which makes life significantly easier than developing sites for the desktop. WebKit boasts some of the best rendering of CSS and HTML and some of the fastest JavaScript of any web browser.

Mobile-ize me!

But more often than not, you purchased this book because you have a particular website in mind that you need to "make mobile". I encourage you to rethink parts of your website that no longer make sense in light of a mobile paradigm. Things such as hover-based navigation, photo galleries of large-sized images, music, flash, or high-speed repeating animation on the home page will frustrate your mobile users for reasons we will make clear. So to "mobile-ize" your website, you may need to do more than just come up with some slick tricks and mobile-friendly CSS to make the mobile experience of your site first-rate.

Throughout this book, we will be working through an example website. *Chapter 2*, *Setting up a Local Development Environment*, will get the example set up, but if you have an existing site, feel free to set up a development environment of your own and add mobile additions to your existing site. The book is set up to work through the example adding features one-by-one, but the chapters will be generally self-contained and you can take them a la carte.

All-in-all, there are about 6500 different web-capable mobile devices. We will attempt to create a site that is viewable on the vast majority of those devices.

But all of this begs the question that is discussed in the section that follows.

What is mobile?

Is mobile just a piece of hardware? We've talked about hardware, some of the software, iOS versus Android, and WebKit versus WAP browsers. But what exactly is "mobile"?

Mobile is a context—the context of the user rather than the context of the computer. With most computing, the assumption is made that it will be done on a desktop or laptop and there will be a storage device, a display device, and multiple user inputs such as keyboards, mice, and trackpads.

Mobile relinquishes those assumptions and assumes only the user. Nothing about the device and screen may be assumed, because we could be talking about a mobile phone with a four-line screen and WAP browser. Nothing about the keyboard can be assumed because very few mobile phones have physical keyboards these days.

A "mobile" website must be adaptive and able to be displayed wherever and whenever the user needs or wants to and cannot make any hardware or software assumptions.

And that is why, in my opinion, it should be a different website from your standard desktop version.

The theme-ability of the content and agile nature of the CMS make Drupal a perfect tool to solve some of these design challenges, as you will soon see.

The "One Design" myth

So there's a theory of design. It states that we should design the website so that all the pieces work together in both a desktop context and in a mobile context. Use smaller design pieces that are adaptable to small screens and use CSS that will optimize the display for whatever the user chooses to view the website on. This is sort of a "one ring to rule them all" mentality.

I would disagree with this strategy. Not only is this an incredibly difficult task for a designer, but I think this approach overlooks a basic constraint. The ways in which we consume information on a desktop machine and that on a handheld device are completely different.

I believe the act of reading itself is completely different for all. More than that, I find the websites on which I'm most compelled to view content have two versions (a desktop version and a mobile version).

Feel free to disagree with me, but this is primarily the approach we will be taking in this book. And with Drupal as the CMS, you will see this approach becomes easy with the addition of contributed modules that separate content and allow different themes and JavaScript for different domain names.

For every rule there is an exception and there is, indeed, a place where the two-pronged site approach breaks down. Designing a website for tablets requires aspects of mobile design, a clear attention to touch events, and a design that's not quite tailored to the desktop and yet not quite a mobile phone.

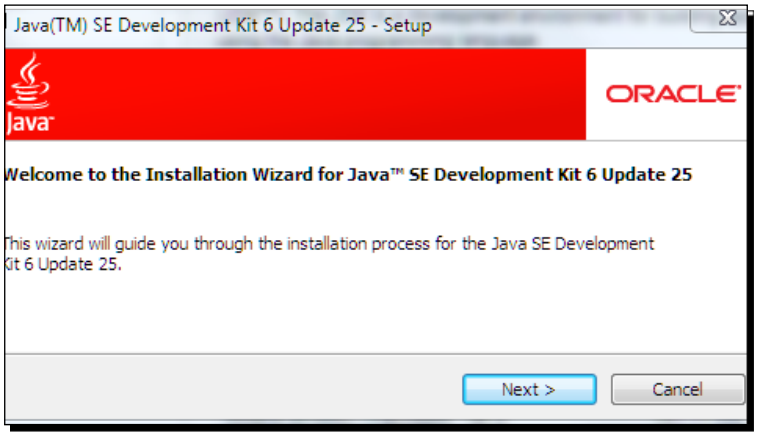
We will discuss the ways Drupal can power your tablet strategy and try to cover the gap between the smaller mobile site and the full-version desktop site.

Mobile simulators

Before we set up a development environment, we need to set up a simulation of our mobile devices. We do that with either of the two packages—the Android development package or the XCode/iOS development package. XCode is available for Mac OS X only, but the Android development environment is Java-based and will work with any modern OS that has a Java Development Kit installed.

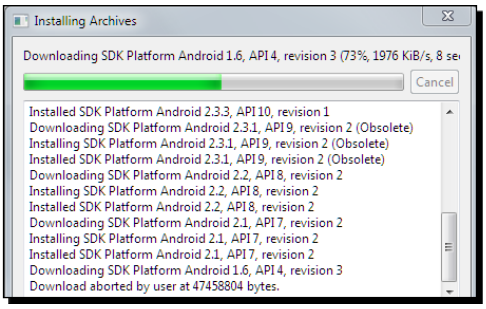
Time for action – installing an Android development package

1. To download and install the Android development package, go to `java.com` and download the **Java Development Kit** (**JDK**). Just having a regular Java package installed is not enough. You need to install the developer tools, too. Go to `http://developer.java.com` and search for the JDK developer download. It will be a `.exe` file. Run the installer:



2. Once JDK developer is installed, the next step is to get the Android development tools. These are available at `http://developer.android.com`. Download the `.exe` version of the Windows installer.

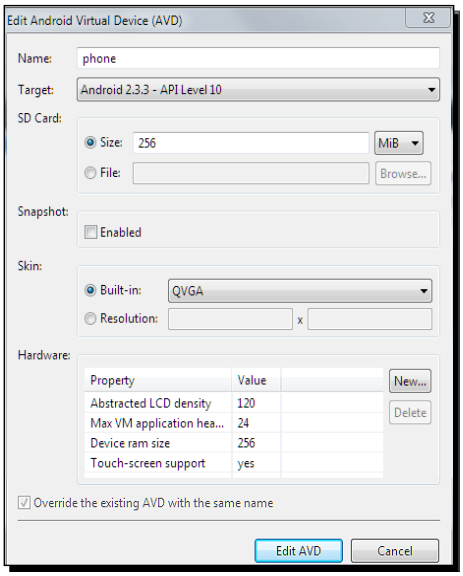
3. After the first Android installer runs, it will launch the Android **Software Development Kit** (**SDK**) and **Android Virtual Device** (**AVD**) manager. Yes, I know these are a lot of acronyms. Java developers love their three-letter acronyms. The SDK and AVD manager will ask to download the various android platform-specific APIs. Go ahead and choose the default options. Android will go through a series of downloads and install them after they are downloaded:



4. We then need to create an Android Virtual Device. Click on the **Virtual devices** tab and then click on the **New** button. Let's create one for Phone emulation and one for tablet.

5. Name the first one **phone** and select **Android 2.3.3-API Level 10** (which is the latest, at present).

6. Choose a screen size. The **QVGA** skin should work for most phones. Then click on the **Create AVD** button to launch the Virtual Device:



What just happened?

One of the current problems with the Android platform is called **fragmentation**. This means that there are a lot of devices that have been sold with various versions of the software on them. As Google produces new distributions of Android, some devices get the updated versions and some do not. Depending on how much the device manufacturer and carrier have customized the Android software, they may make the decision that it's in their best interest to force the user to upgrade to a different device rather than to distribute the updated software to existing devices.

As a developer, if you create an Android application, you have a situation where there are hundreds (maybe thousands) of different combinations of hardware and software to test with your application. In addition, because of the open source nature of the Android OS, each manufacturer of hardware has the ability to alter the Android experience. This includes adding applications that showcase their hardware's features and adding APIs that allow their phones to connect to special networks or services that only they offer. As if that wasn't enough, phone hardware manufacturers often alter the hardware or software of a phone in response to a request from the phone carrier. Verizon is known for not allowing certain phone features to be used without their permission or to be used in a way that allows Verizon to charge for the service.

The myriad of combinations of hardware, hardware manufacturer additions, Android-based OS versions, and carrier additions has the ability to make potential Android developers homicidal. Google is taking steps to minimize fragmentation, but at present, we have a hodge-podge of customer conditions that can only be described as "less than optimal". This makes mobile web development that much more compelling because the version of WebKit that ships with them has very little differences between the various software updates.

iOS

As a platform, iOS has the same problem with fragmentation that Android does. Their application development has a tendency to be a bit tyrannical. No software can be installed on an Apple device without first being submitted to the Apple's application review process. Legend has it that Steve Jobs' insistence on Apple's control over the user experience was a big stumbling block for Verizon and Sprint in the early days of developing the first iPhone and it led to them pairing with AT&T in the US for its initial release. Once the iPhone released and was being sold very well, AT&T asked Apple to limit the amount of YouTube videos or the quality that could be played on an iPhone. Apple's response was something to the effect of "No, we're not dumbing down our user experience for you." Apple's laser beam focus on the user and user experience is simultaneously iOS's biggest draw and its greatest developer gripe.

There's an active community who regularly jailbreaks iOS devices, or rather enables applications from other sources to be installed. In terms of numbers, it represents a very small proportion of total iOS device users and there's always the possibility that jailbreaking your iOS device could lead to "bricking" (Apple rendering the device useless or "like a brick"). And bricking your iPhone is, most assuredly, not covered under the warranty.

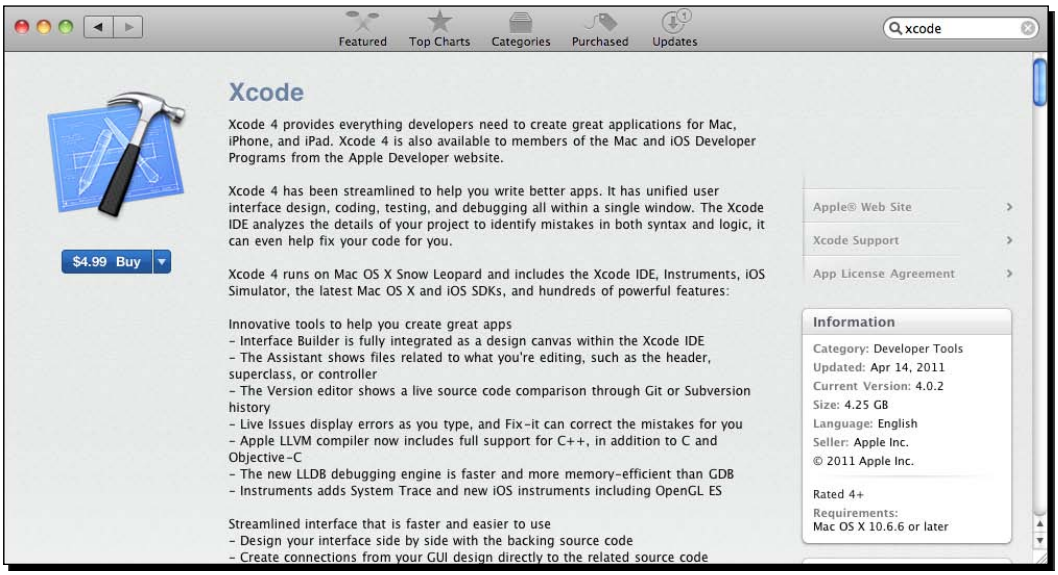
All software on an Apple device is written in Objective C. This is a dialect of the very popular and proven C programming language that's pretty much only used by Apple. When faced with learning a completely new programming language and Apple's API additions to the language, combined with Apple's App Store policy and the fact that any application you develop has no guarantee that it will be accepted and appear in Apple's App Store, many developers, again, turn to the mobile web.

Mobile websites need only a URL to be visible from any iOS device. But the quirks of the iOS web browser—Mobile Safari—make for a lot of trial and error JavaScript and CSS. Fortunately, Apple has mobile web simulation software available with its iOS developer toolkit that will allow us to preview our site before uploading our changes to the live version.

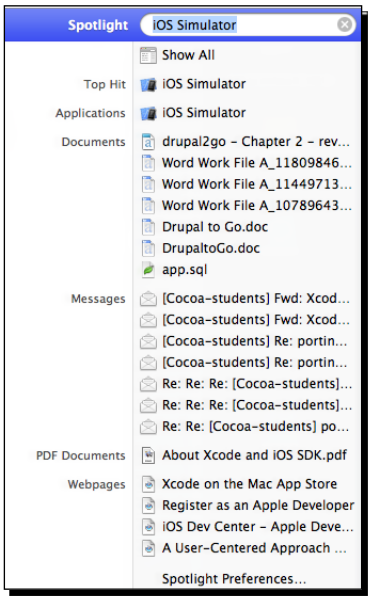
Time for action – installing the Mac OS developer's package

1. In order to download and install the Mac OS developer's package, you first need to register with Apple as a developer. Go to `http://developer.apple.com/programs/register/`.
2. If you have an iTunes username and password, you're welcome to use it or create a new one specifically for your development work. There are two options. You can register as an iOS developer, which costs $99 for a year and allows you to submit iOS applications to the iOS App Store. If you're not planning on doing any iOS development, you can simply download XCode 4 from the Mac App Store.

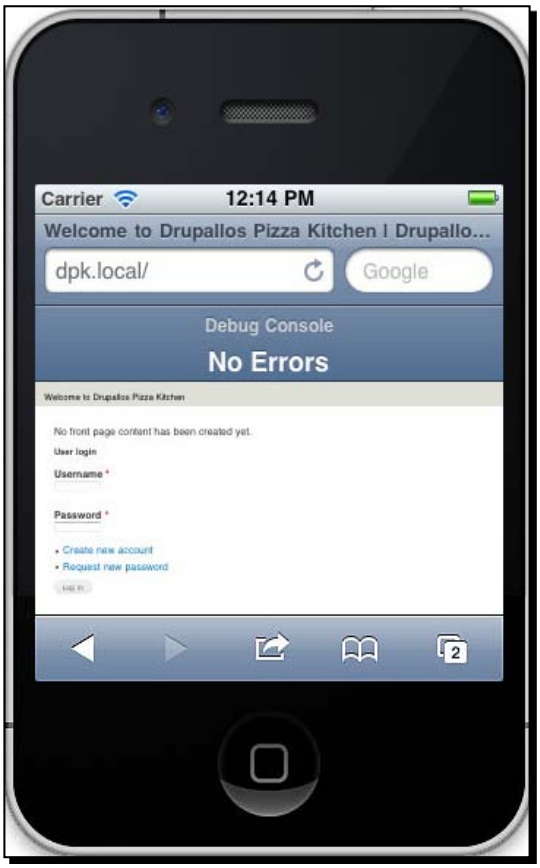
3. Once you've gone through the free registration at Apple's site, launch the Mac App Store and search for **xcode**.



4. Once XCode is installed, you can use the Mac's computer-wide search to find the **iOS Simulator**, as shown in the following screenshot:



5. Launch it and click on the **Mobile Safari** icon in the bottom tray. Go to `http://dpk.local`. It should appear in your desktop browser, as shown in the following screenshot:



What just happened?

Debugging CSS and JavaScript for web pages is never easy. It's made even more difficult with the proliferation of desktop operating systems and, now, handheld and mobile operating systems. No matter which browser claims to have "emulation" or modes that simulate this other browser or that, it's not a substitute for looking at the web page in the actual browsers. Debugging pages in Internet Explorer, requires an install of Windows XP and the browser itself. Don't settle for anything less. With handheld devices, you really need a virtual environment where the handheld OS is running a real version of the browser.

In this exercise, we created such a virtual environment. The XCode package creates the environment for the iPhone and iPad, and the Android development environment creates the virtual machine for any OS capable of running Java.

Summary

The changing mobile landscape requires a web content management system able to creatively address the problems of modern websites. Drupal is just such a CMS. I stated it earlier in the chapter, but it bears repeating—Drupal's worldwide developer base ensures that any problem you ever attempt to solve has been solved by multiple other developers in several different contexts.

The first task at hand is to set up a mobile simulator so we can view web pages as they appear on the device. We did that by installing an Android simulation environment on Windows and XCode from Apple's Mac App Store.

In the next chapter, we will take a look at our example site and get a sane development environment in which we'll build this fantastic mobile site.

We're going to outline a workflow that will allow developers in teams to work with Drupal in a way that allows everyone to do their jobs without stepping on anyone's feet, and at the same time, minimizing the amount of overhead work any developer needs to do. Ready to go? Let's get started!

2
Setting up a Local Development Environment

Working on a server, especially a server that is serving live traffic, is a little like walking a tightrope with no net. Only do it if you're willing to risk head trauma. This book attempts to introduce best practices to your development process. Frequently, we'll venture off the topic of developing, specifically for mobile. This will allow you to catch a glimpse of how, in a perfect world, a development team produces a professional Drupal website locally. From there, we'll push the new features the team has created up through different testing environments. Finally, we'll go live with the new code, or "production" as we will be calling it.

In this chapter, we will cover the following:

- Introduction to our example site, `drupallospizzakitchen.com`
- Best practices for local and remote development
- **Source Control Management** (**SCM**)
- Cloning a copy of our development site's unique files locally
- Some basic Drush commands
- Using a Drush make file to set up a complete and functioning site with a few simple commands
- Installing an AMP stack to allow a local version of our website using WAMP for Microsoft Windows and MAMP for Mac

- Resetting the Drupal root user's password in the database with an SQL command
- Installing an iOS or Android mobile web simulator to test websites locally before pushing them live on the Internet

So, let's get on with it.

Drupallo's Pizza Kitchen

Papa Vito Drupallo left Italy in the 1920s for New York with nothing but a tomato sauce recipe and a dream. He set up a small Pizza Shop in Brooklyn where he served up classic New York style pizza with old world flavor. Over the years, he passed the **Drupallo's Pizza Kitchen** (**DPK**) recipes and location down from one generation to the next. But, by the third generation, James Vincent (also known as Jimmy-V) Drupallo III was ready for some warmer weather and an easier life. Jimmy packed up the family into the Dodge Caravan and left New York for St. Petersburg, Florida. Jimmy runs Drupallo's Pizza Kitchen on St. Petersburg Beach during the day and in his off time, devotes himself to enhancing the family's Sangria recipe at their house a few blocks from the restaurant (the secret, of course, is soaking the fruit in brandy, but you didn't hear that from me).

© iStockPhoto/Duncan Walker

Just last year, Jimmy's son—Little Jimmy—created a great website for the business in Drupal, and by chance, is allowing us to create a mobile version of the website as an example for this book. We thank the Drupallo family for their gracious hospitality and be sure we'll come in for a slice and some Sangria next time we're in town.

Little Jimmy learned the art of tossing pizza dough from his father and makes some of the best New York style pizza crust in central Florida. His web development skills, however, are self-taught, so Little Jimmy doesn't even know about many of the best practices used in more structured corporate environments.

We'll introduce Little Jimmy to a development workflow that will ensure he never puts the family website at risk, and produces the best possible website on his local machine, before pushing the new features up to the live website for the world to see.

The Drupallo family's reputation goes into every slice of their pizza and is on the line with each page view of the website.

Averting disaster

Up until now, Jimmy (Little Jimmy, not Jimmy-V) has primarily developed the website on his own. He uses an FTP program to work directly off the server and the changes that he makes are essentially live as he makes them. He's now adding a few other developers (us) to help him out with the "mobile-izing" of his website, so he'll need to get everyone on the same page.

As previously stated, having multiple developers working off the same server is a recipe for disaster. Let's say you edit a file on the server. Another developer then uploads a newer version of the same file. A third developer uploads a newer version with your change but not the second change. The situation quickly becomes unsustainable. One solution to this is Source Code Management. **Source Code Management** (**SCM**) is a way of storing files that allows them to be progressively altered and stored in a way that allows everyone to access every revision of every file.

The two major SCM systems in use today are **Subversion** (**SVN**) and GIT. SVN is the older and more established version management system. However, recently, the entire Drupal community has moved to GIT, so for this example, we will use GIT. The primary difference between the two is that Subversion has a single-parent structure. There's an SVN repository that is authoritative for any given project. Every other instance is a "local checkout," and every time code is committed, the changes must be checked into the primary parent repository. It is a tree that has a single trunk.

GIT is known as a distributed version management system. Each GIT repository has no dependency on network access or any other repository. You can check files in and commit them to your local version without updating any remote version. The emphasis with GIT is on speed and non-linear development. It's very easy to branch a GIT repository (take the project in a completely new direction). GIT also has the concept of pushes and pulls. When you're making a change and you want the change to be reflected in all copies of the repository, you push your changes out. When you're gathering changes others have made, you're pulling data from one or more origins.

There are several commercial SCM suites available and many corporate programming environments purchase commercial software and have it tailored to their specific development needs. For our project and our needs, GIT will be more than sufficient.

So, the question now becomes, what do we manage? It seems silly to version manage the entire Drupal installation because everything, with the exception of a few files from the theme and custom modules, can be downloaded directly from `drupal.org` and remains unchanged from the version management system `drupal.org`.

Well, that problem is solved by a system called **Drush** and Drush's companion project, **Drush Make**. With Drush and Drush Make, we can describe a version of the Drupal core and a series of projects (modules) and libraries that make our own custom distribution of Drupal.

You may think that this is a lot of "command-line stuff", particularly if you're a frontend developer and used to using a GUI for all of your work. But stay with me; I promise there's a method to my madness and the time you spend learning the command line will pay off. In the final chapter, we'll show you how to roll that distribution with deployment scripts and create Drupal instances that build themselves with a few clicks of the mouse.

A **distribution**, or **distro**, is a series of open source projects assembled into a working group for a single purpose, so that it may be distributed and re-used. Ubuntu and Red Hat are distros of Linux, the open source operating system from Linus Torvalds. Drupal has several popular distributions and, in fact, many developers create their own distro for use in their projects with many of the modules and libraries they consistently use.

A word about platforms

Before we begin, it needs to be noted that the tools for Drupal development work best on Unix-based systems, for example, Mac OS X and Linux. Most of the tools themselves were built in Linux and then back-ported to Microsoft Windows, so there will be gaps in their compatibility with Windows. We will give examples for Microsoft Windows, but based on your version of Windows and the software you already have on the computer, it would be easy for a Windows local development buildout to go horribly wrong. Try to

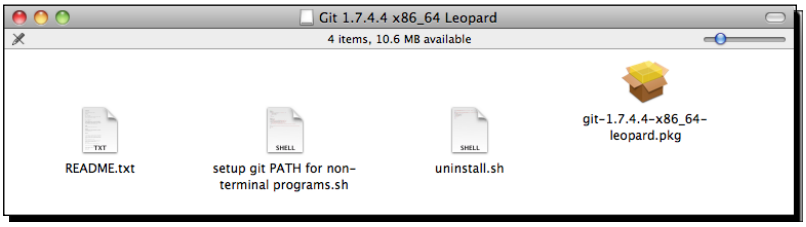
follow the examples as closely as possible and don't skip steps. Google is your best ally for troubleshooting build errors.

The only other caveat is that the iDevice simulator is only available for the Mac OS. Google's Android tools are Java-based and will work on either desktop platform. As a Mac OS X user, you will only need to test your site with the iPhone and iPad simulator, and Microsoft Windows developers can feel confident testing only with the Android simulator's web browser. The version of WebKit that's on Android is sufficiently close to the same one that's on the iPhone. Pages that work well on Mobile Safari should work equally well on Android's web browser.

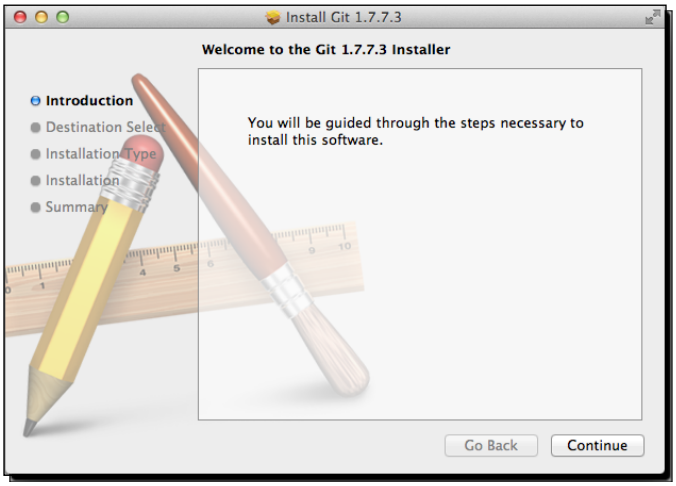
This book assumes that if you are technically astute enough to use Linux, you understand the differences between Mac OS X and Linux and can translate Mac instructions into linux-ese.

Time for action – downloading the SCM client for Mac

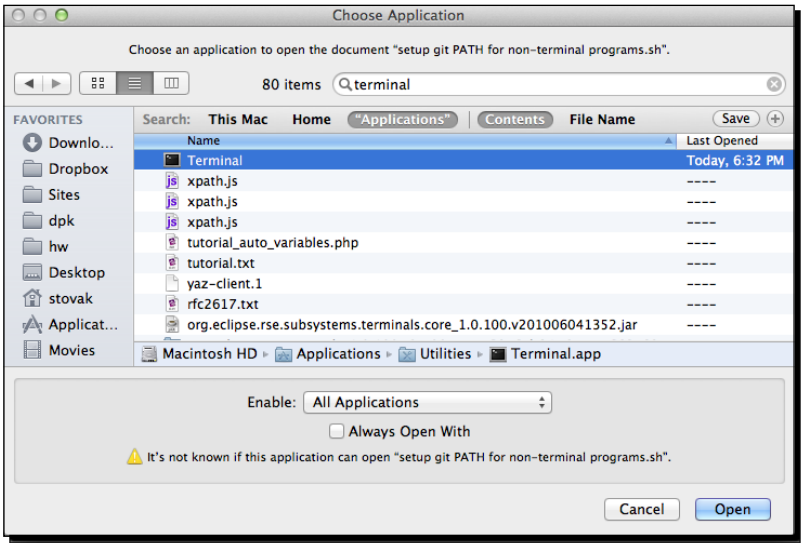
1. Navigate to `http://git-scm.com/download` and click on the **Mac OSX** link.



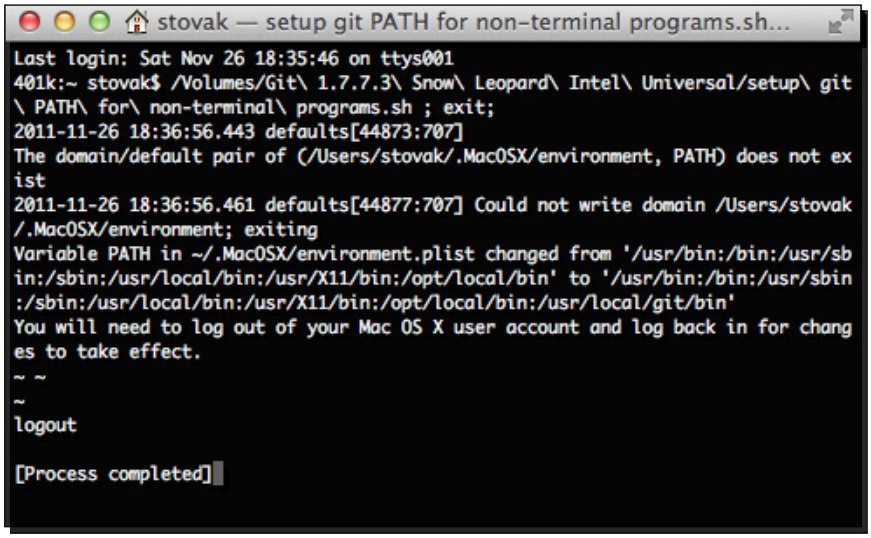
2. Select the package to install and the installer will start:



3. After the installation finishes, right-click on the `setup git PATH for non-terminal programs.sh` script and choose **Open With**. Select the **Terminal** application. If the **Terminal** application doesn't appear in your **Choose Application** list, you can select **other** and find it on your Mac. You can find it easily by typing **terminal** into the Search box, as shown in the following screenshot:



It should be under **Applications** | **Utilities**. After it executes, you can close this window:



4. Open a new terminal window by going to **Shell** | **New Window** and enter the following command:

   ```
   which git
   ```

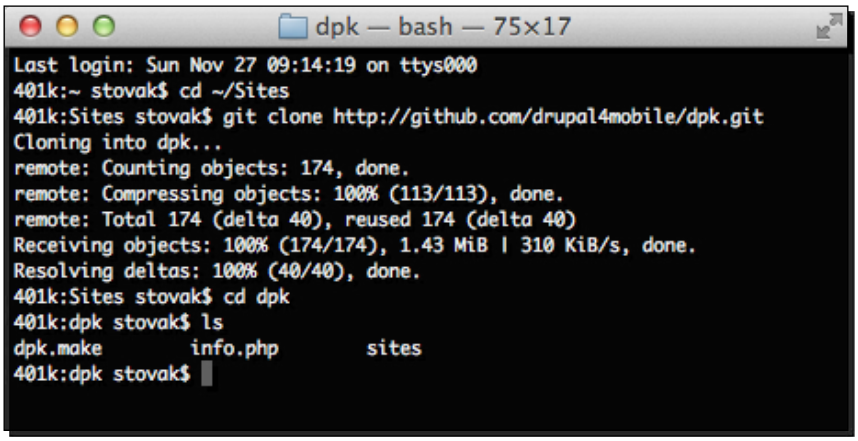
   The output, in response, should be `/usr/bin/git`. If you do not get the correct output, you have missed one of the previous steps. If this should happen, go back and repeat the previous steps.

5. Open your home directory. Under `/users/YOURUSERNAME`, there's a folder called `Sites`.

6. Now, open a new terminal window by going to **Shell** | **New Window**.

7. In the new window, enter the following:

   ```
   cd ~/Sites
   git clone git://github.com/drupal4mobile/dpk.git
   ```



What just happened?

In the preceding example, `cd` stands for "change directory". By entering this command, we moved into the directory that we created a few steps back.

GIT clone creates a local copy of the repository of the files for this site. In the directory, there should be several files that we will use to recreate the DPK website.

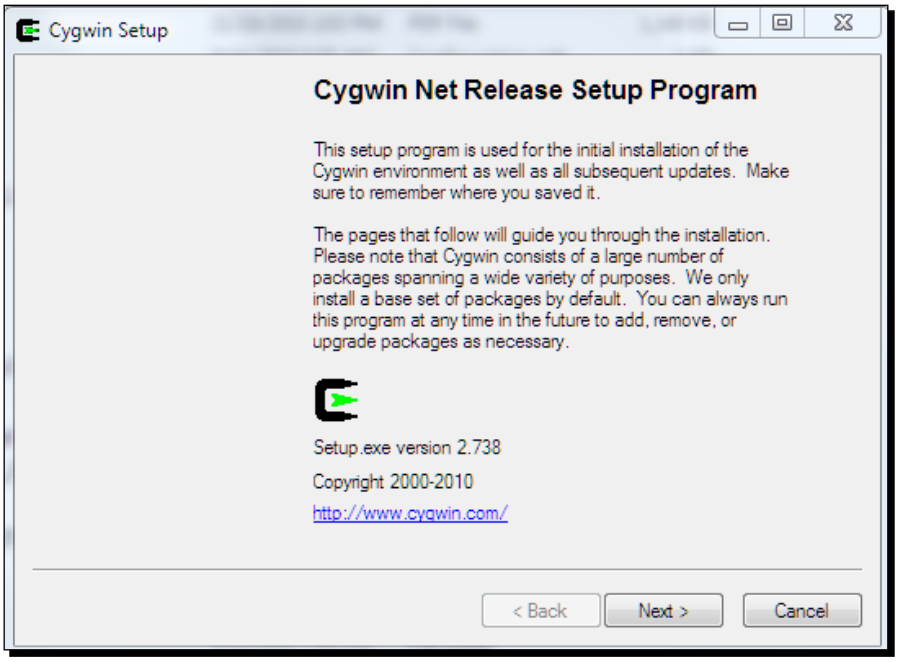
Congratulations! You've installed GIT and cloned your first repository. If you were successful, inside the `dpk` folder there should be a `dpk.make` file, an `info.php`, and a `sites` folder.

Cygwin and GIT for Microsoft Windows

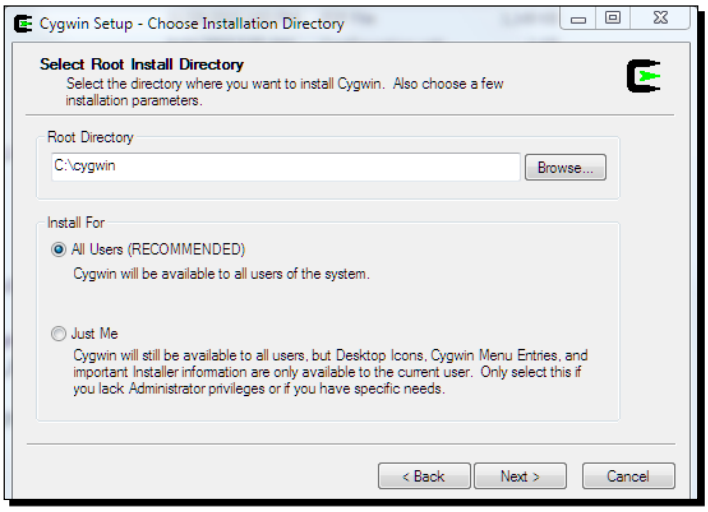
First of all, if you're a Mac or Linux user, there is no need for you to read this section. Skip down to the *Text editors for Mac OS X and Microsoft Windows* section. This is for Windows users only. Second, you must have administrative access to your machine in order to do any of the installations in this book. Almost all of these installations require you to be a Windows Administrative user for the local machine.

Time for action – installing a development environment using Windows and Cygwin

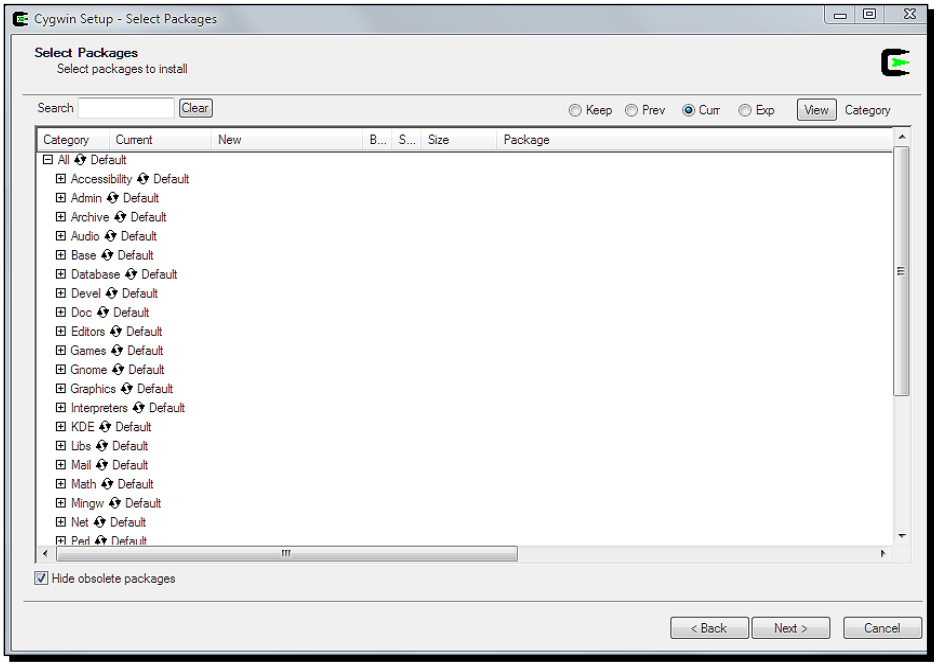
1. Download the installer from `http://www.cygwin.com`. Be sure to save the `setup.exe` installer somewhere you remember. You may need to run it again if another program requires a library that you didn't install on the first go-around:



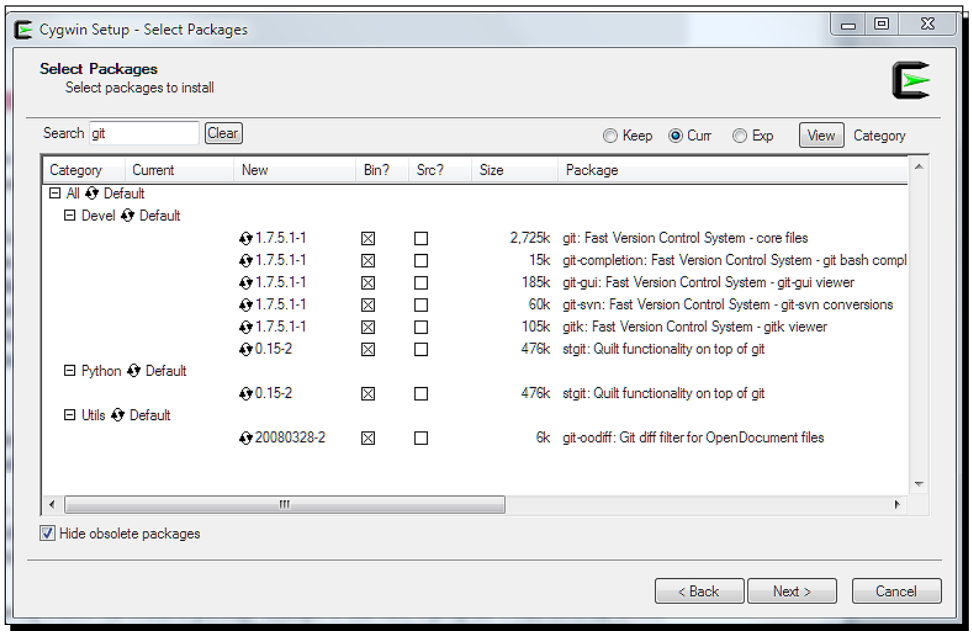
2. The installer takes you through several steps. For each step, choose the default option until you get to the **Select Packages** screen:



3. The **Select Package** screen (shown in the following screenshot) allows you to select which packages will be installed. After the installation, you can return to this screen to add or remove packages:



4. As shown in the following screenshot, enter **git** in the **Search** box and select the GIT packages to install:

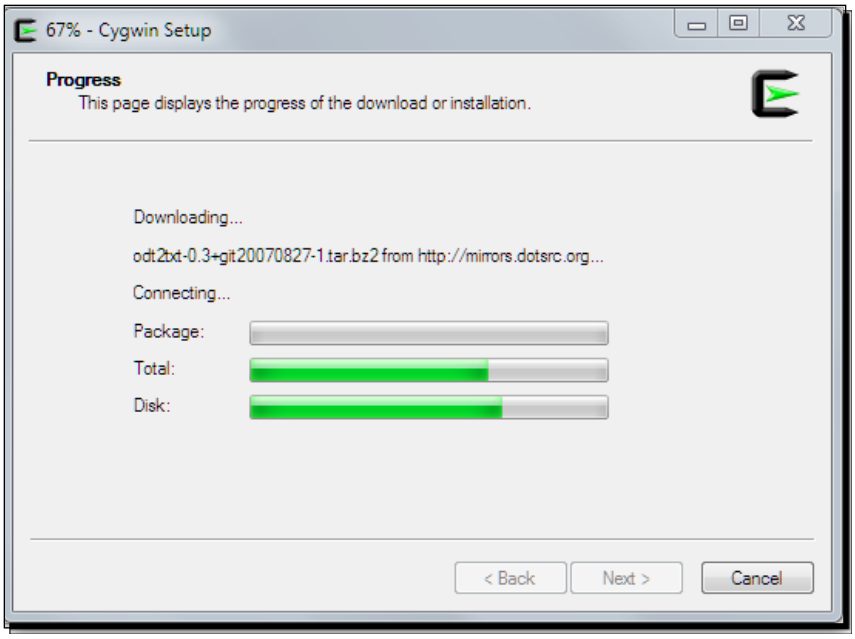


If the package is not installed, in the **New** column, you will see the **Skip** button. Clicking on **Skip** will show the version number to be installed and the **Bin?** checkbox will be checked. Bin stands for binary and Src stands for source. There is no need to install the source. Using this method, mark the following packages for installation:

- `git`
- `tar`
- `gzip`
- `bzip2`
- `ncurses`
- `ImageMagick`
- `zip`
- `unzip`
- `curl`
- `libcurl`
- `libcurl4`

- `libcurl-devel`
- `wget`

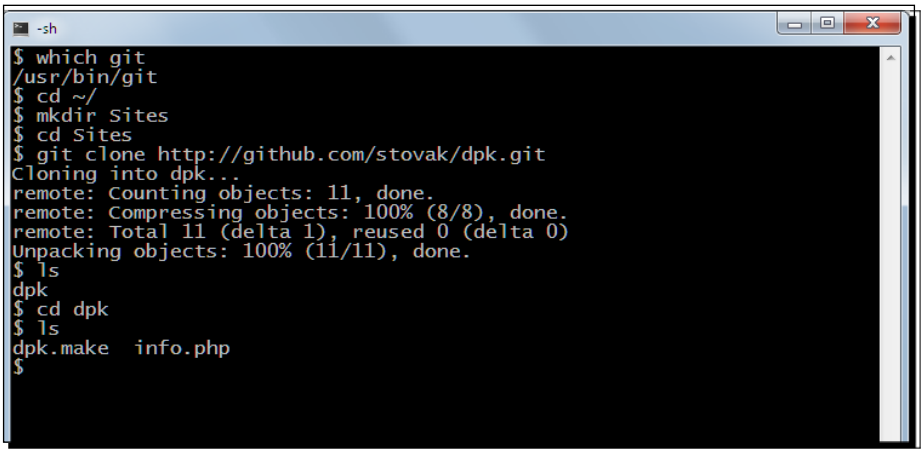
5. Click on the **Next** button and the installation will begin:



6. Once Cygwin has completely installed, go to **Start** | **All Programs** | **Cygwin** | **Cygwin Bash Shell**. The first time the terminal is launched, it will write several files that will help the command line to find the programs you need. When you have a blinking cursor, you can verify if GIT has been installed by entering the `which git` command. The output should be `/usr/bin/git`.

7. Now, it's time to clone your first GIT repository. Enter the following commands into the shell:

```
cd ~/
mkdir sites
cd sites
git clone http://github.com/0drupal4mobile/dpk.git
```



What just happened?

In many Unix-based operating systems, you have the ability to open what is called a "shell". In simple terms, a **shell** is a command-line interface with the computer's operating system. Microsoft Windows has one, but it's proprietary. It's Microsoft's own shell, and out of the box, it doesn't work with some of the basic utilities provided in open source Unix-like alternatives, such as Linux and Mac OS X's BSD shell.

Cygwin is a project for Microsoft Windows that brings the open source utilities to Microsoft Windows. It allows Windows to use many of the tools available to Linux and Mac OS X shell.

In this exercise, we installed Cygwin and added some libraries that we will need later in the book. Once Cygwin was installed and started, we used GIT to clone the repository where the code for our example project is stored.

GitHub, if you've never heard of it, mixes GIT version control with some of the functionality of a social network. It's a kind of "Facebook for nerds". We've chosen to host our code there because it's a great place to host public projects "in the cloud". They also have a paid service that allows you to host private projects. The idea is that you never have to configure any of the base software that runs the servers and you never have to worry about backing up your project.

Congratulations! You've installed GIT and cloned your first repository. Inside the `dpk` folder, there should now be three items: `dpk.make`, `info.php`, and a `sites` folder.

Text editors for Mac OS X and Microsoft Windows

Developers get very attached to their text editors. You learn the keyboard shortcuts and how to extend commands over the course of a few projects and your productivity takes a hit if you have to switch. In many development shops, there's a company-supplied text editor, but you're free to use the one you prefer. Before we get into some sort of text editor "West Side Story," with Sharks on one side and Jets on the other, please take these text editor recommendations as they are given. You're free to edit with whichever editor you feel most comfortable.

A good text editor can make all the difference when it comes to speed. That's why I prefer TextMate (`http://macromates.com/`) for Mac OS X and its companion, E-TextEditor (`http://e-texteditor.com`) for Microsoft Windows. I find them to be incredibly powerful and extendible in the programming language of your choice. If you don't have a good text editor, download one of these and use the trial version. I'll wager you'll be willing to buy the full version before the end of the book.

If you already have a text editor that you use and love, you're probably technically astute enough to translate any editor-specific parts of the book into the commands for your editor.

Also (and this is the general consensus of most people I know who program for a living) if you say your text editor is Adobe Dreamweaver, you will be laughed out of nerd school. Dreamweaver was great for crafting from start to finish HTML pages back when HTML was only just HTML, but it's completely inappropriate for coding the complex mix of HTML elements, JavaScript, and PHP that make up a standard Drupal install.

Serving web pages on your computer

LAMP is an acronym for **Linux, Apache**, **MySQL**, and **PHP**/**Perl**. It has become the world-wide standard for easily code-able, approachable web development and is the platform on which Drupal was built. Once upon a time, serving web pages required several server technicians, specialized in administering complex Unix installations. But not anymore!

Setting up a web server isn't rocket science. It takes a few simple open source tools that can be easily downloaded and installed. This makes it easy to run Drupal locally and push your changes to servers out on the Internet.

Setting up a local version of Apache and PHP used to take the better part of a day. Now there are installers that will do it for Windows or Mac with a few clicks. **WAMP**—acronym for **Windows**, **Apache**, **MySQL**, and **PHP**—for Windows and **MAMP**—acronym for **Macintosh**, **Apache**, **MySQL**, and **PHP**—are single-file installers that put the power of a web server on your local machine. If you're using any recent version of the Mac OS, it comes with Apache and PHP already installed. If you're adept at configuring them and understanding how to set them up for virtual hosting of multiple domains, knock yourself out. You've no need to read this section any further. It should also be noted that to preview websites on an iPad or iPhone, you'll need a Mac. The iOS developer tools aren't available for Windows.

If you're a Windows geek, you may be tempted to want to set up PHP to run with Microsoft Windows' built-in web server, IIS. However, IIS is completely inappropriate for a Drupal install, even a local one.

Furthermore, as someone who has had to work on Drupal sites served from a Windows-based server, I don't recommend Windows for anything other than development and then, only if you have to. There's a bunch of reasons why, but making the case is out of the scope of this book. Trust 15 years of web development experience. Using a Unix-based environment for as much of the process as possible will make your life more trouble-free and you will have less time wasted in making it work with Windows.

MAMP

MAMP is the acronym for **Mac OS X**, **Apache**, **MySQL**, and **PHP**. The Mac download of the MAMP version comes in two flavors, namely, the free version and the Pro version. If all you want to do is develop these examples, you can do so with the free version. If you're going to develop multiple sites and will need to quickly and easily switch between them, you'll want the Pro version. For whichever package you choose, download and run the installer as you would do for any other Mac program and then we'll configure our first virtual host. For the example, we'll be using the Pro version. If you're following along, the Pro version is free for 15 days.

Downloading the example code

The code for this book is stored on GitHub at `http://github.com/drupal4mobile/dpk` and we'll use the GIT version control system throughout the book. I've organized the code for each chapter into a branch on GitHub. As this code is progressive from beginning to end, you can obtain the code for each chapter by checking out the GIT branch for that chapter. Feel free to fork any code on the repository and use it as you see fit. Any code in this book should be considered open source and released under the same license as Drupal.

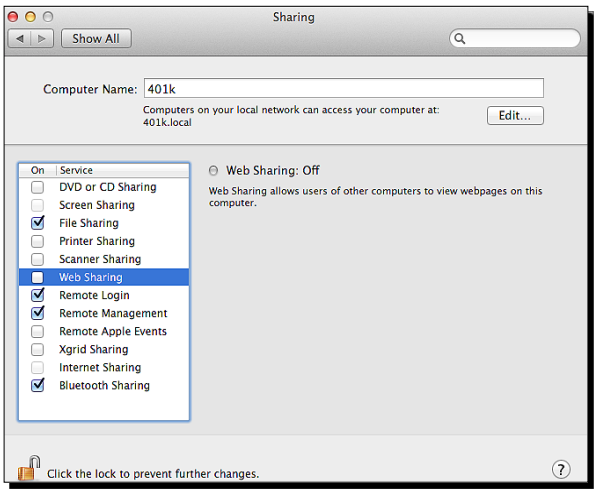
After this book's publication I will attempt to take some of the custom modules used in the book to Drupal contrib module status. If I am successful, I will note such a change in the `dpk.make` file in the root of the install directory.

Also, 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.

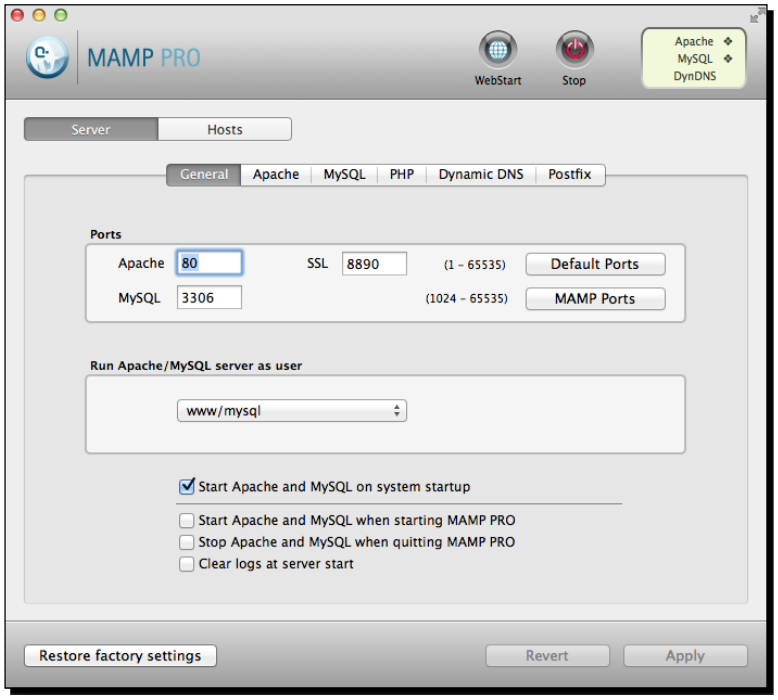
Time for action – configuring our first virtual host

We're going to configure our local web server to display `dpk.local`. You must be an administrator on your machine to finish this task.

1. Go to the Apple menu | **System Preferences** | **Sharing**. Make sure **Web Sharing** is disabled. This is the Mac OS Default install of Apache. We want to disable it in favor of the MAMP one. If you've never done web development on your local machine, you should not have a copy of MySQL installed. If you do, you should know that you do. Uninstall and/or disable your computer's current MySQL instance:



2. Launch the **MAMP Pro** application. As shown in the following screenshot, there are two main panels (**Server** and **Hosts**) and several subpanels under each main panel. Click on the **Server** panel and then on the **General** subtab. Click on the **Default Ports** button. If you have administrative access to your machine, you can change the user to **www/mysql**. This means the computer will run the processes of the web server as generic users created specifically for those processes. If you don't know what administrative access, is or are pretty sure you don't have it, feel free to run the web server under your username. Uncheck the **Start Apache and MySQL when starting MAMP PRO** and **Stop Apache and MySQL when quitting MAMP PRO** checkboxes and make sure the **Start apache and MySQL on System Startup** checkbox is checked:

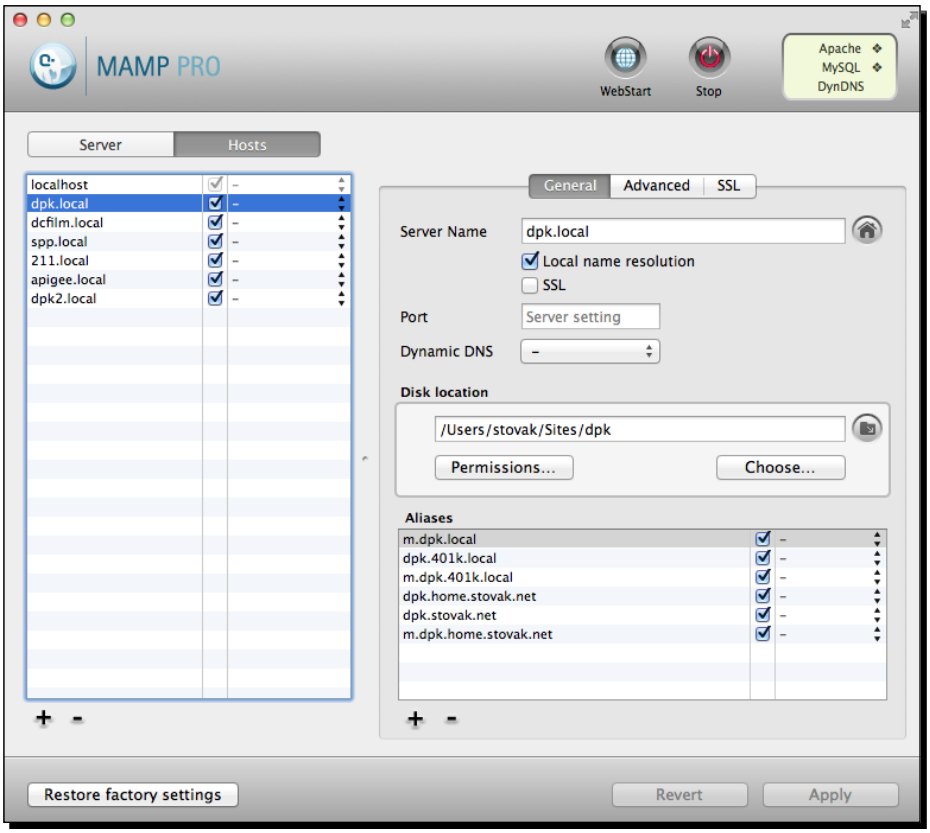


3. Click on the **PHP** panel and choose a version of PHP. You will see a **5.2.x** and a **5.3.x** version. If you're not sure which version to use, just use the 5.3.x version. Click on the **Apply** button in the bottom-right-hand corner to apply all settings, which will restart the servers.

5.2.x versus 5.3.x

For the examples in this book, you're free to use PHP 5.2 or 5.3. The latest core of both Drupal 6 and Drupal 7 is 5.3 clean, so there's no reason not to use the latest version unless you know you are going to be using a module that demands the older version and is incompatible with 5.3. Because 5.3 has some optimizations and 5.2 does not, with 5.3 you get the above 10 percent speed bump.

4. Click on **Hosts**. At the bottom of the virtual host list, there is a **+** (plus sign). Click on the **+** (plus sign). This will create a new item in the virtual host list. Name that item **dpk.local**. In the area labeled as **Aliases**, add **m.dpk.local** as an alias.

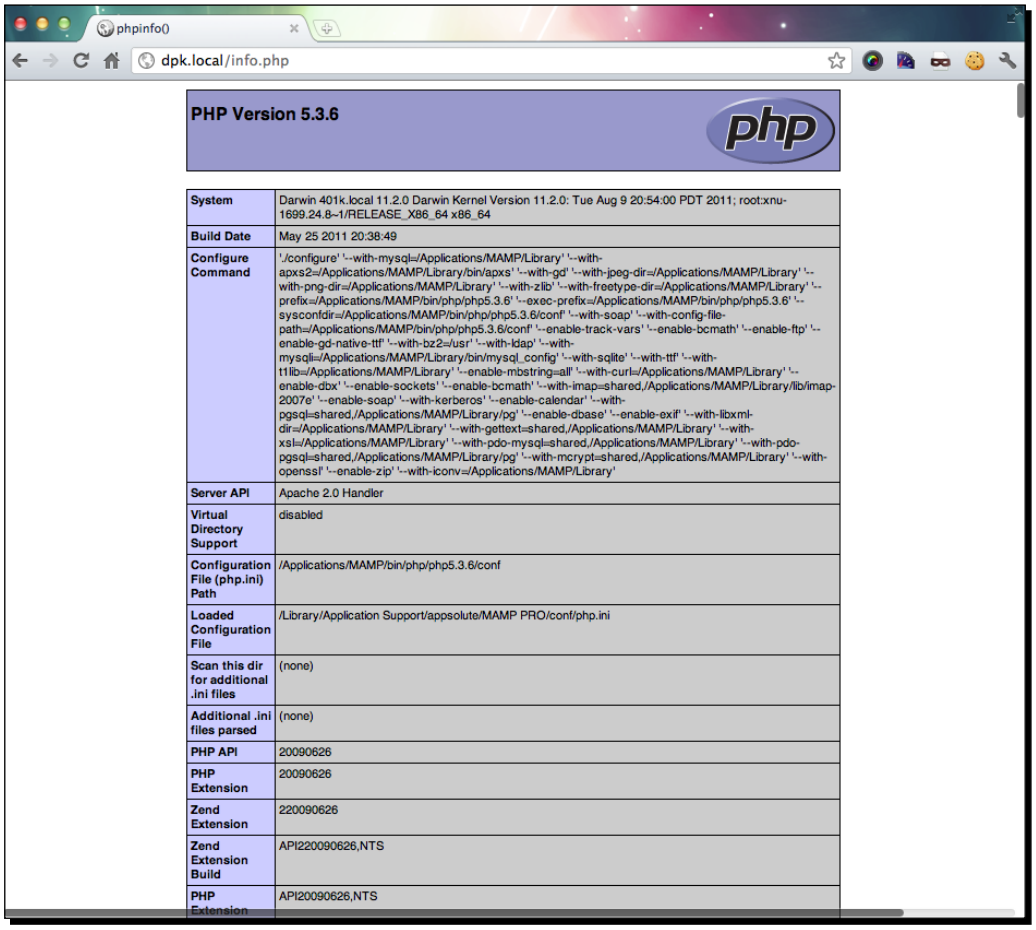


Make sure the **Local name resolution** checkbox is checked.

5. Now, choose the `dpk` folder you created earlier: **/Users/<YOURUSERNAME>/Sites/dpk**. Click on **Save**.

6. At the top of the MAMP Pro window, there's a button labeled **Restart services**. Click on that button to restart Apache and MySQL.

7. In the Safari browser, go to `http://dpk.local/info.php`:



System	Darwin 401k.local 11.2.0 Darwin Kernel Version 11.2.0: Tue Aug 9 20:54:00 PDT 2011; root:xnu-1699.24.8~1/RELEASE_X86_64 x86_64
Build Date	May 25 2011 20:38:49
Configure Command	'./configure' '--with-mysql=/Applications/MAMP/Library' '--with-apxs2=/Applications/MAMP/Library/bin/apxs' '--with-gd' '--with-jpeg-dir=/Applications/MAMP/Library' '--with-png-dir=/Applications/MAMP/Library' '--with-zlib' '--with-freetype-dir=/Applications/MAMP/Library' '--prefix=/Applications/MAMP/bin/php/php5.3.6' '--exec-prefix=/Applications/MAMP/bin/php/php5.3.6' '--sysconfdir=/Applications/MAMP/bin/php/php5.3.6/conf' '--with-soap' '--with-config-file-path=/Applications/MAMP/bin/php/php5.3.6/conf' '--enable-track-vars' '--enable-bcmath' '--enable-ftp' '--enable-gd-native-ttf' '--with-bz2=/usr' '--with-ldap' '--with-mysqli=/Applications/MAMP/Library/bin/mysql_config' '--with-sqlite' '--with-ttf' '--with-t1lib=/Applications/MAMP/Library' '--enable-mbstring=all' '--with-curl=/Applications/MAMP/Library' '--enable-dbx' '--enable-sockets' '--enable-bcmath' '--with-imap=shared,/Applications/MAMP/Library/lib/imap-2007e' '--enable-soap' '--with-kerberos' '--enable-calendar' '--with-pgsql=shared,/Applications/MAMP/Library/pg' '--enable-dbase' '--enable-exif' '--with-libxml-dir=/Applications/MAMP/Library' '--with-gettext=shared,/Applications/MAMP/Library' '--with-xsl=/Applications/MAMP/Library' '--with-pdo-mysql=shared,/Applications/MAMP/Library' '--with-pdo-pgsql=shared,/Applications/MAMP/Library/pg' '--with-mcrypt=shared,/Applications/MAMP/Library' '--with-openssl' '--enable-zip' '--with-iconv=/Applications/MAMP/Library'
Server API	Apache 2.0 Handler
Virtual Directory Support	disabled
Configuration File (php.ini) Path	/Applications/MAMP/bin/php/php5.3.6/conf
Loaded Configuration File	/Library/Application Support/appsolute/MAMP PRO/conf/php.ini
Scan this dir for additional .ini files	(none)
Additional .ini files parsed	(none)
PHP API	20090626
PHP Extension	20090626
Zend Extension	220090626
Zend Extension Build	API220090626,NTS
PHP Extension	API20090626,NTS

What just happened?

Apache is a program on your computer that serves web pages to the outside world. It is the most successful open source project in the short history of open source software. Apache runs some 50 percent of the active domains on the Web.

Virtual hosts are Apache's way of serving multiple websites out of multiple directories off the same computer and on the same IP address.

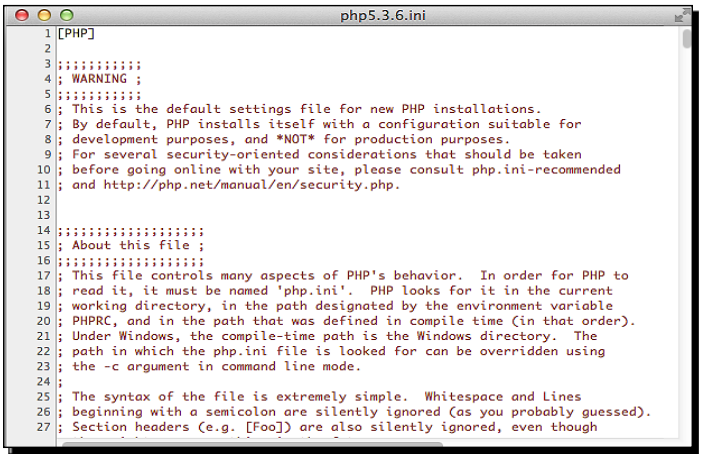
In this exercise, we've shut down the Mac's default install of Apache favoring our own. We created a virtual host named `dpk.local` and configured Apache to serve web pages out of the directory we created with the DPK website we cloned in the last exercise. If you've completed all these steps correctly, you should be greeted with a PHP info page.

One thing to note here is your PHP version. It's at the top of the page you just viewed. In this example, it's 5.3.6. Wherever the PHP version 5.3.6 is used in any of the examples in the rest of this chapter, please substitute your version.

But there are a few problems we have to correct. Some of the default settings for MAMP need to be changed in order for Drupal to function correctly. We'll need to do that by editing the default configuration files for both PHP and MySQL. Fortunately, MAMP makes this very easy.

Time for action – changing the default configuration for MAMP

1. Launch **MAMP Pro**.
2. Go to **File** | **Edit Template** | **PHP** | **5.3.6** (your version may be different. It's the 5.3 that's the important part, not the last number):

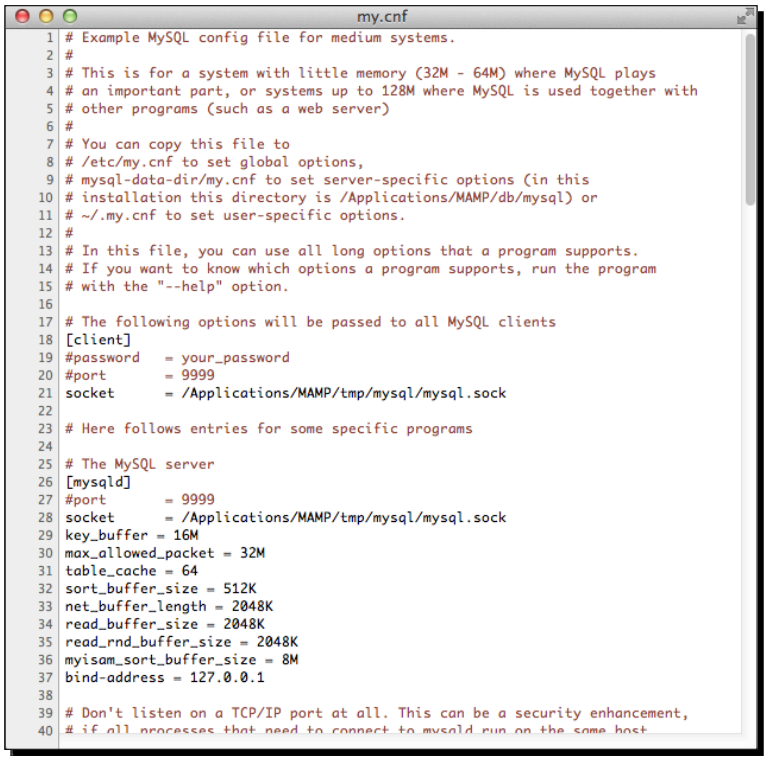


3. On the left-hand side of the text editor are line numbers. Around line 230, you'll find three lines: `max_execution_time`, `max_input_time`, and `memory_limit`. Change their values as follows:

```
max_execution_time = 999;
max_input_time = 999;
memory_limit = 256M;
```

4. Save and close the file after altering the values.

5. Go to **File** | **Edit Template** | **MySQL my.cnf**.



6. As shown in the preceding screenshot, raise the **max_allowed_packet** size to **32M**, raise the **net_buffer_length**, **read_buffer_size**, and **read_rnd_buffer_size** each to **2048K**. Save and close the file.

7. Using the red button at the top right of the MAMP window, stop and start the server's services.

8. Navigate again to `http://dpk.local/info.php`. Look for the `memory_limit` value on the page. It should be the same as the value you set in step 3, `256M`.

What just happened?

The default settings for MySQL and PHP are a bit on the conservative side. We changed some of the defaults to allow Drupal some more memory to play in and gave MySQL some more cache for it to communicate with PHP more efficiently. We then restarted the server to see our settings take effect.

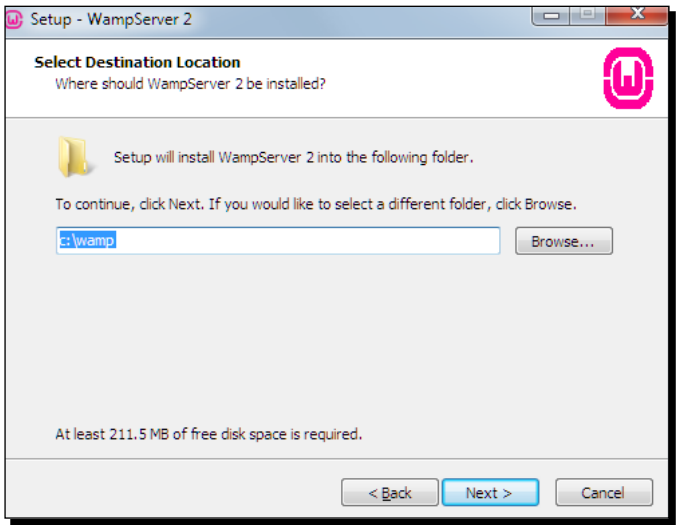
WAMP (Windows, Apache, MySQL, and PHP)

Again, if you have a Mac, you can skip the instructions in this section.

As with everything, things on Microsoft Windows are a bit more complex. You'll need to download and install the WAMP package, but the GUI to configure virtual hosts is not nearly as intuitive.

Time for action – installing WAMP

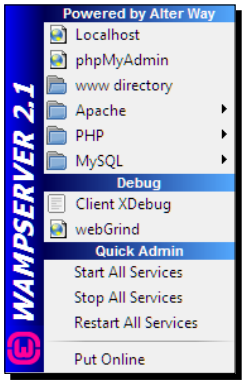
1. Download the WAMP installer from `http://www.wampserver.com/`. The default site is in French. Click on the **ENGLISH** link to get the English version of the site. In the English version, there is a link to download the latest version of the WAMP server. Once it is downloaded, run the installer.
2. Run the installer in the root directory of your `C:` drive. Believe me, you'll be happier installing everything there:



3. Step through the remaining screens of the install wizard using the default choices.

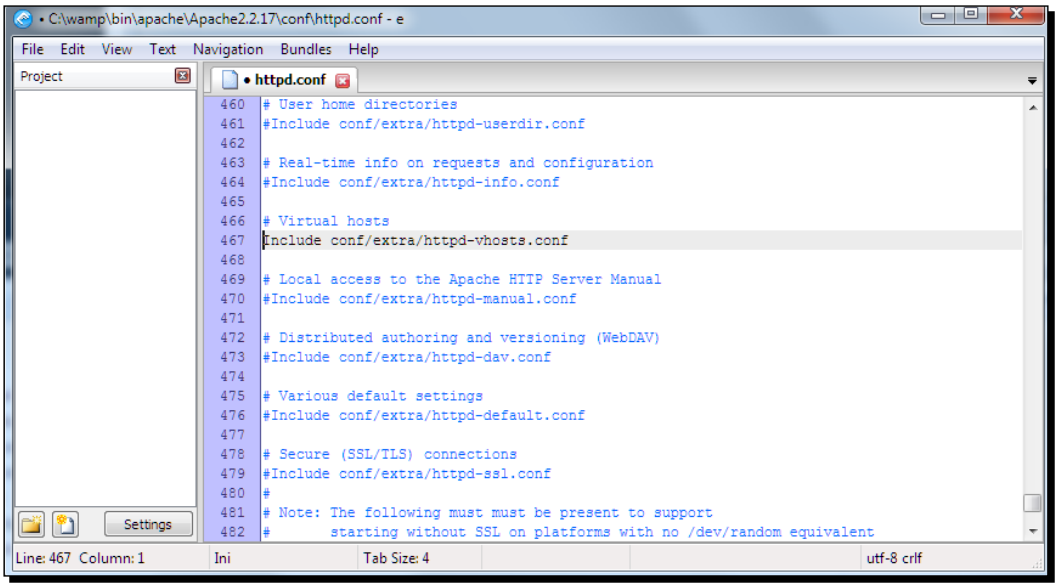
4. Go ahead and start up the WAMP server, if it hasn't already been started for you. You should see the W icon in your system tray.

5. Right-clicking on the W icon gives you the WAMP server menu, as shown in the following screenshot:



6. In Windows Explorer, navigate to `C:\wamp\bin\apache\Apache{VERSION}\conf` where `VERSION` is your Apache version number.

7. Right-click on the `httpd.conf` file and select **Edit with e** (or whatever your favorite text editor is).



8. Uncomment the Virtual hosts inclusion line (number **467** in our version of Apache). This will allow us to add virtual hosts to Apache without further changes to the `httpd.conf` file.

9. Inside the `conf` folder, there is another folder called `extras`. Open the `extras` folder.

10. Right-click on **vhosts.conf** and select **Edit with e**.

11. Add the following lines at the bottom, then save and close the file:

```
<VirtualHost *:80>
  ServerName dpk.local
  DocumentRoot "C:\cygwin\home\[YOUR USER NAME]\sites\dpk"
  <Directory "C:\cygwin\home\[YOUR USER NAME]\sites\dpk">
    Options Includes FollowSymLinks
    AllowOverride All
    Order allow,deny
    Allow from all
  </Directory>
</VirtualHost>
```

12. From the WAMP icon in your taskbar, choose **Restart All Services**.

13. Right-click on the **E Text Editor** icon, select **Run as** and run it as an administrator.

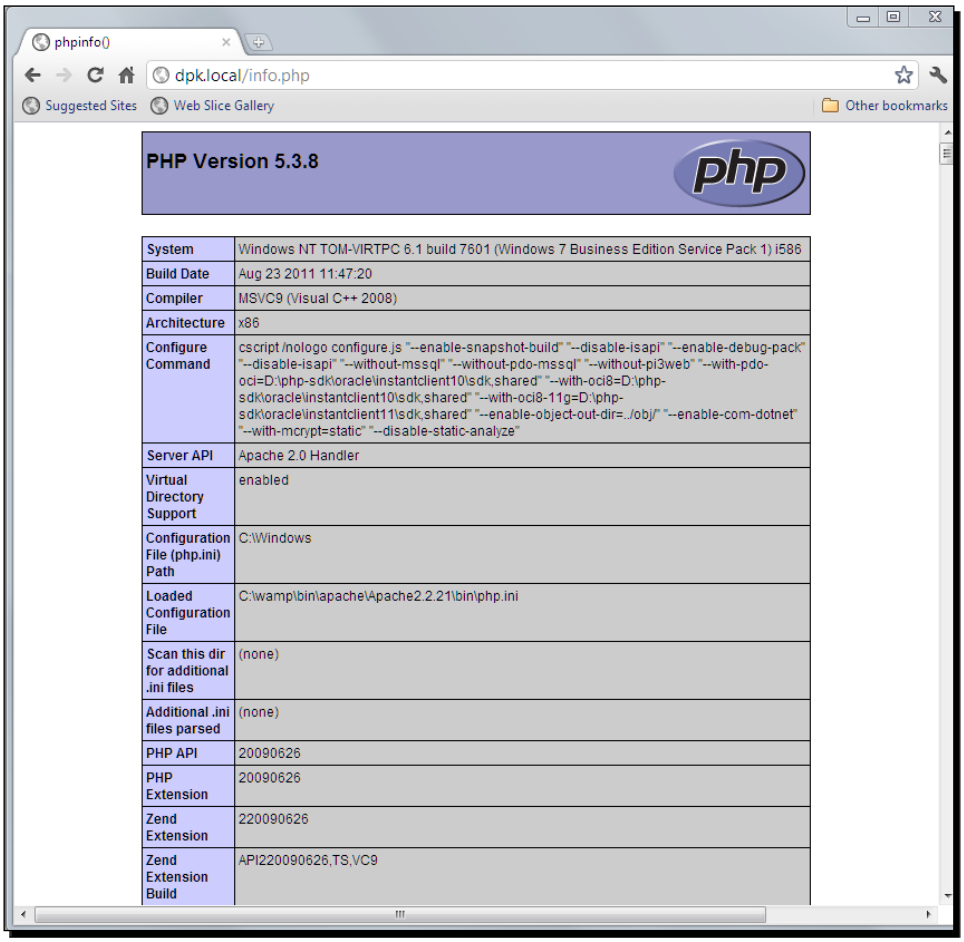
14. Choose **File** | **Open**.

15. Navigate to `C:\Windows\System32\drivers\etc` and open the `hosts` text file. Add the following line:

```
127.0.0.1       dpk.local
```

16. Save and close the file. Exit out of the editor.

17. Open your web browser of choice and go to `http://dpk.local/info.php`. You should be greeted by a PHP info page, as shown in the following screenshot:



System	Windows NT TOM-VIRTPC 6.1 build 7601 (Windows 7 Business Edition Service Pack 1) i586
Build Date	Aug 23 2011 11:47:20
Compiler	MSVC9 (Visual C++ 2008)
Architecture	x86
Configure Command	cscript /nologo configure.js "--enable-snapshot-build" "--disable-isapi" "--enable-debug-pack" "--disable-isapi" "--without-mssql" "--without-pdo-mssql" "--without-pi3web" "--with-pdo-oci=D:\php-sdk\oracle\instantclient10\sdk,shared" "--with-oci8=D:\php-sdk\oracle\instantclient10\sdk,shared" "--with-oci8-11g=D:\php-sdk\oracle\instantclient11\sdk,shared" "--enable-object-out-dir=../obj/" "--enable-com-dotnet" "--with-mcrypt=static" "--disable-static-analyze"
Server API	Apache 2.0 Handler
Virtual Directory Support	enabled
Configuration File (php.ini) Path	C:\Windows
Loaded Configuration File	C:\wamp\bin\apache\Apache2.2.21\bin\php.ini
Scan this dir for additional .ini files	(none)
Additional .ini files parsed	(none)
PHP API	20090626
PHP Extension	20090626
Zend Extension	220090626
Zend Extension Build	API220090626,TS,VC9

What just happened?

The series of directives we just added to the Apache file allow pages to be served on your local machine. The first line tells Apache we'll be serving on Port 80, which is the default port for web servers. The second line gives the server the name `dpk.local`. The third line points the server towards the document root we created in a previous step. The next five lines tell Apache to make special considerations for the `dpk` folder and allow us to serve web pages out of that directory. For security purposes, we deny Apache the ability to serve web pages out of most of the hard drive by default and each site we create has to have an exception created to that rule.

One thing you will need for the next chapter is your PHP version. It should be at the top of the info page at the end of this exercise. For this example, it's **5.3.8**.

Drush and Drush Make

Drush is a command-line shell and Unix scripting interface for Drupal. These few words from the Drush's `README.txt` file (`http://drupal.org/project/drush`) only hint at the power Drush gives you over a Drupal website. Simple tasks such as clearing the Drupal cache and backing up the database become scriptable and allow you to set up jobs to do them automatically. Complex tasks such as checking out code and syncing databases across a development environment become child's play, and you can do them with a few simple keyboard commands.

If you are a designer and are relatively new to web development, you might have a lot of reasons to fear using the command line. The ideas involved are not complex. Embrace the command line and it will simplify many of the common tasks you find complex and time consuming.

Drush Make (`http://drupal.org/project/drush_make`) adds the functionality of being able to describe a Drupal site with a few information files, download the files it needs, install the files, run any necessary database updates, and check the unique code out of a version management system, thus "making" a website.

Drush and Drush Make are Drupal version agnostic. They don't depend on a particular version. The latest version of both will work with Drupal 6 or Drupal 7.

If you are using version 5.0 or later of Drush, Drush Make has been integrated into the Drush core and there is no need to install Drush Make as a separate package.

Time for action – installing Drush and Drush Make for Mac OS X

Open the Mac OS X terminal and enter the following commands. Your PHP version may be different. Substitute your PHP version for the version number in the command:

```
echo "export PATH=/Applications/MAMP/Library/bin:/Applications/MAMP/
bin/php/php5.3.6/bin:$PATH" >> .bash_profile
export PATH=/Applications/MAMP/Library/bin:/Applications/MAMP/bin/php/
php5.3.6/bin:$PATH
sudo mv /Applications/MAMP/bin/php/php5.3.6/conf/pear.conf /
Applications/MAMP/bin/php/php5.3.6/conf/backup_pear.conf
sudo pear upgrade
sudo pear install console_table
sudo pear channel-discover pear.drush.org
sudo pear install drush/drush
```

```
cd ~/
mkdir .drush
cd .drush
git clone --branch 6.x-2.x http://git.drupal.org/project/drush_make.
git
which drush
```

The output on the console should be something like as follows:

```
/Applications/MAMP/bin/php/php5.3.6/bin/drush
```

The correct answer is any answer other than "Not found".

What just happened?

We first added the new `php` and `mysql` path to the `.bash_profile`. We then added the paths to the active profile of the terminal window. The third line moves the default `pear.conf` file out of the way because there's an issue with the version that ships with MAMP. We then upgrade all the pear modules to the latest version. Drush requires the `console_table` PEAR library. The second `pear` command installs it. We then added the Drush pear channel and installed Drush from the pear channel. We then cloned the `drush_make` repository in our home folder in a folder called `~/.drush`. This is the default location for Drush aliases and settings for any given user.

After we installed Drush and Drush Make, we double-checked the installation by showing the Drush install, which is now in the executable path.

Time for action – installing Drush for Windows

1. For Microsoft Windows, choose **Start** | **All Programs** | **Cygwin**, right-click on **Cygwin Terminal**, select **Run as** and then run it as an administrator. Enter the following commands:

   ```
   mkpasswd -l > /etc/passwd
   mkgroup -l > /etc/group
   ```

2. Close the terminal window and navigate to `http://drush.ws/drush_windows_installer` in the web browser of your choice and download the Drush Windows installer.

3. Once downloaded, run the Drush Windows installer.

4. Open a terminal window and enter the following commands:

   ```
   cd ~/
   mkdir .drush
   cd .drush
   ```

```
git clone --branch 6.x-2.x http://git.drupal.org/project/drush_
make.git
which drush
```

The console should answer **/cygdrive/c/ProgramData/Propeople/Drush/drush** or something similar. The correct answer is any answer other than "Not found".

What just happened?

First, we used the Cygwin terminal as a root to set up permissions for our users.

Next, we set up the user's profiles and we ran the Drush Windows installer. The installer put the latest version of Drush in the executable path. After Drush was installed, we installed Drush Make.

Congratulations, you now have a working Drush environment. Let's build a website!

Building a Drupal website with Drush Make

First, let's take a look at the `.make` file for this website and learn a little about what each line is telling Drush Make to do. In the `dpk` directory that you created in a previous step, open the file `dpk.make` in a text editor of your choice. You should see the following code:

```
core = 7.x
api = 2
;core
""projects[drupal][type] = "core"
projects[drupal][download][type] = "git"
projects[drupal][download][tag] = "7.9"
projects[drupal][download][url] = "http://git.drupal.org/project/
drupal.  git"""
```

The first few lines seem easy enough. We're telling the `drush_make` command that we're using Drupal 7.0 core, Drush Make API 2, and then giving `drush_make` a specific method for downloading the Drupal 7.0 core. Notice that, in this instance, we've linked the project to a specific filename and a specific version of Drupal.

The first law of Drupal states, "Thou shalt not hack the core". The second law is similar; it states, "Seriously, dude, don't hack the core. Yes, this means you".

Some time ago, Drupal dropped using "versions" on its GIT code checkins and switched to tagging. Code is checked into the master branch and tagged with the correct release ID, which, at the time of writing, is 7.9. We have adopted that tagging here by adding it to the make file.

Putting a clean download of the build and contrib modules in the make file enforces that convention. If you ever need to patch a contrib module or (may the heavens forbid) patch the core, you can write the patch into your make file, so that the code downloads and then is patched by the patchfile of choice:

```
  ; Contrib projects
projects[admin_menu][subdir] = "contrib"
projects[addressfield][subdir] = "contrib"
projects[backup_migrate][subdir] = "contrib"
projects[contentapi][subdir] = "contrib"
projects[contentapi][version] = "1.0-alpha2"
projects[ckeditor][subdir] = "contrib"
projects[commerce][subdir] = "contrib"
projects[commerce_custom_line_items][subdir] = "contrib"
projects[commerce_custom_line_items][version] = '1.x-dev'
projects[context][subdir] = "contrib"
projects[ctools][subdir] = "contrib"
projects[diff][subdir] = 'contrib'
projects[domain][subdir] = "contrib"
projects[domain_ctools][subdir] = "contrib"
projects[domain_ctools][version] = "1.1"
```

Projects that you download from `drupal.org` are called `contrib` modules. Modules you create for a specific project are called `custom` modules. The best practice is to put these modules in different directories. Each one of these `contrib` modules has a "project name" in the first set of brackets. This project name corresponds to its project name on `drupal.org`. You can enter `http://drupal.org/project/PROJECTNAME` into your web browser for any of these contrib modules and get the page with the latest information about that contributed module.

The `[subdir]` value tells Drush Make, once it has downloaded the archive containing the project, to unpack the project inside the `sites/all/modules/contrib` directory.

The `[version]` value is, of course, the version number. In a `.make` file, it is best to use no version number because make assumes the latest release for the core version you're building. This ensures that the latest modules version is downloaded every time a build is performed. However it sometimes becomes necessary to link a project with a specific modules version. In that instance, module versions can be specified in one of two ways. First, by major release number, which is specified by an unquoted number (for example, `3` or `4`). Module versions can also be specified as a development version and the "extension" can be added, but in this case, the module version must be quoted and the Drupal version must be left out. For example, 3.x-dev in this make file would specify the module version number 7.0-3.x-dev, because the make file has been declared a Drupal 7 core and the module has been declared 3.x-dev. In either case, if you can get away with it, leave the version number out and have Drush Make download the latest stable version.

In between the time this book is written and the time you're actually reading it, many of these modules will be updated several times and have newer (hopefully more stable) versions of their contributed modules. We want each one of these `contrib` modules to have the benefit of the latest bug and security fixes without actually breaking the functionality of our site. So, we specify a primary version and let Drush Make follow the recommendations of the contrib module maintainers on which is the most stable release. At any time, we're free to go into any of these versions and "lock in" a specific number, if we believe downloading any other will break site functionality; but for now, we're going to stick with this type of version numbering in our make file.

But, why is the linking different for core? Core releases can require major database updates and sometimes break compatibility with contrib modules. For the core release, we need to be very purposeful about how and when we upgrade our core modules:

```
; jQuery Cycle
libraries[jquery_cycle][download][type] = "get"
libraries[jquery_cycle][download][url] = "http://www.malsup.com/
jquery/cycle/release/jquery.cycle.zip?v2.86"
libraries[jquery_cycle][directory_name] = "jquery.cycle"
libraries[jquery_cycle][destination] = "libraries"
libraries[json2][download][type] = 'git'
libraries[json2][download][url] = "https://github.com/
douglascrockford/JSON-js"
libraries[json2][directory_name] = "json2"
libraries[json2][destination] = "libraries"
```

Third-party libraries are a part of almost all modern websites. Drush Make allows you to specify which libraries you want to include in the build and how to get them. Because many of these libraries have version-specific features you're using on your site, you're probably going to want to link them to specific versions so that you'll get the features for which your site has been designed.

Ok, so let's run the make file.

Time for action – building a Drupal install from a make file

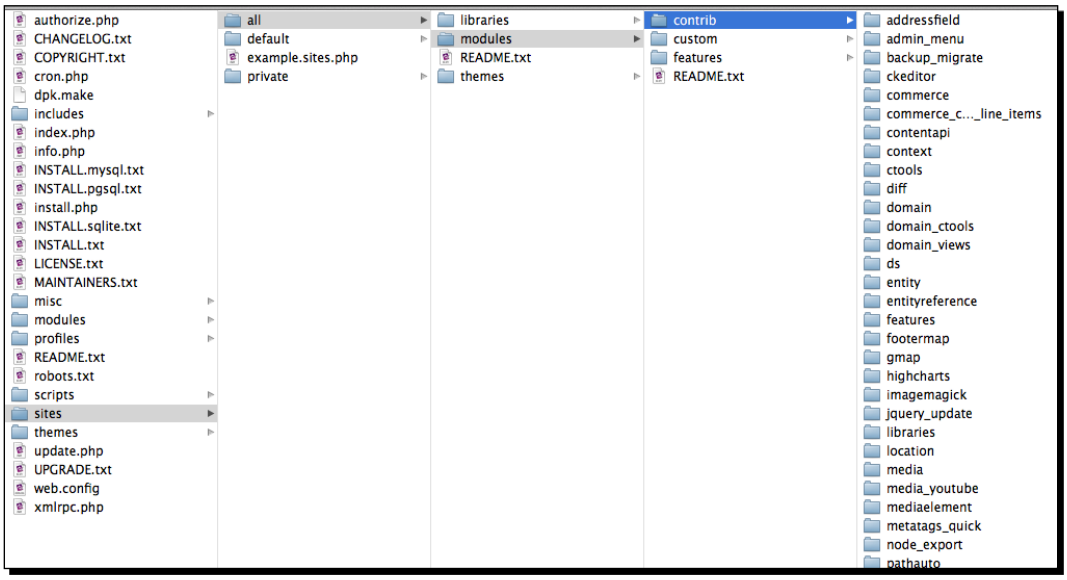
1. Open a terminal window and change the directory to your DPK directory using the `cd` command we created earlier.

2. Then enter `drush make dpk.make`. Drush will ask you if you want to build in the current directory. Enter `y` for yes and then return twice. Your output should be similar to the following screenshot:

```
Last login: Sat May 14 10:47:13 on ttys000
401k:~ stovak$ cd ~/Sites/dpk
401k:dpk stovak$ drush make dpk.make
Make new site in the current directory? (y/n): y
Project information for ctools retrieved.                                    [ok]
Project information for ds retrieved.                                        [ok]
Project information for gmap retrieved.                                      [ok]
Project information for location retrieved.                                  [ok]
Project information for pathauto retrieved.                                  [ok]
Project information for references retrieved.                                [ok]
Project information for token retrieved.                                     [ok]
Project information for views retrieved.                                     [ok]
Project information for features retrieved.                                  [ok]
Project information for backup_migrate retrieved.                            [ok]
drupal downloaded from http://ftp.drupal.org/files/projects/drupal-7.0.tar.gz.  [ok]
ctools downloaded from http://ftp.drupal.org/files/projects/ctools-7.x-1.0-alpha4.tar.gz.  [ok]
ds downloaded from http://ftp.drupal.org/files/projects/ds-7.x-1.1.tar.gz.   [ok]
gmap downloaded from http://ftp.drupal.org/files/projects/gmap-7.x-1.x-dev.tar.gz.  [ok]
location downloaded from http://ftp.drupal.org/files/projects/location-7.x-3.x-dev.tar.gz.  [ok]
pathauto downloaded from http://ftp.drupal.org/files/projects/pathauto-7.x-1.0-beta1.tar.gz.[ok]
references downloaded from                                                   [ok]
http://ftp.drupal.org/files/projects/references-7.x-2.0-beta3.tar.gz.
token downloaded from http://ftp.drupal.org/files/projects/token-7.x-1.0-beta1.tar.gz.  [ok]
views downloaded from http://ftp.drupal.org/files/projects/views-7.x-3.0-beta3.tar.gz.  [ok]
features downloaded from http://ftp.drupal.org/files/projects/features-7.x-1.0-beta2.tar.gz.[ok]
backup_migrate downloaded from                                               [ok]
http://ftp.drupal.org/files/projects/backup_migrate-7.x-2.1.tar.gz.
jquery_cycle downloaded from                                                 [ok]
http://www.malsup.com/jquery/cycle/release/jquery.cycle.zip?v2.86.
json2 cloned from https://github.com/douglascrockford/JSON-js.               [ok]
Make new site in the current directory? (y/n): y
401k:dpk stovak$
```

What just happened?

Drush Make used the `.make` file to go to the `drupal.org` repository and download clean fresh versions of the Drupal core and all the modules listed for use with this site:



In addition, the make file pulled copies of the listed libraries and stored them under `sites/default/libraries`.

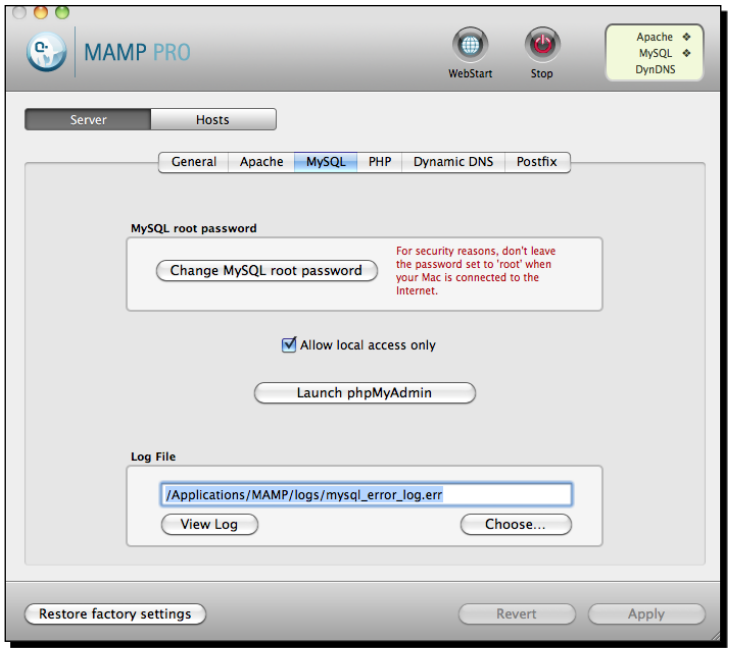
The local database

We need to create a local copy of the website database and alter the Drupal connection settings to properly connect. To do that, we can use phpMyAdmin that was installed with the MAMP/WAMP package, earlier.

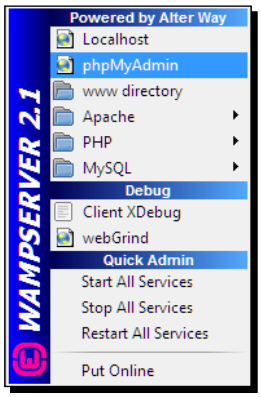
Time for action – creating a database

Drupal stores its data and much of its configuration in a database. In the code you checked out from GitHub, there are database backups for each of the chapters in this book. You can restore the website back to the way it was at any given point in the progression of the site via the databases that are in the **Backup and Migrate** module's database backups.

1. Open the phpMyAdmin page in your default web browser on the Mac OS by launching MAMP Pro. Go to **Server** | **MySQL** and click on the button labeled **Launch phpMyAdmin**, as shown in the following screenshot:



In Windows, choose the WAMP tray icon and click on **phpMyAdmin**, as shown in the following screenshot:

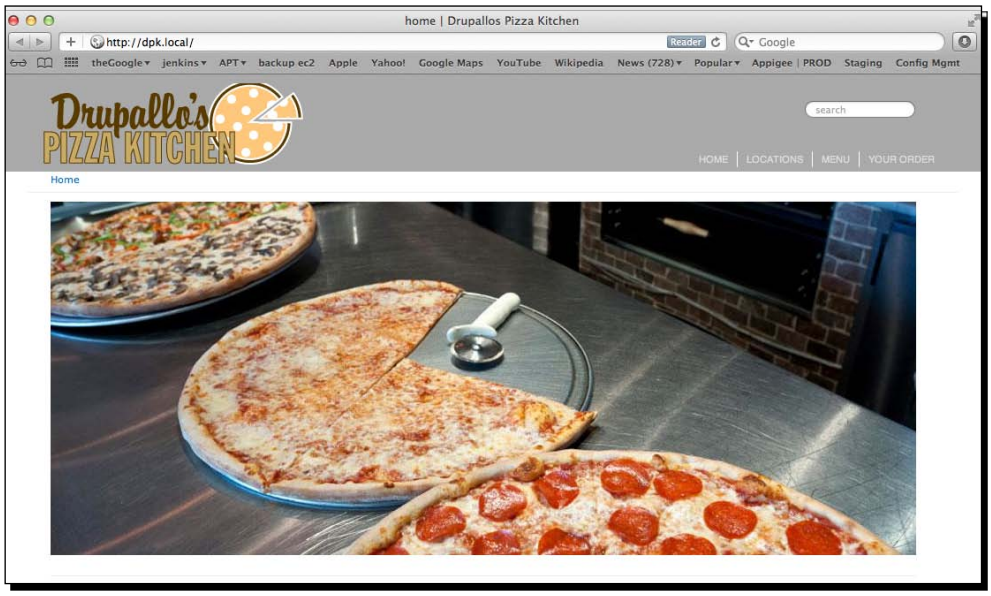


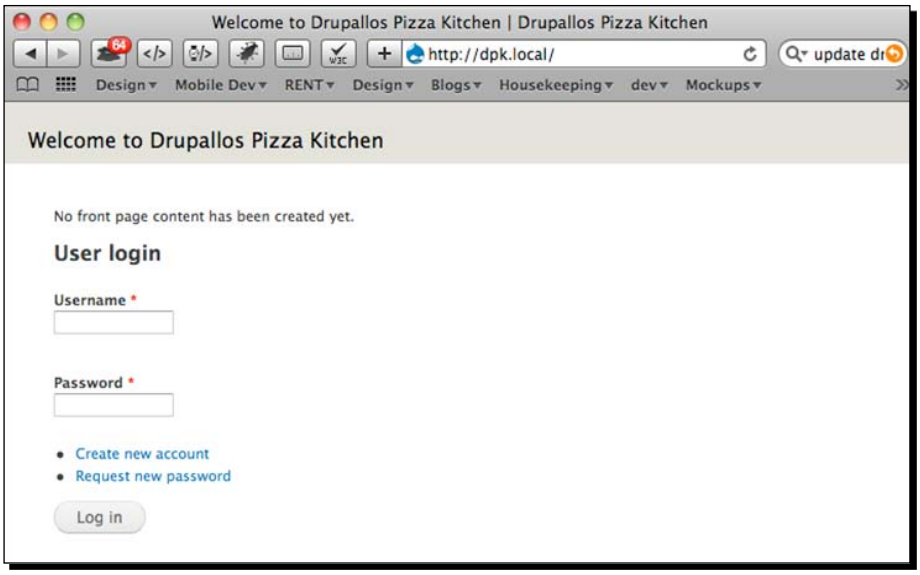
2. Once you have the window, create a new database and call it simply **dpk**. Make sure the encoding is **utf8_unicode_ci**, as shown in the following screenshot:

3. We want to take a database snapshot file that we've downloaded from the repository and do a one-time restore of our site's database. In the window of the new database you've just created, click on the **Import** button. You will see a file dialog box.

4. Choose `sites/private/backup_migrate/manual/SNAPSHOT-CHAPTER2.mysql`. Then click on **Go** and the database will fill up with the file's content.

5. The only thing that's left is to give Drupal a way to connect to the MySQL database. I've provided two setting files that should work with the default WAMP and MAMP installations in `sites/default`. Copy the one that works with your platform to `settings.php` and then go to `http://dpk.local/` and you should be presented with the Drupal login screen, as shown in the following screenshots:

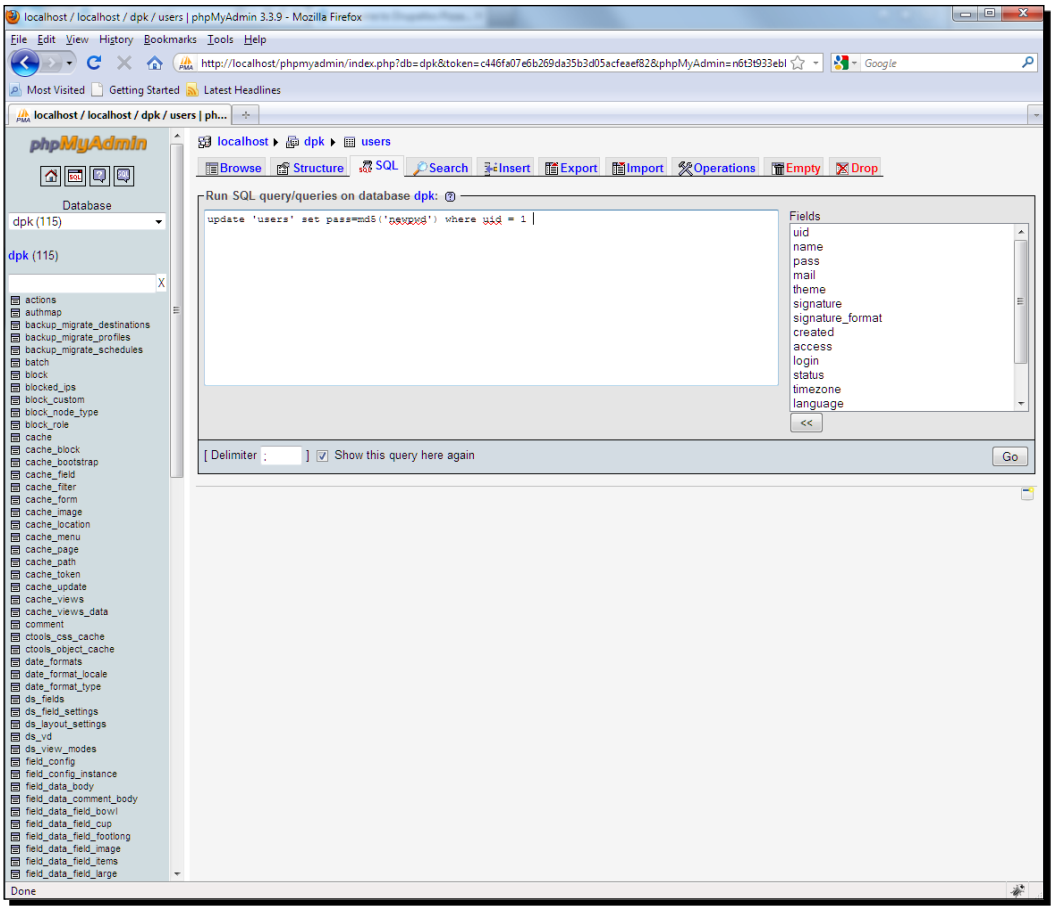




6. One more task to complete and we should be ready to go. We need to change the Drupal user's password to something you'll remember.

7. Open a command line and change the directory to your Drupal site installation. Enter the following command:

```
drush user-password admin --password="NEWPASSWORD"
```



What just happened?

The Drush command changes a user's password quickly and easily.

If you ever forget your admin password, this is a quick and easy way to reset the admin user's password on just about any Drupal database.

Now, try logging in with the username as admin and the new password. You should be able to click **content** and get a list of content for the site.

Now, let's take a quiz to review what we've learned throughout this chapter.

Pop quiz

1. SCM stands for:
    a. Shared Coastal Memories
    b. Standard Code Management
    c. Source Code Management
    d. None of the above
2. Examples of SCM are:
    a. GIT
    b. Subversion
    c. Mercurial
    d. All of the above
3. In the acronyms LAMP, WAMP, and MAMP, the M stands for:
    a. Misty
    b. Mountains
    c. Monitors
    d. MySQL
4. When you make a change to Apache's configuration file, what must you do to see the change in action?
    a. Restart the Apache service
    b. Stop the service completely
    c. Nothing
    d. None of the above
5. Drupal modules included in the base Drupal install are said to be:
    a. Well written code
    b. Part of core
    c. The `contrib` modules
    d. Under the astrological sign of cancer

6. The command-line utility that will allow you to clear the cache in your Drupal site is called:
    a. Rush
    b. Windsong
    c. Grupal
    d. Drush
7. The module for the command line that allows you to build a Drupal install from a single file is called:
    a. `drush_can`
    b. `drush_minister`
    c. `drush_drupal`
    d. `drush_make`
8. Drupal modules created by members of the Drupal community and listed on `drupal.org` are called:
    a. Contrib modules
    b. Under the astrological sign of Capricorn
    c. Of questionable parentage
9. In a make file, you should number module version numbers:
    a. By the days of the week when the module was coded
    b. You should leave out the version number and let `drush_make` download the latest stable release
    c. The Drupal core number plus the maintainer's first child's birthday
    d. None of the above
10. A virtual host:
    a. Is created in Apache's `config` file
    b. Allows many hostnames to be served on a computer
    c. Has an associated port number and host name
    d. All of the above

Summary

In this chapter, we installed the pieces of our development environment and set ourselves up for success for the rest of the chapters of the book. Then, we put the pieces together. We covered the following:

- We installed GIT Source Code Management (SCM).
- We downloaded the source code for our example website from GitHub.
- For Mac, we installed an all-in-one LAMP solution that will allow us to quickly deploy local versions of our websites and serve them on virtual hosts on our own machine.
- For Windows, we installed Cygwin so that our scripting tools will work with Windows' limited shell utilities. We installed a standard WAMP solution to allow PHP-based websites on our Windows machine to develop locally.
- We installed the Drupal scripting layer, Drush.
- We installed Drush's companion module (Drush Make) that allows us to create an entire Drupal installation with one single file.
- We built a Drupal install from our codebase, that is, Drush and Drush Make.
- We installed a new database for our Drupal site.
- We changed the admin user's password using Drush.

We've got our example site installed and we've got our development environment set up. Now, it's time to move on to some buildout!

3
Selecting the Right Domain for your Mobile Site

What does it mean to make a mobile website? For that matter, what does the word "mobile" mean? Increasingly, tablet computers have helped to blur the line between the traditional desktop web browser and handheld devices. In addition to our tablet viewers, we'll need to make sure our desktop viewers still maintain the highest quality of website experience while giving our handheld users the most full-featured website experience that is possible with their device. There are a few different ways to get there.

In this chapter, we will:

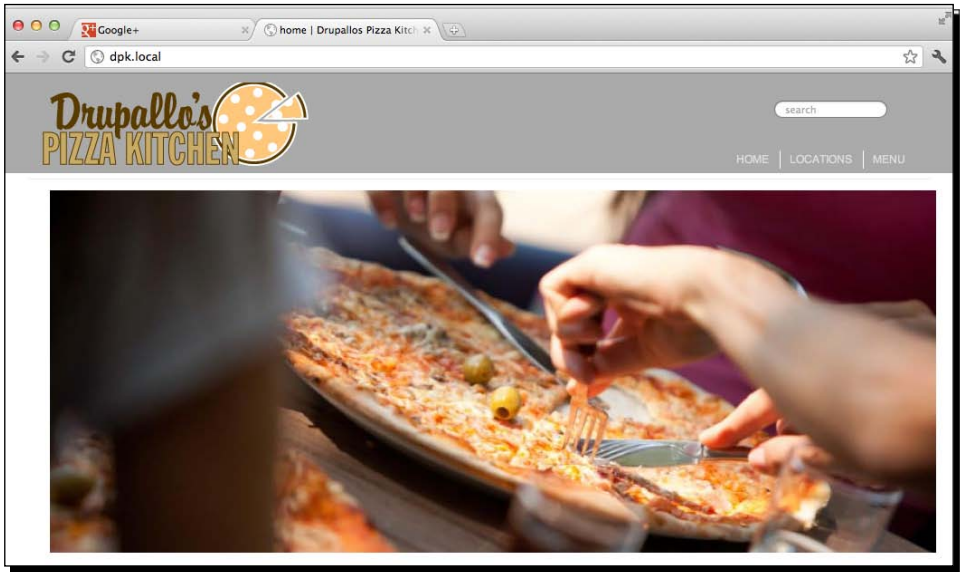
- Learn about the ways in which we can sabotage non-desktop users with outdated desktop design patterns
- Learn about `.mobi` domains
- Learn about modules that allow us to serve the same content on different host domains with different Drupal themes
- Back up our work in the database to a local directory with the **Backup and Migrate** module.
- Learn about the best practices on exporting the settings functionality into a "feature"
- Check our new feature into version management and deploy it to a **User Acceptance Testing** (**UAT**) environment

So let's get on with it.

Once upon a website...

When Little Jimmy started the DPK website, he really didn't know anything about "design" per se, but knew a little bit about setting up Drupal and maintaining the website once it was set up. He asked his friend, Claire Romano, to help him with the design. Claire had a few years of design school and has a small design shop she runs out of her home when not caring for her children. Claire primarily does "print" design—flyers, newspaper ads, and so on. She created a Photoshop `.psd` file of the DPK home page and Little Jimmy did his best at adapting the `.psd` files and creating the DPK theme. But let's take a look at Jimmy's work and view it through the screen of a handheld.

Let's launch our handheld emulator and point the web browser to `http://dpk.local` and take a look at what we see there:



Our handhelds and tablets have a few problems with the design, which we can see more or less immediately:

- First and foremost is the navigation. Jimmy did the animated hovers in Adobe Flash and many handheld devices (especially the Apple devices) just can't display Flash content (More on that in *Chapter 9*, *Putting it Together*). Having Flash navigation completely blocks handheld users who don't have Flash from navigating your website. Luckily, Jimmy put a secondary navigation at the bottom of the page or the website would be completely useless.

- Notice the GIF text. It looks nice, but it's unavailable to mobile clients to read. For a handheld user to dial the number and order a pizza or get a map to the address, they'll have to memorize the phone number, change applications, and dial the number by hand. This could have been avoided by just displaying the phone number and address in text with the proper semantic markup, so the handheld would recognize it as a phone number and address. Most handheld devices have "detectors" for those kinds of information and such information can be made clickable. When a user clicks on a detected address or a phone number, it opens the appropriate application with the data already entered. No need to memorize the number, as the handheld dials it for you. The map application brings up the address automatically.
- The hover state items are revealed only when you "hover" over them. On a touchscreen, there is no equivalent for hovering. These hover state items are unusable on a handheld.
- The **MENU** link at the top navigation links to a 2 MB download of the PDF version of the menu. On a handheld, if my data limit for the month is 200 MB and I click on your menu link, I've used 1 percent of my monthly allotment just to figure out if the restaurant serves Calamari. If it's the end of the month and I'm close to my bandwidth limit, downloading this menu might result in overage charges. If the user is downloading a file, it's common courtesy to let them know what type of file it is and how big the download is. Also, we need a plain HTML alternative to the 2 MB file download.
- One thing that makes sense when you hear it but might not be so obvious is the large "billboard" photo. This is a 100 K+ photo and really doesn't need to be shown on a handheld; or maybe it needs to be shown in a different size and/or aspect ratio. Either way, it's unacceptable to have a 100 K+ image on a home page being viewed by a handheld. Similarly, for the photos that pop up in a lightbox, we'll need to adjust the way they're viewed. Right now, the lightbox is a little less than "handheld friendly".
- In the **ABOUT** page, much of the content is in a lovely font in a GIF/JPEG image. This looks fantastic on a large screen, but the handheld needs to zoom to read small text. In addition, zooming, in this case, doesn't make the small text more legible, because it's in an image format. Also, when zoomed, the text doesn't automatically wrap so that the whole paragraph is legible in the handheld's screen. It requires the user to scroll to read a single line of text. This is less than optimal. Not to mention the fact that the text in the image is completely invisible to search engines.
- Notice the "billboard" photos. The first issue is their aspect ratio. Having photos of this size on the homepage will make everything else on the page so small as to be illegible. Second is the sheer file size. Having multiple 100 K+ photos on the homepage is probably a payload and the average person doesn't want to download that.

Some of these issues can be solved by using simple CSS or markup additions to the theme template (which we'll discuss in *Chapter 4*, *Introduction to a Theme*). Some are going to require something more. But there's a larger philosophical question to answer. We've designed this experience for a desktop user. Is it acceptable to compromise the experience of the desktop for the sake of the handheld or vice versa? I would say "No, it's not acceptable to compromise the design, look, or feel of one use case for another, no matter the validity of either. This is not a universal opinion."

One ring to rule them all

Many developers, particularly the more philosophical of our ilk, would say that you need to design a single markup solution that will work with both mobile handhelds and with the desktop browsers and be able to fit on a screen of any size. And, I guess in a perfect world, we'd polish a design so that it works on a screen of every size, and keep revisiting it as new technology emerges, to ensure that it has an optimal user experience on every new piece of technology. We do not, however, live in a perfect world. In my estimation, that view is a little unrealistic and, again, in my opinion, doesn't lead to the best user experience. It's unrealistic because universal designs take significantly more time to create and debug, and on many projects, the client simply isn't willing to invest that kind of money for something that will pay off minimally. Also, from a UI perspective, the ways in which we consume information on a handheld and on a desktop browser are completely different. Therefore, the user design should be different.

In *Chapter 10*, *Tabula Rasa: Nurturing your Site for Tablets*, we will discuss where and when this strategy breaks down. If you own a tablet computer, you will already have experienced it by being redirected to a mobile site when the screen is more than able to handle the desktop design.

But, in point of fact, the way we consume information on a small-screen handheld is radically different from our desktop computers. Not only do we use the information in different ways, but we need different pieces of information. Each use case offers a specific type of user looking for specific data and it's our job, as the web developer, to get them to that data in the most expeditious manner possible using the best their device has to offer. That's why I believe it is to your benefit to have multiple site designs for multiple purposes, to sniff out the most obvious intent and then to offer links to the other site designs, if the user would prefer using them.

More than that, the search for the one true design that will work on every device is the search for the Holy Grail. It will never be found and will never end. As a developer, I desire closure, if for no other reason than to get paid for the project.

In 2005, the **Internet Committee for Assigned Names and Numbers** (**ICANN**) saw the development of the mobile web as an opportunity and created the `.mobi` top-level domain. Any address that you can get a `.com` domain for, you can now get as a `.mobi`. Whether you create a `.mobi` site or move your mobile site to a subdomain of the primary domain, such as `m.drupallospizzakitchen.com` is up to you. The issues surrounding the secondary domain are the same regardless of the hostname.

When I told Little Jimmy about the `.mobi` top-level domain, he went ahead and reserved `DrupallosPizzaKitchen.mobi`, so we're covered. But right now, the `.mobi` domain is pointed to the current web server and the same site serves on both domain names. We know that we're going to want the same content on both domains, but with two different themes. In addition, there might be some coupons that are only available to mobile users to encourage pizza delivery, so there needs to be some scheme of telling Drupal what to do with requests coming in for the `.mobi` domain versus requests for the `.com` address. On our local version, we will use the `m.<website>` URL pattern to signify the local version. You're free to use either or both on your live site. But how do you get content on your new URL? The answer is by entering the Domain Access module.

Domain Access versus Multisite

Drupal already has, built into the core, a mechanism for hosting multiple sites out of a single Drupal installation. This is commonly called **multisite installs**. You can create folders inside your `Sites` folder with the hostnames and each folder can have its own custom modules and `settings.php` file.

What we want to do here is a bit different than simply hosting two sites from a single core and module install. We want the two sites to share everything with the possible exception of themes. What domain access does is that it gives you options for content and theme segregation. The Drupal website project is actually called the Domain module, but it's commonly referred to as **Domain Access module**. We will use the names interchangeably.

Let's install the **Domain Access** module and take a look at it.

Time for action – installing Domain Access module

In Mac OS X, go to **Applications** | **Utilities** | **Terminal.app**. In Windows, go to **Start** | **All Programs** | **Cygwin Bash Shell**. Enter the following commands:

```
cd ~/Sites/dpk
drush dl domain domain_views domain_ctools
drush pm-enable domain domain_views
```

What just happened?

The first command moved us into the directory where the Drupal installation resides. The second command downloaded the **Domain** module from the Drupal website. The third command enabled the module. We did all of this from the command line, but you could have just as easily downloaded the module by hand, dropped it into `sites/all/modules/contrib`, and then used the Drupal admin screens to enable the module. But typing these three lines seems easier to me. Do it in whichever way you feel most comfortable.

We're installing two modules here; these are the Domain and Domain Views. Domain allows Drupal to segregate content based on the URL and Domain Views adds the fields to the view for adding and excluding that content in a view. Domain CTools allows domain settings to be exported.

Domain management

Now, the first part of domain management is to get the domain registered and pointed to the server in question. We're going to do this locally and we'll go over how to do it on the server when we roll out our changes to the live site.

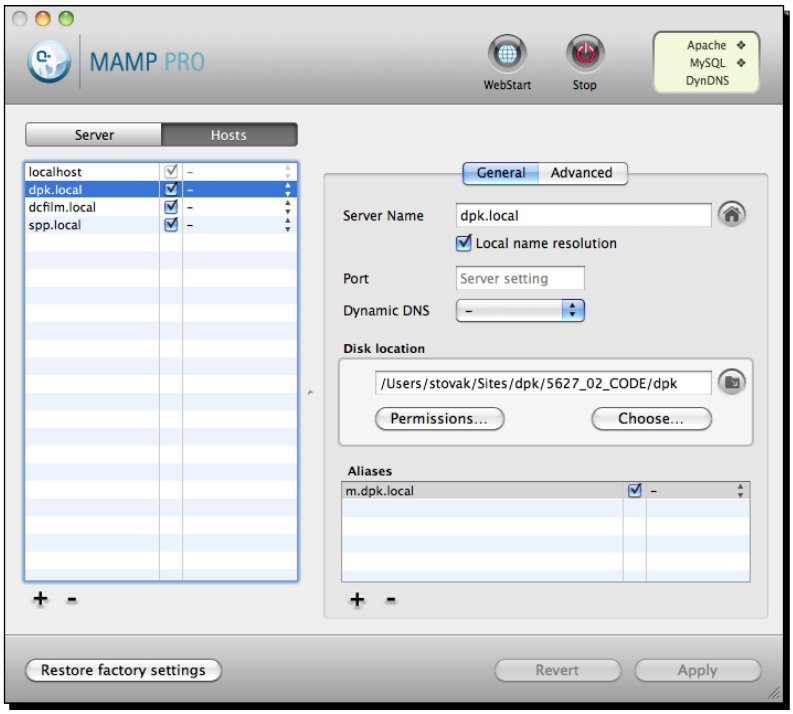
Server Alias is an Apache directive for virtual hosts. It allows Apache to serve multiple domains on a single virtual host. As its name suggests, server alias adds a second hostname to a named virtual host.

Time for action – configuring Apache

First, we need to establish the second URL locally, which means configuring Apache to respond and then configuring the development host to resolve.

For Mac, follow these steps:

1. Launch MAMP Pro.
2. Under the entry for `DPK.LOCAL`, add the site alias as `m.dpk.local`.
3. Click on the **Apply** button:



MAMP will warn you that it needs to restart Apache and MySQL. It will then do so. You may need to put in your administrator's username and password to approve the restarting of services.

For Windows, follow these steps:

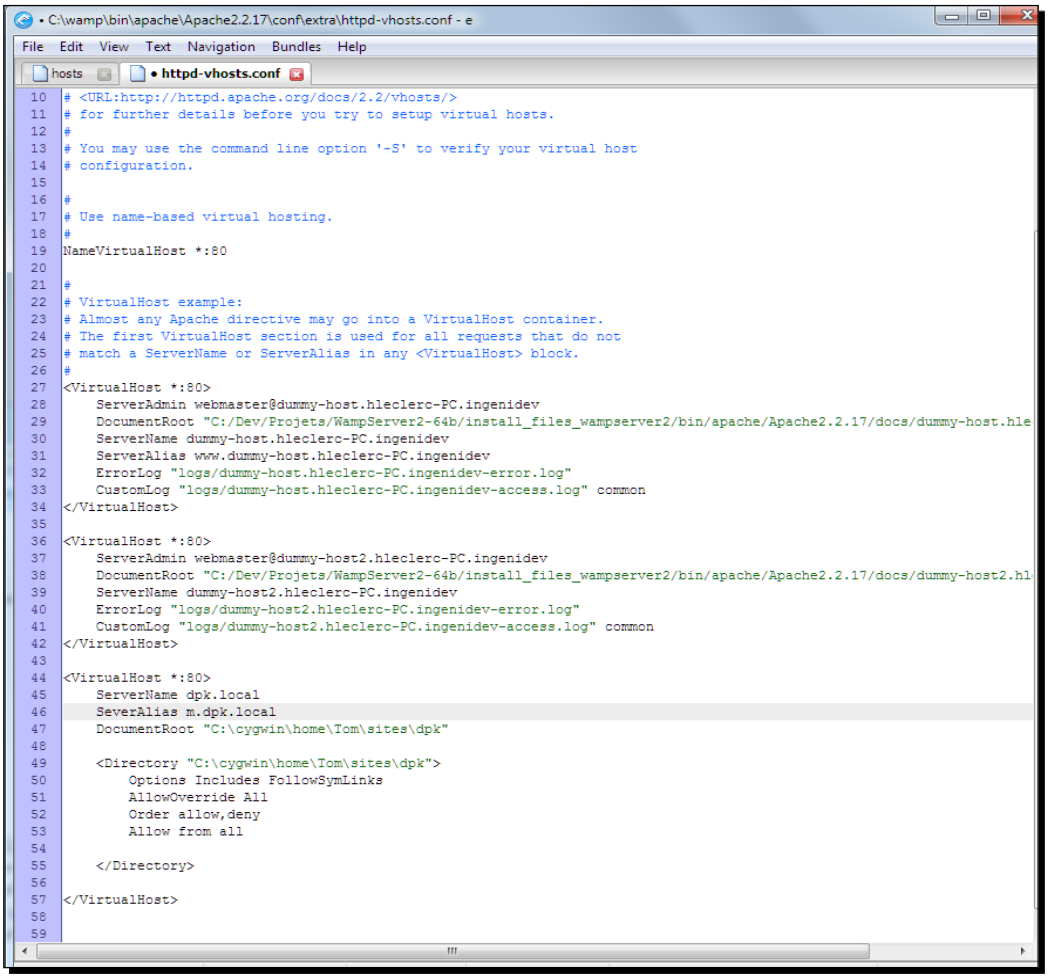
1. Edit the Apache's `vhosts.conf` file, adding the highlighted line in the following code to the `dpk.local` virtual host definition:

```
ServerAlias m.dpk.local

<VirtualHost *:80>
  ServerName dpk.local
  ServerAlias m.dpk.local

  DocumentRoot "/Users/stovak/Sites/dpk"

  <Directory "/Users/stovak/Sites/dpk">
    Options Indexes Includes FollowSymLinks
    AllowOverride All
    Order allow,deny
        Allow from all
```

```
    </Directory>

</VirtualHost>
```

2. Right-click on the **E editor** icon in the programs menu, select **Run as** and then run as administrator.

3. As shown in the following screenshot, add `127.0.0.1 dpk.local` to `C:\windows\system32\drivers\etc\hosts`:



4. Restart Apache using the context menu in the tray.

Now that Apache is configured, open the browser and go to `http://m.dpk.local`. You should be presented with the same site as the one on the primary URL.

What just happened?

We need to make Apache respond to the URL, `m.dpk.local`. Doing that requires two different parts. The local name resolution, which takes place in the local `hosts` file, and the Apache configuration. Adding a "server alias" to Apache makes it reuse the virtual host for a second URL. On the Mac OS, the configuration application takes care of the heavy lifting behind the scenes. On Windows, we did these two tasks by hand.

Bootstrapping the domain

In order to function correctly, the Domain Access module needs a bootstrap in the Drupal installation's `settings.php` file. Let's add that.

Bootstrapping is a process by which the base code is loaded for a specific environment. The bootstrap for an OS loads a basic amount of code, so that it is able to function, but not the entire load of the OS. With Drupal, there's a file called `bootstrap.php` in the site's root. When included in your script, everything that Drupal needs to execute the functions will be loaded by the script. Many simple backend scripting functions of Drush are executed by simply loading a bootstrap from your installation and executing Drupal functions from there.

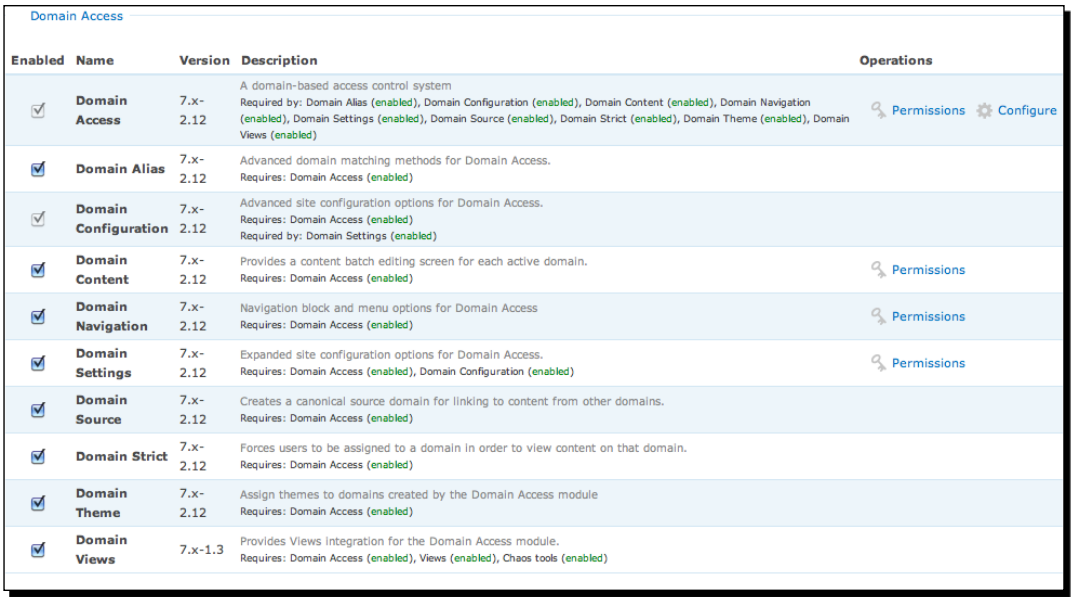
Time for action – bootstrapping the Domain Access module

1. Edit the `sites/default/settings.php` file, adding the following line. You may have to edit this file as an administrator. TextMate on the Mac may ask you for a Username and Password of an administrator for the machine. On the PC, right-click on the **E editor** icon and choose to run as administrator, as you did in the *Time for action – installing the Domain Access module* section and enter the following line:

```
include DRUPAL_ROOT . '/sites/all/modules/contrib/domain/settings.inc';
```

2. First, go to the development URL, `http://dpk.local`. If you're not logged in as the administrator, log in now.

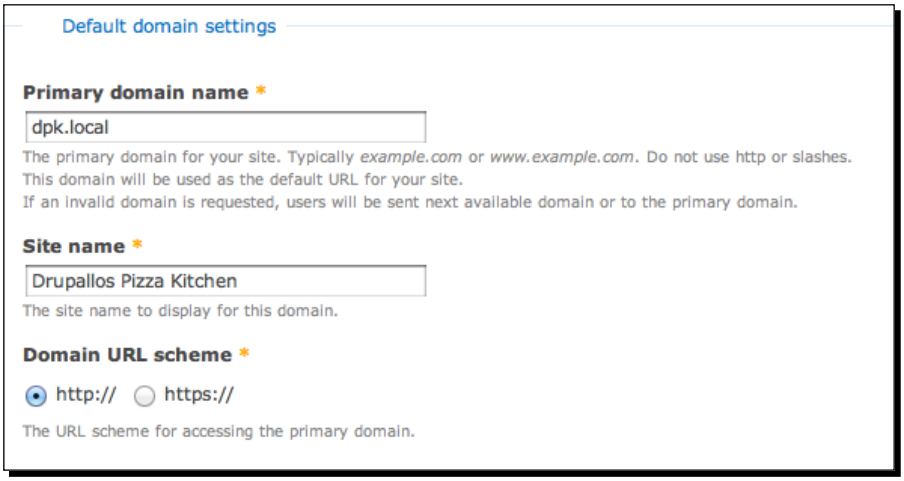
3. First, we need to enable all the modules in the Domain Access suite. Go to **Modules** and find the **Domain Access** section. Enable them all for now. We'll turn off the ones we don't need when we make the changes to the site live:



Domain Access

Enabled	Name	Version	Description	Operations
☑	**Domain Access**	7.x-2.12	A domain-based access control system Required by: Domain Alias (enabled), Domain Configuration (enabled), Domain Content (enabled), Domain Navigation (enabled), Domain Settings (enabled), Domain Source (enabled), Domain Strict (enabled), Domain Theme (enabled), Domain Views (enabled)	Permissions Configure
☑	**Domain Alias**	7.x-2.12	Advanced domain matching methods for Domain Access. Requires: Domain Access (enabled)	
☑	**Domain Configuration**	7.x-2.12	Advanced site configuration options for Domain Access. Requires: Domain Access (enabled) Required by: Domain Settings (enabled)	
☑	**Domain Content**	7.x-2.12	Provides a content batch editing screen for each active domain. Requires: Domain Access (enabled)	Permissions
☑	**Domain Navigation**	7.x-2.12	Navigation block and menu options for Domain Access Requires: Domain Access (enabled)	Permissions
☑	**Domain Settings**	7.x-2.12	Expanded site configuration options for Domain Access. Requires: Domain Access (enabled), Domain Configuration (enabled)	Permissions
☑	**Domain Source**	7.x-2.12	Creates a canonical source domain for linking to content from other domains. Requires: Domain Access (enabled)	
☑	**Domain Strict**	7.x-2.12	Forces users to be assigned to a domain in order to view content on that domain. Requires: Domain Access (enabled)	
☑	**Domain Theme**	7.x-2.12	Assign themes to domains created by the Domain Access module Requires: Domain Access (enabled)	
☑	**Domain Views**	7.x-1.3	Provides Views integration for the Domain Access module. Requires: Domain Access (enabled), Views (enabled), Chaos tools (enabled)	

4. On the line with the primary **Domain Access** module, there are two links, namely, **Permissions** and **Configure**. Click on the **Configure** link.



Default domain settings

Primary domain name *

dpk.local

The primary domain for your site. Typically *example.com* or *www.example.com*. Do not use http or slashes.
This domain will be used as the default URL for your site.
If an invalid domain is requested, users will be sent next available domain or to the primary domain.

Site name *

Drupallos Pizza Kitchen

The site name to display for this domain.

Domain URL scheme *

◉ http:// ○ https://

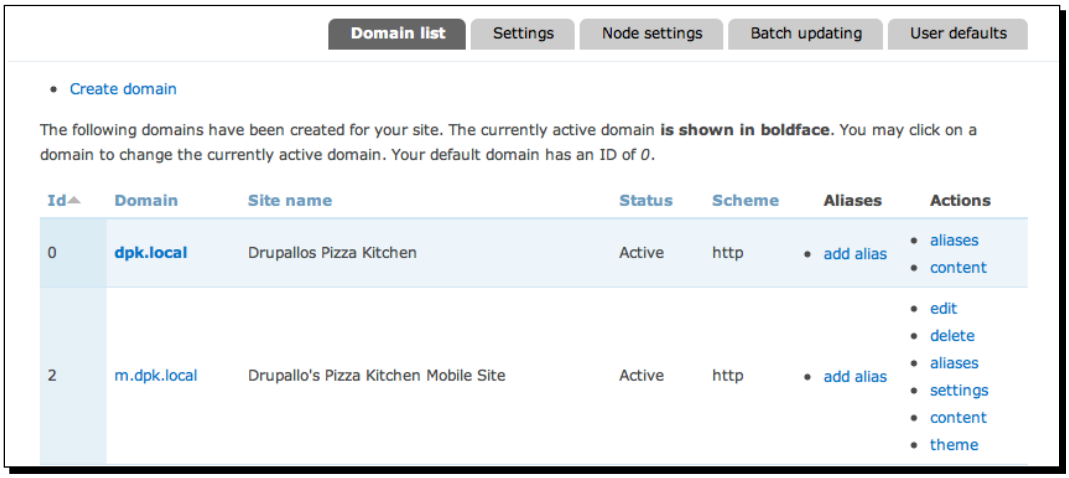
The URL scheme for accessing the primary domain.

5. Make sure the primary domain value is `dpk.local` and not the `m.dpk.local` URL.

6. Save these settings and then click on **Domain List**. It should list your primary domain that you just saved.

0	**dpk.local**	Drupallos Pizza Kitchen	Active	http	• add alias	• aliases • content

7. We need to add the mobile domain. Click on **Create Domain**.



Domain list | Settings | Node settings | Batch updating | User defaults

• Create domain

The following domains have been created for your site. The currently active domain **is shown in boldface**. You may click on a domain to change the currently active domain. Your default domain has an ID of *0*.

Id	Domain	Site name	Status	Scheme	Aliases	Actions
0	**dpk.local**	Drupallos Pizza Kitchen	Active	http	• add alias	• aliases • content
2	m.dpk.local	Drupallo's Pizza Kitchen Mobile Site	Active	http	• add alias	• edit • delete • aliases • settings • content • theme

8. Under the **Domain** column, enter **m.dpk.local**. Under the **Site name** column, put **Drupallo's Pizza Kitchen Mobile Site**, or some such descriptive text. Save the new domain.

9. Now, we have two domains. Add the local version, as we did before, and your domain list should look like the one shown in the preceding screenshot.

What just happened?

We added both the live site and the development local version to the **Domain list** as active domains so that Drupal could distinguish traffic on each domain and treat it accordingly. In the next chapter, we'll assign a theme to the mobile domain and begin the process of dividing content.

Until then, let's take a look at some of the other obvious issues with the home page, starting with the address link.

For now, we need to do one more thing. Package up this site and get it ready to move up to the live site. For that, we're going to need a relatively new module called **Features**.

Introduction to the Features module

Features is a relatively new concept in the Drupal universe. The idea behind the **Features** module is that, for Drupal websites, it's very difficult to version manage the settings in the database without doing a full database backup and then restore. So what we'd prefer to do is just export the stuff we're checking into Version Management and create a standalone module to install those new settings into any Drupal site. **Strongarm** is the module that allows features to talk to system variables. Let's install and enable them both.

Time for action – installing and creating your first feature

The **Features** module bundles functionality into an easily installable package. Let's create one.

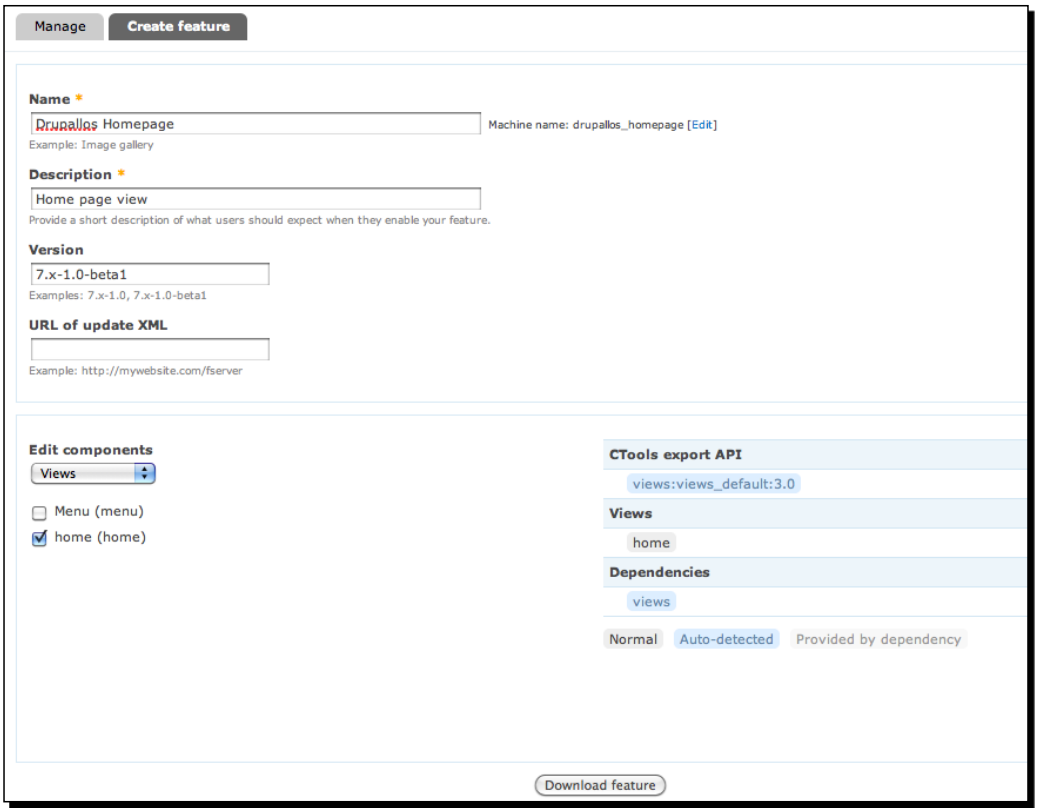
1. In Mac OS X, go to **Applications** | **Utilities** | **Terminal.app**. In Windows, go to **Start** | **All Programs** | **Cygwin Command Shell**. Enter the following commands:

   ```
   cd ~/Sites/dpk
   drush dl features strongarm
   drush pm-enable -y features strongarm
   ```

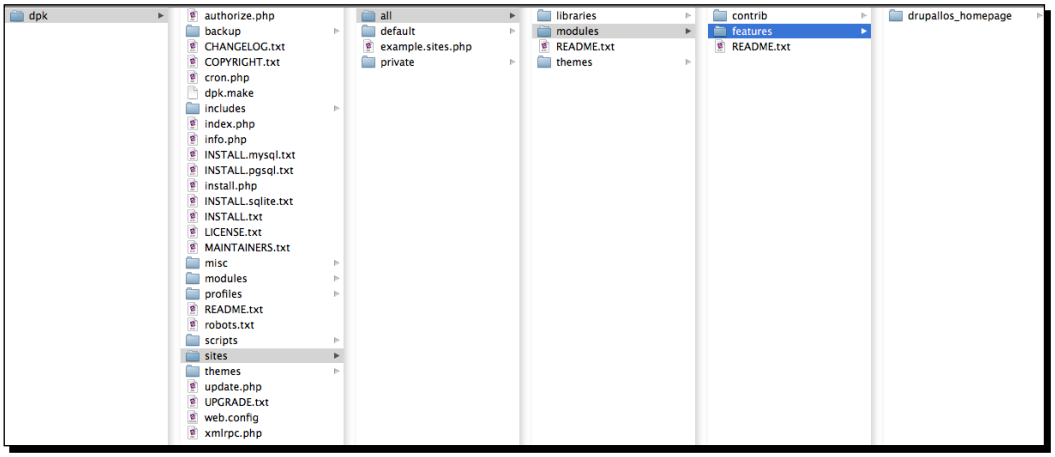
2. Go to **Structure** | **Features** | **Create feature.**

3. As shown in the screenshot that follows, let's name this feature **Drupallos Homepage** and give it a description as **Home page view**. The version number needs to be **7.x-1.0-beta1** for the reasons we discussed in *Chapter 2*, *Setting up a Local Development Environment*.

4. Now, under **Edit components**, choose **Views**. Underneath will be a listing of all of the views that are currently part of the site; select **home (home)**. Click on the **Download feature** button and the feature will be downloaded as a tarball to your hard drive.

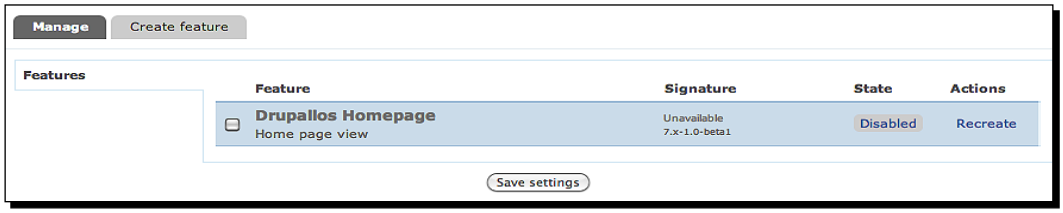


5. In your local install, if the folder `sites/all/modules/features` does not exist, create it and unpack the contents of the new feature module into the newly created folder:



6. Open a browser window and go to `http://dpk.local`. Choose **Structure** then **Features**. You should see the new feature in the **Features** module list (see the screenshot that follows).

7. Enable the feature by checking its checkbox and then clicking on the **Save settings** button:



What just happened?

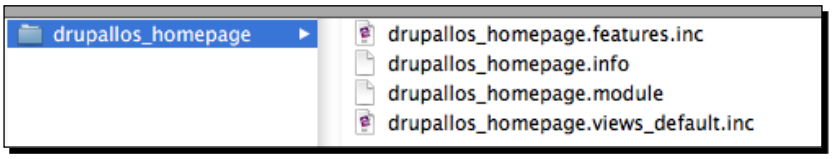
Jimmy originally created the home page view to return nodes of the "billboard" content type, and using the jQuery Cycle Plugin, cycle through the billboard images. We want to make sure that any changes we make to this view:

- Get version managed so we can revert to it if the client doesn't like the change
- Get pushed from our local environment to the staging environment and then on to the production server, so we don't have to make the changes by hand on every server

The **Features** module was created to allow us to do just this.

In the first command, we used Drush to download and install the new **Features** module.

Next, we used the **Features** module to create a feature for our home page view. The following screenshot allows us to take a look at the files in that module:



For those of you who have ever looked at a Drupal module's structure, the `.info` and `.module` files should be familiar to you. The `.module` file contains a single line of code and it's as follows:

```
include_once('drupallos_domains.features.inc');
```

The `.info` file contains some things that might be familiar and some things that might not. Let's go through it line-by-line. The first line of code is as follows:

```
core = "7.x"
```

This line tells Drupal that it works with the 7.x version of the core. The next line of the code tells Drupal that it has a dependency on the views module:

```
dependencies[] = "views"
```

The next line of code is the description field we entered when we created the module:

```
description = "Home page view"
```

The next line of code is a message to `ctools` to load and use the Strongarm module to process the variables below:

```
features[ctools][] = "views:views_default:3.0"
```

The next line of code is a listing of the view we're managing. If there were multiple views in this feature, there would be multiple values here:

```
features[views_view][] = "home"
```

The next line of code is the name variable we set when creating the feature:

```
'name = "Drupallos Homepage"
```

The next line puts the features under the `Features` package. When you have a larger site and start to have 10 to 15 features, it's helpful to put them into groups for better management.

```
package = "Features"
```

The next line of code specifies a machine-readable version of the name we entered when we created the feature.

```
project = "drupallos_homepage"
```

The last line of code shows the `version` we set when we created the feature.

```
version = "7.x-1.0-beta1"
```

Features has created for us two more include files, `features.inc` and `views_default.inc`. The `features.inc` file includes a single function:

```
function drupallos_homepage_views_api() {
  list($module, $api) = func_get_args();
  if ($module == "views" && $api == "views_default") {
    return array("version" | 3.0);
  }
}
```

This function tells views how to use and which version of the views plugin to use.

The `.views_default.inc` file contains the meat and potatoes of our feature. If you take a look at the file, there's only one function that begins and ends as such:

```
function drupallos_homepage_views_default_views() {
  $export = array();
…
  return $export;
}
```

This code takes all of the settings in our local home view and puts them in code so that when the feature is installed on another site, the other site will have an exact copy of our view. Each line of the views feature code sets a value for our view that can be edited via the Views GUI. If you've built a view from scratch and you take a look at the settings, the settings should look familiar to you.

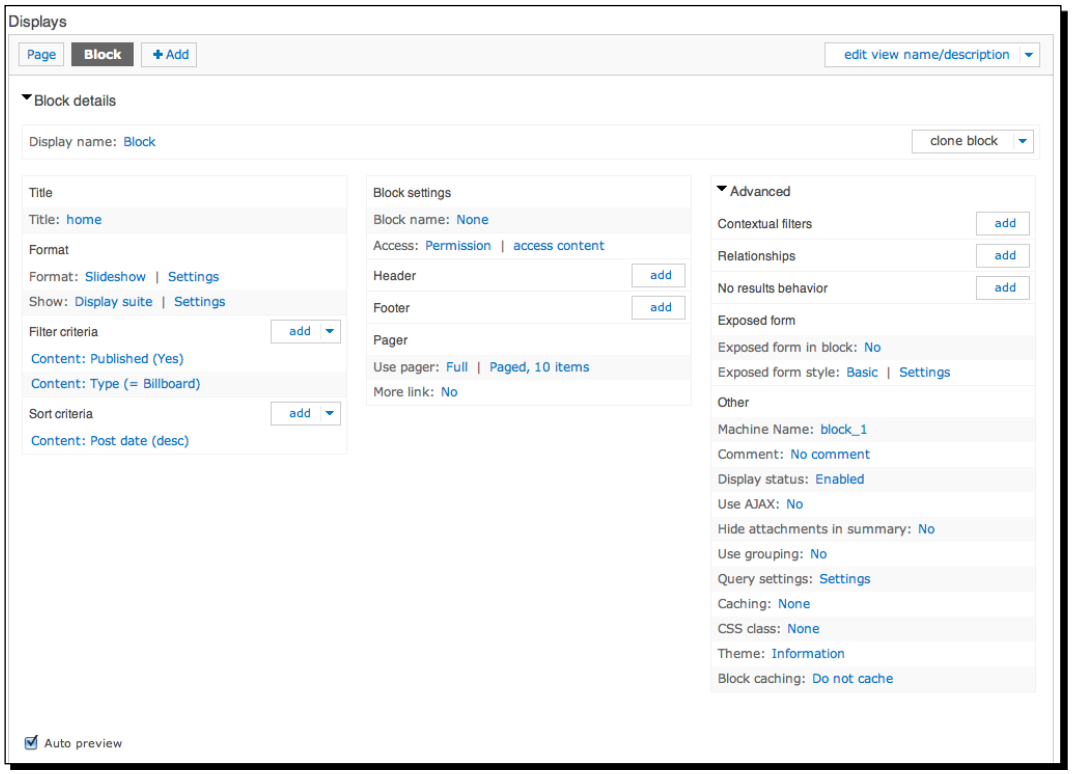
Also notice that, for each of our view displays, there's a corresponding series of lines that begin with:

```
$handler = $view->new_display('page', 'Page', 'page');
```

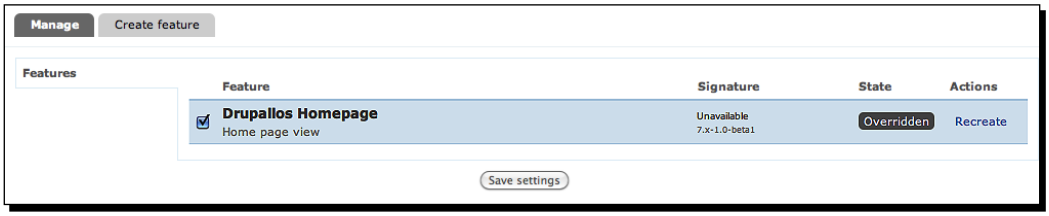
Time for action – updating the feature with new settings

Let's try updating the feature:

1. Go to **Structure** | **Views** | **Home**.
2. Add a block by going to **Displays** | **Add** | **Block**.
3. Save the view with the **Save** button at the top right. All of the default values are fine.

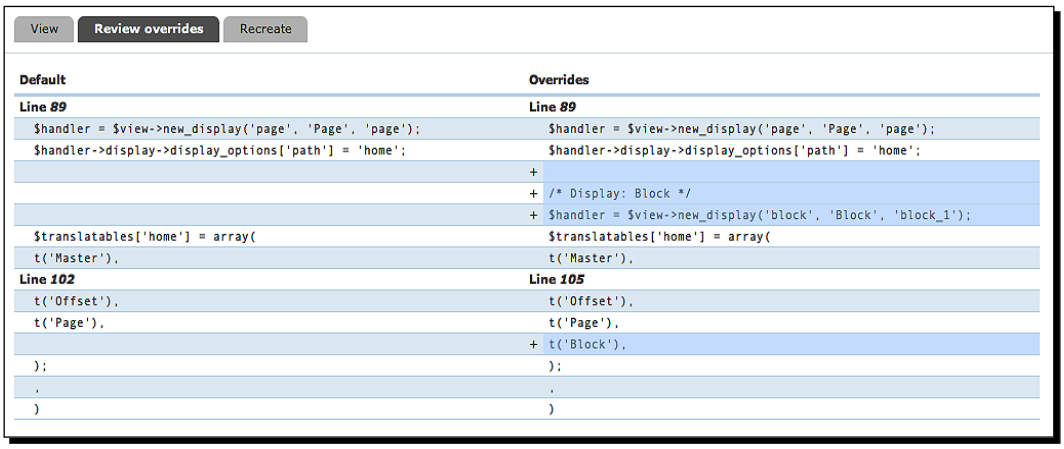


4. Go back to **Structure** | **Features**. You will notice that the **State** column value is no longer **Default** but **Overridden** (see the following screenshot). Click on the **Overridden** state:



5. The features information shows you that the views in this feature have been overridden. Click on **Overridden** again to see the differences.

6. A list of differences shows the new "handler" block we just added:



7. Now, open a terminal and change the directory to your site directory (`~/Sites/dpk`)

8. Enter the following command:

```
drush features-update drupallos_homepage
```

9. Drush will ask you if you want to update the feature. It will let you know that a feature with the name **drupallos_homepage** exists and will ask: **Would you like to overwrite it?**. Enter `y` for yes.

10. Hop back over to our browser, refresh the page, and take another look at our view. You'll see that the state is now **Default** instead of **Overridden**.

11. Taking a look at the `.views_default.inc` file in the module reveals the new block handler we created when editing the view.

12. You may install this module on another website (or our live site). This will prevent the view from being deleted and also will add our new block view we just created without any admin changes to the view itself.

What just happened?

How many times have you deleted a view accidentally, only to have to spend 45 minutes recreating it? How many times have you made changes to a view and needed those changes either on another site or on another environment in the project workflow? Features solves a myriad of development problems with its fantastic Dark Magic. When a change is made to the site's structure you use Drush to update the feature and it takes the change you just made on the site and moves it to your code. Once the feature is in your code, the view can

never be deleted. You can easily add the view to another website by copying the feature to another `sites/all/modules/features` folder.

Features-based App Stores:

With the advent of App Stores such as the Apple App Store and Android Market, an idea being floated around the Drupal community is one of an "App Store" for Drupal-based bundled features that you add to your site via downloading Drush. Notice that all the features have a value called **XML Update URL**, which could easily be linked to a remote App Store that provides updates to the features on a regular basis. One can only hope that this strategy is successful because it may be the tipping point for Drupal to truly be the default choice for much of the content-managed sites in the world.

The missing piece of our development puzzle is deployment. How do we get the code we're working on, onto the live site in the easiest way, at the same time, managing the risk of putting new code out onto the Internet?

Have a go hero – adding hooks to the feature

If you've ever created a customized module, you know that you interact with the Drupal core by using a series of *hooks*, which are functions named in a correct pattern to be executed at the occurrence of a Drupal event.

Because features are, themselves, custom modules, you can add a hook for your content type. For this exercise, add a hook to your new Drupal feature module that will check values when a node is saved.

Deployment—best practices

Little Jimmy had a conversation with Big Daddy Jimmy about us helping out with the mobile work and they both agreed that we should probably go through a testing and acceptance phase, where they take a look at the development we've done and test the site before it goes live. Big Jimmy has enlisted his wife Adrianna and a few of Adrianna's friends to look over the site for mistakes as well as our designer, Claire Romano, who will probably have some feedback about the way the design is implemented.

We completely and wholeheartedly agree, so we're going to set up a UAT server so that the Jimmys and the entire team can take a look at the changes and sign off on them before they go public.

For the purposes of this example, we're going to simulate the UAT environment locally with new URLs on our local install, but you could just as easily add a remote host and create your own UAT on your remote host. Or better yet, as we've done with this book's code repository, host the code in the cloud on GitHub or a similar service (such as BeanStalk). Not only will your code be protected from hardware failure, but also you'll have a common place for access for all developers for the project.

If you remember correctly, we checked this code out of the DPK GitHub project. We'll need to specifically not push our code out there because that code is offered by Packt to buyers of the book. We want this new code just in our UAT environment until it's tested and ready to go. Best testing practices say the code is assumed to be bad until tested. Do not assume any change is good until it's completely tested and signed off, either by the client or the project manager in charge.

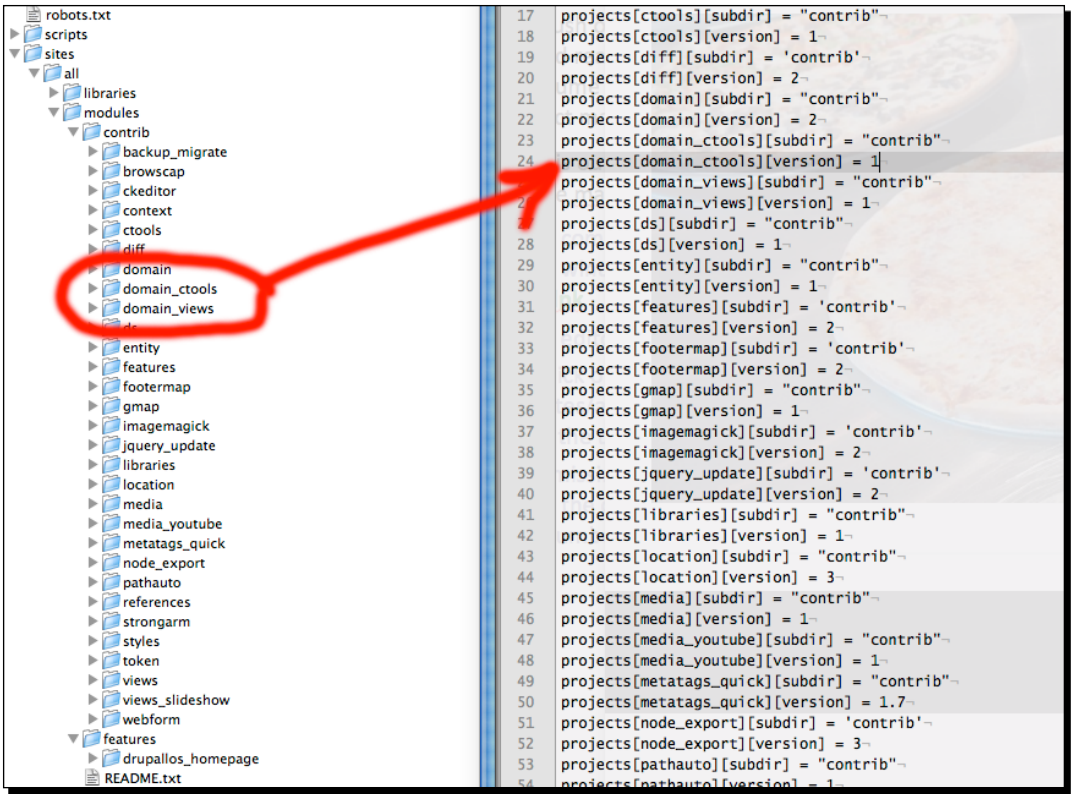
Let me say one more word about testing by telling you what it is not. Testing is not a critique of the design (for example, one may say, "I like blue; I don't like green"). At this point in the project, you should be well past the design stage and the design should be established and set. Testing is making sure the page works in every available browser in as many versions as is in scope and that the design implementation is as close to the design as the browser allows. You should at least test in the latest version of Internet Explorer plus one version older, latest version of Firefox, and either Chrome or Safari. As we go farther in the mobile device themes, you'll want to recruit your friends' various cell phones and carriers to help you test on a variety of mobile devices, including the iPhone and iPad simulator you installed in *Chapter 1*, *When is a Phone Not a Phone?*

Now let's check in our code.

Time for action – code check-in and deployment

Our make file is out-of-date and so is our latest database backup. Throughout this chapter, we've added a few modules. Now, we need to update the make file to reflect the changes:

1. In the command prompt on Mac, enter `mate ~/Sites/dpk` (or whatever site root you're using). From the Windows Cygwin command line, substitute `e` for `mate`: `e ~/Sites/dpk`.
2. Your text editor will open up with a project view of the entire folders of the site.
3. Double-click on the `dpk.make` file in the root directory. As shown in the following screenshot, reveal the contents of under `sites/all/modules/contrib` and line them up side-by-side so you can compare the two lists. This is a good way to make sure your `.make` file includes all the `contrib` modules you've recently installed:



4. Add any missing modules. You can get the module's base version number by revealing the contents of the module folder and double-clicking on the `.info` file. The version number will be listed in the `.info` file. Add the following lines:

```
projects[domain][subdir] = "contrib"
projects[domain][version] = 2
projects[domain_ctools][subdir] = "contrib"
projects[domain_ctools][version] = 1
projects[domain_views][subdir] = "contrib"
projects[domain_views][version] = 1
```

5. Open a terminal window and change the directory to your site root (`~/Sites/dpk`).

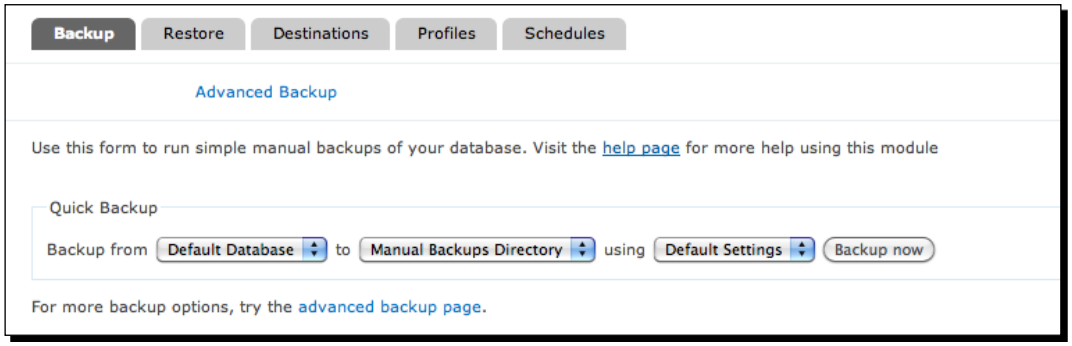
6. Enter the following commands and keep the terminal window open:

```
> git add dpk.make
> git commit -m "adding missing modules"
```

7. Now we create the UAT instance. You can do this locally or on a remote server. Open a new terminal window. In the separate terminal window, enter the following commands:

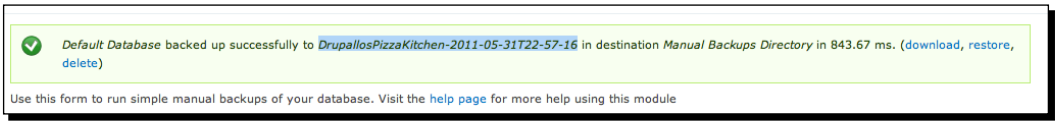
```
> cd ~/Sites
> git clone ~/Sites/dpk ~/Sites/uat.drupallospizzakitchen.com
```

8. In your web browser, go to the development URL, `http://dpk.local`. If you're still not logged in, log in as the administrator. Choose **Configuration | Backup and Migrate** from the admin menu.



9. Under the **Backup** tab, choose **Quick Backup** of the **Default Database** to the **Manual Backups Directory** using **Default Settings**. Click on the **Backup Now** button.

10. On the resulting screen will be the new file created by the backup. Highlight the name with the mouse, right-click on it, and copy the name to the clipboard.



11. Open a terminal window and change the directory to your site root (`~/Sites/dpk`).

12. Enter the following commands. You can paste the filename you just copied to the end of the first line:

```
> git add sites/private/backup_migrate/manual/ [paste the filename you just copied]
> git commit -m 'current database backup'
```

What just happened?

GIT is a distributed version management system. What that means is that there's no one "big daddy" repository for any code set. Every repository has the ability to be both a pusher and a puller from other clones. We checked out code into the development repository. If this were a real project, we'd have hosted the project on GitHub and then pushed our updated code out to the cloud GitHub host.

However, what you really don't want to do is to deploy from a make file. Because between the time you check code in and the time you build the code there may be changes introduced that are not accounted for. That's why it's my polite suggestion that deployment environments use the `rsync` command to copy a deployment from a tested lower environment. That way no code enters the production stream that hasn't been thoroughly tested and your UAT and Production environments are an exact copy of the development snapshot.

We then used the **Backup and Migrate** module to create a local backup of the Drupal database. We added that database to the version management payload and committed it to the local repository. If UAT and Production environments have no unique data, you can do a database update during deployment. Most of the time, your production database should not be updated and you'll need to add any deployed nodes to your feature.

Pushing out features

Now, let's add that new `features` module and push it up to UAT.

Time for action – check in your features module

Open a terminal window, go to your development site root (`~/Sites/dpk`), and enter the following lines:

```
> git add sites/all/modules/features/drupallos_homepage/*
> git commit -m 'adding homepage view module'
> cd ~/Sites/uat.drupallospizzakitchen.com
> git pull
```

What just happened?

The first line of code added the next `drupallos_homepage` module to the local development version of the repository. The second line committed the changes to the repository locally. The third line changed us over to the "remote" version of the repository. The fourth line pulled all the changes we just checked in to the UAT repository.

You could repeat these commands on a remote server and have a working make file and codebase to set up a UAT server. We won't go into detail with the steps involved in setting up the UAT server because it's exactly the same as setting up our local development, but here they are:

1. Run the `drush make dpk.make` command on the `.make` file, as you did in *Chapter 1*, *When is a Phone Not a Phone?*
2. Create a new database and put the connection settings in `settings.php`.
3. Restore the latest database backup to the new empty database.
4. Add the new UAT hostnames to the domain access "alias" list for each of the primary domains.

Pop quiz

1. Which module segregates content based on the hostname?
    a. Views
    b. CCK
    c. Fields API
    d. Domain Access
    e. None of the Above
2. For what daemon or service is `ServerAlias` a directive?
    a. Apache
    b. Linux
    c. MySQL
    d. PHP
3. Bootstrapping mainly concerns itself with:
    a. Starting up an environment's basic functions
    b. Killing all processes on a running virtual host
    c. Creating virtual hosts
    d. Backing up databases

4. Apache and the Domains module can handle how many domains?
    a. 1
    b. 3
    c. 10
    d. For as many or as few as your server can handle traffic
5. We package similar Drupal structural items in a feature:
    a. So the structures can be version-managed
    b. So the structures can be copied easily
    c. Both a and b
    d. Neither a nor b
6. Views cannot be version-managed.
    a. True
    b. False

Summary

We learned a lot in this chapter about managing development environments, domains, and features.

Specifically, we:

- Used Drush to download and install the Domain and other related modules
- Configured the Domain and related modules locally and were able to segregate content on the two domains
- Backed up the database to a local directory with the **Backup and Migrate** module
- Built our first **Feature** module
- Checked our changes into version management and updated a "remote" server with our changes

Now that we've learned about managing and changing this environment, we're ready to start working on the mobile theme.

4
Introduction to a Theme

The British have an expression that aptly describes the current state of Drupal 7 theming for mobile. The phrase "the dog's dinner", while of unknown origin, has come to mean "a recipe fit only for the mouth as a last resort". Much of the work that was done to retrofit Drupal 6 for mobile isn't needed in Drupal 7, but Drupal 7 is still new enough that the heavy hitters of the module world are being ported to Drupal 7 without much thought as to the changes Drupal 7 brings to the job of theming. In fact, many of the modules that were commonly used in Drupal 6 simply aren't ready for a live Drupal 7 site. We will attempt to make sense of this "dog's dinner" and get a "working mobile site" up quickly. We'll then, in a later chapter, go back and refine the mobile site to make it more graceful and tailored to our client's needs.

In this chapter, we will:

- Introduce the changes which HTML5 brings to theming
- Review the concepts of the "mobile theme"
- Learn about redirection and guessing the intent of a browser
- Progressively enhance our mobile site

So let's get on with it...

Progressive Enhancement

Grandpa Fiorello Drupallo still has his e-Machines PC, running Windows 2000 that he bought in 1999. It hasn't broken down and it still "gets the Internet" very well, thank you, so why should he buy a new one? Why indeed! In this economy, with prices the way they are? Little Jimmy tried to get grandpa a new one for Christmas, but grandpa is insistent. He likes his old machine and is quite comfortable with it running and working the way it does. Problem is, it's still running IE 6 and when grandpa views Drupallo's website, he wants to see a great looking site on his ancient (in Internet terms) web browser.

The bane of every frontend web developer's experience is **Internet Explorer** (**IE**), more specifically, IE 6 and IE 7. They have consistently given the worst browser experience, the worst standards support, and the worst markup rendering. But the customers continue to hold on to older equipment and fail to upgrade their software to the latest browser versions. More often, the users are browsing with company-owned computers where they are reticent or simply unable to upgrade the browser. Either that or they have their own Grandpa Drupallo as one of their stakeholders.

With mobile, we are in the same conundrum. We have many users who don't want the complexity or expense of a smart phone and who are content to use their five-year-old Nokia phone until it completely dies. At the same time, from the statistics collection, we know that the majority of mobile web traffic happens from WebKit-based browsers in Apple's iOS and Google's Android phone operating systems.

When Little Jimmy began the project, he had the discussion with Big Daddy Jimmy about how it would be very difficult to support older technology, but he was going to do his best to make everything look really good, no matter what the customer was using. Big Daddy Jimmy agreed to progressive enhancement, as it's called in the web development business. He agreed, in theory.

Progressive Enhancement is the idea that the browser should display the page as best it can and older browsers with less features simply "miss out" on the better looking page. In the heat of the project, though, stakeholders start looking at web traffic statistics and see that people using IE comprise a much higher percentage of the web traffic and begin to balk and want a pixel-perfect experience.

The frontend developer uses CSS that rounds up the corners of squares and the project manager demands that the site "looks wrong" in browsers that don't support the rounded corners CSS3 property. The developer uses some CSS3 gradients to color the background of a div and to provide the color spec of a solid color for browsers that don't support the gradient. The project manager (or designer) then freaks out that the design looks flat in the older browser.

It is easy to get the project stakeholders to agree in theory to progressive enhancement, but very few actually put this theory into practice. And thus we get page bloat with hundreds of `<div>` tags within `<div>` tags within `<div>` tags, because the stakeholder demands the design to be pixel-perfect to the mockup. I had one stakeholder that actually took a screenshot in IE6 and put it in the `Photoshop psd` file as an onion skin to compare the HTML render with the original mockup.

The web is not the print, the print is not the web, and the desktop web is not the mobile web. Trying to make any one of these into the other does a disservice to each.

Big Jimmy agreed to progressive enhancement in theory, but when the website didn't look quite right in Grandpa's web browser, Grandpa made Little Jimmy sit down for a "talk" and Grandpa explained how he would be "very disappointed" if any of his friends looked at this website and didn't see the quality, which he had worked on for all of his life, put into the pies they make. To make peace in the family and to prevent any further patriarchal disappointment, Jimmy agreed to make the site compatible with IE 6. Grandpa was pleased and family dinners were considerably less adversarial.

Little Jimmy knows his grandmother has an older Nokia phone that doesn't show HTML pages with any fidelity, but he also knows that there is a large audience of tourists who have mobile phones, in the hotels around the pizzeria on the beach, that he would like to make sure can view coupons and a menu, and hopefully get them to call in and order.

So how do you do it? How do you implement the progressive enhancement on an existing site? I think the answer can be actually found in two priorities. First, make sure your mobile customers are taken care of, and then implement a design that puts mobile first and then adds desktop compatibility on top of the good mobile strategy. We'll build it from the ground up, starting at the new HTML5 DOCTYPE.

HTML5 and the simplified DOCTYPE

Drupal 7 takes the steps to make its generated HTML more HTML5-compliant and there is currently a core initiative for Drupal 8 to be completely HTML5 clean (`http://groups.drupal.org/html5`). So in the words of Maria VonTrapp, "Let's start at the beginning. It's a very good place to start. When you read, you begin with A-B-C." When you begin with web markup, you begin with the DOCTYPE.

The Document Type Definition or DOCTYPE is, or should be, the first element of any HTML document. The DOCTYPE describes the way in which the markup, CSS, and the JavaScript should be rendered by the browser's rendering engine. An improper form on the DOCTYPE could lead to the poor rendering of standard CSS in some browsers.

During the late 90s, it was common in Microsoft IE for only XHTML pages to begin the document with an XML declaration to aid in XML parsing:

```
<?xml version="1.0" encoding="UTF-8" ?>
```

This should never be used on any website. If there is an XML declaration at the beginning of an HTML document, Internet Explorer versions 9 and below will render that document in what's called "Quirks Mode". The Quirks Mode is an evil machination of Dark Forces that provides for non-standard CSS rendering in IE. It is also my belief that its use will lead to hunger and famine in many parts of the world, although direct cause-and-effect has not been proven. The point is, it's never a good idea to start an HTML document with an XML declaration. Someone's life may depend on it.

There are really only two DOCTYPE use cases that work best for modern HTML pages. The first is the older XHTML Strict, which is quite verbose:

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

The second is the HTML5 simplified version:

```
<!DOCTYPE html>
```

The HTML5 version is much easier, isn't it? This is one of the many ways HTML5 simplifies and clarifies markup while making it more semantic and descriptive of its content. This DOCTYPE invokes standards-based CSS rendering on every browser in general use today, including all versions of Internet Explorer:

- **Semantic web**: The semantic web refers to a web of data that can be consumed by other web entities, for example, search engines or news aggregators. A good example is the new **Resource Description Framework** (**RDF**) for web pages that allows information in a website to be read by other websites that aggregate the headlines. Screen readers and accessibility assistance devices rely on good semantic markup to help the visually challenged make sense of a web page. Search engines understand more about web content if more semantic markup is used.
- **RDF**: The RDF is a language for representing information about resources in the World Wide Web. It is particularly intended for representing metadata about web resources, such as the title, author, and modification date of a web page, copyright, and licensing information about a web document, or the availability schedule for some shared resource. However, by generalizing the concept of a web resource, RDF can also be used to represent information about things that can be identified on the web, even when they cannot be directly retrieved on the web. Examples include

information about items available from online shopping facilities (for example, information about specifications, prices, and availability), or the description of a web user's preferences for information delivery (for more details, refer to the W3C RDF Primer document at `http://www.w3.org/TR/rdf-syntax/`).

New HTML5 semantic elements

The other thing that HTML5 brings to the table is a slew of new markup elements, many of which are semantic replacements to using the more generic `<div>` tag everywhere. These are becoming, and will continue to become, more facilitating for mobiles as they will aid in defining the information mobile phones will use to make more sense out of the page.

The problem with introducing new elements is the issue with versions of Internet Explorer (and older versions of Firefox) that don't render the elements correctly. But since version 4.0, IE has had one interesting feature. You can create any element and use it in a stylesheet, as long as the element is created by JavaScript before the page loads. So in order to get HTML5 elements to render correctly in older versions of IE, you'll need a small JavaScript that creates the elements in the head. There's a small JavaScript included on Google Code that will help us with this, called the HTML5 Shim: `http://code.google.com/p/html5shim/`.

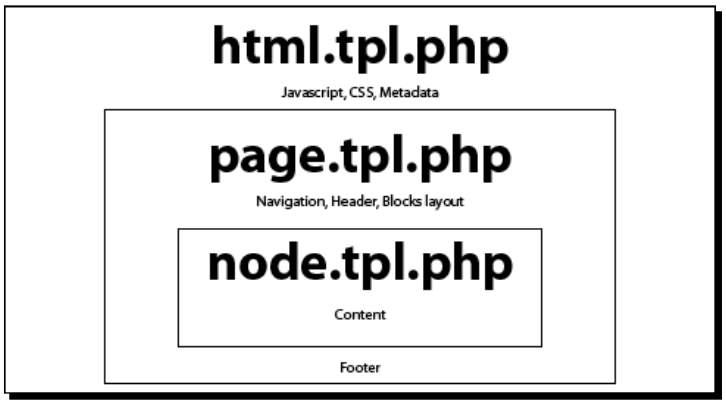
It involves adding a single conditional JavaScript in the page head. It has been added already to the DPK `html.tpl.php` file. We'll use a copy of that file for our new mobile theme, later in the chapter.

Drupal 6 versus Drupal 7 theming

One of the primary ways Drupal 6 theming differs from Drupal 7 theming is in the `page.tpl` file. In Drupal 6, the `page.tpl` held the entire markup for the page and became the base of all of the page's theming. In Drupal 7, that's divided into three different files, and all three are optional to your theme. If you do not have any of the following three in your theme, the default one will be used:

- `html.tpl.php`: Holds the base html, DOCTYPE, head, and body tags
- `page.tpl.php`: Everything that goes inside of the body tags

- `node.tpl.php`: Structure surrounding the `content` variable from the `page.tpl.php`



Let's take a look at a base `html.tpl.php` file:

```
<?php print $doctype; ?>

<html lang="<?php print $language->language; ?>"
  dir="<?php print $language->dir; ?>"
  <?php print $rdf->version . $rdf->namespaces; ?>>

  <head <?php print $rdf->profile; ?>>
    <?php print $head; ?>
      <title><?php print $head_title; ?></title>
      <?php print $styles; ?>
      <?php print $scripts; ?>
      <!-- backwards HTML5 compatibility for IE -->
        <!--[if lt IE 9]>
      <script src="http://html5shim.googlecode.com/
      svn/trunk/html5.js"></script>
    <![endif]-->
  </head>

    <body class="<?php print $classes; ?>" <?php print
    $attributes;?>>

    <?php print $page_top; ?>
    <?php print $page; ?>
    <?php print $page_bottom; ?>

  </body>
</html>
```

The first line allows Drupal to manage the DOCTYPE statement. The second line adds the RDF additions to the HTML tag as well as the default Drupal translation. The `<header>` tag is a standard Drupal 7 style and script, with RDF additions, if needed by Drupal. Notice the inclusion of the conditional statements for backwards HTML5 compatibility for IE.

Conditional statements

Conditional statements are codes that are invisible to browsers other than IE. You can use them to target specific versions of IE or a range of versions with the `if` syntax in the first conditional statement. The statement, "`if lt IE 9`" gets translated to "if less than IE version 9". As of version 9, IE renders the new HTML5 elements correctly.

Body classes and attributes will be handled in the pre-process functions in `template.php`. The `$page_top` and `$page_bottom` variables are added for Drupal's overlays and the admin menu and the `$page` variable hold the rendered content of the `page.tpl.php` file. Let's look at a standard `page.tpl.php` file:

```
<?php echo render($page['header']); ?>
<header id="main-header" class="clearfix">
  <div class="container">
  <h1><?=l($site_name, "<front>");?></h1>
    <nav id="menu">
      <?php if ($primary_nav): print $primary_nav; endif; ?>
      <?php if ($secondary_nav): print $secondary_nav; endif; ?>
    </nav>
    <?php if (isset($main_menu)) { ?>
      <?= theme('links', $main_menu, array('class' => 'links',
      'id' =>'main')) ?>
    <?php } ?>
    <?php if (isset($secondary_menu)) { ?>
      <?= theme('links', $secondary_menu, array('class' => 'links',
      'id' => 'secondary')); ?>
    <?php } ?>
    <?=$search_form;?>
  </div>
</header>
```

The header consists of the site name, primary, and secondary menus as well as the search form. Notice we have used the HTML5 `<header>` element to define the header rather than the more generic `<div>` with a class or ID of `<header>`:

```
<section id="main" class="clearfix">
  <?php if ($page['sidebar_first']): ?>
```

```
      <aside id="sidebar-first" class="sidebar">
        <?=render($page['sidebar_first']); ?>
      </aside>
    <?php endif; ?>
    <?php print $breadcrumb; ?>
    <?php if ($page['highlighted']): ?>
      <div id="highlighted">
        <?=render($page['highlighted']); ?>
      </div>
    <?php endif; ?>
    <a id="main-content"></a>
    <?php if ($tabs): ?>
      <div id="tabs-wrapper" class="clearfix">
      <?php endif; ?>
        <?=render($title_prefix); ?>
        <?=render($title_suffix); ?>
      <?php if ($tabs): ?>
    <?=render($tabs); ?></div>
    <?php endif; ?>
    <?php print render($tabs2); ?>
      <?=$messages; ?>
      <?=render($page['help']); ?>
      <?php if ($action_links): ?>
        <ul class="action-links">
          <?php print render($action_links); ?>
        </ul>
      <?php endif; ?>
    <article class="clearfix">
      <?php print render($page['content']); ?>
    </article>
    <?php if ($page['sidebar_second']): ?>
      <aside id="sidebar-second" class="sidebar">
        <?php print render($page['sidebar_second']); ?>
      </aside>
    <?php endif; ?>
  </section>
```

The markup is pretty straightforward and very similar to the content area of a Drupal 6 `page.tpl.php`. Notice the addition of the new HTML5 elements `header`, `article`, and `section` that add to the semantic description of the content on the page.

More semantic > less semantic:

In the early days of search engines, we learned to do things such as put lots of links to ourselves throughout the text so the search engine would think lots of pages linked to our content. We put repeated text in the `<meta>` tags and had `<meta>` tags with hundreds of words to try and come up with every eventuality of searching combinations. This voodoo markup was what was commonly meant by **Search Engine Optimization** (**SEO)**. Google has evolved the search engine since then. They've gotten better at looking at content and there's one thing I promise – their ability to program their servers to understand content is better than your ability to try to trick them. They have more money and better programmers and are better at search theory than you.

The current approach to SEO is to try to "describe" the content as entirely as possible. What I mean by "describe" is that, as developers, we need to remove as much inaccessible content as possible and to add semantic tags to help search engines understand the context of the content as much as possible. People will immediately close a window that doesn't have the content they want. Don't try to trick Google to get page views because those type of page views have no value to advertisers. The HTML5 structure tags like `<header>`, `<footer>`, `<article>`, and `<aside>` will help with SEO and further describe your content. The more accessible, semantically described content you have, the more page views you will get.

The `<footer>` element appears at the bottom of the page and contains the `<footer>` region for the theme and utilizes the new HTML5 `<footer>` element. Refer to the following code snippet:

```
<footer id="main-footer" class="clearfix">
  <?php print $feed_icons ?>
  <?php print render($page['footer']); ?>
</footer>
```

Now let's install an even simpler theme for our new mobile site.

Time for action – installing the default mobile theme

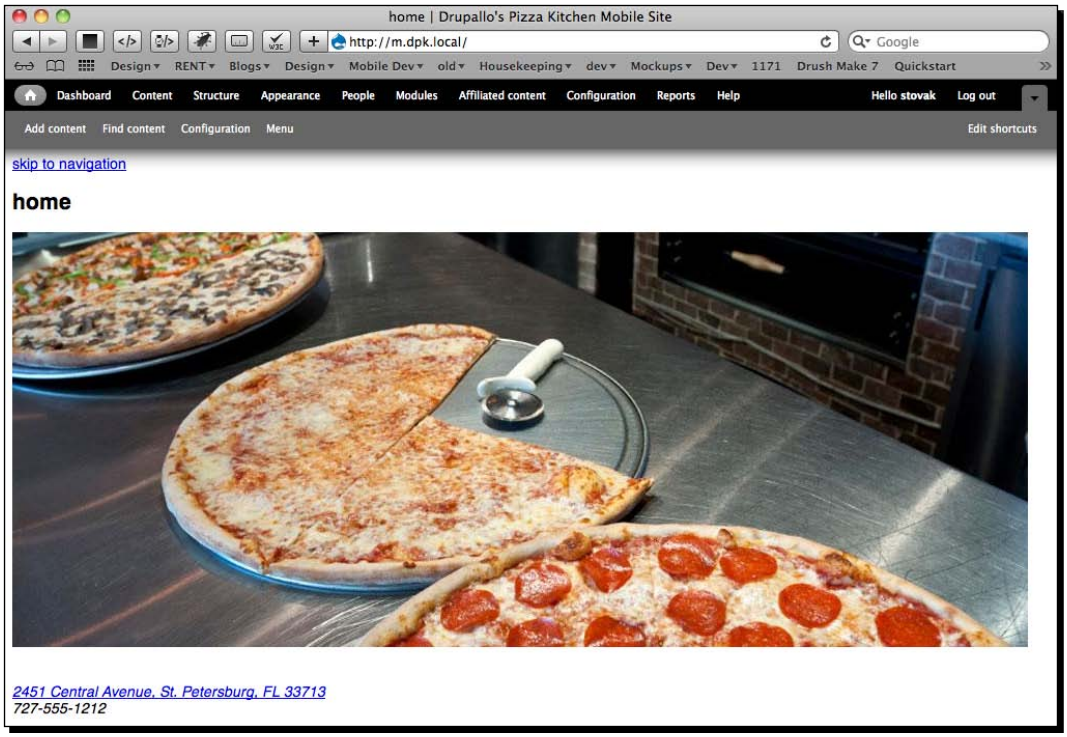
1. Open a new terminal window, change the directory to your site root, and let's install the mobile theme:

   ```
   drush dl mobile
   drush pm-enable mobile
   ```

2. Next open a browser window and go to the `http://dpk.local website.`

3. Choose **Structure** | **Domains**.

4. From the **Domains** list, choose the mobile domain's theme settings.
5. Change the theme to **mobile.**
6. Point the browser to the mobile domain (**http://m.dpk.local**) in this example. You should see the simple theme:



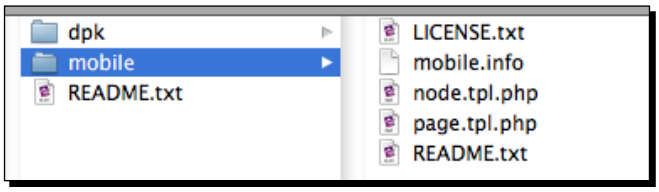
What just happened?

The first line tells Drush to download (`dl`) the `mobile` theme. The second line tells Drush to enable the module you just downloaded. After each of the `enable` commands, you'll need to approve with a *y* for yes.

Then we assign a different theme to the mobile URL.

The simple life

Let's take a look at the newly-downloaded **mobile** theme:



As you can see, the **mobile** theme is very simple—no custom CSS and no customized JavaScript. Notice the lack of `html.tpl.php`. As mentioned before, the default `html.tpl.php` is used.

What we need to do is add a link from the low resolution site to the high resolution one on both sites. The `mobile.info` file looks something like this:

```
name = "Mobile"
description = "Ideal for small devices and/or for low-bandwidth
consumers."
core = 7.x
engine = phptemplate
regions[help] = Help
regions[navigation] = Navigation
regions[content] = Content
; Information added by Drupal.org packaging script on 2010-12-11
version = "7.x-1.x-dev"
core = "7.x"
project = "mobile"
datestamp = "1292026750"
```

The mobile template basically has three regions – "Help", "Navigation", and "Content". The `page.tpl` looks like this:

```
<a href="<?php print url($_GET['q'], array('query' => NULL,
'fragment' => 'nav', 'absolute' => TRUE)); ?>"><?php print t('skip to
navigation');?></a>

  <?php print render($title_prefix); ?>
  <?php if ($title): ?><h2 class="title" id="page-title"><?php print
    $title; ?></h2><?php endif; ?>
    <?php print render($title_suffix); ?>

    <p id="help"><?php print render($page['help']); ?></p>
    <?php if ($messages != ""): ?>
```

```
    <div id="message"><?php print render($messages) ?></div>
  <?php endif; ?>
  <?php if ($tabs != ""): ?>
    <?php print render($tabs); ?>
  <?php endif; ?>

  <?php print render($page['content']); ?>
    <a name="nav"></a>
  <?php print render($page['navigation']); ?>
```

The first line is a link to quickly scroll to the bottom navigation. The next section is a "headline" area, followed by the standard Drupal messages and tabs sections. Let's add some CSS to this very basic theme.

Media queries

From the WC3 working draft of CSS3:

> *"A media query consists of a media type and zero or more expressions that check for the conditions of particular media features. Among the media features that can be used in media queries are width, height, and color. By using media queries, presentations can be tailored to a specific range of output devices without changing the content itself."*

The sales pitch does that by using media queries, we can use the same markup on multiple screen sizes and resolutions to produce the best possible design, given the current device. But soon, as we will see, there's no silver bullet to mobile site design.

Media queries can be used in different ways. You can use them in the link tag to specify the situations in which the CSS file should be loaded and used:

```
<link rel="stylesheet" media="screen and (min-device-width: 800px)"
href="example.css" />
```

This would load the referenced CSS only on screens with a device width of `800 px` or larger. They can also be used in the CSS itself with the `@media` rule, as shown in the following code snippet:

```
<style>
  @media all and (orientation:portrait) { … }
  @media all and (orientation:landscape) { … }
</style>
```

But what about Internet Explorer? Older versions don't work with these fancy new CSS3 directives. Well, actually, the absence of media queries is, in and of itself, a media query. You can use the fact that a browser does not support media queries to supply default CSS in the event the others fail.

Now, at this point, it would be tempting to go through the original site `global.CSS` and use the media queries to design three or four screen use cases and call it a day. But honestly, there's more to mobile web development than that. There are several reasons this is true:

- If you have images on the page and the image is set to `display:none`, those images are still downloaded
- Adding mobile CSS doesn't fix issues with JavaScript load and execution times
- The concern with using a mobile browser is not just how the information looks, but what information shows and where it shows on the small screen

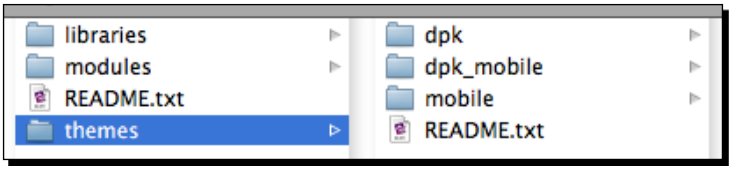
Time for action – personalizing the mobile theme

Let's copy the "mobile" theme into a new theme and make our changes to the new one:

1. Open a terminal window and change the directory to your site root:

   ```
   cp -R sites/all/themes/mobile sites/all/themes/dpk_mobile
   ```

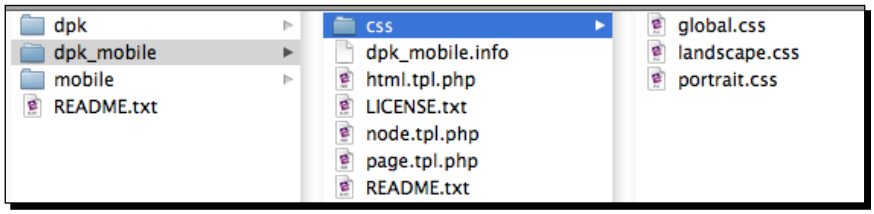
2. As shown in the following screenshot, you should now have three themes in the `sites/all/themes` directory:



Rename the `.info` file to `dpk_mobile.info`. Edit the new `dpk_mobile.info` file, changing the `name` variable to `DPK Mobile`, and add three new stylesheets:

```
name = "DPK Mobile"
stylesheets[all][] = css/global.css
stylesheets[screen and (orientation:landscape)][] = css/landscape.
css
stylesheets[screen and (orientation:portrait)][] = css/portrait.
css
```

3. Copy the `HTML.tpl.php` file from the DPK theme to the `dpk_mobile` theme.
4. Create a new folder called `css` inside the `dpk_mobile` theme directory.
5. Add three new empty files to the `CSS` folder: `global.css`, `portrait.css`, and `landscape.css`:



6. Add the following rule to `landscape.css`:

```
header { border: 5px solid  orange; }
```

7. Add the following rule to `portrait.css`:

```
header { border: 5px solid  green; }
```

8. Open a browser window, go to `http://dpk.local`, and choose **Appearance**. The new DPK **mobile** theme should be listed but disabled. Enable it by going to **Structure | Domains**. Make the default mobile theme for the `mobile` domain the new `DPK Mobile` theme we just created.
9. Copy the `template.php` file from the main theme to the mobile theme.

```
cp sites/all/themes/dpk/template.php sites/all/themes/dpk_mobile/
template.php
```

10. Change all the function names in `template.php` from `dpk_` to `dpk_mobile_`.
11. Go to **Configuration | Performance | Clear cache.**
12. After a page refresh, the PHP notices should disappear.

What just happened?

First we copied all the files from the `mobile theme` project.

The conditional stylesheets are added to the site by the new entries to the `mobile theme.info` file. When you view the source of the mobile site, you should see the new stylesheets we just added to the theme; two of which have media queries. Refer to the following code snippet:

```
<style type="text/css" media="all">
  @import url("http://m.dpk.local/sites/all/themes/
  dpk_mobile/css/global.css?ln1wht");
</style>
<style type="text/css" media="screen and (orientation:landscape)">
  @import url("http://m.dpk.local/sites/all/themes/
  dpk_mobile/css/landscape.css?ln1wht");
</style>
<style type="text/css" media="screen and (orientation:portrait)">
  @import url("http://m.dpk.local/sites/all/themes/
  dpk_mobile/css/portrait.css?ln1wht");
</style>
```

If you open the iOS or Android simulator, launch the mobile browser and open the mobile site URL. You can switch orientations from the **Hardware** menu. Notice how switching orientations changes the color of the border around the graphic on the main page.

The functions in `template.php` are not as straightforward. If you've done any Drupal theming, the `dpk_mobile_preprocess_page` function will look familiar to you. It is implemented as it is in Drupal 6, only it is executed before `page.tpl.php` is processed. Let's take a look at the search lines at the bottom of the function:

```
function dpk_mobile_preprocess_page(&$vars) {
  $search = Drupal_get_form('search_form', NULL, (isset($searchTerm)
  ? $searchTerm : ''));
  $search['basic']['keys']['#type'] = "search";
  $search['basic']['keys']['#size'] = "20";
  $search['basic']['keys']['#attributes']=array("placeholder" =>
  "search", "autocapitalize" => "off", "autocorrect" => "off");
}
```

There are several items to notice here. First, the `['#type'] = 'search'.` `<Input type='search' />` is a new HTML5 element alternative to `'type=text'`. It is one of the ways HTML5 is "more semantic" than its predecessor. Also notice the placeholder attribute. Placeholder is a new attribute for the `form text` fields that will put "grayed out" text that disappears on focus. `autocapitalize` and `autocorrect` turn off the handheld's automatic capitalization and text completion functions for this input control only.

We've also added a `dpk_mobile_preprocess_html` function that does similar things for the `html.tpl.php` file. Refer to the following code snippet:

```
function dpk_mobile_preprocess_html(&$vars) {
  if (module_exists('rdf')) {
    $vars['doctype'] = '<!DOCTYPE html PUBLIC "-//W3C//DTD HTML+RDFa
    1.1//EN">' . "\n";
    $vars['rdf']->version = 'version="HTML+RDFa 1.1"';
    $vars['rdf']->namespaces = $vars['rdf_namespaces'];
    $vars['rdf']->profile = ' profile="' . $vars['grddl_profile'] .;
  } else {
    $vars['doctype'] = '<!DOCTYPE html>' . "\n";
    $vars['rdf']->version = '';
    $vars['rdf']->namespaces = '';
    $vars['rdf']->profile = '';
  }
}
```

In this function, we set the DOCTYPE and RDF information if the RDF module is enabled. If the RDF module is not enabled, we use the simple HTML5 DOCTYPE declaration.

We've also added two other functions:

- An `html_head_altar` hook: This function adds the meta information for the UTF8 character set.

  ```
  function dpk_mobile_html_head_alter(&$head_elements) {
    $head_elements['system_meta_content_type']['#attributes'] =
    array(
      'charset' => 'utf-8'
    );
  }
  ```

- The `_preprocess_search_block_form` hook is one last hook, changing the `type=text` to `type=search` for any `search` form not caught by the page preprocess hook. Sometimes the `search` form is used on other parts of the page and we want to make sure when it is used that it's the HTML5 version and not the more generic "text" version.

Redirecting mobile clients

To create a better experience for our users, it is standard practice to scan the client for certain criteria and redirect them to a site that would better serve their device. There are basically two approaches for redirection. The first involves sniffing the "user agent".

The word **request** is defined as, "The act of asking for something to be given or done." A client using a browser enters a URL into the address bar and we refer to that as a request. The browser's host networking protocol then does a lookup on the domain name of the address in the bar. Once the domain's IP has been resolved, it then makes a `GET` request of the remainder of the URL, or the index, if there's nothing behind the domain. The browser sends its identity along with the request in the form of a user agent string. Each browser will have a slightly different user agent string. Here's one from a browser I'm currently using:

```
Mozilla/5.0 (Macintosh; Intel Mac OS X 10_6_7) AppleWebKit/535.1+
(KHTML, like Gecko) Version/5.0.5 Safari/533.21.1
```

Just a day after I wrote this, Apple released an update changing the string to this:

```
Mozilla/5.0 (Macintosh; Intel Mac OS X 10_6_7) AppleWebKit/534.42
(KHTML, like Gecko) Version/5.1 Safari/534.42
```

I also have several other browsers installed on my hard drive:

```
Mozilla/5.0 (Macintosh; Intel Mac OS X 10.6; rv:2.0) Gecko/20100101
Firefox/4.0
```

Here's another one:

```
Mozilla/5.0 (Macintosh; Intel Mac OS X 10_6_7) AppleWebKit/534.30
(KHTML, like Gecko) Chrome/12.0.742.91 Safari/534.30
```

And these are just a few.

These strings tell the browser about the capabilities of the client making the request. The problem is, for as many browsers and devices on the Internet, there are that many user agent strings. Imagine every version of Windows ever published. Every possible configuration; each one has its own user agent string. Every handheld and tablet has its own string, and every time the software on any of them gets updated, the user agent string changes slightly. As you can imagine, this makes sniffing this string difficult for a programmer, with too many use cases to be able to do so with any accuracy. Enter a duo of open source projects; `browscap` and `wurfl`. Both libraries operate on the same principle. They keep track of all the different user agent strings and can allow you to perform redirection based on a series of user agent string matches. They each have a companion Drupal module. The libraries are constantly updated and you can set up a `CRON` task to download the latest version of the libraries' base files when new versions are released.

But this brings us back the questions, "What is mobile?", "Why are we building this second website anyway?", "Who should we redirect to?", "Under what conditions should they be redirected?", and "What if the user prefers the full site on their mobile phone?", "Should they not be able to get that version?"

The `mobile_tools` module implements much of the functionality we have developed so far and also has a `browscap` redirect. There are a couple of reasons I don't like the implementation. The first is because when you set mobile tools to redirect, it always redirects the client to the mobile site if it thinks you're using a handheld. I think that this is a wrong approach. I think the user should be redirected, but if they express a preference for one site over the other, that preference should be honored. Indeed to my way of thinking, if you have a "second URL" for handhelds, what you should target is the screen size. What you're really doing with the two sites is making the website legible for two different screen sizes. Which brings me to the second way to sniff for client redirection – through what's called "capabilities scan".

Until we get the unified theme designed, it's my belief that the two users can be distinguished by simply using JavaScript to redirect, based on screen size. Anything under 500 pixels gets the small site. Anything over 500 pixels should get the full site, unless a user expresses a preference for either.

So let's write some JavaScript that will perform redirection for our themes.

Time for action – writing JavaScript redirection for our theme

We'll add a `global.js` file and some Drupal behaviors to our theme to redirect our users, as we anticipate they will want to be redirected:

1. First we'll add a `global.js` to both themes. Edit the `.info` file for each of the two themes and add the following line:

   ```
   scripts[] = js/global.js
   ```

2. Next, create a folder `js` and add an empty `global.js` file to the folder.

3. Go to **Configuration** | **Performance** and click the **Clear cache** button.

4. Go to both sites and verify that the `global.js` is now in the "head" as a `javascript` include:

   ```
   <script type="text/javascript" src="http://dpk.401k.local/sites/
   all/modules/contrib/views_slideshow/contrib/views_slideshow_cycle/
   js/views_slideshow_cycle.js?lnb1w2"></script>
   <script type="text/javascript" src="http://dpk.401k.local/sites/
   all/themes/dpk/js/global.js?lnb1w2"></script>
   <script type="text/javascript" src="http://dpk.401k.local/sites/
   all/modules/contrib/views/js/jquery.ui.dialog.patch.js?lnb1w2"></
   script>
   ```

5. Go to **Structure** | **Blocks.**

6. There should be a domain list navigator block in the disabled blocks list. Click **Configure**.

7. Leave the **Block title** blank. Under paths, make sure they "link to the home page" and change the **Link theme** to an **Unordered list of links**.

8. Go to **People** and then click the tab **Permissions**. Scroll down to the **Domain content** section and there's a permission called **Access domain navigation links**. Make sure all user roles are checked. We want the menu to show up for everyone. Scroll to the bottom of the page and save the permissions you just changed.

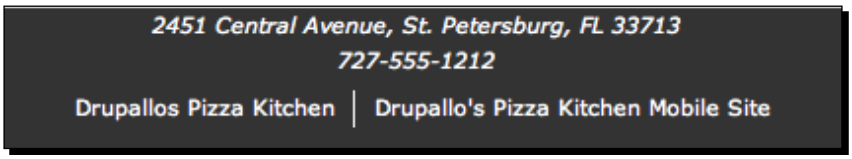
9. Navigate back to your block's admin screen and under **Region settings**, put the list in the **Footer** on the **dpk (default theme)** and put in the **content** section in the **DPK Mobile** theme. Save the block, and on the next page, save your new "blocks" configuration.

10. Now, at the bottom of the page, there should be two links to the two DPK domains, namely, **Drupallo's Pizza Kitchen** and **Drupallo's Pizza Kitchen Mobile site**.

11. Add the following to `global.css` in both the **dpk** and **DPK Mobile** themes:

```
.block-domain-nav ul li
  { display: inline-block;  border-left: 1px solid white;}
.block-domain-nav ul li:first-child
  { border-left: 0px none;}
.block-domain-nav ul li a
  { display: block; margin-left: 5px; padding-left: 5px; }
.block-domain-nav ul li:first-child a
  { margin-left: 0px; padding-left: 0px; }
```

Refresh the page and your footer should look like this:



What just happened?

Well, first, we created a `global.js` file for the theme. This will hold JavaScript that is specific to the theme, mainly the re-routing scripts.

Second, we added that JavaScript to the `.info` file. The reason we do this is because Drupal has some special "behind the scenes" handling of JavaScript files where it can compress all the `.js` files into a single file, making it faster for loading and execution on the live site.

Next, we grabbed the jQuery "Cookie" plugin and cloned it in the `Libraries` folder (we'll need to remember to add that to the `.make` file later) and then to the theme's `.info` file.

Next, we added off-the-shelf domain navigator from the `domains` module. We configured it to list the domains at the bottom of the page on both the mobile and the full-site theme and then styled the list accordingly.

So why did we add global JavaScript files to both themes? Well, yes, there is a method to this madness.

Give them what you think they need until they tell you what they want

What we need to do now is come up with a series of rules to intelligently redirect browsers to the website that we can intelligently discern they should have, and then allow them to express a preference, if they care to do so. It is also important to me to be unobtrusive about this redirection. To my way of thinking – giving a user a pop up that asks them to express a preference is a bigger sin than giving them the wrong site. It's based on my core belief that pop ups are the root of all that is evil in UI. Feel free to disagree.

The redirection flow, as described below, is based on my logic and doesn't represent any definite way to do anything. Feel free to come up with a better way to do this. I'd love to hear your ideas. Here are the use cases I came up with:

- The majority of users, both desktop and mobile, will go to the main URL. Entering the URL to the mobile site is, in and of itself, expressing a preference.
- Desktop users should never be redirected. If no preference is expressed, users of desktop and larger tablet browsers should get the desktop site. If they land on the `.com` site, they get that site and are not redirected. If a desktop user lands on the mobile site, this act expresses a preference for the mobile site. Give them the mobile site with no redirect. If they then go back to the primary site, do not redirect.
- If no preference is expressed, users of smaller-screen mobile devices should get the low-end mobile version. If a mobile browser accesses the desktop site for the first time, redirect them silently and give them the mobile site with a flag at the top of the page telling them what we did.
- If the mobile user clicks the link to go back to the site, from now until they express another preference, the desktop site is preferred. Never redirect them again to the mobile site, but allow them to see the link at the bottom of the page if they care to express another preference. It's very annoying to be on a mobile site, as it doesn't have the information you need, so you go to the full site. The full site loads, only to be redirected again to the mobile one.

With these rules in mind, let's write some JavaScript that allows mobile users to be redirected, but allows them to override the redirection when they express a preference.

For those of you who have written JavaScript before, it would be tempting to sit down and write a quick jQuery function inside a `jQuery onDocumentReady` script. And that would probably work without any issue. But you are using Drupal. And Drupal has a method for adding scripts to the `onDocumentReady` process that allows items that are loaded via AJAX to be processed incrementally to eliminate duplicate processing of JavaScript objects. It won't be an issue with this particular script, but we'll code this script in the correct Drupal way, to get into the habit of using Drupal behaviors.

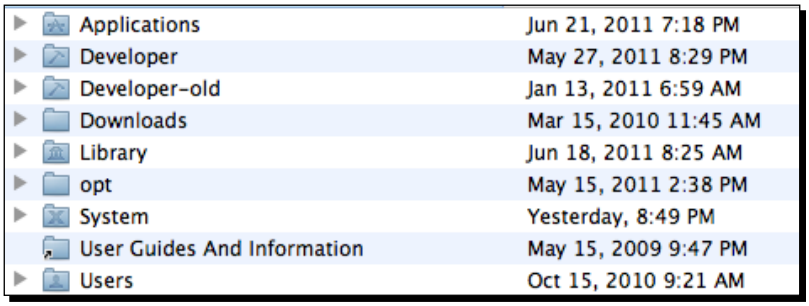
Behave yourself

We live in a world where the static web page isn't what it used to be. Many times, there is JavaScript activity on the page that loads new items into the DOM. If you have JavaScript that processes the page, that processing needs to be done again on the newly-loaded items. Many times, in carelessly written JavaScript, the processing happens multiple times for all the items on the page each time new items are loaded into the DOM.

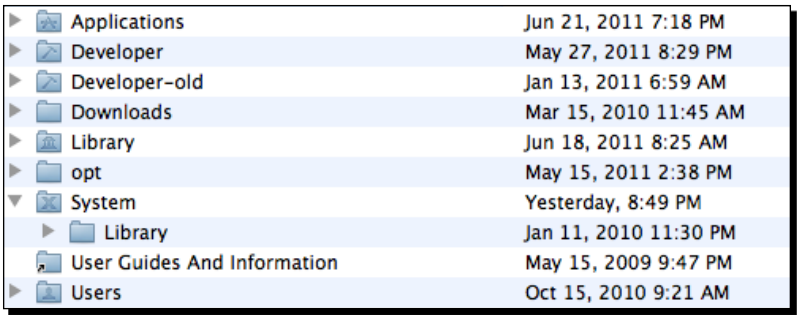
Document Object Model (DOM)

It is a cross-platform way of interacting with HTML. Once the page is loaded, the browser is able to access every item on the page using the DOM of addressing. It is looking at the web page through a JavaScript lens and is able to interact with every item on the page through JavaScript.

Let's have a look at a hierarchical listing:



Each item in the listing has a "reveal" triangle. Each time we click on the triangle, we want to show the children of the folder. But we don't want the whole page to load each time we want to see children. The script we wrote scans the list, and then attaches a JavaScript behavior to the triangle that makes an AJAX call to the server to load the children when the triangle is clicked and load them in a way that shows them underneath the parent. Take a look at **System** in the following screenshot. We have revealed the child **Library** underneath it:



The problem is, the child (children) now needs the same "click-reveal" behavior attached to it so we re-execute the same script. Now each one of the triangles has two behaviors attached to it, so clicking one of the original items results in two AJAX calls and two copies of the children loaded underneath the parent. You can see how quickly this can get out of control.

What we need is a way of executing JavaScript so that, on the initial page load, the whole document is processed, but on each successive AJAX call, only the new nodes are processed.

Drupal behaviors

Drupal behaviors are a site-wide JavaScript convention that gives context to each script so that behaviors are not mistaken and attached to a DOM object twice. Let's examine the following small behavior:

```
Drupal.behaviors.displayChildrenOfFolder = {
  attach: function(context) {

    $(".module-name-folder:not(.module-name-processed)",
    context).click(function(){

    // There's a better way to do this ajax, but one concept at a
    time
      $("#container-for-children").load(URL, {
        success: function(blah, blah) {
        Drupal.attachBehaviors($("#container-for-children"))
        }
      });
    }).addClass(".module-name-processed");
  }
}
```

First thing we notice is that there is a global object called **Drupal**. All behaviors are methods of `Drupal.behaviors`. The `Drupal.behaviors` are all called when Drupal executes an `AttachBehaviors`. This happens on load, but we can trigger it too. When the page loads, Drupal executes `attach` for all the methods in the `Drupal.behaviors` object and it executes them within the context of the entire page. So, our jQuery script finds all of the objects that do not have the class `module-name-processed` and processes them with a `click` event listener. The `click` event listener then makes the AJAX call, but after the call has succeeded and loaded the data into the child container, the script then runs the `Drupal.attachBehaviors` on all of the newly-loaded HTML.

Now, you might be thinking, "What in the world does this have to do with the redirection we need to do on the DPK site?" Well, nothing… except for the fact that we're going to use Drupal behaviors and I wanted to make a case for why we code this way. We code this way to avoid JavaScript event listener bloat and collisions with existing function names. On this specific example, it isn't technically "necessary" to do it the right way, but it's always good to know why we code to standard.

Now, let's write a Drupal behavior that will add a listener to the full site for mobile clients and redirect, based on the rules we laid out.

Time for action – redirection with a cookie to remember state

So many times, when you make a choice or preference on the website, you return to the website only to discover that the preference is lost. If you prefer one version of the website to another and make that preference known by choosing, the website should store your choice in a cookie and remember it when you come back. We are going to implement this in Drupal:

1. Edit the `sites/default/settings.php` file. Search for `cookie_domain`. Uncomment it out by deleting the `#` character at the start of the line and change the `cookie_domain` value to `dpk.local`:

```
$cookie_domain = "dpk.local"
```

2. Edit the `template.php` of both sites, adding the highlighted lines to the `dpk_mobile_preprocess_html` function:

```
function dpk_mobile_preprocess_html(&$vars) {

  global $cookie_domain;

  drupal_add_library("system", "jquery.cookie");

  drupal_add_js(array("cookie_domain" => $cookie_domain),
  "setting");
  if (module_exists('rdf')) {
```

```
    $vars['doctype'] = '<!DOCTYPE html PUBLIC "-//W3C//DTD
    HTML+RDFa 1.1//EN">' . "\n";
    $vars['rdf']->version = 'version="HTML+RDFa 1.1"';
    $vars['rdf']->namespaces = $vars['rdf_namespaces'];
    $vars['rdf']->profile = ' profile="' . $vars['grddl_profile']
. '"';
  } else {
    $vars['doctype'] = '<!DOCTYPE html>' . "\n";
    $vars['rdf']->version = '';
    $vars['rdf']->namespaces = '';
    $vars['rdf']->profile = '';
  }
}
```

3. Edit the `sites/all/themes/dpk/js/global.js` file. Add the following lines:

```
Drupal.behaviors.mobileRedirect = {
  attach: function (context, settings) {
    if (Number(window.screen.width) <= 500 && jQuery.cookie("dpk
    -site-preference",  { "domain": Drupal.settings.cookie_domain,
    "path": "/" }) != jQuery(".block-domain-nav .content ul li
    a.active").attr("href")) {
      jQuery.cookie("dpk-has-been-redirected", "YES",
      { "domain": Drupal.settings.cookie_domain, "path": "/" });
      document.location.href=jQuery(".block-domain-nav .content ul
      li a").not(".active").attr("href");
    }
  }
}
```

4. Edit the `sites/all/themes/dpk_mobile/js/global.js` file. Add the following lines:

```
Drupal.behaviors.handleRedirect = {
  attach: function (context, settings) {
    if (jQuery.cookie("dpk-has-been-redirected", { "domain":
    Drupal.settings.cookie_domain, "path": "/" }) == "YES") {
      jQuery(document.createElement("div")).html("<h3>We've taken
      the liberty of redirecting you to our mobile site. <a
      href='javascript:setCookieAndRedirect(); return false;'>Go
      back to our FULL site</a></h3>").prependTo("body");
    }
  }
}

function setCookieAndRedirect() {
  fullSite = jQuery(".block-domain-nav .content ul li
```

```
    a").not(".active").attr("href");
    jQuery.cookie("dpk-has-been-redirected", null);

    jQuery.cookie("dpk-site-preference", fullSite, { "domain":
    Drupal.settings.cookie_domain, "path": "/" });
    document.location.href=fullSite;
}
```

5. Open a command-line terminal and change the directory to your site's root. Add the new files to your local GIT repository:

```
git add sites/all/themes/dpk_mobile
git add sites/all/themes/dpk
git commit –m 'new mobile theme'
```

6. Clear the site cache by either doing a `drush cc all` from the command line or going to **Configuration** | **Performance** and clicking the **Clear cache** button.

What just happened?

Cookies are bits of information that are stored inside the browser. They are used for a lot of different things. For security purposes, they are restricted to a single domain so, for instance, no one can read your saved password for another website. The `settings.php` file contains the default settings for the Drupal installation. What we are doing is making `m.dpk.local` and `dpk.local` use the same cookie so they can see the message we are sending from the full site.

The second item adds three lines to the HTML preprocess hook. We are declaring the `$cookie_domain` variable we set in the `settings.php` to be global and available to this function, then adding it to the `Drupal.settings` array. This is a JavaScript array that holds a myriad of Drupal settings for JavaScript to be used on the site. The second line adds the `jquery.cookie` library, so we can use it to set and retrieve cookie values.

The third item is a Drupal JavaScript behavior that does the redirection from the full site. It looks at the screen width and, whether there is an expressed site preference, and if there is no preference, and the screen width is under 500 pixels, it redirects to the mobile site and sets the "I've just been redirected" cookie, so that the mobile site can warn the user.

The fourth is an addition to the mobile site's `global.js` that warns the user on redirect that they have, indeed, been redirected. It also offers them the option to go back to the original site and gives them a link.

The second function handles that link's click. The function sets the "I have a site preference" cookie, unsets the "I have been redirected" cookie, and redirect the browser's URL to go back to the full site.

Then we should commit all our changes to GIT and clear Drupal's cache. If we deploy right now, we'd do a `git push` to push our changes out to our website, but you would have to wait until the next chapter, as we will adapt the home page design for both full site and mobile.

Pop quiz

1. The idea that web page design should start with the lowest common denominator of functions and work upwards is called:
    a. Search Engine Optimization
    b. Module Creation
    c. Progressive Enhancement
    d. Structural Arrangement
2. The next major version of HTML markup is commonly referred to as:
    a. HTML 4.02
    b. HTML Extreme
    c. HTML for IE
    d. HTML5
3. HTML5 is ready to be used in browsers today:
    a. True
    b. False
4. The difference between Drupal 6 and Drupal 7 theming is:
    a. There are three default templates instead of a single one
    b. There are multiple hooks for new default templates
    c. The RDF module ships with Drupal 7 and can be worked into the theme
    d. Both a and b
    e. All of the above
5. HTML5 has:
    a. More semantic structure
    b. New structural elements
    c. New form controls
    d. New text formatting structures
    e. All of the above
    f. None of the above

6. What is the Drupal global object JavaScript namespace variable name?
    a. D
    b. Drupal
    c. Behaviors
    d. Window
7. Coding to the Drupal JavaScript standard is important. Why?
    a. To avoid duplicate event firing and event bloat
    b. To keep functions from avoiding namespace collisions
    c. Both a and b
    d. Neither a nor b
8. Drupal JavaScript behaviors typically have two functions. What are they?
    a. Attach and Detach
    b. Catch and Release
    c. Begin and End
    d. Starsky and Hutch

Summary

We started out introducing you to the idea of progressive enhancement. Rather than "degrading gracefully", web developers should start with the least common denominator and build upon it until the website provides the best user experience for the browser being used.

We then turned our attention to the new web standard HTML5 and its new simplified DOCTYPE. Everything about HTML is designed to help build robust web-based applications including new semantic elements, which we then saw put to good use in the basic Drupal project "mobile", which is a theme targeted at mobile user websites. We downloaded, mapped, and installed this mobile theme on our mobile URL.

We talked a bit about the new RDF standard for building websites that are SEO friendly and how HTML5 helps with the semantic markup for RDF.

We then began the process of customizing our default mobile theme, that is, putting in stylesheets for various aspect ratios of the document on handheld devices using media queries in the theme's `.info` file.

Finally, we used Drupal JavaScript behaviors to redirect handheld devices to the handheld URL of our website.

We've learned the basics of customizing the theme for our mobile site. In the next chapter, we will begin the process of customizing the content.

5
A Home with a View

Right now, the home page of our example site is, for the most part, static. It's a view that returns a series of billboards in a jQuery Cycle slider. What we need is a version of the home page that's aware of the domain it's on and responds accordingly.

In this chapter, we will work again on the content of the site for the mobile by:

- Adjusting the size of the home page graphics to support faster downloads and smaller screens
- Using sprites for the site's icon graphics to minimize request calls to the server
- Moving content from a large PDF file to nodes and styling them using the Display Suite module
- Using node reference fields to group menu items
- Creating a view to show the menu in the same format as the print version

So let's get on with it...

The Context and Display suite modules

In the last chapter, we used the built-in Drupal block system to put the Domains link at the bottom of the page. The built-in blocks system has the following problems. Some of these problems are obvious and some are not as obvious:

- The appearance and lack thereof can only be triggered by the path variable.
- Caching of the block is done sitewide; so generally speaking, PHP results are executed once and cached. They can't be individual to the logged-in user or even individual to the request.
- Because of the way Drupal builds a page, every block is built with every page view, even if the block's path criteria isn't met. You wouldn't want any complex PHP to execute with every page refresh action because on pages where it wasn't necessary, it would greatly slow down the site.
- Block views, because they are built once and cached, and do not change when the page arguments change.
- When used with the Domains module, the block is cached once per page load and not per domain. The block will appear the same on every domain with the same path, regardless of whether the content is specific to that domain or not.

The solution to most of these problems is the **Context** module. Context allows one or more conditions to be met for a block to appear and those conditions may be a path or it may be any one of several other conditions. In addition, the **Context** module has the ability to render only those blocks which are necessary for page display.

The **Context** module also allows you to turn off the default Drupal block system triggers. That way, the default block system doesn't try to build every block with every page view. If you have complex blocks, this can speed up the page load time significantly. You can also have a view block, that is specific to a domain, that only pulls content that belongs to that domain, which in our case, is what we want to do.

The other module we're going to introduce you to is the **Display suite** module. Display Suite was first introduced with Drupal 6 and has since been updated and the submodules consolidated for Drupal 7. It is now integral to Drupal 7 with support for the new Entities API.

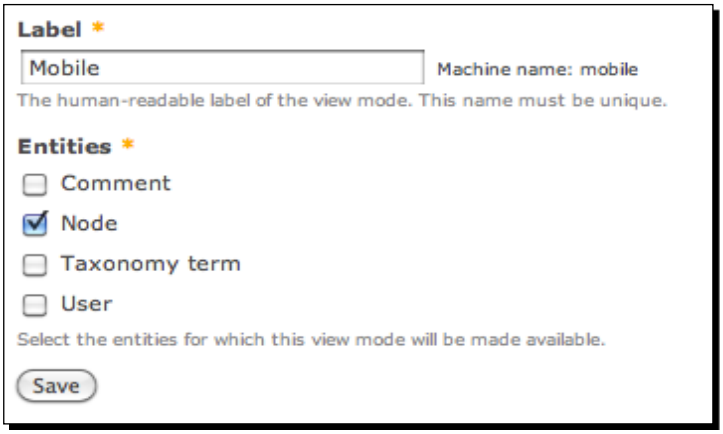
What Display Suite adds to Drupal is the ability to style nodes in multiple ways. The default build modes in Drupal are Full and Teaser. Display Suite allows an infinite number of build modes and allows the fields of the node to be styled and laid out in multiple ways in those build modes.

Right now, the billboards on the home page are displayed in the full build mode. We'll create a new build mode for the mobile site.

Time for action – creating a mobile-friendly home page

Let's turn the home page image into a block and have it show lower resolution images on the mobile home page:

1. Open a browser window and go to the main site. Make sure you're logged in.
2. The modules we're using—**Display suite**, **Views**, and **Views slideshow**—were listed in our Drush Make file and should have downloaded with our initial build. Let's verify that it's on. Navigate to **Structure** | **Modules** | **List**. Verify that all of the **Display Suite** modules are checked and turned on.
3. Go to **Structure | Display suite | View modes**. Choose **Add new view mode**. Call it **Mobile**. As shown in the following screenshot, select **Nodes** in the **Entities** section, so that it will be used on nodes only. Save this configuration:



4. Go to **Configuration** | **Image styles** | **Add a new style** and call it **mobile**.

5. You'll be presented with a screen where you can add adjustments to the image style (see the following screenshot). Add a new **Scale and crop** adjustment. Enter **480** in the **width** box and click on **Update style**:

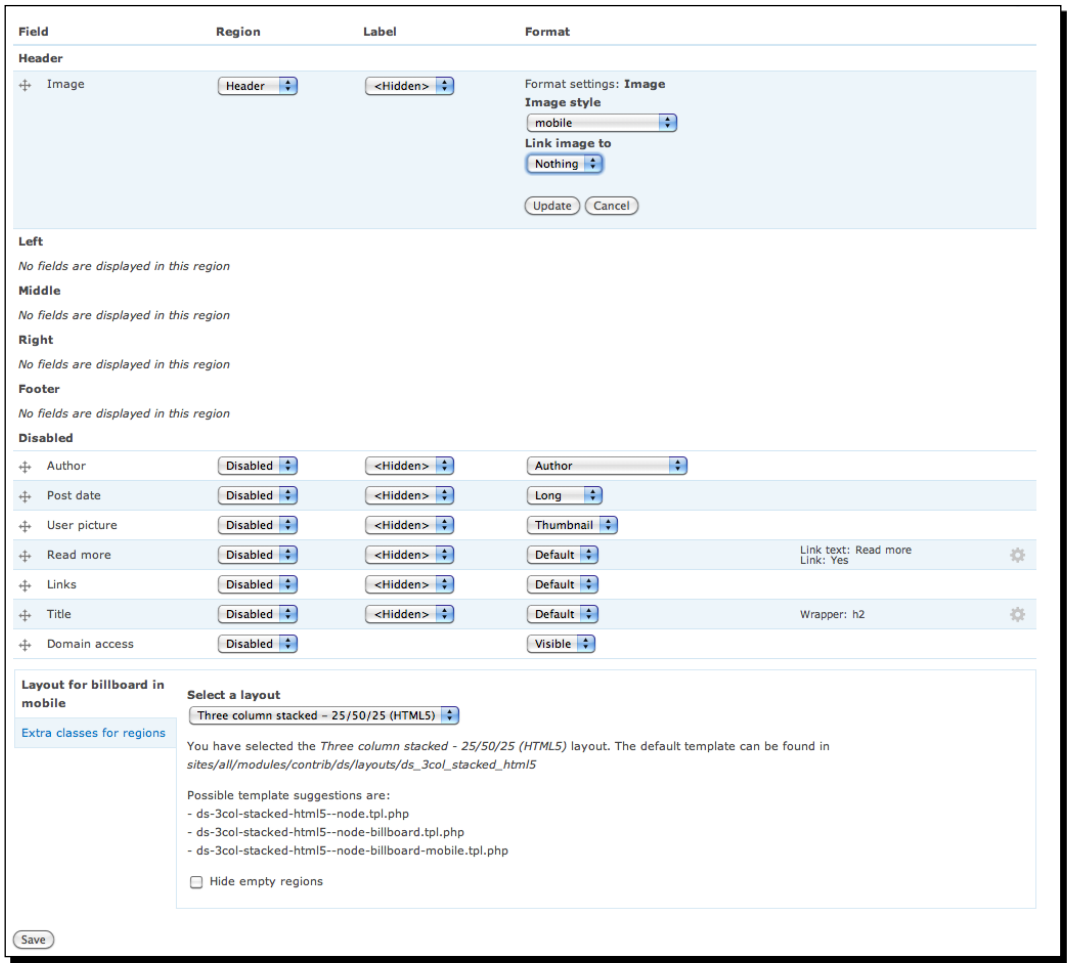
6. Go to **Structure | Display suite | Layout**. The front page content types were created by our client and should have been restored with the database you restored in *Chapter 2*, *Setting up a Local Development Environment*. The front page items are called **billboard** and under the node type, click on **Manage display** for the billboard.

7. At the bottom of the page, under **Custom display settings** | **Use custom display settings for the following view modes**, select **Mobile**, and then click on **Save**.

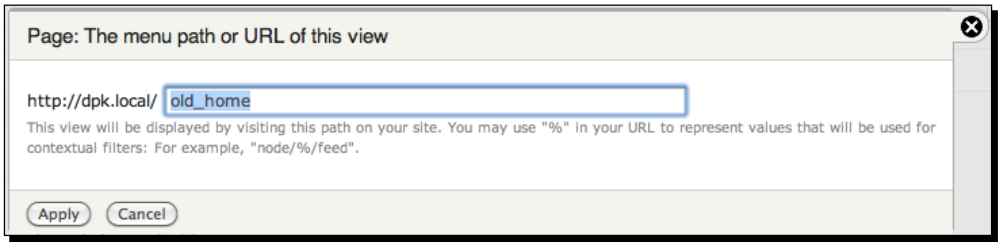
8. After the page saves, you'll see a new **Mobile** tab underneath the tabs. Click on **Mobile** to alter the node's mobile build mode.

9. At the bottom of the page, under **Select a layout**, choose **Three-column stacked - 20/50/25 (HTML5)**, and then click on **Save**. There should now be five regions—**Header**, **Left**, **Middle**, **Right**, and **Footer**.

10. Drag the **Image** item up to the **Header** region and select **mobile** under **Image style**. Save the settings.



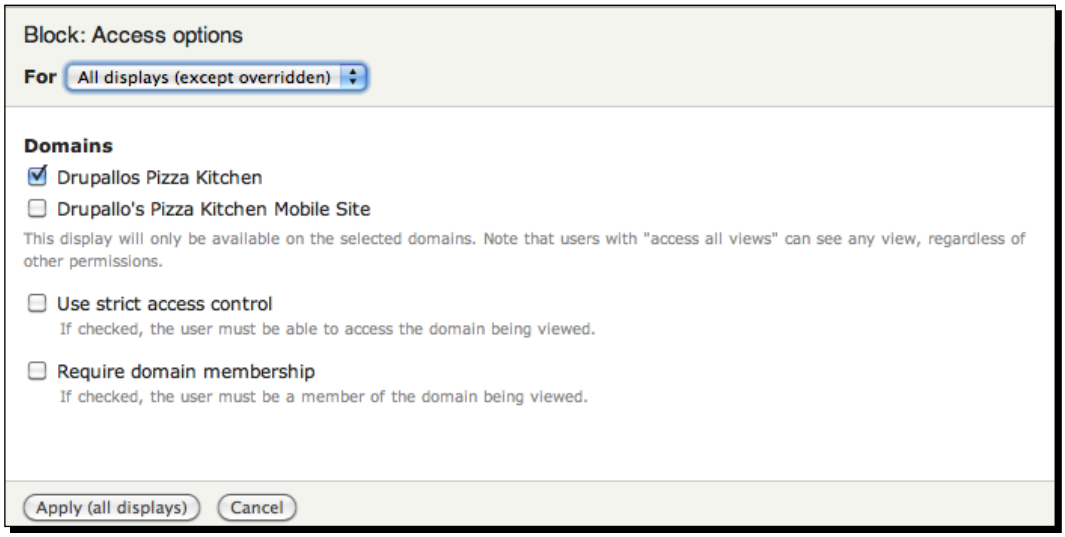
11. Go to **Structure | Views**. Edit the **Home | Page** view. The first display is the current home page. As shown in the following screenshot, change the URL to **old_home** to get it out of the way:



There should already be a "block" for this view. If not, create a default one.

12. Under the **Format: Slideshow** area, click on **settings**. Under **How should this block be styled?**, the first block will be the default settings for the view or **All displays (except where overridden)**. We do the same thing with the line underneath it: **Show: Display suite**. The **Display suite settings** mode should be **Full content**.

13. Notice the settings section titled **Access**. Change the access mode to **Domain** and choose the default or main **Drupallos Pizza Kitchen** domain. Both settings, again, should apply to all the displays (except for the overridden ones).

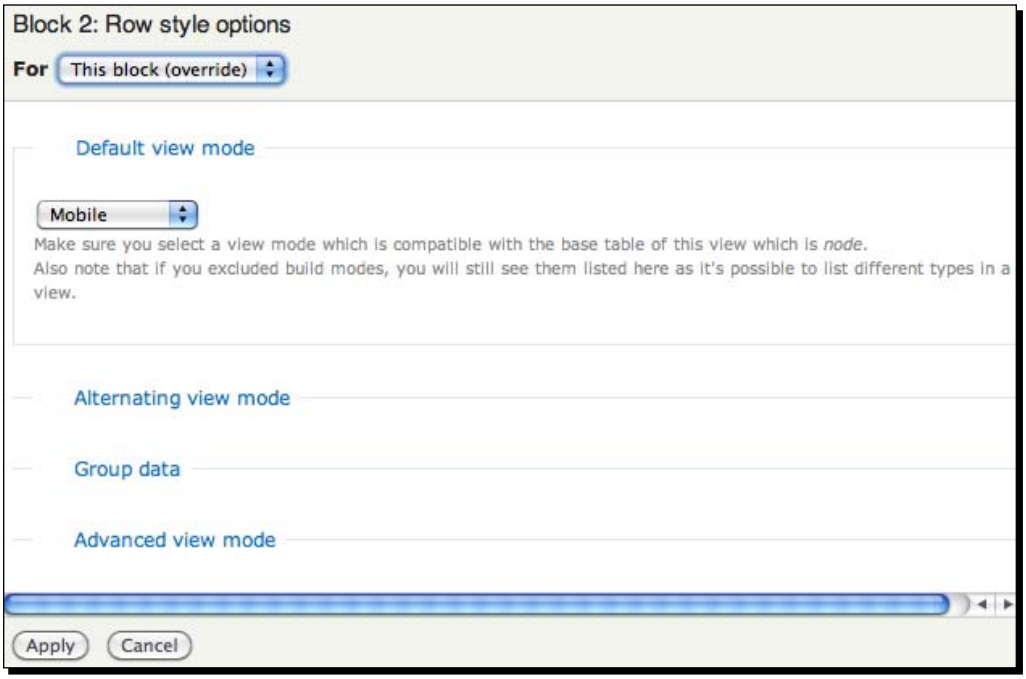


14. Create a second "block" display for this view (there should be two blocks and one page view).

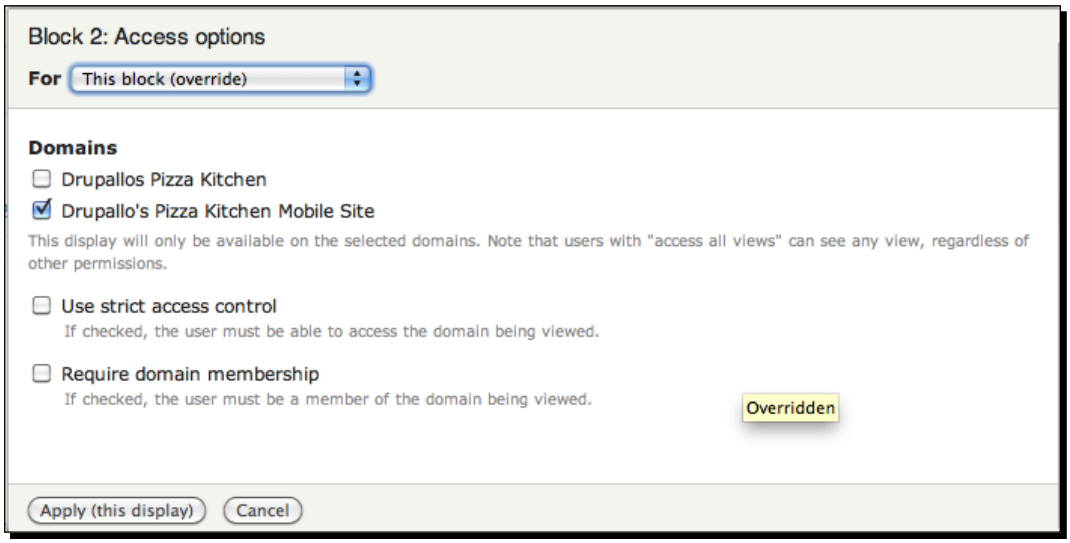
15. Click on **Add** | **Block**. When editing the new block, under **Format**, there will be a **Show: Display Suite** area.

16. Click on **Settings** and change the **Default view mode** to **Mobile**, under **How should this block be styled?**

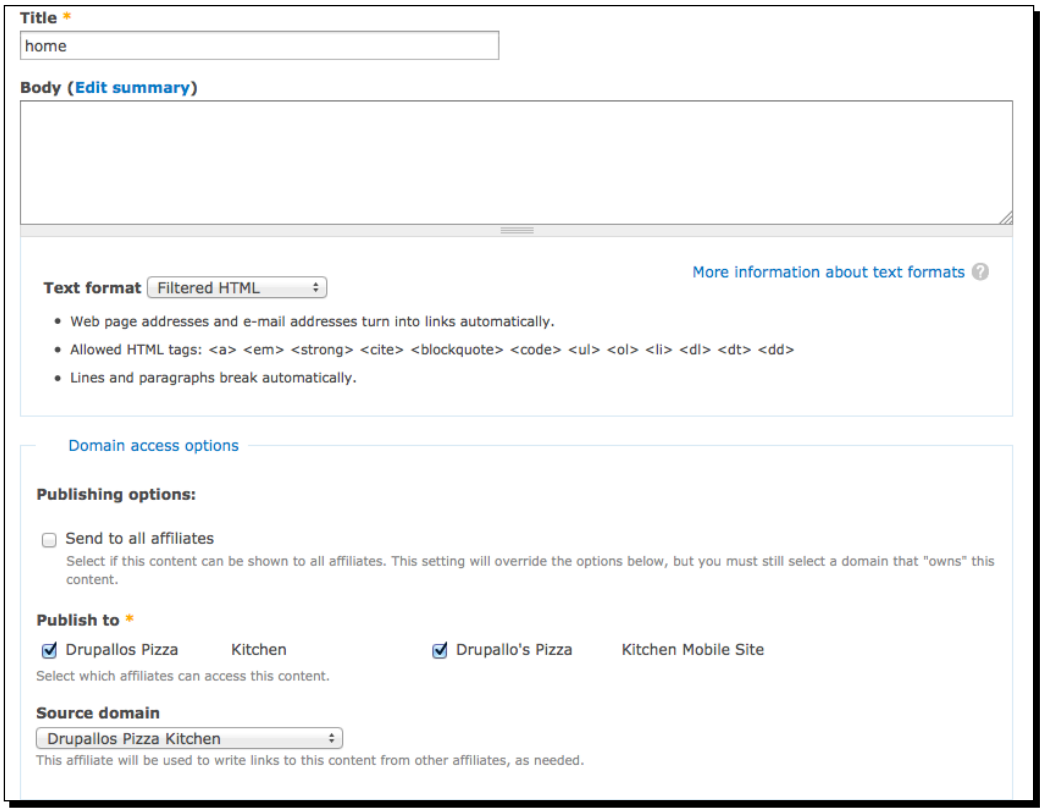
17. The second block will be **This block (override)**, under both **Format** and **Show** settings.



18. Notice the area titled **Access**. Access mode must already be **Domain** because that is the default mode for the view. Click on **Domains**. Set the **Access options** setting to **This block (override)**. Uncheck the primary domain and check the mobile domain, **Drupallo's Pizza Kitchen Mobile Site**. Click on **Apply (this display)** to save these settings.

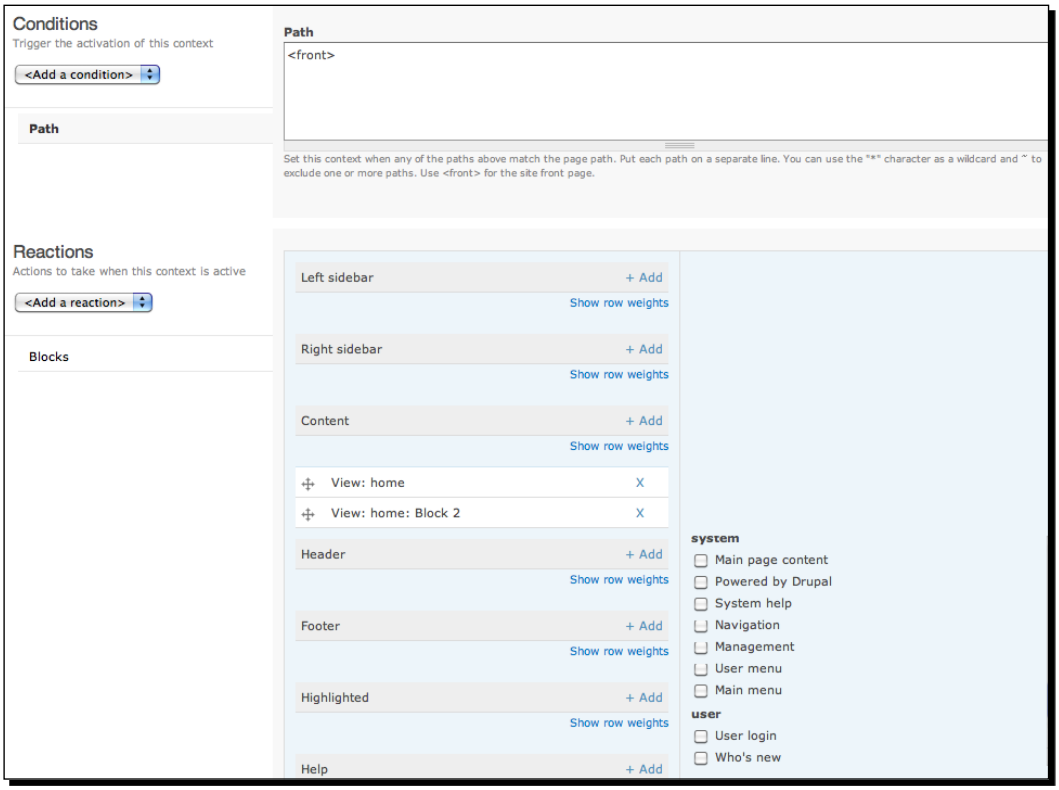


19. Go to **Content | Add content | Basic page**. As shown in the following screenshot, label it as **home**. Under the **Publish to** section, check that the node is published on both the primary and mobile sites. In the bottom tabs, make sure the URL alias is simply **home**. Save the new page.



20. Make sure that the **Context** module is installed and enabled. It should have been downloaded with your original `drush make` command, but check it again. You can do this by entering `drush pm-list | grep 'context'` from the command line or by going to the Drupal administration menu and choosing **Modules**. The **Context** module consists of three modules, **Context**, **Context Layouts**, and **Context UI**. Make sure all three are enabled.

21. Go to **Structure | Context** and then choose **Add** a new context. Name it something logical such as **Homepage**. Under the **Conditions** section, pull down the option **Path** and enter **<front>** in the resulting text area, as shown in the following screenshot. Under the **Reactions** section, choose **Blocks**. On the right-hand side of the resulting screen, a list of blocks will be displayed. Scroll to the bottom of the list and check our two view blocks we just created. In the middle column under **Content**, click on **Add**. Save the new context:



What just happened?

We first created a new build mode called **Mobile**. We used that build mode to show nodes in a more abbreviated form. Next, we created a new **Image style**. To use it with our new build mode, we chose to size down the image to 480 pixels, which is the maximum width of a standard phone screen.

We then changed the **Display Suite** settings for the billboard content type in its **Mobile** display mode to use the new image sizing.

We then moved the home page view to two new blocks: one for the full website, one for the mobile site and restricted access, based on the domain.

We added a blank page to serve as the home page for both sites. Contexts basically are made up of two parts: **Conditions** and **Reactions**. We can match this page with several different conditions. We can use the node ID or the URL path, or because this is the home page, we can use **<front>** as our matching path condition. We used this path to show the view blocks in the Content area.

Pushing changes from one environment to another

So now that we've made these changes on our machine, we need to get these changes to a staging server for the client to see. So that means we have to repeat them again on a second server, right? For most web developers, this is an unacceptable option. Programmers, by nature, are a lazy bunch and would never duplicate work. And by programmers, I mean mostly, me. I'm lazy, and any time I spend doing duplicate work is the time out of my life that I will never get back. Well, either that or we could export a database and restore the database on the new server!

There are many reasons why constantly updating the database on a live server from a staging or local environment is a really bad idea. The first of which is that, most live sites have people making changes on them that are happening in parallel with development. Let's say you had user groups and forums on your site. You can't update the database, as all the user group changes and postings would be lost. New users would have to re-register. For most websites, this is an unacceptable compromise, and in general, creates a poor user experience.

I usually take for granted that users know these things, and if I am going over something that is repetitive to you, I apologize, but there are newbie Drupal developers who will choose one of those two really bad options when, in fact, there is another way.

The Features module gives us a way to push these changes from one environment to another via code, rather than making the changes by hand or by database restoration.

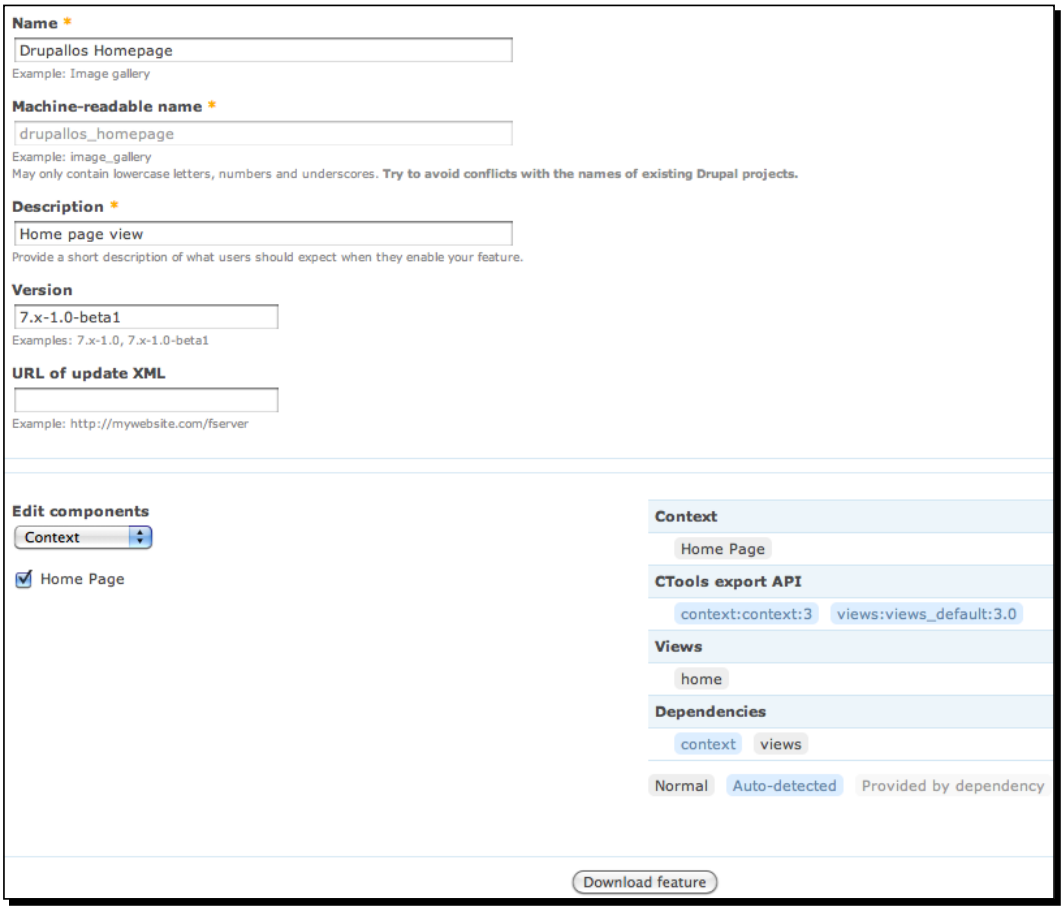
Time for action – updating the Home Page feature

Let's update the feature with changes to the current functionality. Add the new context functionality, download the updated feature, and take a look at the added code:

1. Open a terminal window and change the directory to the site root. Enter the following commands:

```
drush features-update -y drupallos_homepage
```

2. Open a browser window, choose **Structure**, and then **Features**. You should see the **Home Page** feature listed (now in its default state). Click on **Recreate**. Under **Edit components**, choose **Context**. You should see the **Home Page** context listed. Check the **Home Page** context, and it will be added to the bundle:



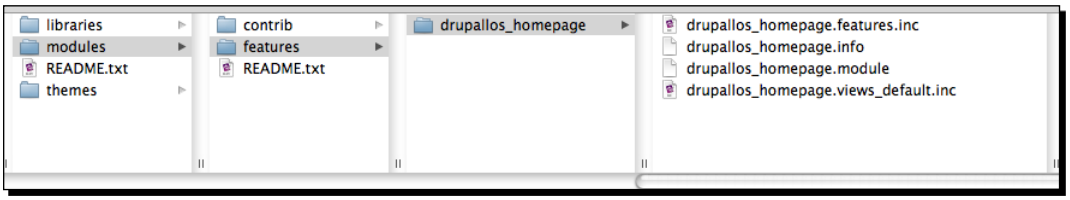
3. Click on **Download feature** and the feature will download to your browser's **Downloads** folder. Double-clicking on it should expand it (on a Windows machine, you may have to install some other software). Copy `drupallos_homepage.info` and `drupallos_homepage.context.inc` from the newly downloaded directory to the `sites/all/modules/features/drupallos_homepage` folder. It's okay to replace the current `.info` file with the newly downloaded one.

4. Open a browser window and log completely out of both the sites. You will see that the mobile site has the **Mobile** view and the full site has the **Hero** view.

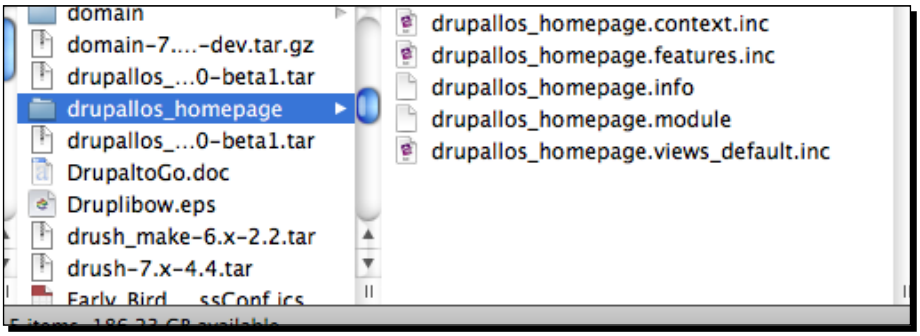
If you have not logged out, you will see both views on both sites. When you're logged in as administrator, you have the privileges, by default, to **See all views** and **See all domain content**. These administrator privileges override the domain access that we set up in this task.

What just happened?

Once we had altered the views accordingly, we updated the view's feature to save the new settings to code with the Drush command, `features-update`. We will later check the updated feature into version management. We then added the new context to the feature by recreating it. Notice the difference between the downloaded folder and the original feature. The following screenshot shows the original feature:



The following screenshot shows the new downloaded feature:



In the new downloaded feature, a file has been altered and a `.context.inc` file has been added. Let's take a look at the `.info` file. Notice that the highlighted lines in the following code snippet are different:

```
core = "7.x"
dependencies[] = "context"
dependencies[] = "views"
description = "Home page view"
features[context][] = "Home Page"
```

```
features[ctools][] = "context:context:3"
features[ctools][] = "views:views_default:3.0"
features[views_view][] = "home"
name = "Drupallos Homepage"
```

The new context has been added and a CTools setting for the context has been added to the `features[ctools]` property.

The context itself has been turned into a series of Drupal-style variables:

```
<?php
/**
 * @file
 * drupallos_homepage.context.inc
 */

/**
 * Implementation of hook_context_default_contexts().
 */
function drupallos_homepage_context_default_contexts() {
  $export = array();

  $context = new stdClass;
  $context->disabled = FALSE; /* Edit this to true to make a default
context disabled initially */
  $context->api_version = 3;
  $context->name = 'Home Page';
  $context->description = '';
  $context->tag = '';
  $context->conditions = array(
    'path' => array(
      'values' => array(
        '<front>' => '<front>',
      ),
    ),
  );
  $context->reactions = array(
    'block' => array(
      'blocks' => array(
        'views-home-block_1' => array(
          'module' => 'views',
          'delta' => 'home-block_1',
          'region' => 'content',
          'weight' => '-10',
        ),
```

```
          'views-home-block_2' => array(
            'module' => 'views',
            'delta' => 'home-block_2',
            'region' => 'content',
            'weight' => '-10',
          ),
        ),
      ),
    );
    $context->condition_mode = 0;
    $export['Home Page'] = $context;

    return $export;
  }
```

In the different settings we created with the GUI, the conditions and reactions have been saved to a standard configuration object and wrapped in a function that conforms to `hook_context_default_contexts`. More savvy programmers could use this as a template to create multiple contexts without actually using the Drupal UI.

Now, let's turn our attention to other areas of the site content that need to be "mobile-ized".

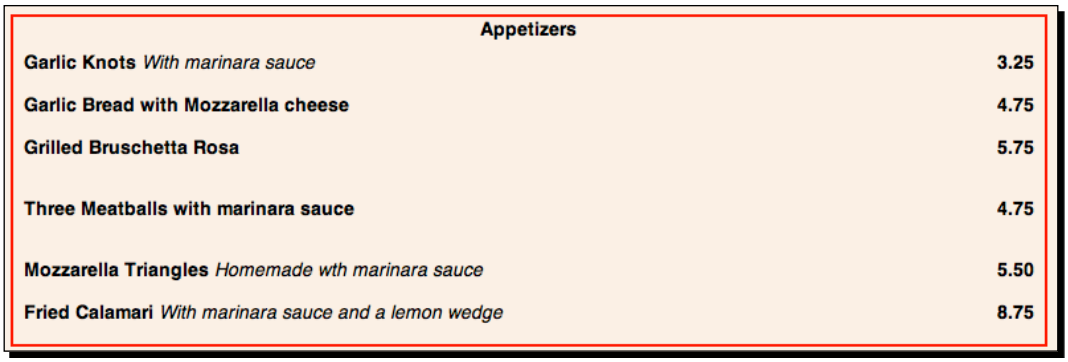
The menu

In this part of the chapter and from this moment on in the book, we will talk about the word "menu" in two different contexts.

In Drupal, **menu items** are snippits for navigation. The Menu module is a Drupal core module that allows you to create navigation structures for stringing together content in a hierarchical manner. In the restaurant world, a menu or course is a list of items the restaurant serves. I will do my best to make it clear to which menu I'm referring.

So many restaurants do it and, in their defense, it's the path of least resistance. Take their current menu, turn it into a PDF file, and upload it to the website. Boom! The world has a copy of the menu. But this can be problematic for mobile devices. Menu's PDF files, typically, are larger than 2 MB, which, on a mobile phone, will be a very slow download. Also, once the mobile phone downloads it, typically you're left with an 8.5 by 11 page squeezed into a 480 pixel by 320 pixel screen, not an optimal scenario for legibility by any stretch of the imagination.

If we're going to have a truly mobile version of the website, we need a menu that's managed by the CMS, so that the display on the full site will mimic the menu, and the display on the mobile site will be legible on the phone:



Appetizers	
Garlic Knots *With marinara sauce*	**3.25**
Garlic Bread with Mozzarella cheese	**4.75**
Grilled Bruschetta Rosa	**5.75**
Three Meatballs with marinara sauce	**4.75**
Mozzarella Triangles *Homemade wth marinara sauce*	**5.50**
Fried Calamari *With marinara sauce and a lemon wedge*	**8.75**

The Drupallo's carryout menu is divided into sections: Appetizers, Soup and Salad, Pizza, and Dinners and Specialties. We're going to want to keep those "sections" but develop a view to list them and control the order in which they are listed.

Drupal has always maintained two content type fields, namely, body and title. You are then free to add other fields with a widely-used module called the **Content Construction Kit** (**CCK**). With Drupal 7, CCK's name is changed to **Fields** and it became part of the Drupal core. If you are using Drupal 6, you will need to verify that CCK is installed and updated to the latest version. If you're using Drupal 7, please verify that the **Fields** module is enabled.

Time for action – creating the menu content types

We need to add some new content types to handle menu items and the different way the menu items are priced:

1. On the development site, after you are logged in as administrator, go to **Structure** | **Content types**. Select **Add new content type**.
2. Create new content types as shown below with the listed fields. Each new content type will have several "extra" fields provided, for example, **Domain Access** and **URL Path**. Ignore these for now.
3. After you've created the content type, you add the fields by clicking on **Manage fields** from the Content type list.

4. There are two ways to add fields: **Add new field** and **Add existing field**. Field names must be unique. If you need a field named the same thing on two different content types, you can choose **Add existing field**, which effectively re-uses the field from another content type.

   The following table displays the fields for the **Menu Item** content type:

Label	Name	Field	Widget
Title	**Title**	Node module element	
Body	**body**	Long text and summary	Text area with a summary
Price	**field_price**	Decimal	Text field

   The following table displays the fields for the **Pizza** content type:

Label	Name	Field	Widget
Title	**Title**	Node module element	
Body	**body**	Long text and summary	Text area with a summary
Slice	**field_price**	Decimal	Text field
14" Small	**field_small**	Decimal	Text field
16" Medium	**field_medium**	Decimal	Text field
18" Large	**field_large**	Decimal	Text field

   The following table displays the fields for the **Soup** content type:

Label	Name	Field	Widget
Title	**Title**	Node module element	
Body	**body**	Long text and summary	Text area with a summary
Cup	**field_cup**	Decimal	Text field
Bowl	**field_bowl**	Decimal	Text field

The following table displays the fields for the **Sandwich** content type:

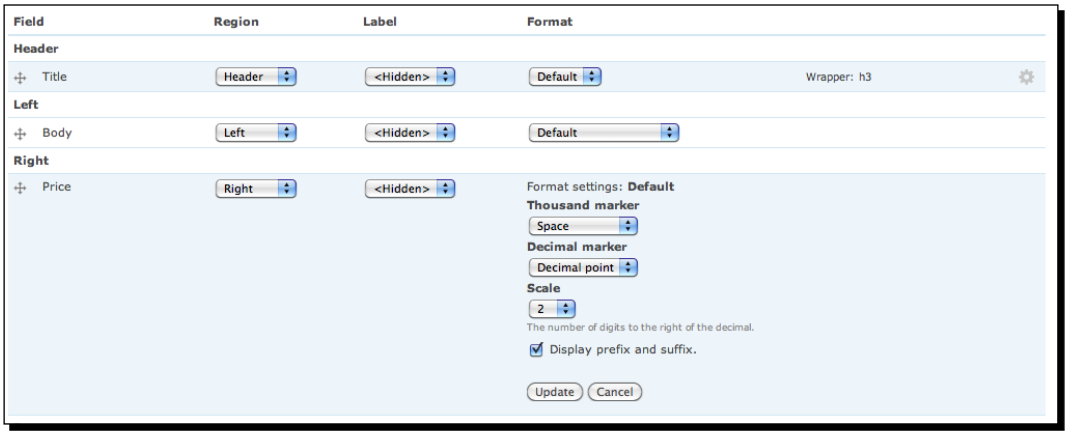
Label	Name	Field	Widget
Title	**Title**	Node module element	
Body	**body**	Long text and summary	Text area with a summary
Regular	**field_regular**	Decimal	Text field
Foot Long	**field_foot_long**	Decimal	Text field

5. Each content type, other than the **Menu item group**, has a title, body, and pricing fields. After you've added the fields, choose **Structure | Content types | Manage display**.

6. At the bottom of the page, there's a tab called **Layout for content type in default**. Change that value to **Two column stacked**. Save the page settings. At the top of the page, you should now see four regions, namely, **Header**, **Left**, **Right**, and **Footer**.

7. For each of the above content types, put **Title** in the **Header** region, **Body** in the **Left** region, and **Price** fields in the **Right** region.

8. At the end of the row is a "gear" icon. Click the gear icon to reveal formatting options for the **Price** field. If there are multiple pricing fields (for example, **Sandwich**, **Pizza**, and so on) put them all in the **Right** region. You can control the vertical order by dragging the right-hand handle. The **Price** field's vertical order should be the same as the preceding table:

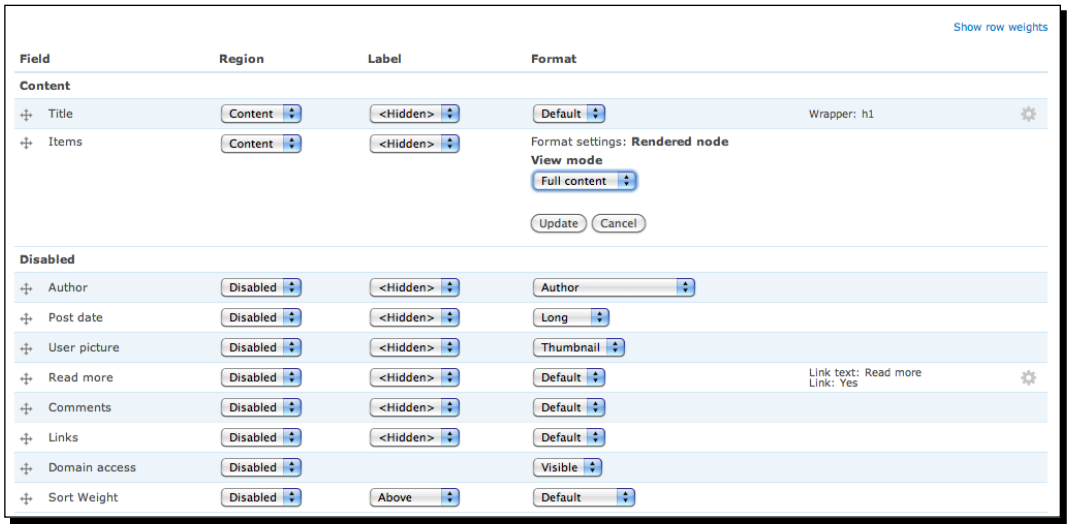


9. Create the **Menu item group** content type, as you did earlier. Notice that the **Items** field is a node reference field, not a text field:

The following table displays the fields for the **Menu item group** content type:

Label	Name	Field	Widget
Title	**Title**	Node module element	
Body	Not used	REMOVE	REMOVE
Items	**field_items**	Node reference	Autocomplete text field
Sort Weight	**field_sort_weight**	Integer	Text field

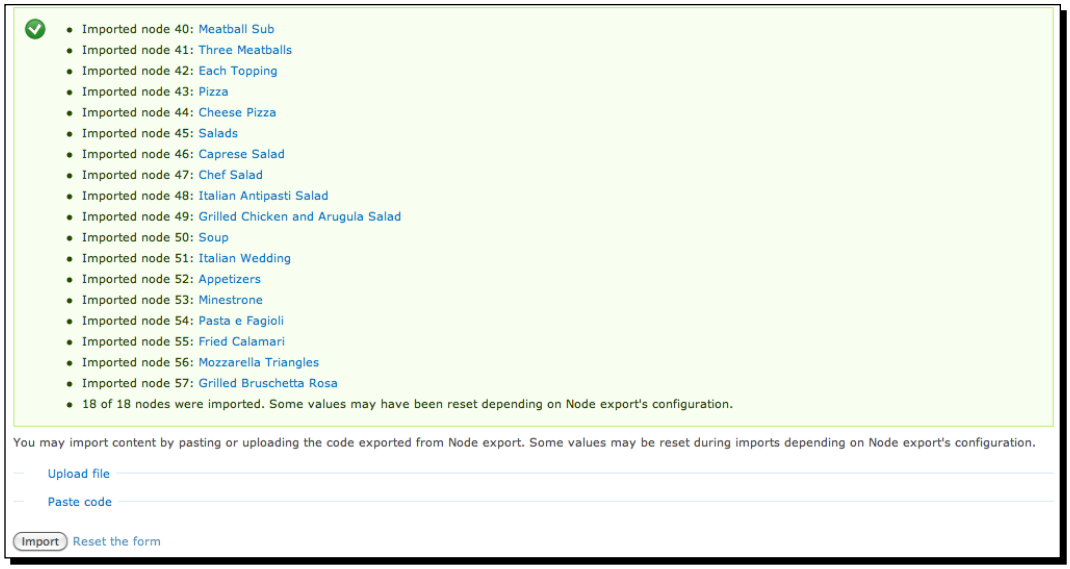
10. At the bottom of the page, there's a tab called **Layout for menu item in full**. Change that value to **One column**. Save the page settings. At the top of the page, you should now see one region, **Content**. Move **Title** and **Items** up to the **Content** region. Change the items' display options to **Rendered node** and in the view mode, select **Full content**:



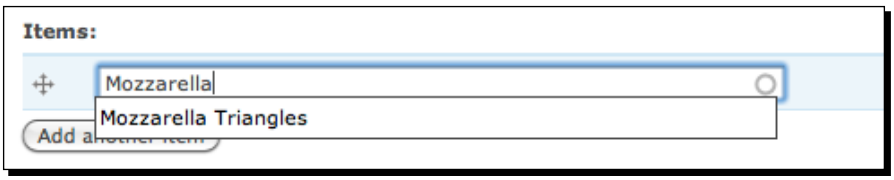
11. To import the content via the command line, open a terminal window, change the directory to your site root, and enter the following commands:

```
drush dl uuid node_export
drush pm-enable uuid node_export
drush node-export-import --file=sites/default/content_imports/
chapter5.export
```

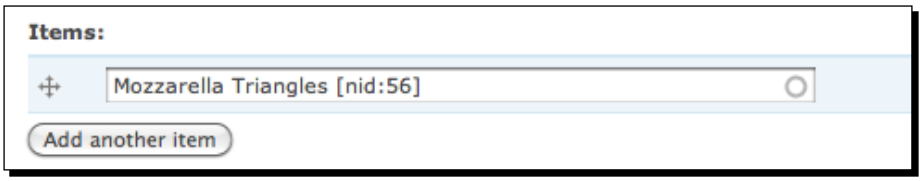
To import the content via Drupal GUI, go to `drupal.org` and download the `node_export` (`http://drupal.org/project/node_export`) and `uuid` (`http://drupal.org/project/uuid`) modules. Copy the folders to `sites/all/modules/contrib`. On the development site, when logged in as admin, navigate to the **Modules** list. Check the box for both **Node_export** and **UUID**. Save the module settings so that the modules are enabled. Go to **Content** | **Add content** | **Node export: Import**. Then upload the file. I've created the `.export` file in `sites/default/content_imports/menu.export`. Choose that file. Then click on **Import**:



12. Go to **Content** | **Add content** | **Menu item group**. Entitle this content object **Appetizers** and under **Items**, begin typing **Mozzarella**. The node reference will suggest the node named **Mozzarella Triangles**. Choose this suggestion by clicking on it, as shown in the following screenshot:



13. The field will now show the node name and the node ID, as shown in the following screenshot. Click on **Add another item** and add the **Bruschetta** and **Calamari** items:



14. Create a **Salads** group. Add the **Italian Antipasti**, **Grilled Chicken and Arugula Salad**, **Chef Salad**, and **Caprese Salad** items.

15. Create a **Pizza** group and add the **Cheese** and **Each Topping** nodes.

16. Create a **Soup** group and add the **Italian Wedding**, **Pasta E Fagioli**, and **Minestrone** items.

17. Choose **Structure** | **Views** | **Add new view**. Label the view **Menu** and the menu title and path should be simply **menu**. Click on **Continue & Edit**.

18. Add the **Filter** criteria as **Content published = yes** and **Content type** = **Menu item group**. Add the **Sort criteria**: **Content**: **Sort Weight**.

19. Add a normal menu item. Call it **Menu**. Save the view.

What just happened?

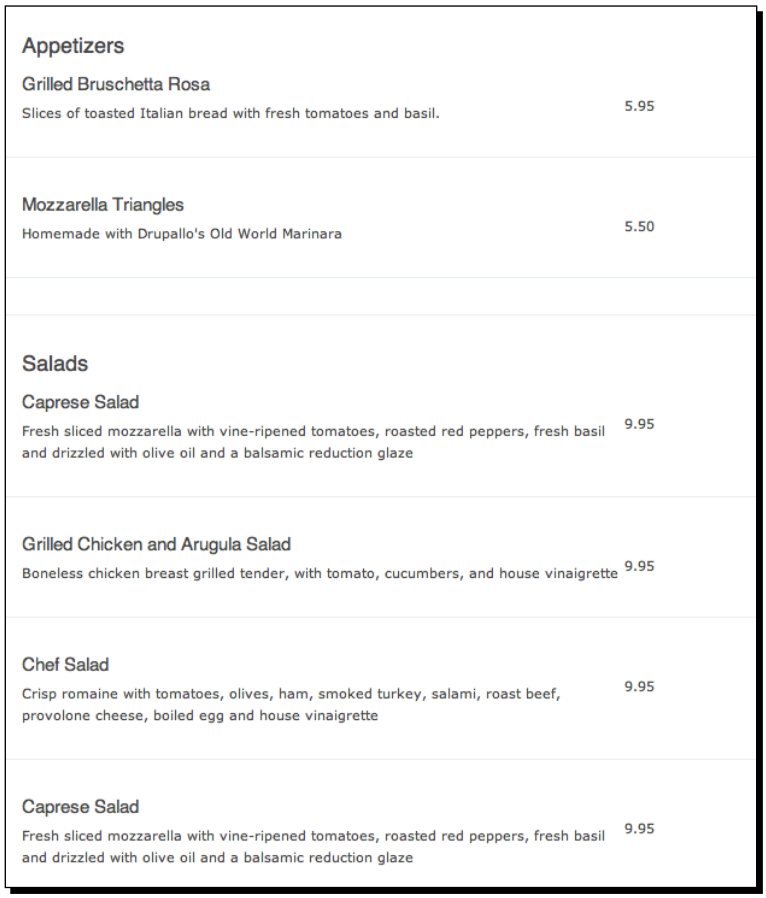
First, we created content types for all the different menu items. The differences in menu items were primarily the pricing structure. Pizza is sold by the slice (14", 16", and 18" pie), whereas soup is sold by the cup and bowl. Appetizers and salads have a single price, whereas sandwiches have a 6" and foot long variety. We also created an organizing content object called the **Menu item group**. The **Menu item group** type has a special kind of data field called **node reference**. Node references are, simply put, pointers to other nodes.

We then created the Display suite builds for each of the new content objects. When we created a build for the group content type, we specified that the node references be displayed in their default Full build mode. When Display Suite puts together the node, it will grab all of those related nodes and display them in the order they are referenced.

We then created a view that returned group objects and sorted them based on their weight.

Display suite is a powerful set of tools to control how nodes are displayed. We'll explain it in more detail in *Chapter 10*, *Tabula Rasa: Nurturing your Site for Tablets* where we will learn how to use it to optimize your mobile site.

If you now go to `http://dpk.local/menu`, you should see a reasonably-nice, well-laid out representation of the menu.



Appetizers

Grilled Bruschetta Rosa
Slices of toasted Italian bread with fresh tomatoes and basil. 5.95

Mozzarella Triangles
Homemade with Drupallo's Old World Marinara 5.50

Salads

Caprese Salad
Fresh sliced mozzarella with vine-ripened tomatoes, roasted red peppers, fresh basil and drizzled with olive oil and a balsamic reduction glaze 9.95

Grilled Chicken and Arugula Salad
Boneless chicken breast grilled tender, with tomato, cucumbers, and house vinaigrette 9.95

Chef Salad
Crisp romaine with tomatoes, olives, ham, smoked turkey, salami, roast beef, provolone cheese, boiled egg and house vinaigrette 9.95

Caprese Salad
Fresh sliced mozzarella with vine-ripened tomatoes, roasted red peppers, fresh basil and drizzled with olive oil and a balsamic reduction glaze 9.95

Bundling up the changes

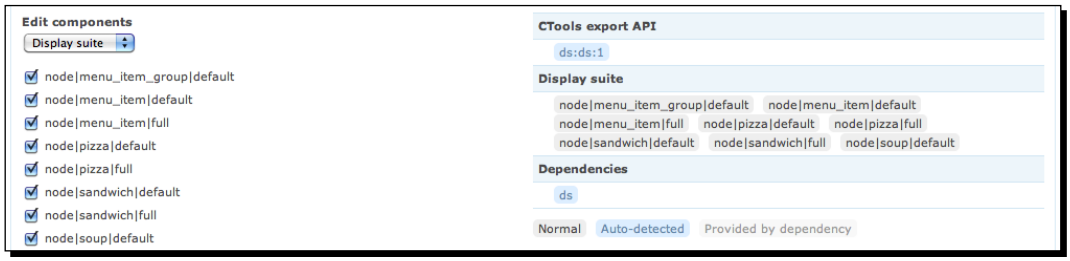
In order to make these changes, we need to bundle them up into the `Feature` package and check them into version management.

Time for action – bundling the changes into a package

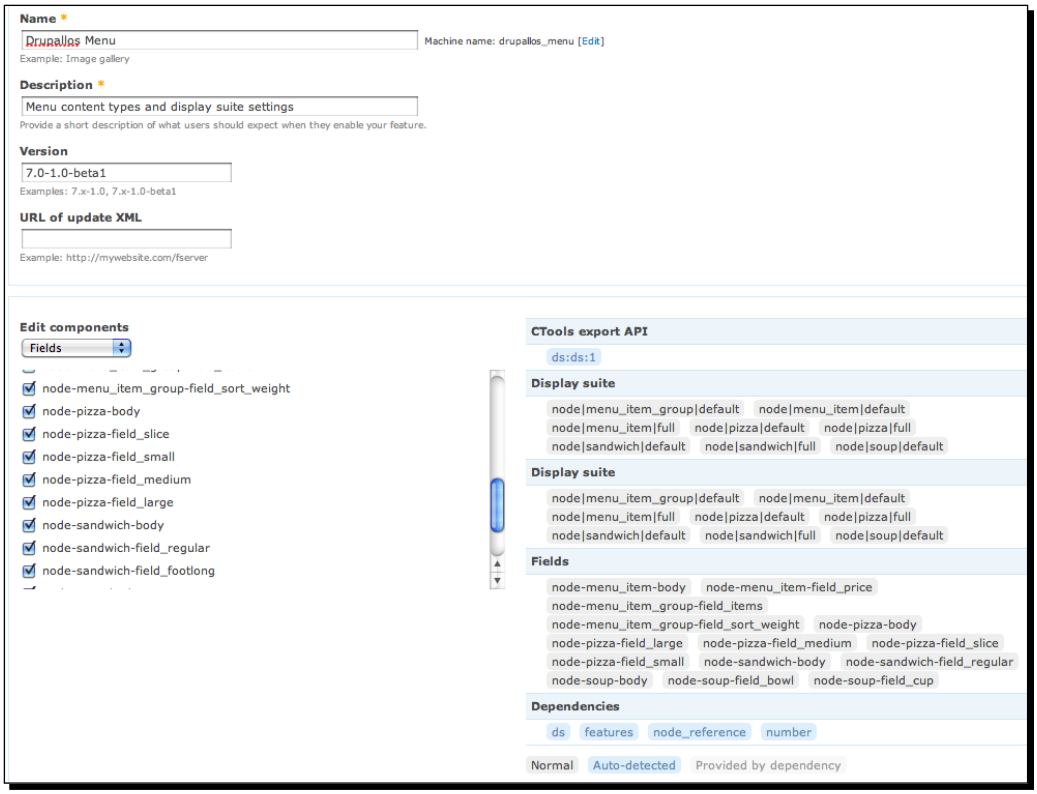
Once the changes have been committed to code, they can be checked into an SCM and version managed. As you evolve the site, you have the versions of the code going backward and can always revert if the site takes a wrong direction:

1. Go to **Structure** | **Features** | **Create feature**. Name the new feature **Drupallos Menu** and put a description, such as **Menu content types and display suite settings**. The version number should be something similar to **7.x-1.0-beta1**.

2. Under **Edit components**, choose the first **Display suite** option. Check all the checkboxes for the node's **Display suite** settings to include them in the **Features** package:

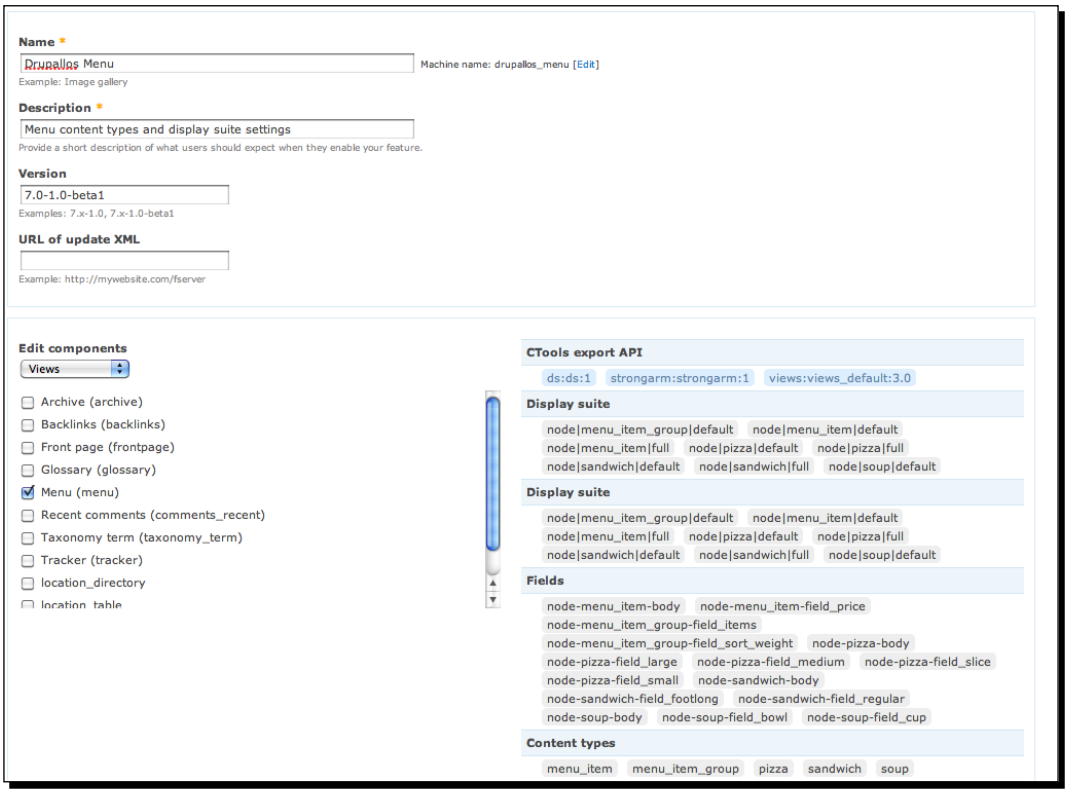


3. Now, select the **Fields** option. Check all of the fields you just created for the new menu content types:

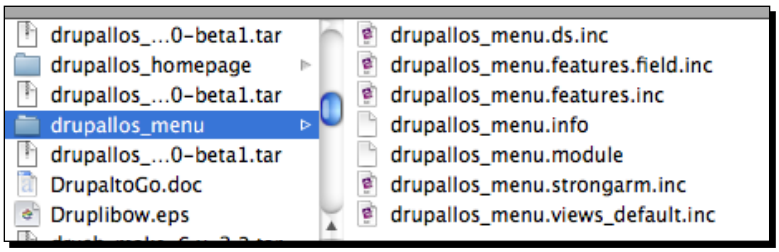


4. Select **Content types** and select the newly-created content types relating to the **Menu** page.

5. Choose **Views** and check the checkbox for the new **Menu** view:



6. Click on **Download feature** to download the newly-created feature package. Once downloaded, copy it to `sites/all/modules/features`:



7. Open a terminal window and change the directory to your site root. Enter the following commands:

```
git add sites/all/modules/features/drupallos*
git commit -m 'menu and home page features'
```

What just happened?

When the `features` module creates a new bundle, it looks at all of the entities available for it to export with various modules, groups them by module, and provides a nice selection interface. As we select items for the `features` module to export, it adds them to a payload list. When the mechanics of the export is actually done, `features` iterates through this payload and uses the `source` module's CTools interface to export these items to files in a private directory. `features` then tars up the file and sends it to the user's browser. It then downloads this file to your desktop or **Downloads** folder.

We now have all of our settings and views rolled into a nice version-managed package. We add it to the source code tree in `sites/all/modules/features`. Once there, we add it to the list of items in Source Code Management with the GIT `add` command and then commit our changes to the repository.

Have a go hero – adding the Print CSS file

Add a Print CSS file that allows the **Menu** page to be printed correctly on an 8.5 X 11 standard letter page.

Pop quiz

1. When there's a `<div>` tag with an `<image>` inside it, what is the behavior of the `<image>` tag if that `<div>` tag's CSS or style tags are set to `display:none`?
    a. It does not download and it does not display
    b. It downloads but does not display
    c. It downloads and displays
    d. It waits for JavaScript to load to display
2. Drupal's default build modes for content nodes include:
    a. Block
    b. Teaser
    c. Left and right
    d. Full
    e. Both b and d

3. The **Display suite** module does the following:
    a. Changes the `<div>` tags with the default layout classes around fields based on content nodes
    b. Turns blocks into nodes and vice versa
    c. Gives node displays unlimited build modes to vary the display based on context
    d. Both a and c
    e. All of the above
    f. None of the above
4. The **Context** module does the following:
    a. Changes nodes into blocks and vice versa
    b. Consists of **Conditions** and **Reactions** to those conditions
    c. Allows you to add blocks to pages outside of the standard block system
    d. Renders blocks on the fly rather than rendering every block with every page load
    e. b, c, and d
5. The **Features** module does the following:
    a. bundles functionality into a downloadable module
    b. renames content types
    c. themes content types
    d. none of the above

Summary

First, we moved the view providing the current home page and replaced it with a blank generic page. We created a block for each of the mobile and desktop sites, and changed the view access settings to the proper home page block, shown on the proper website. We then updated the feature for the home page and added it to version management so we can move the changes to the live site.

Then, we moved the menu from a 2 MB PDF to a page with text, so that our users who don't have the ability to read PDF files will know what good stuff Drupallos offers on its menu. We then bundled the menu functionality into a feature and version managed the files.

In the next chapter, we'll go into further detail on planning a customized unified theme that will work on all browsers, both mobile and desktop.

6
The Elephant in the Room: Audio, Video, and Flash Media

From the beginning of the web, it was the Holy Grail. The ability to show and distribute video over the web has been sought after, achieved, and re-engineered multiple times.

In this chapter, we will:

- Take a look at the current state of video embedding
- Suggest strategies for managing video with Drupal
- Introduce the **Media** module for Drupal 7
- Take a look at the challenges when viewing different media on mobile devices
- Walk through some Drupal strategies for embedding a mobile-friendly video
- Learn to create mobile-friendly charts

But first, let's first look back at how we got to where we are now.

Flash and iOS

Whenever you deal with iOS plusses and minuses, there's one glaring omission that isn't going to change any time in the near future. It's the lack of support for Adobe Flash. It is the proverbial "elephant in the room" of every Apple Store.

Apple's original statement about Flash support wasn't that they were anti-Flash, but they were pro open standards. Their argument was that with HTML5, time-based animation and line-and-circle pen drawing was now possible with CSS, SVG, and the Canvas element. You really didn't "need" Flash to get the effect you wanted with a website. That position has pivoted and become a Holy War against Flash media on handhelds, the main reason being battery life.

In some tests, you can get as much as 25 percent more battery life out of your laptop computer by completely removing or disabling Adobe Flash in your web browsers. It is a battery hog. Adobe has begun to address that and, to their credit, they have released a "lite" version of the Flash plugin for the Android operating system, but that "lite" version can still handle all video and audio on handhelds. The "lite" version of Flash also has some issues with faster frame animations in Flash-based video games.

Then in late 2011, Adobe relented. They as much as admitted that Flash was never going to be what they wanted it to be on mobile devices and stopped development of the mobile version of Flash for Android. Shortly thereafter, Microsoft stated that their new Live Tiles technology, the main technological push behind the next version of Windows, would not, in it's native state, support any browser plugins. The Live Tiles would be built in pure HTML5 and CSS3.

And although, technically speaking, you don't "need" Flash on an iOS device to create the project you want, there's still a ton of legacy Flash development that hasn't yet been ported to HTML5.

Fortunately, though, one-by-one, online video providers are adjusting their code for mobile to allow viewing of their content on both tablets and handhelds. The first, and most significant convert, was Google and, by extension, YouTube.

Incorporating video into your web content

Big Jimmy has made the decision to run some advertising on local cable to help drum up some more business on weekends. In addition, the business has decided to sponsor a young girls' soccer team. For these reasons, we've been asked to help Little Jimmy come up with a plan to host audio and video on the site. With all the work we've done on the mobile site, there's no reason we should add the videos without having a plan for viewing these videos on mobile devices.

Whenever you host a video, the first question to ask is do you want to host it yourself or use a service such as YouTube or Vimeo. Unless there's a specific reason not to, the easiest path to embedding a video on your website is via YouTube.

Most people can't conceive of life before YouTube, but 15 years ago, web video was a competing maze of incompatible formats and competing vendors. RealMedia was one of the pioneers of web video. Microsoft and Apple's QuickTime had competing formats. Then came YouTube.

YouTube made web video accessible to the masses by transcoding uploads and making them playable by the Flash media plugin. Flash came pre-installed on many computers and the website over about a two-year span built up a steady user base and was bought by Google in 2006. Under Google's umbrella, the site has been both a creative and destructive force, beginning and ending careers, businesses, institutions, and in some cases, governments. It has, in many ways, changed the world.

But there are limits to the videos you can upload. YouTube has very strict content rules and a limit of 10 minutes for a video's length. For that reason, many people use a paid video hosting service such as Vimeo. There are Drupal `contrib` modules for hosting a video on any one of 50+ services with a module called **Media** for Drupal 7, and it's called **Embedded Media** in Drupal 6. They both work basically the same way. They allow you to create a content type that has a field pointer to the media in question and then they write embedded tags for that media.

As you may or may not know, most iOS mobile devices do not support Flash embedding, so getting a video to appear correctly can be troublesome. That's where the MediaElement modules come into play. MediaElement will write HTML5-compatible tags for your media.

Time for action – embedding media files

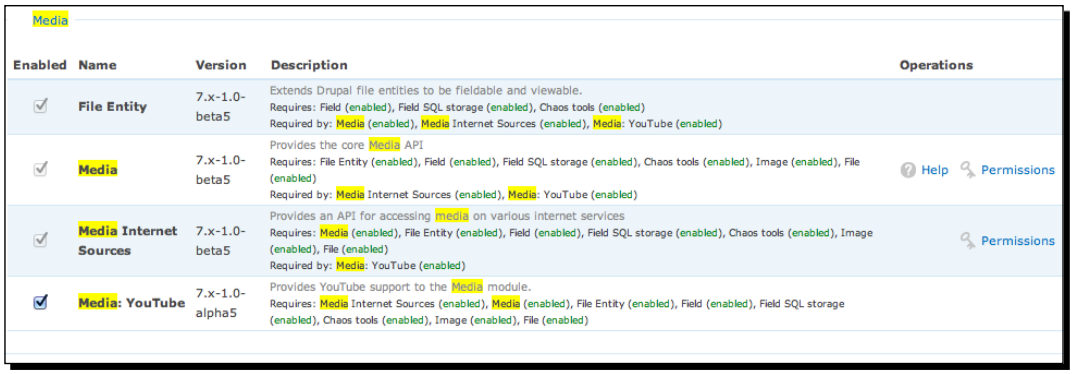
In order to embed media in Drupal, we need to add a few libraries and a module or two. Let's set ourselves up.

1. Enter the following commands in a terminal window:

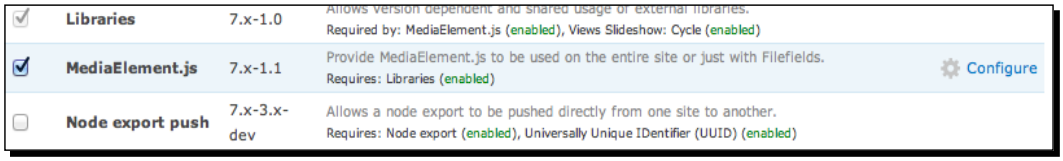
```
cd ~/Sites/dpk/sites/all/libraries
git clone git://github.com/johndyer/mediaelement.git
drush dl media media_youtube mediaelement
drush pm-enable media media_youtube media_internet mediaelement
```

2. Alternatively, you can download them by hand and install them into the `sites/all/modules/contrib` folder of Drupal 7 and download the `mediaelement` library to your `sites/all/libraries` folder.

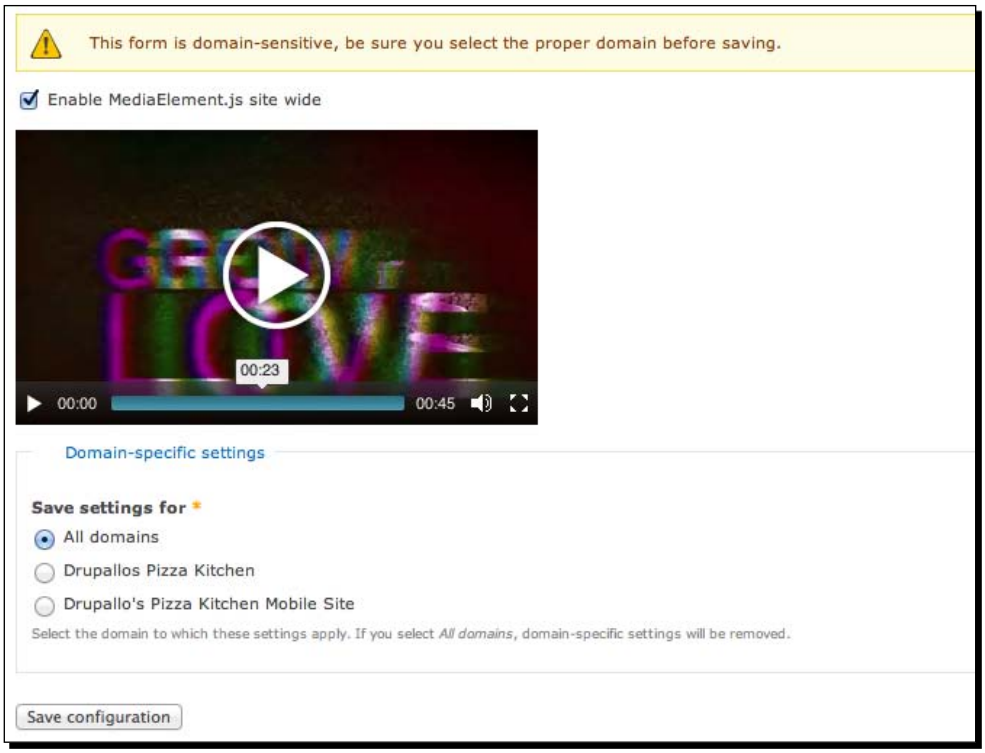
3. Go to the `modules` folder and verify whether all the Media modules are enabled. As shown in the following screenshot, **Media**, **Media Internet Sources**, and **Media: YouTube** are listed in their own group, **Media**:



As shown in the following screenshot, the **MediaElement.js** module is listed under **Other**. If you have other modules turned off, it may request that the dependencies be turned on:

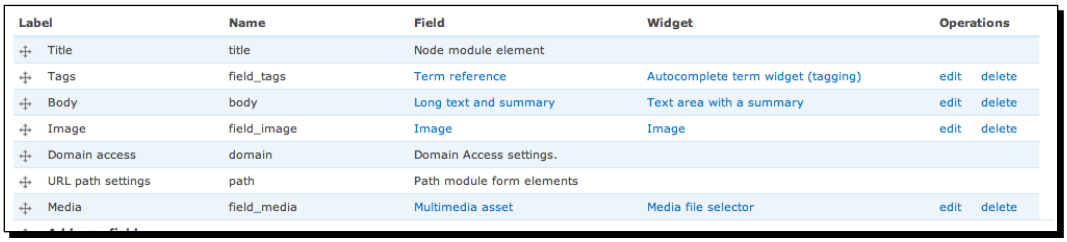


4. Now, navigate to **Structure** | **Configuration** | **MediaElement.js**. Check the checkbox labeled as **Enable MediaElement.js site wide** and change the **Domain-specific settings** to **All domains**. Save the changes by clicking on the **Save configuration** button.



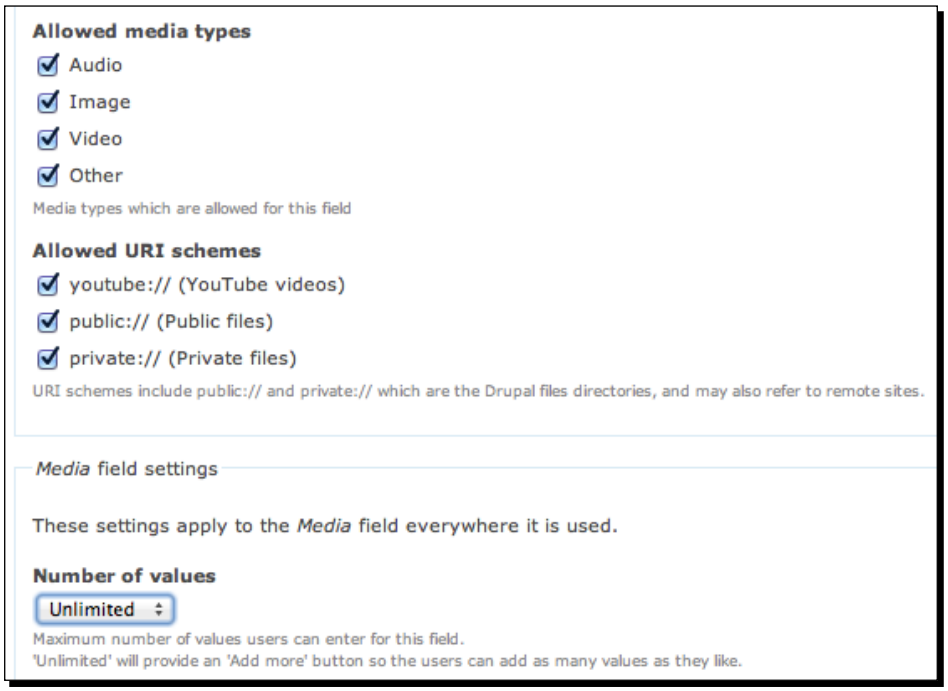
5. Navigate to **Structure** | **Content Types** and edit the **Article** content type.

6. As shown in the following screenshot, create a new field **Media** of the type **Multimedia asset**:



Label	Name	Field	Widget	Operations
Title	title	Node module element		
Tags	field_tags	Term reference	Autocomplete term widget (tagging)	edit delete
Body	body	Long text and summary	Text area with a summary	edit delete
Image	field_image	Image	Image	edit delete
Domain access	domain	Domain Access settings.		
URL path settings	path	Path module form elements		
Media	field_media	Multimedia asset	Media file selector	edit delete

7. Check all the options for **Allowed URI schemes** and **Allowed Media Types**, as shown in the following screenshot, and save these settings:



What just happened?

In the first three steps, we downloaded and installed the `media`, `media_internet`, `media_youtube`, and `mediaelements` modules. These modules are necessary for the Fields API to accept online-embedded media. We ensure that they're enabled and we configure **MediaElement.js** to use the media elements embedded script sitewide on both our mobile and desktop sites. This will ensure we get consistent HTML5 embedding for browsers that can handle it.

Incidentally, there's a series of modules called **Embedded media for CCK** that will do the same thing for Drupal 6 and, in fact, the way these modules work was taken from the embedded media suite.

We added one of the new media fields to our standard **Article** content type. We allowed the new fields to display audio and video, but we could restrict the fields from displaying either of these two media, using the **Allowed media type** checkboxes in the **Media** field configuration page.

We created a new node with a media embed. The way this works, Drupal downloads the entire page to the `sites/default/files` area of the website and uses a regular expression match to pull the embedded tags of the clips out of YouTube's download page. If you're having trouble embedding media in that step, check to make sure your permissions are correct for Drupal to be able to read and write files to that area.

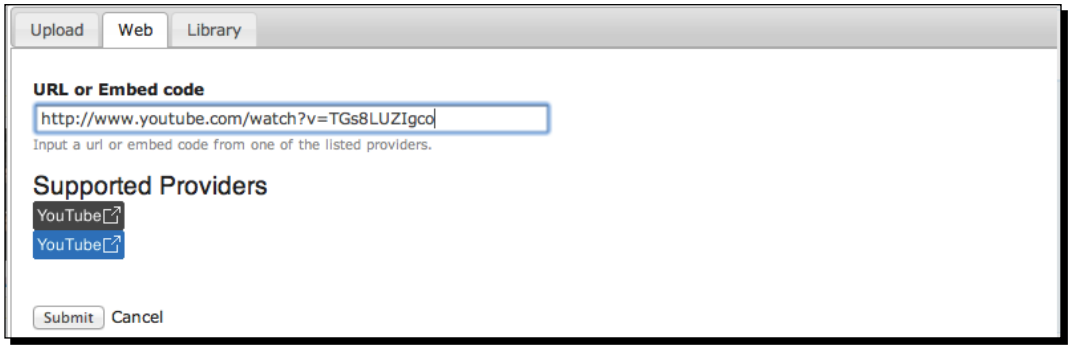
Time for action – adding content

1. Navigate to **Content** | **Add Content** | **Article**.
2. Create an article with the values in the following table:

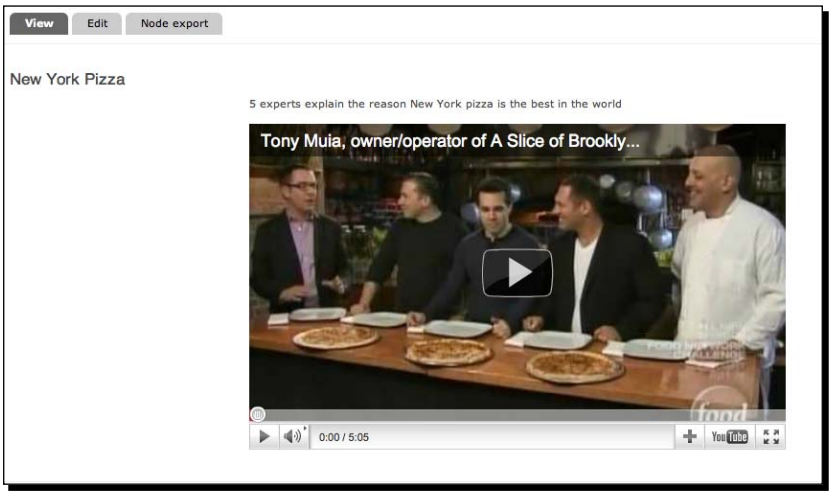
Heading	Value
Title	New York Pizza
Body	5 experts explain the reason New York pizza is the best in the world.

3. Click on **Select Media** and then click on the **Web** tab. Paste the following URL into the field labeled as **URL or Embed code** and then click on **Submit.** If the video at the following URL is not available, any YouTube video that can be embedded will do:

   `http://www.youtube.com/watch?v=TGs8LUZIgco`



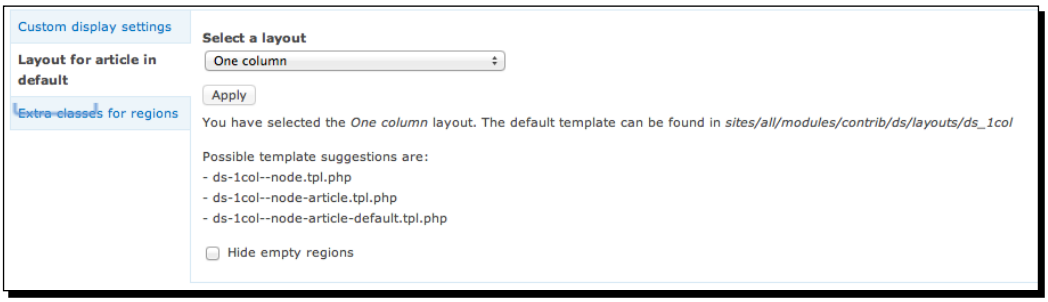
4. If all goes well, you should see a thumbnail of the clip, as shown in the following screenshot:



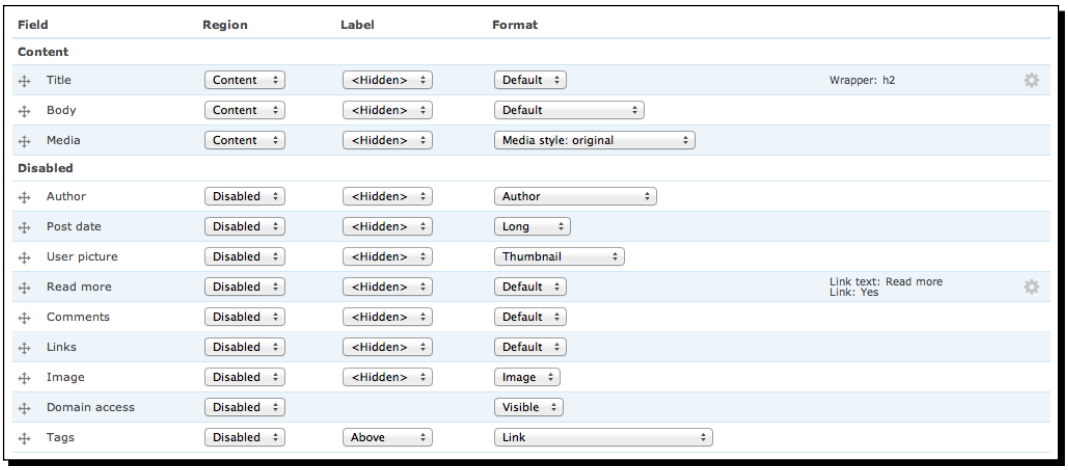
5. Save the new node.

6. Navigate to **Structure** | **Content Types** | **Article** | **Manage Display.**

7. At the bottom of the page, click the **Layout for article in default** tab. Choose **One column** and click on **Apply**.

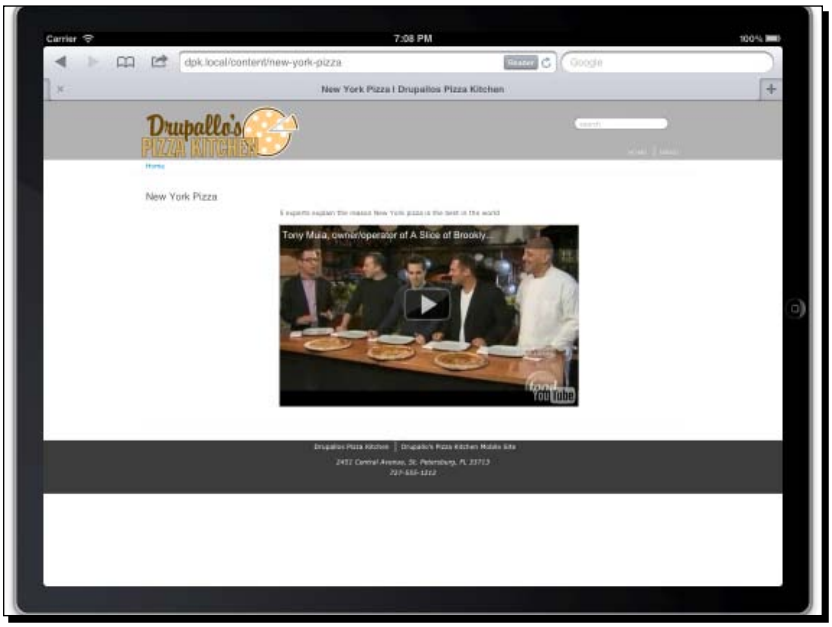


8. Drag the **Title**, **Body**, and **Media** fields into the **Content** area and make their labels hidden, by selecting **<Hidden>** under the **Label** column. Save the new layout. In the **Media** field, under **Format** column, select **Media style: original**.



9. Open a desktop browser window and go to `http://dpk.local/content/new-york-pizza`. You should see the following:

10. Opening the same window in the iPhone or iPad Simulator will reveal something akin to the following screenshot:



What just happened?

In Drupal 7, the Media module provides a framework for handling all types of media content objects—audio, video, and images—whether they are hosted on your site or are pointers to the content of another site.

A word about encoding

For text files on your computer's hard drive, it's easy to simply open those files in a text editor and begin editing the content of the file. But audio and video files on your hard drive are not that straightforward. They are basically a modern-day version of the grade-school animation flip-book. Each video stream is a series of saved images; each is slightly different than the one before and after that, when strung together at around 24 frames per second, make up a moving image that the eye interprets as motion video. Each image is compressed so that the entire image doesn't have to be saved, compressed and re-interpreted for each video impression. Similarly, audio is an aural picture of tones that when strung together, tone after tone, makes up a sonic image of the audio. For this reason, video files that have an audio track are basically two streams strung together. Audio is compressed into formats, of which, MP3 is arguably the most popular. The way each of these streams are compressed are said to be the streams encoding format.

One of the most contentious arguments on the Internet right now is the argument of which encoding formats will HTML5's `<audio>` and `<video>` tags support. A coalition of software companies has backed the MPEG4 format. MPEG at the moment is free, but it is owned by the Motion Picture Expert's Group and is not an open source algorithm and the fear by many open source advocates is that one day this group will begin charging for the privilege of using their formats. Google has backed an encoding format they purchased and then open sourced called **WebM**. They've built the WebM encoding format into Chrome and built a plugin for Internet Explorer. The Mozilla foundation has backed Ogg Vorbis, which is a completely open source encoding format but not widely used outside of open source operating systems.

So, I wrote all that to say that HTML5's `<video>` and `<audio>` tag is not a panacea. Differently encoded videos will play in different browsers that support the video's encoding format. This is yet another reason to use a video service because they will handle much of the heavy lifting with regards to encoding the video into a format that each browser can understand.

"I did it my way"

Sometimes, YouTube won't cut it. Either for content or copyright reasons, the project's requirements can preclude the service's terms. Sometimes you need to, in the words of the immortal Frank Sinatra song, do it your way. Host the video yourself.

My first advice to you as a 15-year veteran of 1000+ websites is, "Don't do it".

My second advice to you is, "Seriously dude, don't do it". Use a service. The Embedded Media module for Drupal 6 has support for just about any video service you would ever care to use. One-by-one, those will be ported to the new **Media** module for Drupal 7. Look for the new media module within a year of the publication date of this book to support mostly every commercial video service on the Internet. I promise you that you'll never do it as well as they do.

Users demand a lot from web video. The video has to start downloading and be viewed more or less immediately and the image be of high-enough quality that the viewer can understand what's going on in the video. You'll never be able to do that with your server as well as someone who just does that with their server. Web video takes massive server and bandwidth horsepower and constant attention to the hardware and network connections. So, take my advice—unless you're prepared to hire multiple server administrators, burning through the Hot Pockets and energy drinks in a basement somewhere to keep your video servers up and running and your video downloads disruption-free and you have a revenue stream that supports their salary, just use a service. Otherwise, no matter how well-intentioned your website is, it's doomed to failure. If you're determined and I cannot dissuade you, check out the Drupal distribution, Videola. It may be what you're looking for.

As Frank sang so many times, "Regrets… I've had a few…" Not a lot of people know this, but one of his regrets is hosting his own web video. True story! Well, it's 90 percent true.

Charting and graphs

A great module called fusion charts takes data in a Drupal view and allows you display a chart of the data. Here's the problem. It requires the Flash plugin to display the chart.

This is true for many of the charting solutions on the web today, with one notable exception—Highcharts. The following screenshot shows the **Home** page of `highcharts.com`:

The talented programmers over at `highcharts.com` have created a JavaScript library that takes data and displays it in charts, rendered with SVG on modern browsers and makes those charts backward-compatible with older versions of IE via Microsoft's **Vector Markup Language** (**VML**). This makes it ideal for mobiles. All of the major mobile iterations of WebKit fully support SVG.

Graphic Formats

Scalable Vector Graphics (**SVG**) is a family of XML-based specifications for 2D graphics based in part on Adobe's PostScript graphic language. In the late 90s, Adobe took what was then "old technology" for them, Illustrator version 3.0's graphic format, and began the process of creating the SVG standard. The standard never really caught on and was usurped by Macromedia's Flash format. Adobe then bought Macromedia primarily because Flash technology had achieved many of the stated goals of SVG. Primarily, resolution independence and a massive install base. Once SVG became an open standard, support was written into the HTML5 draft. All mobile versions of WebKit have support for the format.

Vector Markup Language (**VML**) is Microsoft's proprietary (and deprecated) vector graphic format. There's an old joke. How many Microsoft technicians does it take to change a light bulb? None; Microsoft just declared Darkness, the new industry standard. Subject to both diminishing market share and diminishing influence in the Internet community, Microsoft has seen the light and is embracing HTML open standards and building them (albeit more slowly than their rivals) into IE. Basically, the only implementation of VML is in Internet Explorer versions 8 and under. It's used as backwards compatibility for SVG files with scripting that does an XML transformation to change the SVG code into VML. Going forward, IE will support SVG.

We'll create a simple example of how this module is used and generate a cross-browser, mobile, and table-compatible graph.

Time for action – graphing a view

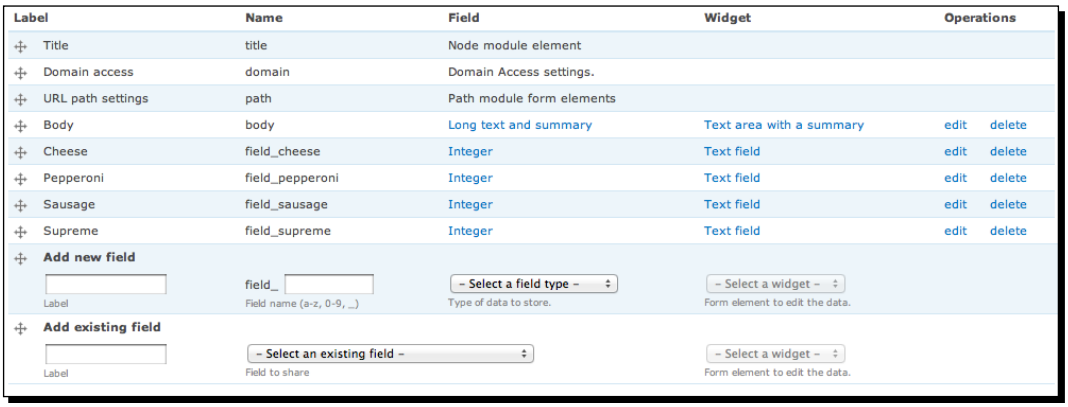
1. Navigate to `http://highcharts.com/download` and download the latest version of the Highcharts JavaScript library. Unpack the latest version and put the folder in `sites/all/libraries`. The module will look for the file `sites/all/libraries/highcharts/js/highcharts.js`.

2. Open a terminal window and enter the following commands:

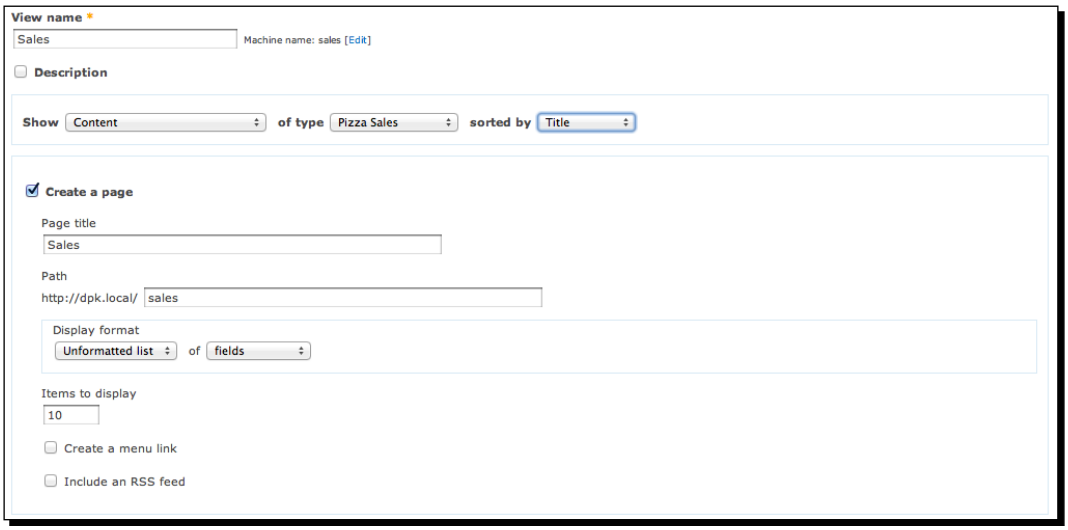
```
cd ~/Sites/dpk
drush dl highcharts
drush pm-enable highcharts
```

3. Navigate to **Structure** | **Content Types** | **Add Content Type**.

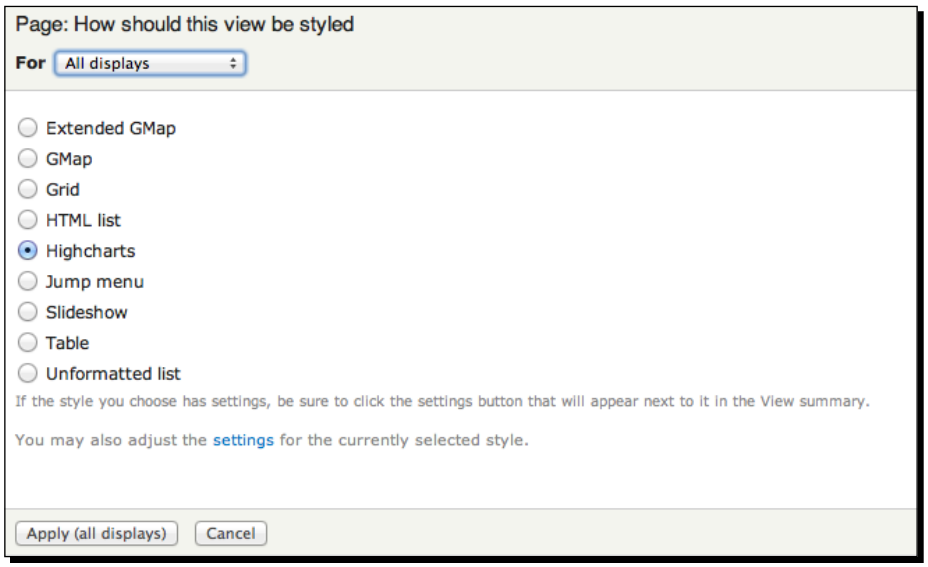
4. Create a new content type **Pizza Sales** with fields for **Cheese**, **Pepperoni**, **Sausage**, and **Supreme**; all of the **Integer** type, as shown in the following screenshot:



5. Navigate to **Structure** | **Views** | **Add new view**. This view will show **Content** of type **Pizza Sales** sorted by **Title**. Create a page display. With the title **Sales** and **Path** set to **Sales** as an **Unformatted list** of **fields**. Click on **Continue and edit**:



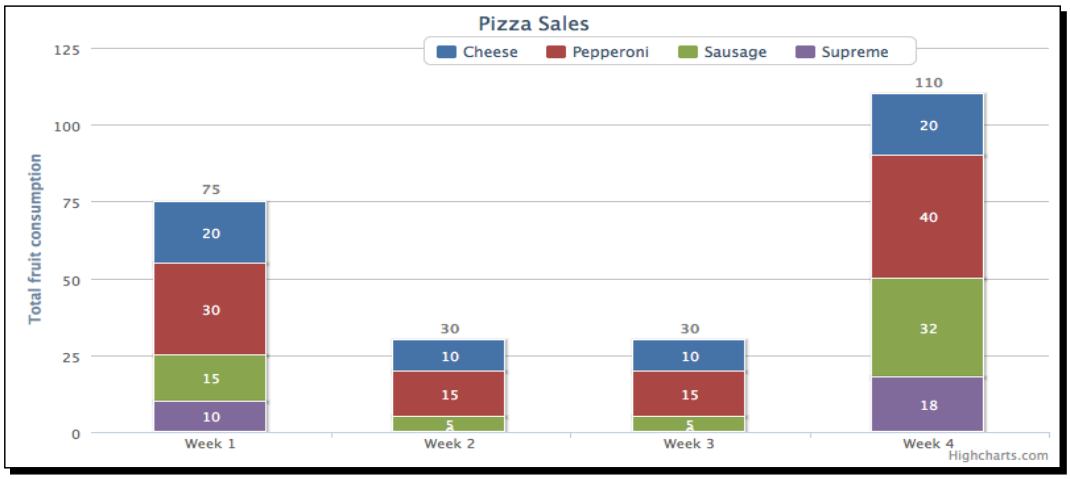
6. Under **Format**, choose **Highcharts**. Click **Apply (all displays)**.



7. At the time of the writing of this book, the settings on the Highcharts module are still in development. You can choose the columns to be incuded with the graph, the type of graph, and the x-axis label. For this example, we used a simple bar graph. We select the four fields, **Cheese**, **Pepperoni**, **Sausage**, and **Supreme** as our fields, with the week as the label field.

8. Navigate to **Content | Add content | Node export: import**. Click on **Upload a File** and navigate the file browser to the `sites/default/content_imports` directory. There should be a **chapter7.export** file that you pulled down when you pulled down the original code. Once the file is chosen, click on **Import**. The import action will list the content nodes that have been created.

9. Navigate to `http://dpk.local/sales`. It should look something like the following diagram:



Pop quiz

1. What types of content does Drupal 7's **Media** module assist you with managing?
    a. Photos
    b. Video
    c. Audio
    d. All of the above
2. Fusion charts are available for Drupal 6, but have a client requirement of what browser plugin?
    a. Flash
    b. RealMedia
    c. Adobe Acrobat
    d. None of the above
3. Highcharts uses which two vector graphic standards to create maps that are mobile-ready?
    a. VMS and SVG
    b. FLV and MP4
    c. PSD and JPG
    d. JPG and TIF

4. The Highcharts JavaScript library is
    a. Free to non-profit websites
    b. Per-website license for commercial sites
    c. Both a and b
    d. Neither a nor b

Summary

Websites with audio and video attract more visitors. Drupal is perfect for embedding audio and video from YouTube or from any number of video content providers. We've walked you through how to add embedding fields to any content type. You can use those fields to add audio and video objects to the Web content.

Charts and graphs are the life blood of business analysis. With Highcharts, Drupal gains a cross-browser charting tool and allows you to use the power of CMS to manage statistical analysis of business data.

In the next chapter, we'll look at Drupal's integration with location services and new open source ways to embed maps into pages.

7
Location, Location, Location

Big Jimmy has been talking to some guys—franchise guys. Jimmy always had visions of opening one or more locations, but never really had that "push" to get the job done. The franchise guys are pushing him to open up a few more locations in the Tampa Bay area to prove the validity of the "pizza kitchen" concept. Then they want to turn the kitchen into a franchised restaurant and start taking on investors and new store owners.

With this information in mind, Little Jimmy and I had a discussion about some of the new information the website would need to accommodate in order to keep pace with his father's dreams for the restaurant. We'll need to be able to store location information about the different stores. We'll also want a list of stores giving the customer the ability to find the store closest to them. Luckily, Drupal can accommodate all of that and do it very well.

In this chapter, we'll:

- Learn about the **Location** and **GMap** modules and how to use them as building blocks to create a rich mobile (and desktop) user experience
- Learn about the `navigator.geolocation` object
- Add location information to node objects
- Geocode a node's location data

A word about browsers

You will need to use Safari, Chrome, Firefox or Internet Explorer 9 or later in order to work on the tasks in this chapter. The Geolocation API wasn't included in IE until version 9. Safari, Chrome, and Firefox have had the Geolocation API for a year or so now, so any version that's less than a year old should work.

Geolocation

Your position on the earth can be derived in one of the two ways. The first being the GPS satellite system. It's the system used in most cars and, lately, most smart phones. The GPS satellites constantly bounces signals off the earth and there's a special receiver in the GPS hardware that decodes that signal and uses it to determine longitude and latitude. The GPS hardware then returns the coordinates to any software needing them.

The second way is a little bit more complex. All over the world, people have implemented Wi-Fi networks and cell phone tower networks. Most of them are stationary and installed in houses, businesses, cities, and roadsides across the country. In the same way that Google Maps has cars traveling the country taking photos of street views, there are geolocation companies that travel the country and take a census of the available Wi-Fi networks and their relative signal strength and the geolocation of the Wi-Fi measuring device. All of the cell phone towers in the US are already geocoded and their longitude and latitude locations are known. Software can then, with an amazing degree of accuracy determine your location from the Wi-Fi and cell networks and relative strengths available to your desktop, laptop computer, or cell phone without any assistance from GPS-specific hardware.

For a year or so now, the desktop versions of Firefox and WebKit (Safari and Chrome) have implemented the HTML5 geolocation interface for JavaScript using the second method. The browsers will always warn the users about beginning a geolocation survey and allow the user to opt out for privacy reasons.

That's why, as a web developer, it's frustrating to me that more web developers don't take advantage of the geolocation JavaScript object when developing their websites, especially when it comes to locating the physical location of their client or their client's stores.

The navigator.geolocation object

The JavaScript object that powers the desktop and handheld browser's location abilities is the `navigator.geolocation` object. Because it can take time to make a call to the geolocation hardware or website that will derive the location, using the object does not stop the JavaScript execution. You hand the `geolocation` object a function to execute once the location is derived and a function to execute on failure, and the JavaScript execution continues while the `geolocation` object goes to work. We'll look more into these calls in the *Time for action – downloding and enabling the close2u module* section.

First, let's create some nodes with the **Location** module and get them geocoded with longitude and latitude coordinates.

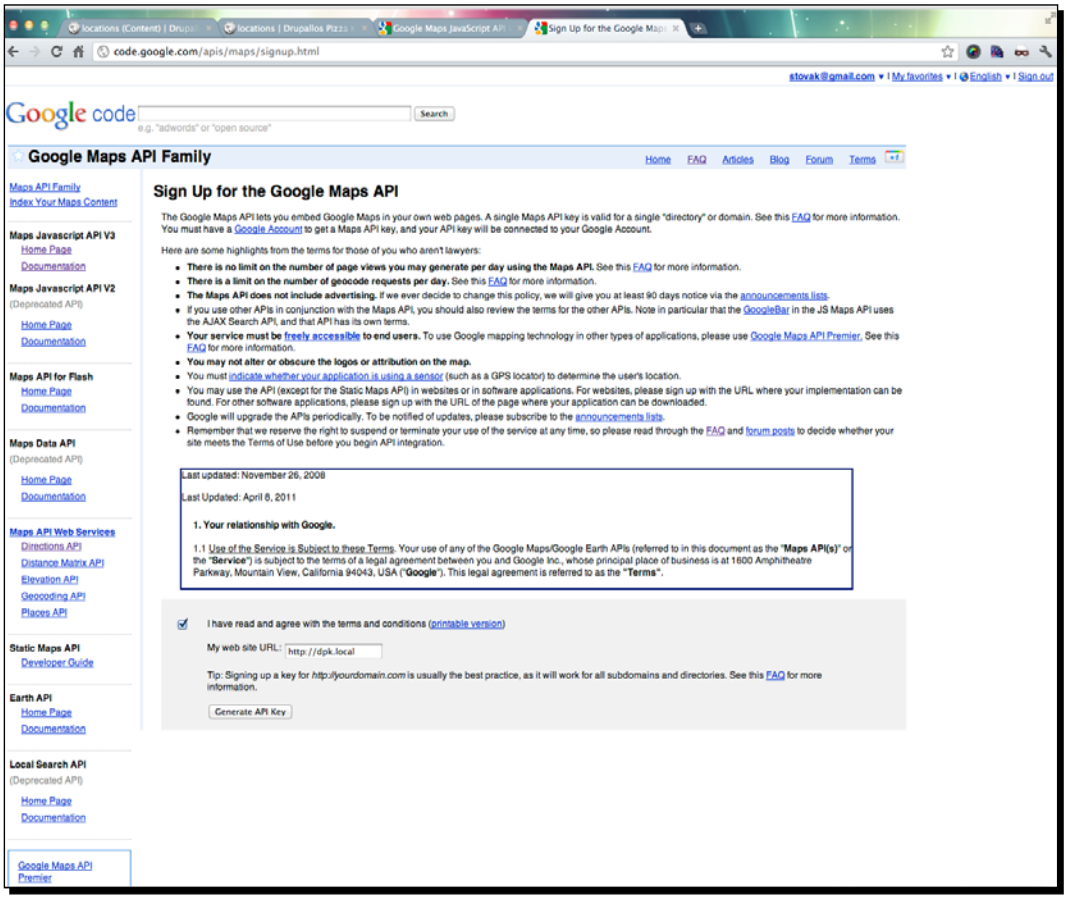
Time for action – adding location data to nodes

If you take a look at the Drupal Database, you'll notice that the location has its own database table. Locations are separate database objects. They are then related to other Drupal data entities such as nodes and users. We're going to add some location data from Google's Map service to nodes in this example, but the location data could just as easily be linked to almost any data entity that Drupal recognizes:

1. Open a terminal window and enter the following commands:

```
cd ~/Sites/dpk
drush dl gmap location
```

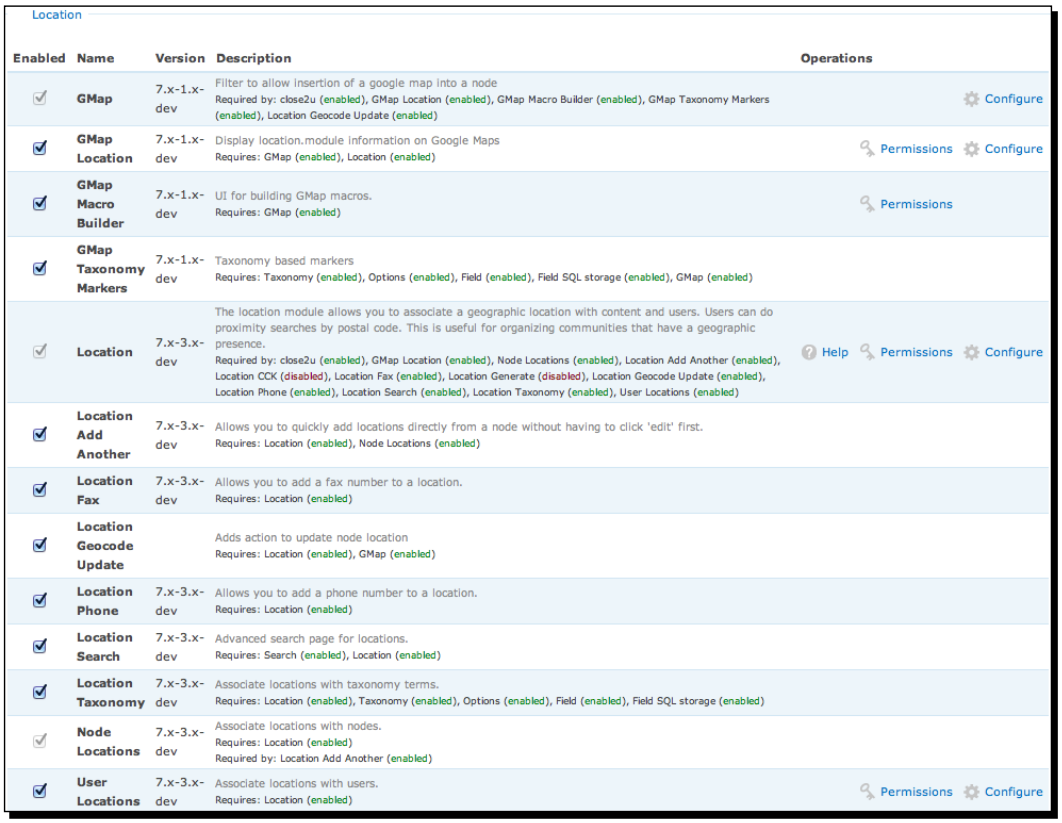
2. Navigate to `http://code.google.com/apis/maps/signup.html`:



3. As shown in the preceding screenshot, enter the URL of your test site (`dpk.local`) and check the **I have read and agree with the terms and conditions** checkbox. You're just using this site for development. You will need to obtain another API key for any live site. Be sure that you actually agree to the terms and conditions if you use the key on a live site. Click on the **Generate API Key** button and on the next page, you will see a long API key, as shown in the following screenshot. Copy and paste it somewhere, where you won't lose it:

Your API key will almost certainly be different from this one but will be a long string of letters, numbers, and symbols like this.

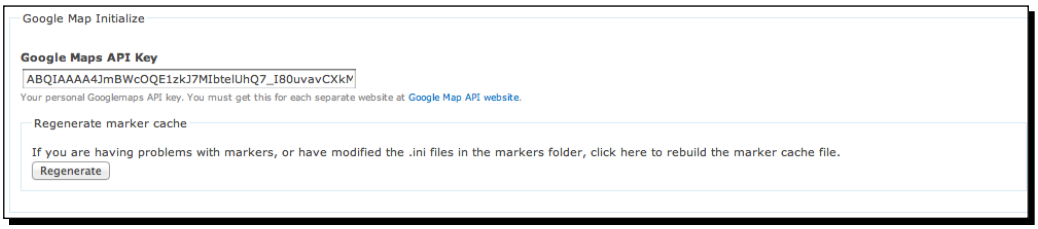
4. Navigate to **Admin** | **Modules** and ensure that all of the modules in the **Location** and **GMap** groups are enabled:



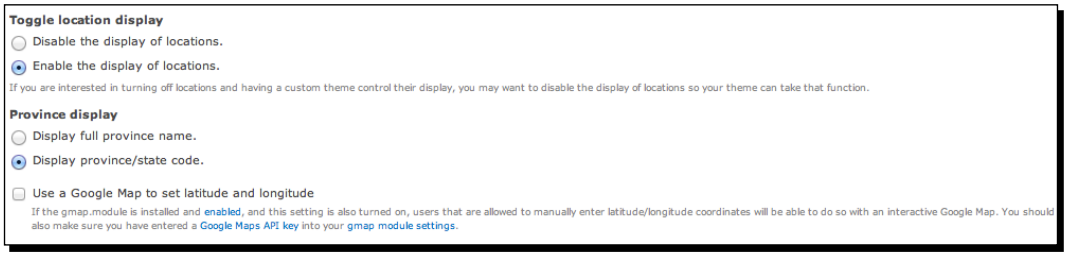
Location

Enabled	Name	Version	Description	Operations
☑	**GMap**	7.x-1.x-dev	Filter to allow insertion of a google map into a node Required by: close2u (enabled), GMap Location (enabled), GMap Macro Builder (enabled), GMap Taxonomy Markers (enabled), Location Geocode Update (enabled)	Configure
☑	**GMap Location**	7.x-1.x-dev	Display location.module information on Google Maps Requires: GMap (enabled), Location (enabled)	Permissions Configure
☑	**GMap Macro Builder**	7.x-1.x-dev	UI for building GMap macros. Requires: GMap (enabled)	Permissions
☑	**GMap Taxonomy Markers**	7.x-1.x-dev	Taxonomy based markers Requires: Taxonomy (enabled), Options (enabled), Field (enabled), Field SQL storage (enabled), GMap (enabled)	
☑	**Location**	7.x-3.x-dev	The location module allows you to associate a geographic location with content and users. Users can do proximity searches by postal code. This is useful for organizing communities that have a geographic presence. Required by: close2u (enabled), GMap Location (enabled), Node Locations (enabled), Location Add Another (enabled), Location CCK (disabled), Location Fax (enabled), Location Generate (disabled), Location Geocode Update (enabled), Location Phone (enabled), Location Search (enabled), Location Taxonomy (enabled), User Locations (enabled)	Help Permissions Configure
☑	**Location Add Another**	7.x-3.x-dev	Allows you to quickly add locations directly from a node without having to click 'edit' first. Requires: Location (enabled), Node Locations (enabled)	
☑	**Location Fax**	7.x-3.x-dev	Allows you to add a fax number to a location. Requires: Location (enabled)	
☑	**Location Geocode Update**		Adds action to update node location Requires: Location (enabled), GMap (enabled)	
☑	**Location Phone**	7.x-3.x-dev	Allows you to add a phone number to a location. Requires: Location (enabled)	
☑	**Location Search**	7.x-3.x-dev	Advanced search page for locations. Requires: Search (enabled), Location (enabled)	
☑	**Location Taxonomy**	7.x-3.x-dev	Associate locations with taxonomy terms. Requires: Location (enabled), Taxonomy (enabled), Options (enabled), Field (enabled), Field SQL storage (enabled)	
☑	**Node Locations**	7.x-3.x-dev	Associate locations with nodes. Requires: Location (enabled) Required by: Location Add Another (enabled)	
☑	**User Locations**	7.x-3.x-dev	Associate locations with users. Requires: Location (enabled)	Permissions Configure

5. Navigate to **Configuration** | **Services** | **GMap** and paste the API key in the **Google Maps API Key** field:

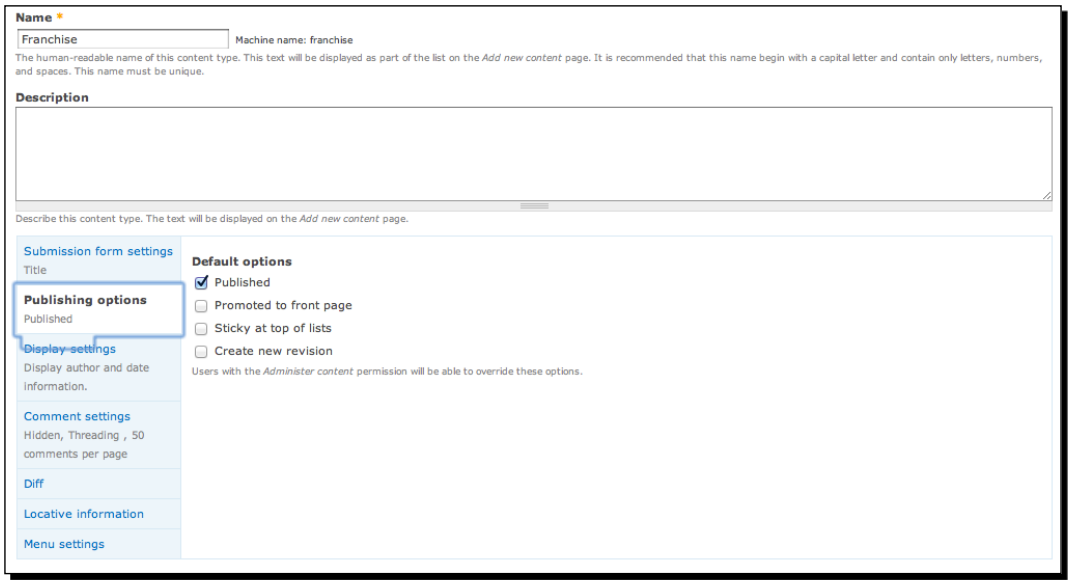


6. Navigate to **Admin** | **Content** | **Location**. Make sure you select the **Enable the display of locations** options:

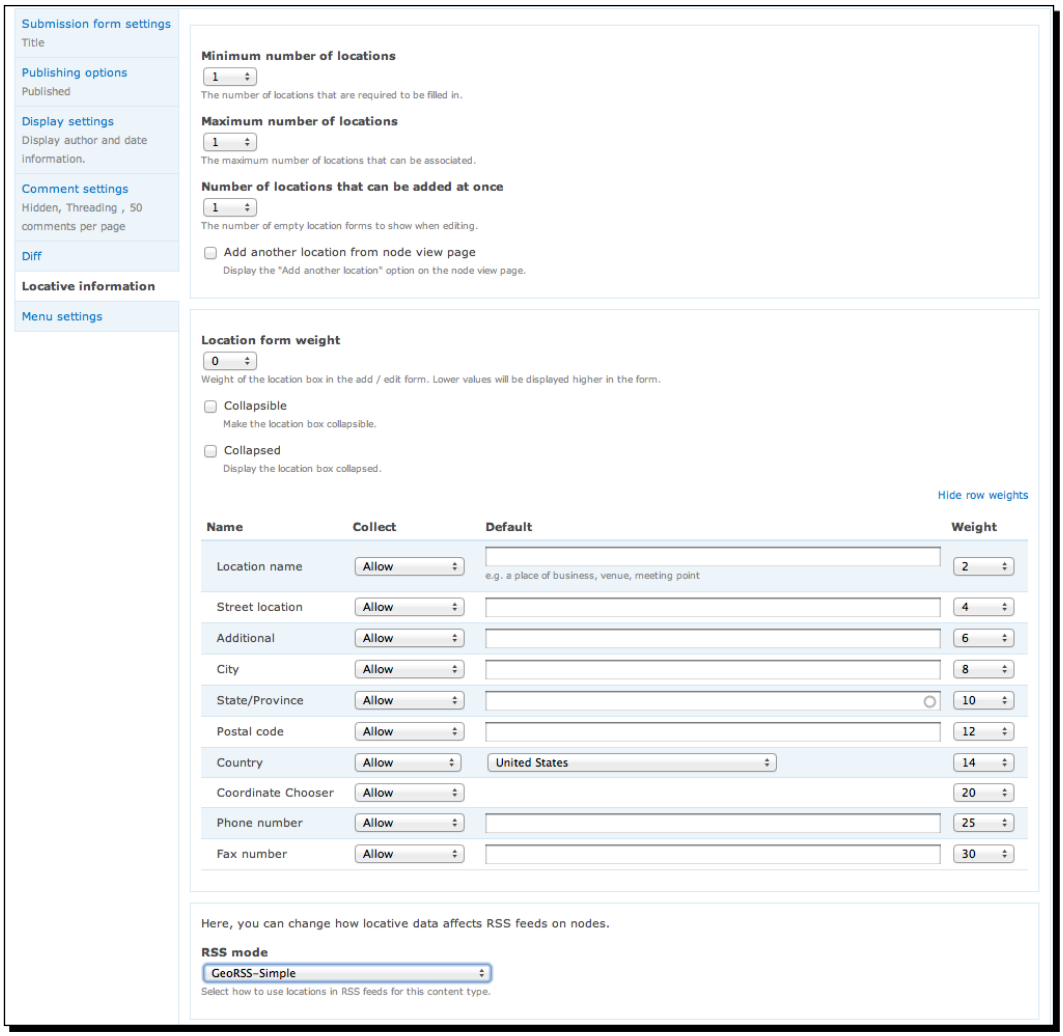


7. Navigate to **Structure** | **Content types** | **Add content type**.

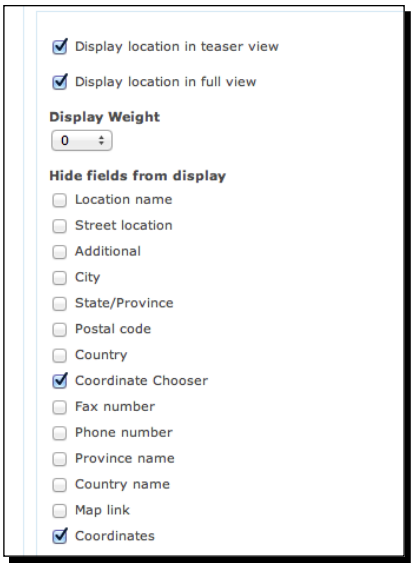
8. We're going to call this content type **Franchise** (see the following screenshot). Under **Publishing options**, select the **Published** option, and the **Promoted to front page** option should be unchecked. Under **Comment settings**, select **Hidden**. Under **Locative information**, change **Minimum Number of Locations** to **1** (with **Maximum number of locations** set to **1**). Save the new content type:



9. Now, navigate back to **Admin** | **Structure** | **Content types**. Under **Locative information**, there are options to gather **Postal code**, **City**, **State**, **Phone number**, **Fax number** and so on. Change them all to **Allow**:

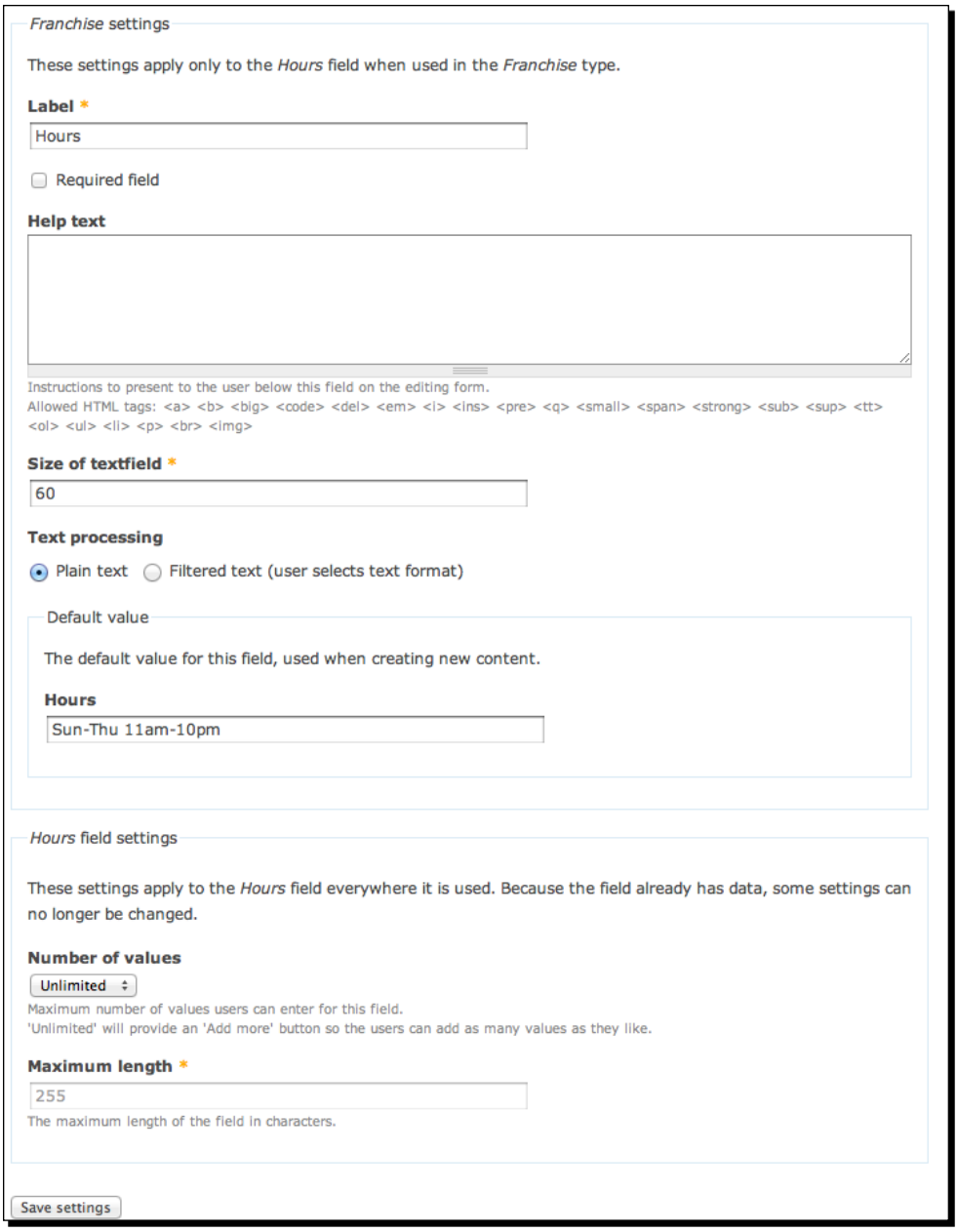

Submission form settings
Title
Publishing options
Published
Display settings
Display author and date information.
Comment settings
Hidden, Threading , 50 comments per page
Diff
Locative information
Menu settings
Minimum number of locations
1
The number of locations that are required to be filled in.
Maximum number of locations
1
The maximum number of locations that can be associated.
Number of locations that can be added at once
1
The number of empty location forms to show when editing.
Add another location from node view page
Display the "Add another location" option on the node view page.
Location form weight
0
Weight of the location box in the add / edit form. Lower values will be displayed higher in the form.
Collapsible
Make the location box collapsible.
Collapsed
Display the location box collapsed.
Hide row weights
Name
Collect
Default
Weight
Location name
Allow
e.g. a place of business, venue, meeting point
2
Street location
Allow
4
Additional
Allow
6
City
Allow
8
State/Province
Allow
10
Postal code
Allow
12
Country
Allow
United States
14
Coordinate Chooser
Allow
20
Phone number
Allow
25
Fax number
Allow
30
Here, you can change how locative data affects RSS feeds on nodes.
RSS mode
GeoRSS-Simple
Select how to use locations in RSS feeds for this content type.

10. At the bottom of the location information, we will have to hide the **Coordinate Chooser** and check the **Display location in teaser view** and **Display location in full view** checkboxes:

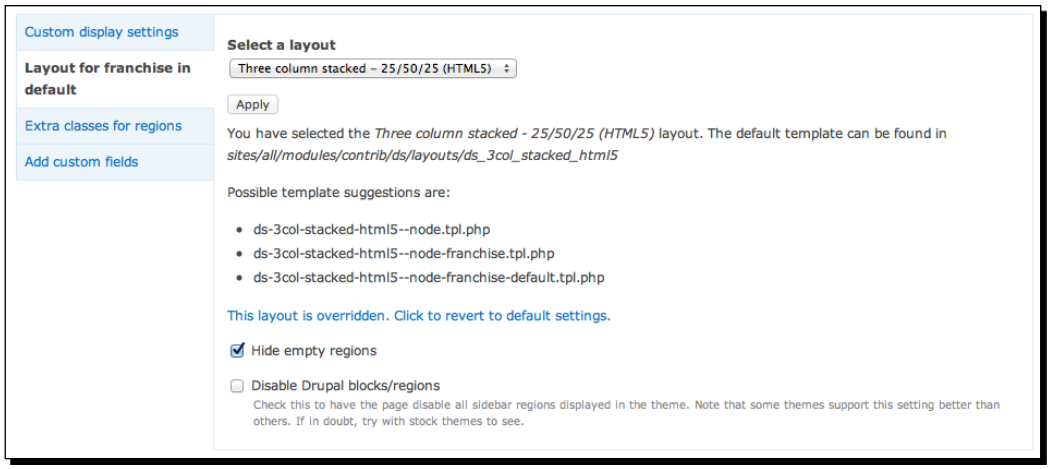


11. Once the content type is saved, navigate back to **Structure** | **Content types** | **Franchise** | **Manage fields**.

12. Create a new field called **Hours** with the machine name as **hours**. Under **Widget Type**, choose **Text Field** and under **Hours fields settings** change **Number of Values** to **Unlimited**.


Franchise settings
These settings apply only to the Hours field when used in the Franchise type.
Label *
Hours
Required field
Help text
Instructions to present to the user below this field on the editing form.
Allowed HTML tags: <a> <b> <big> <code> <del> <i> <ins> <pre> <q> <small>   <sub> <sup> <tt>   <p>
 <img>
Size of textfield *
60
Text processing
Plain text
Filtered text (user selects text format)
Default value
The default value for this field, used when creating new content.
Hours
Sun-Thu 11am-10pm
Hours field settings
These settings apply to the Hours field everywhere it is used. Because the field already has data, some settings can no longer be changed.
Number of values
Unlimited
Maximum number of values users can enter for this field.
'Unlimited' will provide an 'Add more' button so the users can add as many values as they like.
Maximum length *
255
The maximum length of the field in characters.
Save settings

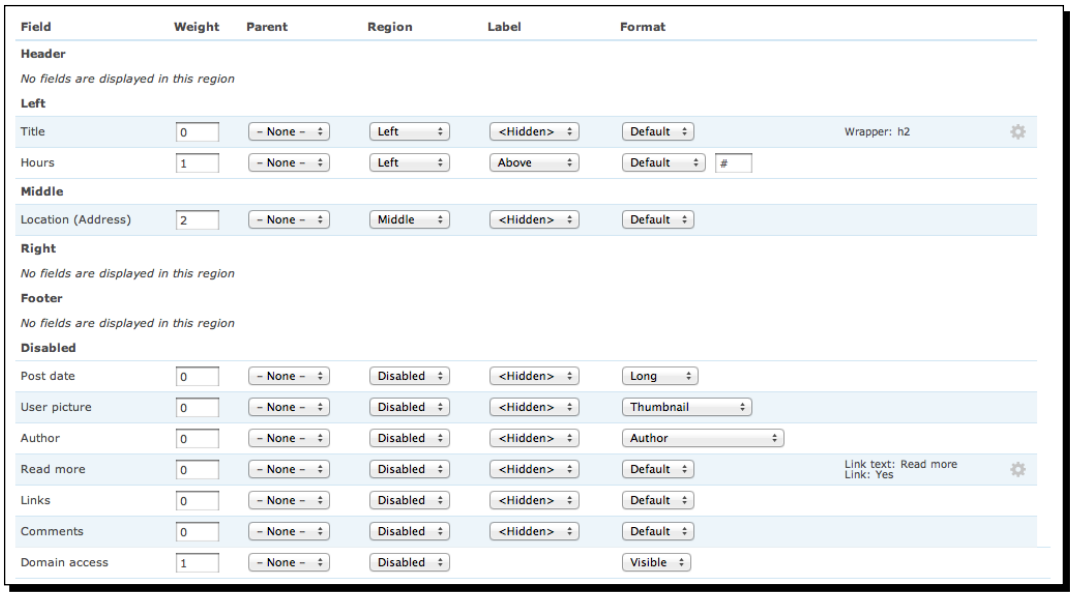
13. Navigate to **Structure** | **Content types** | **Franchise** | **Manage display**. In the **Layout for franchise in default** tab, select **Three column stacked - 25/50/25 (HTML5)** under **Select a layout** and then save the settings:



14. Click on the **Add custom fields** tab and then click on **Add a code field**. Call it **Location (Address)**. Check the **Node** entity. Add the following as the code value, then save the field:

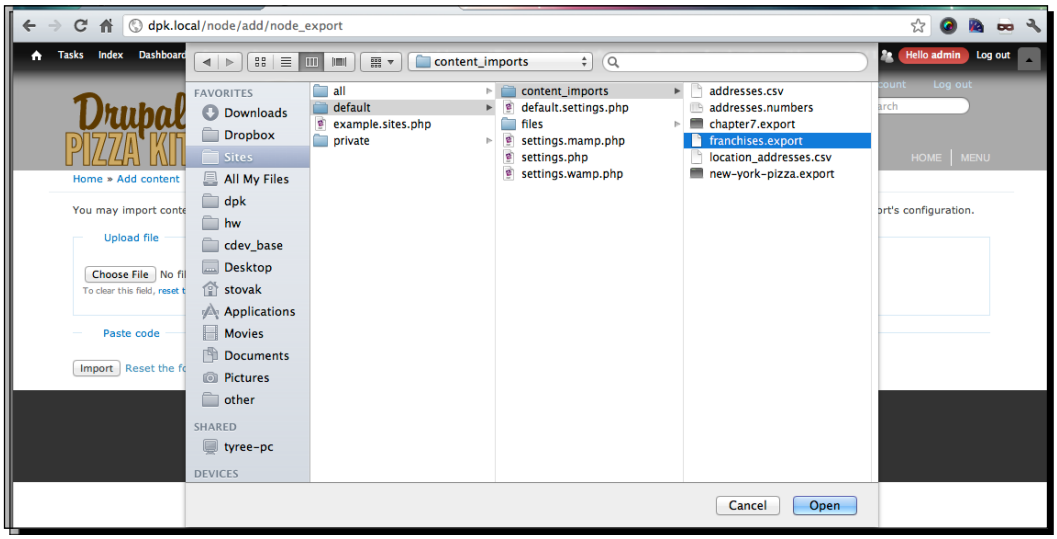
```
<?php
  $settings = variable_get('location_settings_node_' . $entity-
>type, array());
if (isset($entity->locations)) {
print drupal_render(location_display($settings, $entity-
>locations));
  }
?>
```

15. Move the **Title** and **Hours** fields to the **Left** region. Move the newly created **Location (Address)** field to the **Middle** region. Save the node display:



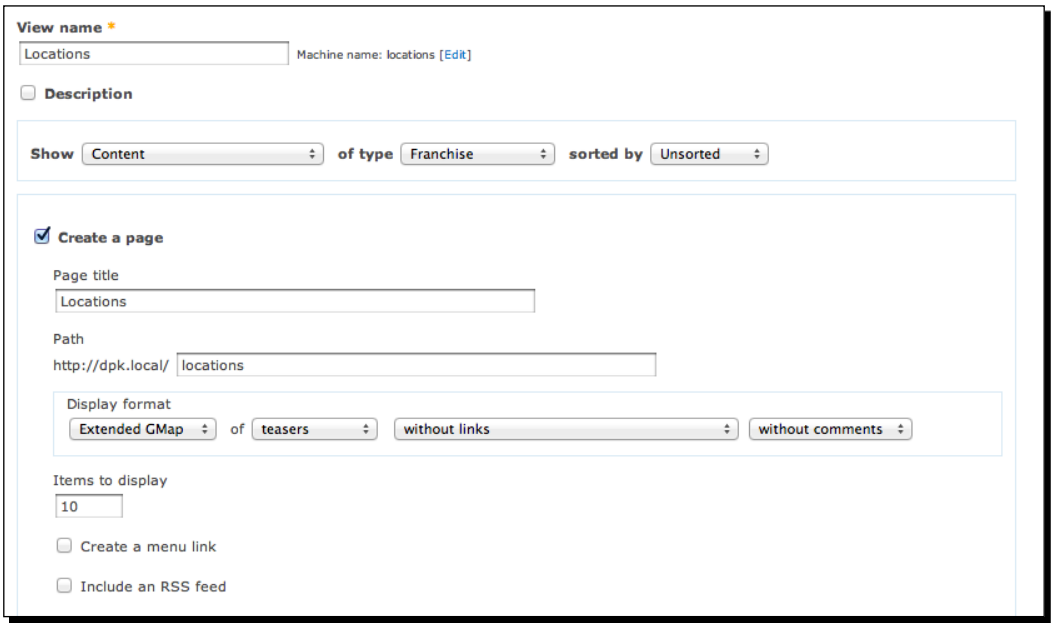
Field	Weight	Parent	Region	Label	Format	
Header						
No fields are displayed in this region						
Left						
Title	0	– None –	Left	<Hidden>	Default	Wrapper: h2
Hours	1	– None –	Left	Above	Default #	
Middle						
Location (Address)	2	– None –	Middle	<Hidden>	Default	
Right						
No fields are displayed in this region						
Footer						
No fields are displayed in this region						
Disabled						
Post date	0	– None –	Disabled	<Hidden>	Long	
User picture	0	– None –	Disabled	<Hidden>	Thumbnail	
Author	0	– None –	Disabled	<Hidden>	Author	
Read more	0	– None –	Disabled	<Hidden>	Default	Link text: Read more Link: Yes
Links	0	– None –	Disabled	<Hidden>	Default	
Comments	0	– None –	Disabled	<Hidden>	Default	
Domain access	1	– None –	Disabled		Visible	

16. Choose **Content** | **Add content** | **Node Export: Import**. Click on **Upload file**. In the `sites/default/content_exports` folder, there should be a file called `franchiese.export`. Choose that file and click on **Upload**. Then click on **Import**:

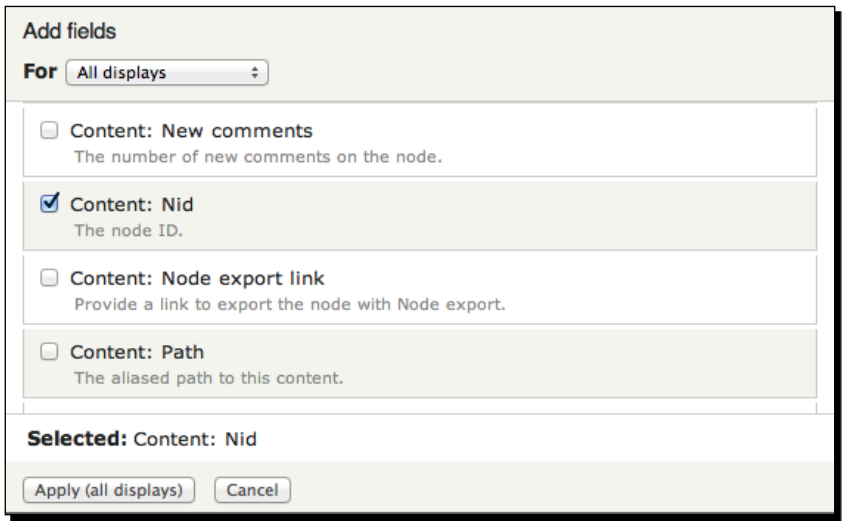


17. As shown in the following screenshot, Drupal will list the nodes that have been imported:

18. Choose **Structure** | **Views**. Choose **Add new view**. Create a new view named **Locations** showing **Content of type Franchise**. The display format should be **Extended GMap of teasers without links** and **without comments**. Click on **Continue & edit**:



19. Under **Add fields**, select the **Content: Nid** option to add the Node ID field:

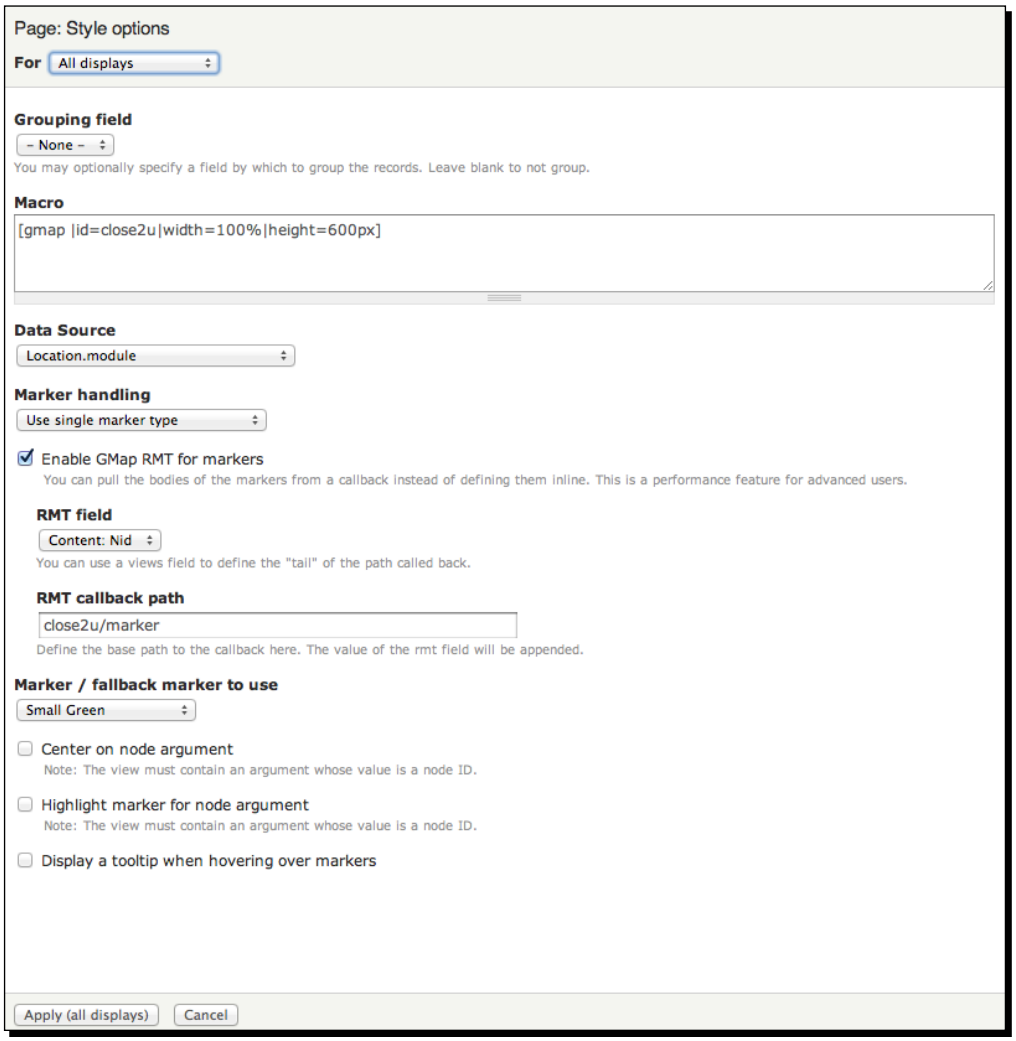


20. Beside the **Format** line is the current format **GMap** and a **settings** link. Click on the **settings** link. In the **Macro** box, add the macro as follows:

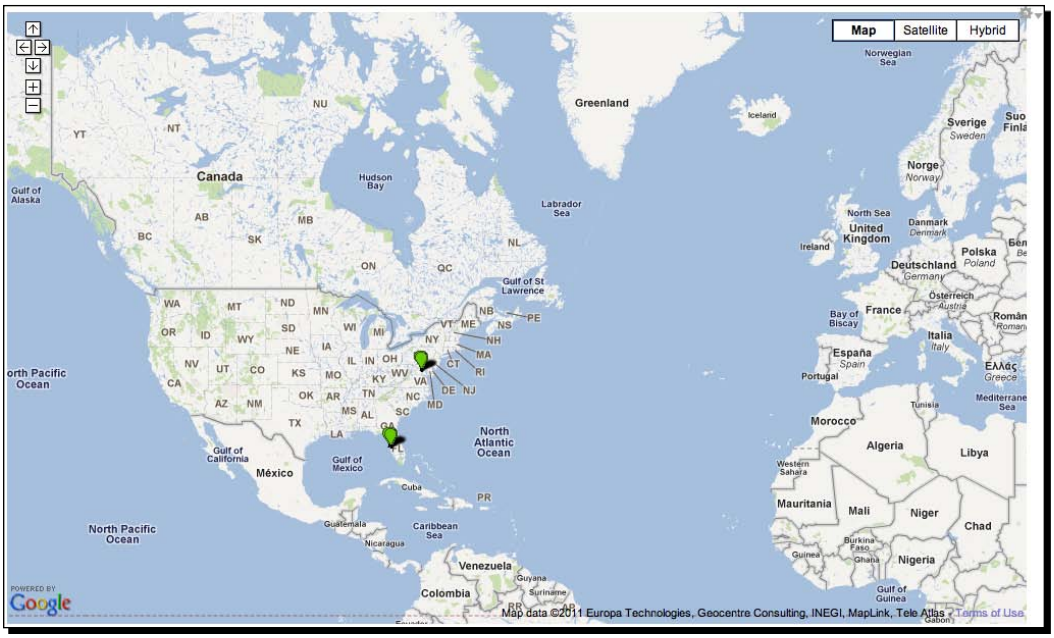
```
[gmap |id=close2u|width=100%|height=600px]
```

21. Under **Data Source** select **Location.module**, and under **Marker Handling** select **Use single marker type**.

22. Under **RMT field** select **Content: NID**. Add a **RMT callback path** of **close2u/marker**. Click on **Apply (all displays)** and save the view:

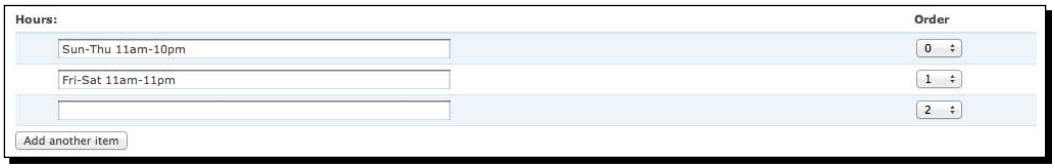


23. If all goes well, you should now be able to navigate to `http://dpk.local/locations` and have a proper Google map with markers for the listed locations. We'll address some of the map issues in the *Time for action – geocoding a node's location data* and *Time for action – downloading and enabling the close2u module* sections, but for now, your nodes are mapped:



What just happened?

The first thing to notice is the way we've done the **Hours** field. Different locations have different hours. Some may need 3 lines to describe their regular hours, some may choose to put all 7 days of the week on a separate line. We can accommodate that by adding a text field and allowing unlimited values in that text field. When entering data for the **Hours** field, you can click on **Add another item** to allow multiple entries:

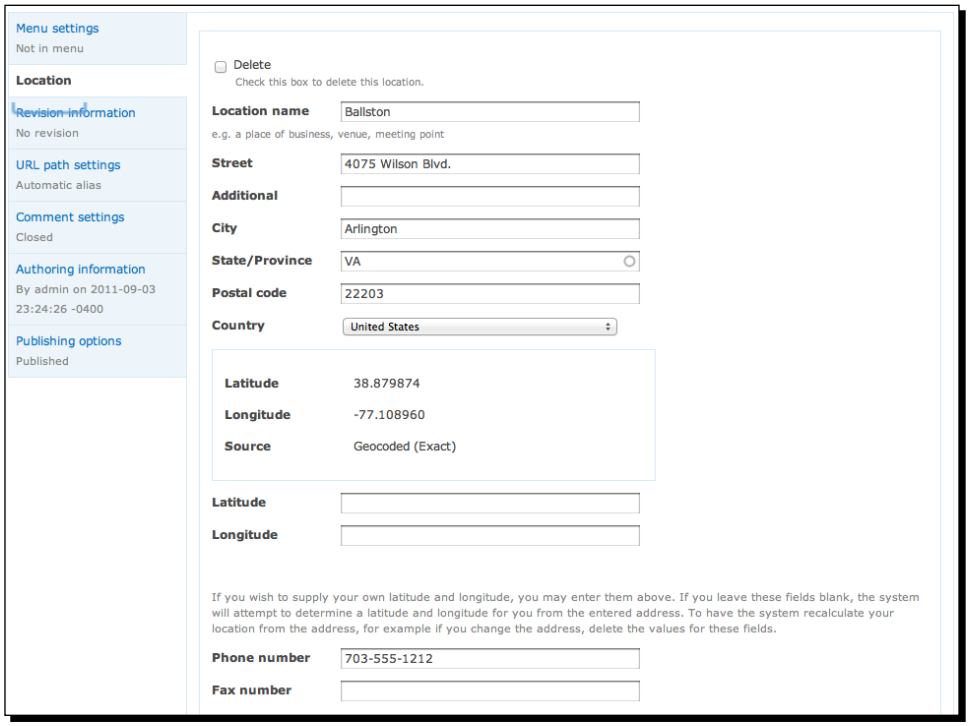


The **Location** module saves the address and geolocation data, and then associates that data with either a node or a user. You can enable the module to work with either. In this example, we've given the **Franchise** content type the ability to attach one or more locations to the nodes. We've then used a standard view and the **GMap** module to mash up those nodes into a GMap.

We've illustrated one of the simultaneously interesting and powerful things about Display Suite's code fields. Display Suite allows you to create customized fields that will execute the PHP code when a node is displayed. We created a Display Suite custom field that shows the address with a Google Maps link. We added the location title and the hours. It is also easy to white-screen your installation with a custom code that fails. Be very careful using this feature.

After we added the code field, we imported some sample data and then created a view that places the newly created nodes on a GMap. We set some values in the macro field of the GMap. These macros define the width and height of the map that is produced. There are other options you can add to determine the look and feel of the maps that the Google API produces. Those options are available in the Google API reference under the **google.maps.MapOptions object** section (`http://code.google.com/apis/maps/documentation/javascript/reference.html#MapOptions`). You should be able to set any map value from that command line using the syntax shown. One we specifically did not set was the MapCenter and the Zoom level. We'll set that a little later in this chapter, based on the user's location data.

If you take a look at one of the nodes we've imported and scrolled down to the node's location data, you'll see the address and a longitude and latitude value that allows Google to plot those coordinates on a map. We'll use those coordinates in the *Time for action – downloading and enabling the close2u module* section to gauge the distance from the user's location:



From address to longitude and latitude

The addresses we've imported had longitude and latitude information encoded in their associated node locations. But unless you regularly list longitude and latitude coordinates with your addresses, you probably don't have this information at hand. You'll need to run the addresses through a process called **geocoding** which sends the address to a service such as Google Maps (or Yahoo! or Bing) and the map system returns a longitude and latitude coordinate for the given address.

There are multiple services that provide geocoding and nothing says you have to use Google or the module in this example. But for Drupal 7, at the time of this writing I found my options limited, so I wrote a companion project for the Drupal 7 **Location** module that I call **Location Geocode Update**. I hope by the time of this book's publication that this module will be a full-fledged Drupal project, but for now it's a sandbox project. Sandbox projects are projects developers create to try new things and use experimental code. The code in this module works for this example and I look forward to sharing it with the Drupal community at large as a full-fledged Drupal project.

Playing in the Sandbox

Drupal projects have several different phases. The first phase should be the sandbox mode. For the `geolocation` module and the `close2u` module I've created them as sandbox modules on `drupal.org`. You can find these at `http://drupal.org/sandbox/stovak/1262930` and `http://drupal.org/sandbox/stovak/1265648`. Use sandbox modules at your own risk. To install a project from the sandbox, you'll need to clone it from the GIT repository. Most sandbox modules have the GIT repository URL. Open the terminal window and change the directory to your project's `sites/all/modules/custom` directory. If your project doesn't have one, create it. Enter `git clone SANDBOX_URL`, where `SANDBOX_URL` is the GIT URL you obtained from the sandbox project listing. GIT will create a local copy of the sandbox. If the Drush commands to find the modules in this chapter fail, you can obtain the latest version by cloning the `drupal.org` sandbox. To do this, search for the module on `drupal.org` using the project name I've given you and clone the project locally. Let's get started!

Time for action – geocoding a node's location data

Geocoding address data is a simple call to Google's API. I've encapsulated the API calls into a module that triggers on node save:

Fake addresses will return incorrect geocoding data and should not be used for this example. We're testing the node's ability to add the longitude and latitude data without looking them up. Also, note that in order for Google geocoding to work correctly, you must provide a valid postal code for the given address.

1. Change the directory to your site's root directory and use Drush to install and enable the module with the following commands:

```
cd ~/Sites/dpk
drush dl location_geocode_update
drush pm-enable location_geocode_update
```

2. Navigate to **Modules** | **Admin** and enable the **Trigger** module from core if it's not already enabled.

3. Navigate to **Admin** | **Content** | **Location**. Check the **Enable JIT geocoding** checkbox.

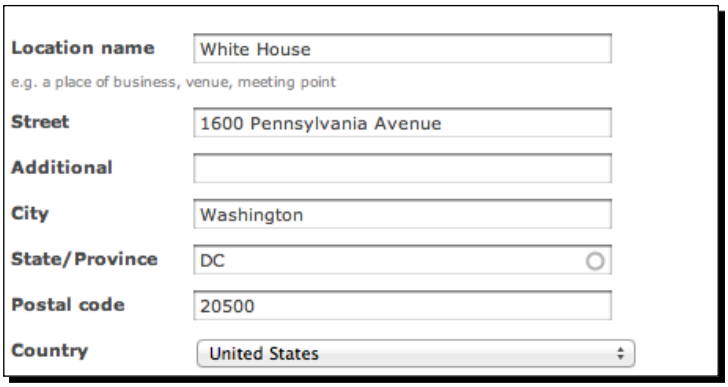
4. Navigate to **Structure** | **Triggers** | **Node**. You will see several triggers listed with an action list of possible actions. Under the first trigger, **When either saving new content or updating existing content**, select the **Update Latitude/Longitude** action and click on **Assign.** The new action should appear in the list.

5. Navigate to **Content** | **Add content** | **Franchise**.

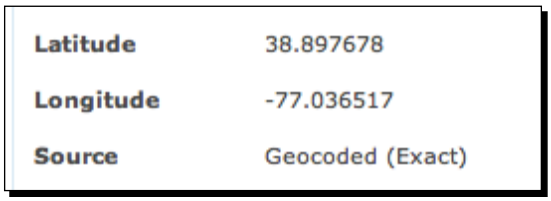
6. Name the franchise as **My Location**.

7. Add the hours as **9AM – 5PM Mon-Fri**.

8. Click on **Location** and add your address or an address you know is valid without looking it up. Save the node:



9. Drupal should take you to a full view of the newly added node. At the top, there will be a selected **View** tab as well as **Edit** and maybe **Export**, too. Edit the node and notice the address should now have longitude and latitude information:



What just happened?

The **Triggers** module allows actions to be triggered (get it?) on predefined events for pieces of content. The Location Geocode Update module created a new action that we added to the node create/update trigger. This trigger will allow any node that allows the location data to be geocoded when saved or created.

The close2u module

The Google map we created in the *Time for action – adding location data to nodes* section has a few problems. The first of which is the zoom magnification. It's way too far out. The second is that the center of the map is traditionally the user's location. The third is that the map itself doesn't have any user location data.

I mentioned earlier that the JavaScript `geolocation` object has been available in browsers for some time now. During the writing of this chapter, I couldn't actually find any Drupal module that used client geolocation data from the browser; so, I wrote this simple module that integrates with **Views**, **GMap** and the Drupal 7 **Location** modules to sort of re-imagine the "store locator" page that so many websites have. It's a good example of how to create a customized Drupal 7 module to solve many of the issues that exist with GMap and location integration in Drupal 7. As of the writing of this chapter, the module is a sandbox module. I hope to take it to full project status before the book is published. We'll step through the module's code after the exercise to show you how the magic happens.

Let's get started!

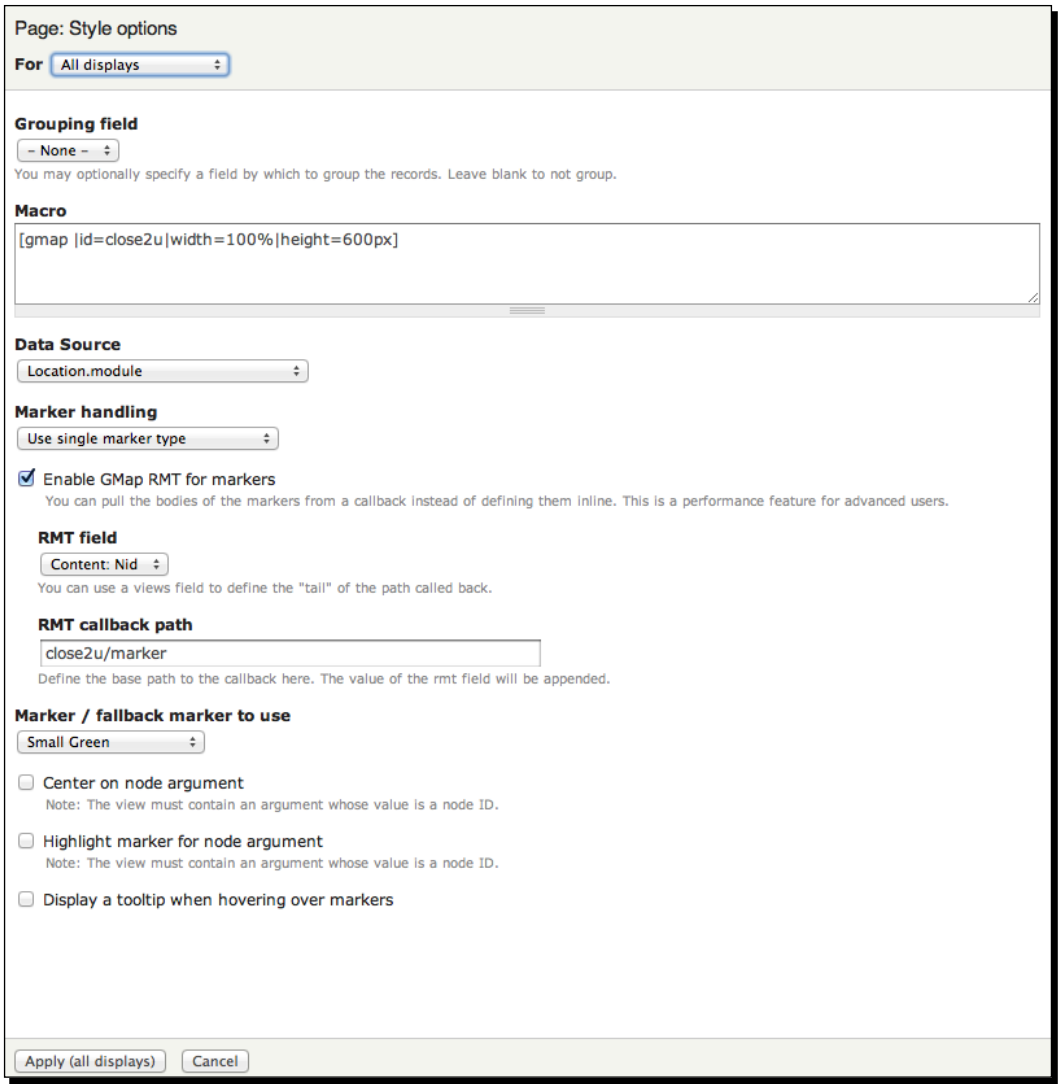
Time for action – downloding and enabling the close2u module

The math to determine the distance between two objects on the earth is not simple. It's a complex formula. Luckily, that's all been worked out in advance for us and the code is in the `location` directory in a file called `earth.inc`. If you care to look that over, open up the file and take a look. If not, just trust that it's there. This module uses `earth.inc` to produce SQL that will sort objects by location:

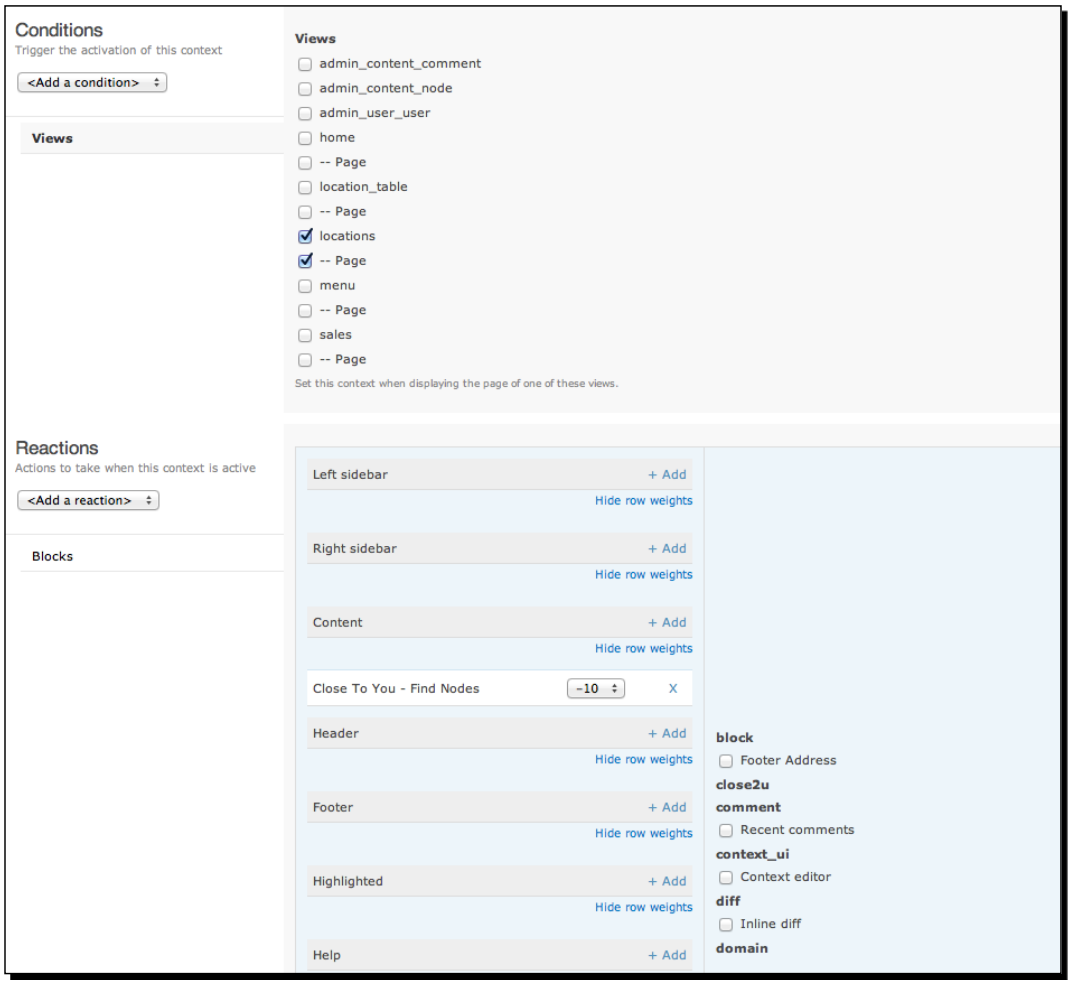
1. Open a terminal window. Change the directory to your site root and use the `drush` command to download and enable the `close2u` module. Note the words at the beginning of the chapter with regards to sandbox modules.

```
cd ~/dpk/
drush dl close2u
drush pm-enable close2u
```

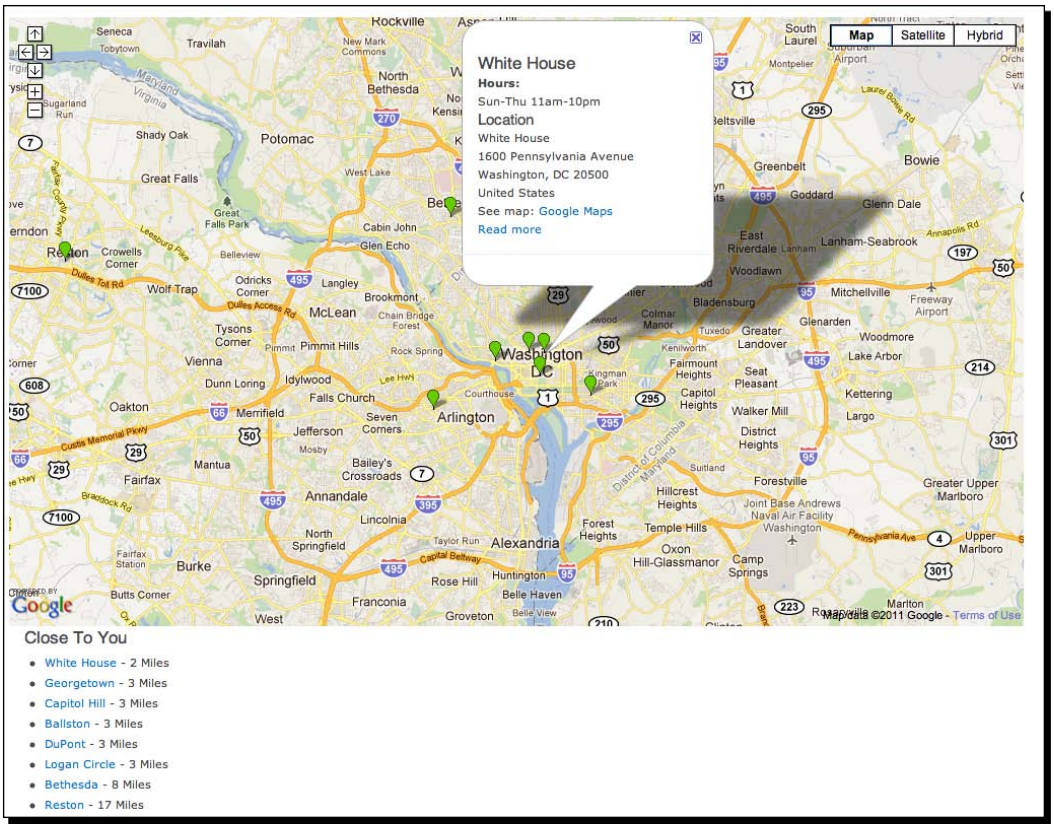
2. First, navigate to **Structure** | **Views** and edit the **Locations** view we created in the *Time for action – geocoding a node's location data* section. Verify that the RMT values match the following screenshot. The **RMT field** value needs to be set to **Content: Nid** and the **RMT callback path** needs to be set to **close2u/marker**. Also of note, we're going to be working with the created GMap directly, so under **Macro**, make sure that the GMap ID value is `close2u`. Otherwise, our JavaScript won't be able to find the map:


Page: Style options
For
All displays
Grouping field
– None –
You may optionally specify a field by which to group the records. Leave blank to not group.
Macro
[gmap |id=close2u|width=100%|height=600px]
Data Source
Location.module
Marker handling
Use single marker type
Enable GMap RMT for markers
You can pull the bodies of the markers from a callback instead of defining them inline. This is a performance feature for advanced users.
RMT field
Content: Nid
You can use a views field to define the "tail" of the path called back.
RMT callback path
close2u/marker
Define the base path to the callback here. The value of the rmt field will be appended.
Marker / fallback marker to use
Small Green
Center on node argument
Note: The view must contain an argument whose value is a node ID.
Highlight marker for node argument
Note: The view must contain an argument whose value is a node ID.
Display a tooltip when hovering over markers
Apply (all displays)
Cancel

3. Navigate to **Structure** | **Context** | **Add**. Title this context as **Locations**. Under **Conditions**, click on **Views** and then select the **locations** view we created earlier along with it's page display (**--Page**). Under **Reaction**, click on **Blocks**. Check the checkbox beside the **Close To You – Find Nodes** block, and besides **Content** click on **Add**. Save the context.



4. Navigate to the **Locations** view and see if it doesn't look a little friendlier:



What just happened?

Let's take a look at what this `close2u` module does and how it interacts with our view.

First, the view creates a block that can be placed on any page. If you're used to Drupal 6's `hook_block` which provides all the information for a block in a single function, you should understand Drupal 7's block system. It's been split up into the actions required to define a block by adding the action to the hook name. For this block, we've created a `hook_block_info` that defines the blocks for the module and `hook_block_view`, which does the heavy lifting of putting the block together:

```
function close2u_block_info($delta = 0) {
 $blocks = array();
 $blocks['find_node'] = array(
   'info' => t('Close To You - Find Nodes'),
     'status' => 1,
   'cache' => DRUPAL_NO_CACHE,
 );
```

```
return $blocks;
}

function close2u_block_view($delta = 0) {
  $block = array();
switch ($delta) {
default:
      $block['subject'] = "Close To You";
      $block['content'] = close2u_page($delta);
  }
return $block;
 }
```

`close2u_block_view` calls a function `close2u_page` to do the action that initiates our block.

```
function close2u_page($delta) {
module_load_include('inc', 'uuid', 'uuid');
  $list_id = "close2u-" . uuid_generate();
drupal_add_js(array("close2u" => array("instances" => array($list_
id))), "setting");
drupal_add_js(drupal_get_path("module", "close2u") . '/close2u.js');
return theme("close2u_container", array("list_id" => $list_id, "delta"
=> $delta));
}
```

This function references the `uuid` module to create a unique ID for our block listing. Calling `drupal_add_js` with an array and a second argument of setting will add this `uuid` to our `Drupal.settings` object in JavaScript at the frontend. We'll use that in the file referenced in the next line, `close2u.js`. We then theme the container that will hold the list of nodes close to us. We've defined that the container in the hook theme and the file to generate the HTML in the templates folder.

```
function close2u_theme() {
  $path = drupal_get_path("module", "close2u") . "/templates";
  $items = array();

  $items['close2u_container'] = array(
    "template" => "close2u_container",
    "arguments" => array("uuid" => NULL, "delta" => NULL),
    "path" => $path,
  );
  $items['close2u_list_item'] = array(
    "template" => "close2u_list_item",
    "arguments" => array("result" => NULL),
```

```
    "path" => $path,
  );
return $items;
}
```

We've also defined a second theming function that we will use to theme the individual items. Our theme file in the `templates` folder is called `close2u_container.tpl.php`.

```
<div class="close2u-enter-location-container" style="display:none;">
  <form action="/close2u/address" method="get" accept-charset="utf-8">
    <label for="close2u-enter-location-text">
Enter an Address or Postal Code</label>
    <input id="close2u-enter-location-text" name="close2u-enter-
location-text" placeholder="Enter an Address or Postal Code">
    <p><input type="submit" value="find &rarr;"></p>
  </form>
</div>
<div class='close2u-container' id='<?php echo $list_id;?>' rel='<?php
echo $delta; ?>'></div>
```

It creates a container for our JSON call and contains a form to enter an address if our client-side geolocation fails.

The JavaScript file `close2u.js` defines the `Drupal.behavior` object that pulls all of this together. Let's take a look at it function-by-function. The basis for all Drupal 7 `Drupal.behavior` files are the `attach` and `detach` methods. We've talked about them in a previous chapter so I won't go over their function. Sufficient to say, the `attach` method gets everything started.

```
Drupal.behaviors.close2u = {
  attach: function(context) {
    if (Drupal.settings.close2u.origin == undefined) {
      if (navigator.geolocation) {
        Drupal.settings.close2u.origin = {
longitude: null,
latitude: null
};
        jQuery(Drupal.behaviors.close2u)
.bind("locationChange",
Drupal.behaviors.close2u.locationChangeHandler);
        navigator.geolocation.getCurrentPosition(
Drupal.behaviors.close2u.saveOrigin,
Drupal.behaviors.close2u.locationFail);
        Drupal.settings.close2u.watchId =
          navigator.geolocation.watchPosition(
Drupal.behaviors.close2u.saveOrigin);
```

```
        } else {
          Drupal.behaviors.close2u.locationFail();
        }
      }
    },
```

In order to determine distance, you need an origin and a destination. The destinations are, obviously, our node locations. The origin is the user. We'll be storing the user's origin in the `Drupal.settings` object. If that origin has not been defined, the function attempts to create it. If the browser supports geolocation, we define the latitude and longitude as null and bind a `location change` event responder to the `Drupal.behaviors.close2u` object. If this event is triggered, the `locationChangeHandler` function will attempt to handle the event. `navigator.geolocation.getCurrentPosition` is the client-side call that attempts to discern a location in the client's browser. The first function references execute if the attempt is successful, the second, if it is unsuccessful. We then put a `watchPosition` on the client's browser. If the client is using a handheld, it's helpful that we respond to their movements as they get closer or farther away. If the client's browser does not support geolocation, execute the same function as a failed geolocation call.

```
saveOrigin: function(position) {
    if (position) {
      Drupal.settings.close2u.origin = position.coords;
      Drupal.settings.close2u.origin.timestamp = position.timestamp;
      jQuery(Drupal.behaviors.close2u).trigger("locationChange");
    }
  },
```

The `saveOrigin` function responds to the `Navigator.geolocation` request. It receives a position object that has the longitude and latitude coordinates as well as some other information.

```
{
accuracy: 38
altitude: null
altitudeAccuracy: null
heading: null
latitude: 39.8624353
longitude: -76.0532586
speed: null
timestamp: 1315139386335
}
```

For some handhelds, you'll also get altitude, speed and other information about where you are. We store this information in the `Drupal.settings.close2u` object and then trigger a `locationChange` event on our behavior object. The `locationChange` event executes the `locationChangeHandler` function.

```
locationChangeHandler: function(evt) {
    if (Drupal.settings.close2u.origin.longitude != null && Drupal.
settings.close2u.origin.latitude != null) {
```

First, we make sure the longitude and latitude have values.

```
      jQuery.each(Drupal.settings.close2u.instances, function(idx,
value) {
```

We loop over the block instances for `close2u`. We store them in the `instances` object.

```
        url = jQuery("#"+value).attr("rel").replace(/_/g, "/");
        jQuery("#"+value).load("close2u/"+url, Drupal.settings.
close2u.origin, Drupal.behaviors.close2u.locationListHandler).
addClass("close2u-processed");
      });
```

For every instance, we execute a `jQuery.loadajax` request and give it the longitude and latitude from our origin. We'll see the response to this request in a minute. We dispatch this request and move on for it to load in the background.

```
      // gmap module integration
      if (Drupal.settings.gmap.close2u != undefined) {
        this.gmapObject = Drupal.gmap.getMap("close2u");
        //center and zoom
         this.gmapObject.map.setCenter(new GLatLng(Drupal.settings.
close2u.origin.latitude, Drupal.settings.close2u.origin.longitude));
        this.gmapObject.map.setZoom(11);
      }
```

We want to re-center the map so that the center is the user's location and then set the `setZoom` value to around `10` or `11`, which will give us a view of the metro area surrounding the user's location.

```
if (Drupal.settings.gmap.close2u != undefined &&
   (Drupal.behaviors.close2u.markers == undefined ||
   Drupal.behaviors.close2u.markers.length == 0)) {
  Drupal.behaviors.close2u.markers = {};
  for(i in Drupal.behaviors.close2u.gmapObject.vars.markers) {
  Drupal.behaviors.close2u.markers[Drupal.behaviors.close2u.
gmapObject.vars.markers[i].rmt] = Drupal.behaviors.close2u.gmapObject.
vars.markers[i];
        }
      }
    } else {
      jQuery(".close2u-enter-location").show();
    }
  },
```

When we created the view, we tell it to use the node ID as the RMT value for the pointer click. This node ID value is stored in every marker object on the page, but we can't just do a search for them. So we need an object that references the node ID so when the node IDs are clicked on the page, we can highlight the marker.

In the `close2u.module` file, we've set up a `hook_menu` that will handle the `.load` request for nodes close to the client.

```
function close2u_menu() {
  $items = array();
  $items['close2u'] = array(
    "page callback" => "close2u_page",
    "page arguments" => array(2),
    'access callback' => TRUE,
    "type" => MENU_CALLBACK,
  );
  $items['close2u/find/node'] = array(
    "page callback" => "close2u_find",
    "page arguments" => array(2),
    'access callback' => TRUE,
    "type" => MENU_CALLBACK,
  );
  $items['close2u/marker/%node'] = array(
    "page callback" => "close2u_marker_retrieve",
    "page arguments" => array(2),
    'access callback' => 'node_access',
    'access arguments' => array('view', 2),
    "type" => MENU_CALLBACK,
  );
return $items;
}
```

The `close2u/find/node` menu item will respond to our `jQuery.load` request by triggering the `close2u_find` function. Let's take this line by line.

```
//default search is nodes, but you can also search for users by type =
'uid'
function close2u_find($type = "node", $origin = NULL) {
  $toReturn = "<ul class='close2u-list'>";
if ($origin == NULL) {
if (isset($_REQUEST['longitude']) &&isset($_REQUEST["latitude"])) {
      $origin = $_REQUEST;
    }
elseif (property_exists($GLOBALS, "origin")) {
global $origin;
```

```
    }
else {
drupal_set_message("in order to find something close to you, I must
have an origin. Set \$GLOBALS\[\'origin\'\] or use as second argument
to close2u_find.", "error");
return FALSE;
    }
  }
```

We begin the list to be returned, and we set the origin, if it's not already set.

```
  $query = db_select("location", "l")->fields("l", array("lid",
"longitude", "latitude"));
```

Using Drupal 7's new database API, we create a database query object that will get details of the `location` field:

```
module_load_include("inc", "location", "earth");
  $query->addExpression(earth_distance_sql($origin['longitude'],
$origin['latitude']), "distance");
```

Calculating the distance is not as simple as subtracting one value from the other. The earth is a sphere and we calculate distance along the outside of that globe in a unit called **radians**. There's some complex math at work so we add a calculated field called distance, with the math contained in the earth functions of the location module. The `earth.inc` include has all of that worked out in advance for us.

```
  $query->join("location_instance", "li", "li.lid = l.lid");
  $query->fields("li");
  $max_distance = (array_key_exists($origin['max_distance']) ?
    $origin['max_distance'] : variable_get("close2u_default_max_
distance", NULL)
  );
if ($max_distance != NULL) {
    $query->having("distance < :max_distance ", array(
        ":max_distance" => $max_distance,
      ));
  }
```

We can add a `max_distance` to the request if we want to exclude nodes that are over a certain distance from the origin. For this experiment, I have the default `max_distance` set to `null`. Eventually, we should write a `hook_block_config` function that sets the system variable, `close2u_default_max_distance` and by the time of publication of this book, the module may have that. But for development purposes, we always want to show something in the results no matter how far it is from the user.

```
  $query->orderBy("distance");
```

Sort the returned objects by the calculated distance.

```
$query->range(0, 20);
```

Limit the results to `20` nodes.

```
$foreign_alias = substr($type, 0, 1);
$foreign_key   = $foreign_alias . "id";
$join_clause   = "li." . $foreign_key . " = " . $foreign_alias . "."
. $foreign_key";
$query->join($type, $foreign_alias, $join_clause);
```

So, this is a little complicated. We want to use a query that will work for either users or nodes so we create a join based on the word "node" and do a SQL join on the nodes table to get the node ID (`nid`).

```
$results = $query->execute();
```

The preceding command executes the query.

```
while ($result = $results->fetchObject()) {
    $result->node = node_load($result->nid);
    $toReturn .= theme("close2u_list_item", array("result" =>
$result));
  }
```

Retrieve and theme the results.

```
$toReturn .= "</ul>";
echo $toReturn;
exit();
}
```

Close the list, print the list, and echo the results. We then want to stop execution because we don't need the entire page's HTML, just the list itself.

In our JavaScript file, we process the incoming list when it loads:

```
locationListHandler: function(evt) {
    jQuery(".close2u-list-item")
      .not(".close2u-list-item-processed")
      .find("a.close2u-click-marker")
      .click(Drupal.behaviors.close2u.locationListItemClickHandler)
      .attr("href", "javscript:;")
      .addClass("close2u-list-item-processed");
    //first in list should be closes, click it.
    jQuery(".close2u-list-item:first-child a").click();
},
```

We select the list items that have not yet been processed. Select the link inside the list item and on click, give them a function to handle the click. We remove the direct link to the location's node and trigger the click action of the first one, or rather the one that's closest to the user's location.

```
locationListItemClickHandler: function(evt) {
    if(evt) evt.preventDefault();
    google.maps.Event.trigger(Drupal.behaviors.close2u.
markers[jQuery(this)
      .parent().attr("rel")].marker, "click");
    return false;
  },
```

In any event handler, the function's arguments are an event object. For this event, we want to prevent the default action and substitute our own triggers. When a location is clicked, we trigger the event of its complimentary marker being clicked. The Google Map fires another AJAX request to the `close2u` module that we've described in the **RMTcallback path** setting (**close2u/marker**) and append the node ID to that request. The request will be `close2u/marker/32` for node ID as 32. The receiver of that function is very simple:

```
function close2u_marker_retrieve($node) {
  echo drupal_render(node_view($node, "marker"));
  exit();
}
```

Grab the Node ID from the URL and view it in the `marker` build mode. Wait! There is no `marker` build mode! Well that's defined by the last two functions in the module:

```
function close2u_ctools_plugin_api() {
list($module, $api) = func_get_args();
if ($module == "ds" && $api == "ds") {
return array("version" => "1");
  }
}
function close2u_ds_view_modes_info() {
  $export = array();

  $ds_view_mode = new stdClass;
  $ds_view_mode->api_version = 1;
  $ds_view_mode->view_mode = 'marker';
  $ds_view_mode->label = 'Marker';
  $ds_view_mode->entities = array(
    'node' => 'node',
  );
  $export['mobile'] = $ds_view_mode;

return $export;
}
```

These two functions implement a new Display Suite build mode called `Marker`. The first function tells CTools which version of the Display Suite API to use. The second creates the build mode and returns it to Display Suite for using on every node. Every node will now be able to have its own "marker" display in Display Suite. If there's no specific marker display, Display Suite will show the default build.

Finishing the page

We still have a couple of lingering issues with the page. The first of which is that we haven't really coded any alternative to geolocation. What about browsers that don't support the `navigator.geolocation` object or people in witness protection programs that ask you to click **Don't Allow** when you give them a location dialog? We have to plan for not being able to use the geolocation in the browser. Let's create that code together now.

Time for action – finding the closest franchise the hard way

There are multiple reasons to code for situations where the location object is not available. Older versions of Internet Explorer and some marginal browsers do not support it. But there will also be times when the location is found incorrectly or you may want to find franchise locations that are not near the the computer or the handheld's current location. In these cases, the dependance on this feature is an inconvenience to the user. We need to write the code for that use case:

1. Create a form in the `templates/close2u_container.tpl.php` file to handle user input of location data. Add the following lines to the top of the template:

```
<div class="close2u-enter-location-container"
style="display:none;">
  <form action="/close2u/address" method="get" accept-
charset="utf-8" id="close2u-enter-location" name="close2u-enter-
location">
    <label for="close2u-enter-location-text">Enter an Address or
Postal Code</label>
    <input id="close2u-enter-location-text" name="close2u-enter-
location-text" placeholder="Enter an Address or Postal Code">
    <input type="hidden" name="list_id" value="<?php echo $list_
id;?>" id="list_id" class="close2u-enter-location-list-id">
    <p><input type="submit" value="find &rarr;"></p>
  </form>
</div>
```

2. Edit the `close2u.js` file as follows. Delete the `alert("location fail!");` line and add the following lines to the `locationFail` function to show the location form. Bind a `submit` event to the form so it can be submitted via AJAX:

```
    jQuery(".close2u-enter-location-container")
      .show()
      .find("form")
      .submit(Drupal.behaviors.close2u.userEnterLocationHandler);
  },
```

3. Edit the `close2u.js` file as follows. Create a `userEnterLocationHandlerfunction` to handle user-submitted location data and send it to the `close2u` module:

```
  userEnterLocationHandler: function(evt) {
    if (evt) evt.preventDefault();
    jQuery.getJSON(jQuery(this).attr("action"), jQuery(this).
serialize(), Drupal.behaviors.close2u.saveOrigin);
    return false;
  }
```

4. Create an entry in the `hook_menu` for the function that will mimic the call to the `geolocation` object:

```
function close2u_menu() {
  $items = array();
  $items['close2u'] = array(
    "page callback" => "close2u_page",
    "page arguments" => array(2),
    'access callback' => TRUE,
    "type" => MENU_CALLBACK,
  );
  $items['close2u/find/node'] = array(
    "page callback" => "close2u_find",
    "page arguments" => array(2),
    'access callback' => TRUE,
    "type" => MENU_CALLBACK,
  );
  $items['close2u/marker/%node'] = array(
    "page callback" => "close2u_marker_retrieve",
    "page arguments" => array(2),
    'access callback' => 'node_access',
    'access arguments' => array('view', 2),
    "type" => MENU_CALLBACK,
  );
  $items['close2u/address'] = array(
```

```
        "page callback" => "close2u_address_entry",
        "page arguments" => array(2),
        'access callback' => TRUE,
        "type" => MENU_CALLBACK,
    );
return $items;
}
```

5. Edit the `close2u` module. Create the function to mimic the call to the `geolocation` object:

```
function close2u_address_entry() {
  module_load_include("module", "gmap", "gmap");
module_load_include("inc", "location", "geocoding/google");
  $response = google_geocode_location(array("street" => $_
REQUEST['close2u-enter-location-text']));
  if (is_array($response)) {
    jsonjsonechojson_encode(array("coords" => array("longitude"=>
(float)$response['lon'], "latitude" => (float)$response['lat']),
"timestamp" => time(), "list_id" => $_REQUEST['list_id']));
  } else {
    jsonjsonechojson_encode(array("error" => "<h1>Google is unable
to find the location you entered.</h1>"));
  }
  exit();
}
```

6. Alter the `saveOrigin` function in `close2u.js` to handle error messages:

```
  saveOrigin: function(position) {
    if (position) {
      if (position.error != undefined) {
        $("#"+position.list_id).html(position.error);
      } else {
        Drupal.settings.close2u.origin = position.coords;
        Drupal.settings.close2u.origin.timestamp = position.
timestamp;
        jQuery(Drupal.behaviors.close2u).
trigger("locationChange");
      }
    }
  },
```

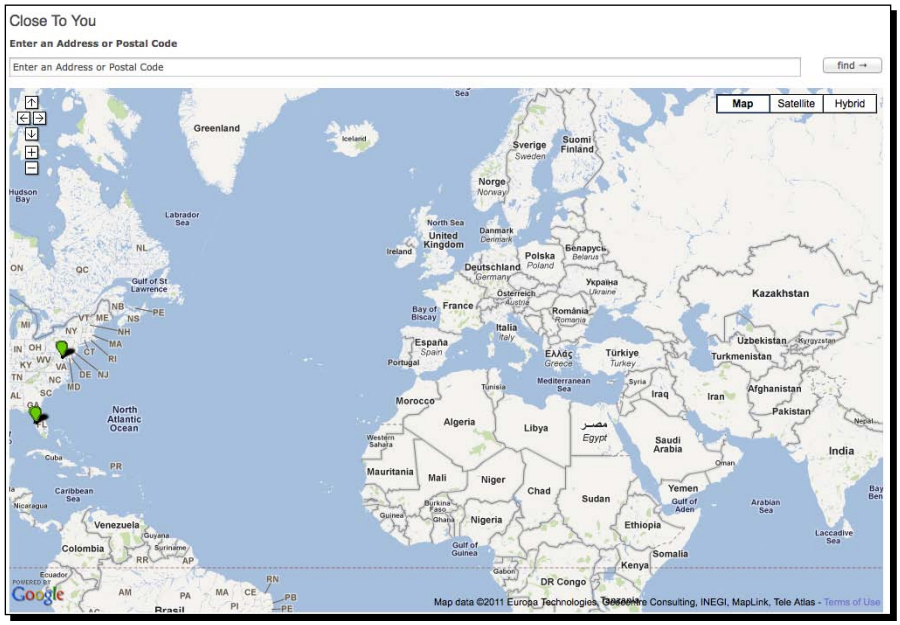
7. Edit the `sites/all/themes/dpk/css/styles.css` file and add the following lines:

```
.close2u-list {
    -moz-column-count: 4;
    -moz-column-gap: 1em;
    -webkit-column-count: 4;
    -webkit-column-gap: 1em;
    column-count: 4;
    column-gap: 1em;
  }
```

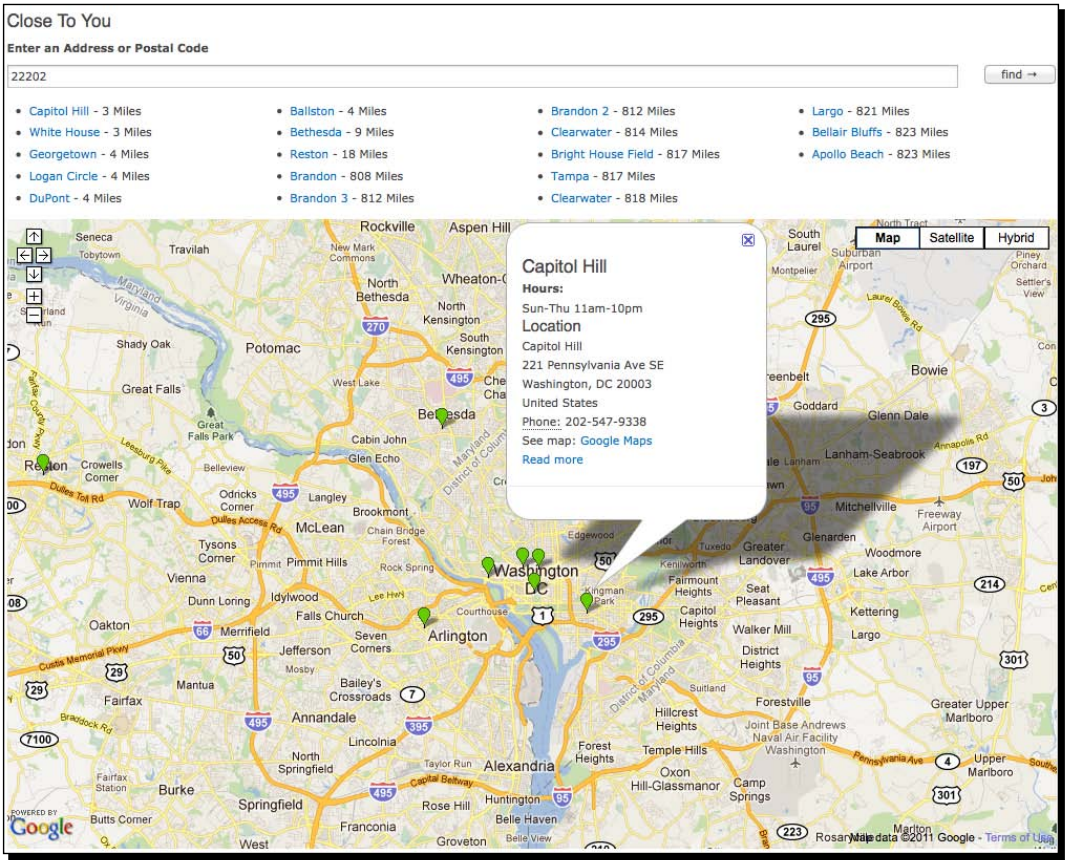
8. Edit the `sites/all/themes/dpk_mobile/css/global.css` and add the following lines:

```
.close2u-list {
    -moz-column-count: 2;
    -moz-column-gap: 1em;
    -webkit-column-count: 2;
    -webkit-column-gap: 1em;
    column-count: 2;
    column-gap: 1em;
  }
```

9. Navigate to `http://dpk.local/locations`. If the browser asks if you want to allow it to use geolocation, click on **Don't Allow**. You should see something like what is seen in the following screenshot:



10. Enter **22202** into the **Enter an Address or Postal Code** text field and click on **find**:



What just happened?

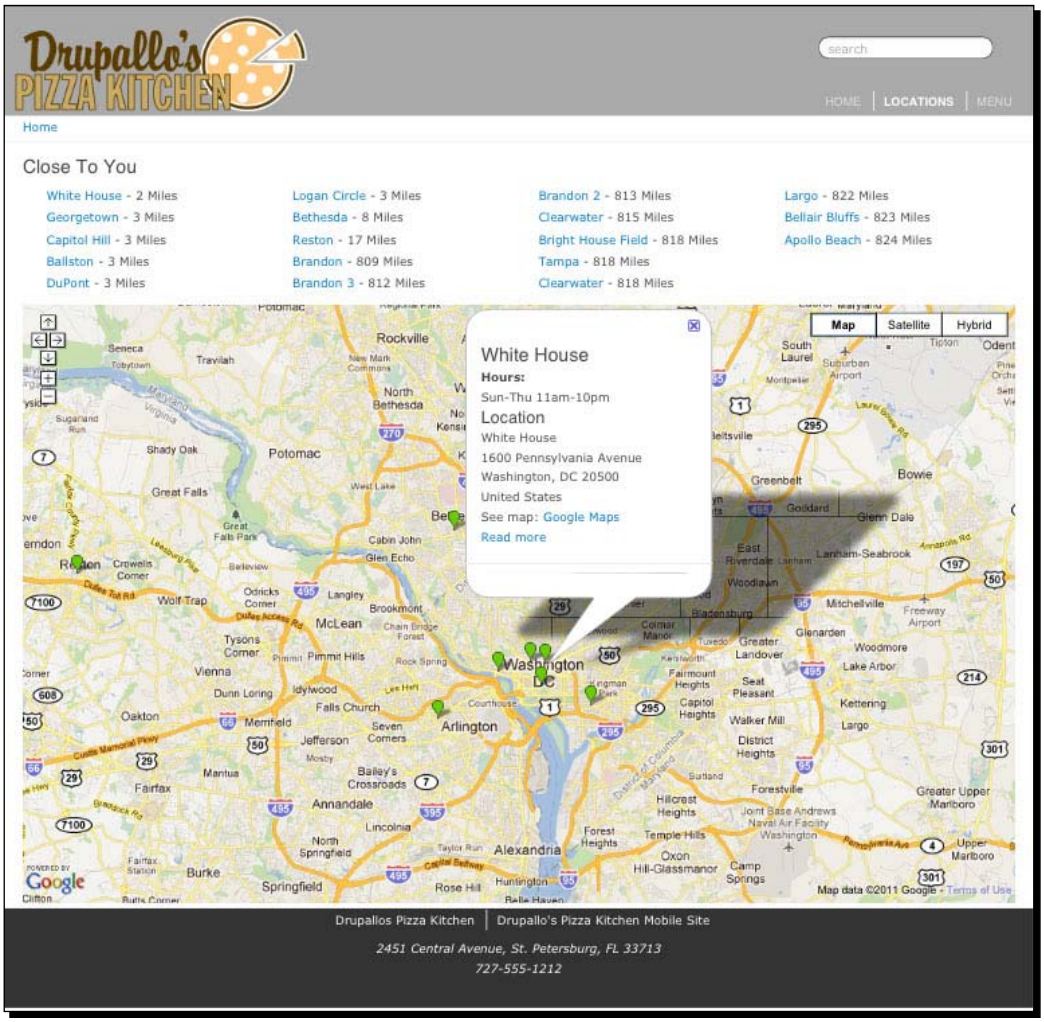
In the first step, we created a standard HTML form to send address data to the module for use in geocoding. In the second step, we intercepted the form's `submit` event and submitted the form via AJAX in exchange for a JSON object.

Step 4 creates a `hook_menu` that will call a function, `close2u_address_entry`. In the function, in step 5, we mimic the activity of the `geolocation` object by returning an object with its longitude and latitude values. If there's a problem with geocoding the address, we get an error.

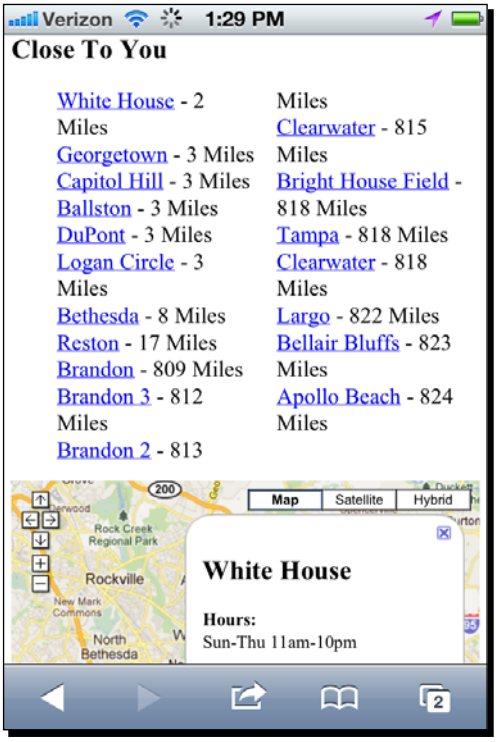
In step 6, we alter the `saveOrigin` JavaScript function to handle error objects as well as geolocation responses.

Finally, we added some CSS code that adds columns to the list of responses. It cleans up the look a little bit. It adds four columns for the desktop version and two columns for the handheld.

Taking a look at this page on your mobile device you can see how well everything we've done works with mobiles. If you click the Google Map links, the links should open in the mobile device's default maps application. The following screenshot is of the desktop version:



The following screenshot is of the mobile version:



Pop quiz

1. The JavaScript object that allows you to read a browser's position is:
    a. `location.storage`
    b. `navigator.geolocation`
    c. `window.location.href`
    d. `document.location.href`
2. The module that allows you to map locations returned in a view is:
    a. `views_location`
    b. `gmap`
    c. `google_maps`
    d. `locale`

3. Address-based locations must first have longitude and latitude added to their data to place the items on a map:
    a. True
    b. False
4. The process of adding longitude and latitude to an address is called:
    a. Mapping
    b. Geocoding
    c. Location sharing
    d. Spelunking
5. You can use PHP to display a field in Display Suite by adding a:
    a. PHP file to the module
    b. New template
    c. Code field
    d. None of the above
6. To calculate the distance between two objects:
    a. You simply subtract the longitude and latitude values of one from the other
    b. Distance cannot be calculated by any method
    c. Use a complex set of radian math to figure the distance given the curve of the earth
    d. None of the above

Summary

In this chapter, we've learned a lot about the **Location** and **GMap** modules and how to use them as building blocks to create a rich mobile (and desktop) user experience. We added location information to node objects and then produced a view of those objects. We've geocoded addresses and learned how to get by, when automatic geolocation isn't available to the client.

In the next chapter, we'll take a look at the **Services** module and other ways to retrieve data from Drupal with API calls.

8
Services with a Smile

As completely awesome as web applications are, there are still things that a web application cannot do. The JavaScript APIs, such as file uploading, still image capturing, and video and audio capturing are still not written in the HTML5 spec and are likely to take at least another year or so from the date of writing this book.

For this chapter, we are going to primarily concern ourselves with a REST interface and JSON data, but the services module accommodates XML-RPC and several other protocols and hopefully if you have an odd protocol, by the end of this chapter you will feel comfortable enough to write your own protocol and/or interface.

So with that in mind, in this chapter we will cover the following topics:

- Enabling the Services module
- Enabling a standard REST service
- Learning how to retrieve multiple types of data (JSON, XML, and so on)
- Retrieving nodes via the REST request
- Authenticating a user via the REST request
- Creating a user via the REST request
- Creating a customized REST service
- Using a customized REST service to power a jQuery mobile site

Note that for some of these exercises, you will need a copy of Firefox and a plugin called POSTER (`https://addons.mozilla.org/en-US/firefox/addon/poster/`). It will allow you to test the REST interface.

Using Drupal to power your native application

The first big native application that I remember using was Facebook. The mobile Facebook application was awesome and allowed you to do almost everything that the website did, but with an interface specially designed for interaction on a hand-held device. Indeed, as popular as Facebook at the time already was, putting a tool that powerful in the hands of iPhone owners all over the world helped Facebook, quite literally, change the world.

For some applications, it may be necessary to write a custom native application for the handhelds to interact with your Drupal website. Android Market and the Apple AppStore have made billions for their respective companies by creating an ecosystem of native hand-held applications. For this application to interact with an Internet-based backend, it's usually better to pick a single method of communication and use that method throughout your application so that you can write a single interface to call and interpret requests to and from the backend site. For this series of examples we will use REST and JSON responses, but as you will see, **Services** module accepts many different protocols and can respond with a few different encodings.

Jargon watch

XML (**eXtensible Markup Language**): In simple words, XML was designed to transport data so as to retain integrity across platforms, programming languages, spoken and written languages, and server platforms.

AJAX (**Asynchronous JavaScript and XML**): A development pattern pioneered by Google and Microsoft where the client's browser exchanges information with the server. Originally that information was in XML form and needed to parse down from XML on both the client and the server side. However, it has come to mean any asynchronous exchange of data and its current preferred response is JSON.

JSON (**JavaScript Object Notation**): A way of storing data so that it can be executed by a JavaScript interpreter and return a proper JavaScript variable object. The values of this object can be accessed by the script. Direct JSON encoding/decoding was added to PHP in version 5.2 and many AJAX.

JSONP (**JSON with Padding**): There are security issues with handing off JSON among domain names. Sometimes it is helpful to "pad" the JSON with a function call that will execute once the object is loaded on the client-side.

REST (**Representational State Transfer**): A way of writing web services that allow the creating, reading, updating, and deleting (CRUD) of records by getting and posting data to and from a website's URL, as well as adding several other customized web-data transactions such as put and delete. REST provides a consistent and stateless interface to allow clients to interact with website data. REST is the foundation of modern mobile applications and the way they communicate with the Internet at large.

XML-RPC (**XML-based Remote Procedure Call**): It passes the XML-based data to and from a website's resource with the intent of some sort of server process acting on that data. XML data is parsed on both the ends and translated into server actions and data storage.

API (**Application Programming Interface**): An API allows multiple platforms and applications to speak a common programming language for the exchange of data and the acting of both client and server processes on exchanged data.

Let us enable the **Services** module and take a look at how it works.

Time for action – creating a REST service

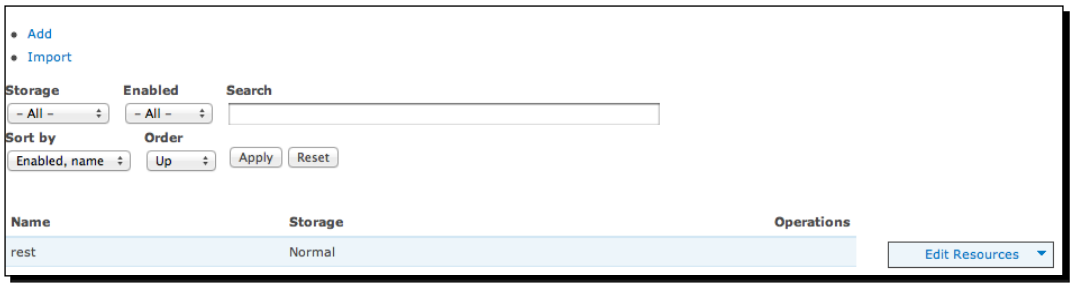
Version 2.x and 3.x of the **Services** module vary greatly. We will be using 3.x for all the following examples.

1. In order to download and enable the `Services` and `Services_rest` modules, open a terminal window and enter the following commands:

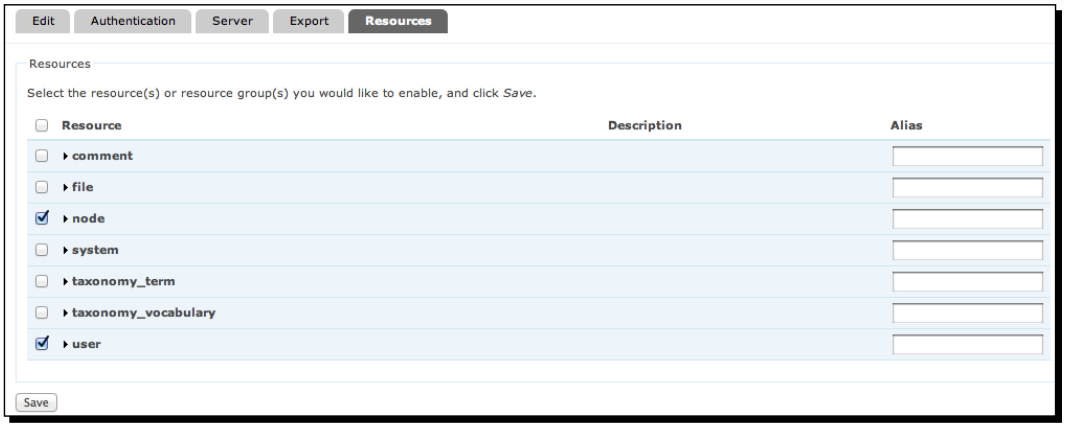
```
cd ~/Sites/dpk
drush dl services
drush pm-enable services rest_server
```

2. Navigate to **Admin** | **Structure** | **Services**. Click on the **Add New Service**. Name the new service as **rest**. Choose **REST** as the **Server** and enter **rest** for **Path to endpoint** as well. Check the **Session authentication** checkbox to allow session authentication. Click on the **Save** button to save the new service, as shown in the following screenshot:

3. Saving the service will take you back to the services list. From the list choose **Edit Resources**, as shown in the following screenshot:

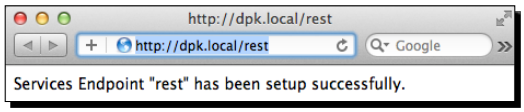


4. Enable the **node** and **user** resources and save the changes.



5. Click on the **Server** tab. Make sure that all the response formatters and the request interpreters are enabled. Save the server settings.

6. Navigate to `http://dpk.local/rest`, as shown in the following screenshot. You now have the basic REST services enabled for the site:



What just happened?

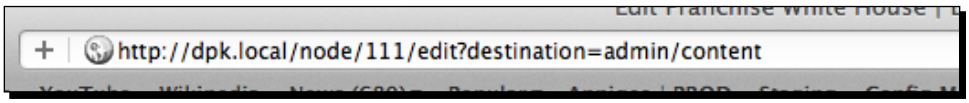
We first downloaded, installed, and enabled the **Services** module. The **Services** module has two default protocols: XML-RPC and REST.

We then created a REST endpoint at the `http://dpk.local/rest` URL by enabling the node and user resources.

Time for action – testing your new REST service

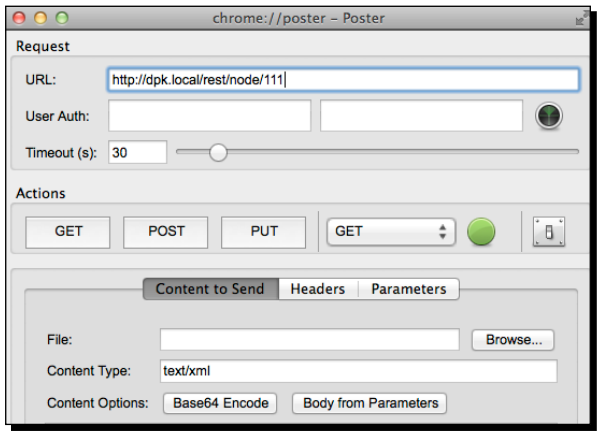
Let us test the new REST resource with some REST calls.

1. Navigate to **Admin | Content** and click **Edit** on the first item in the list. The URL should contain that item's node ID. Remember this node ID. The node ID for this URL is 111. Yours will be different:

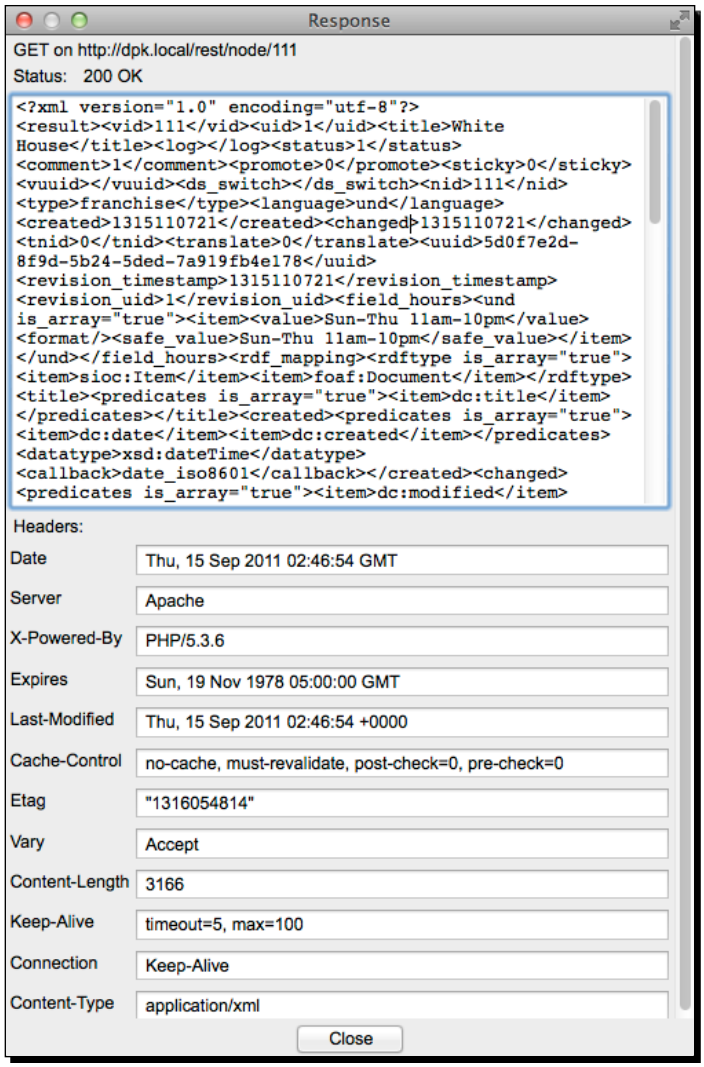


2. If you have not downloaded Mozilla Firefox yet, then go to `http://getfirefox.com` and install the Poster Firefox add-on at `https://addons.mozilla.org/en-US/firefox/addon/poster/`.

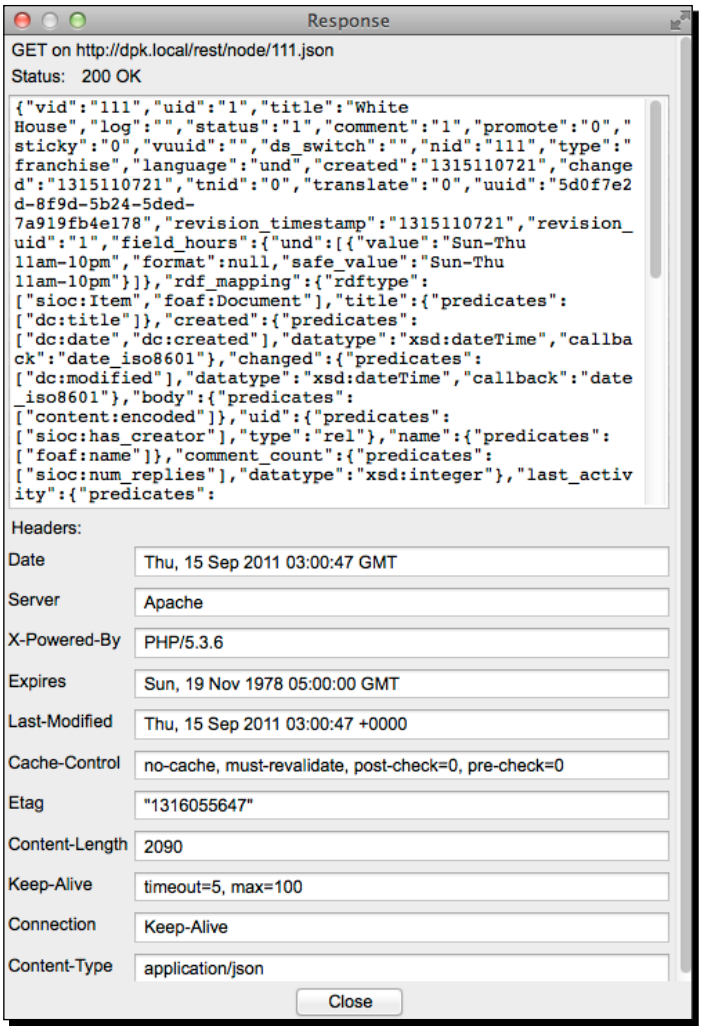
3. Launch Firefox and choose **Tools | Poster**. Under **URL** enter this link: `http://dpk.local/rest/node/` and then the node ID that you have obtained from step 1, which in this case is 111. Now the node URL is `http://dpk.local/rest/node/111`. This is shown in the following screenshot:



4. After you have entered the address, click on the **GET** button. If your node service is set up correctly, it will respond with an XML containing different properties of the node:

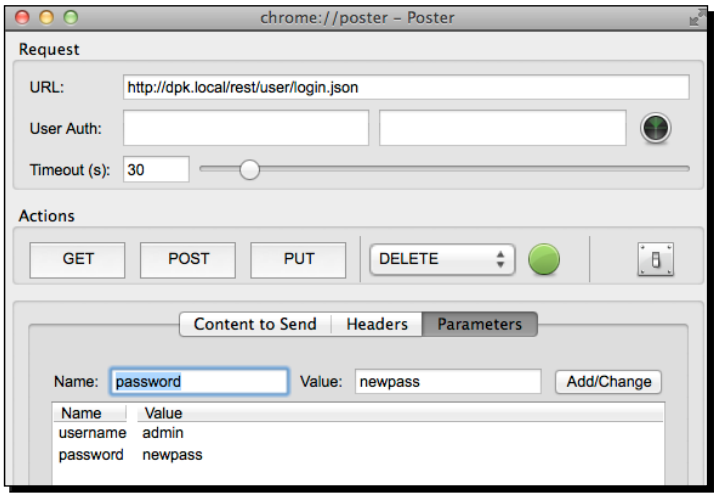


5. What is returned is basically all the properties you would get, had you done a `node_load(NODE_ID)` from within the Drupal API. Add the extension `.json` to the URL so that it becomes `http://dpk.local/rest/node/111.json`. Predictably, what is returned is the JSON data instead of the XML data. The same data, but with two different formatters:

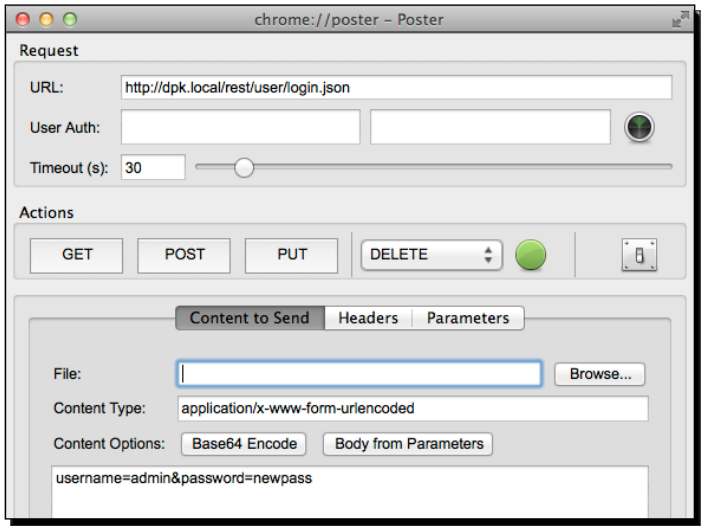


6. Open a new window and navigate back to the website. If you are still logged in, log out by accessing `http://dpk.local/logout`.

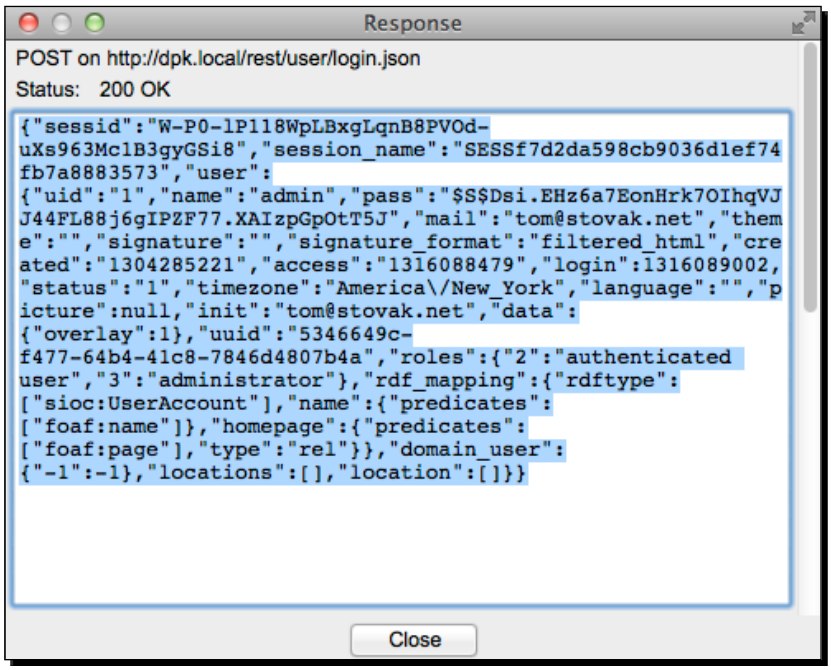
7. Move back to the **Poster** window. Change the URL to `http://dpk.local/rest/user/login.json`. Click on the **Parameters** tab and enter **username** as the **Name**, and the username of your Drupal admin user. Click on the **Add/Change** button. Type **password** in the first box and in the second box, the password of your admin user. Now, click on the **Add/Change** button:



8. Click on the **Content to Send** tab. Click on the **Body from Parameters** button. The username and password you just entered should show up in the body in a serialized string, as shown in the following screenshot:



9. Click on the **POST** button and the following will be shown:



10. This is the response to a successful login API call.

What just happened?

After we created a services endpoint, we began testing that endpoint for responses. By changing the extension of the URL that we were requesting, we can change the way data is sent back to us. The default format is XML, but we can change that to JSON easily by appending `.json` to the URL string.

In the last part of the exercise, we actually logged in via the REST service and obtained a session ID. We can use similar calls to log in a user and allow them access to privileged services, such as content editing and posting comments. We will talk a little more about this later in the chapter.

APIs: The future of the interactive web

The Federal Communication Commission's (FCC) website was consistently voted the worst website in the government for many years. The problems were obvious to any casual visitor. Content was divided into the FCC's bureaus and departments. In order to find the information that you needed, you had to know which department and bureau handled your specific concern, which most people didn't know. The top-down regimented governmental design with little or no search functionality made for a horrible user experience for new visitors.

In 2010, the Seabourne group received the bid to help with the redesign. However, it turned out that the internal politics of managing the website was what dictated the current design. Managers of the bureaus could not agree on formats for data interchange and getting something posted to the website was a Herculean task that involved multiple calls to multiple departments.

Taking a "top-down" approach and dictating that all bureaus and departments store their data in a compatible format or create some sort of conversion algorithm, the Seabourne group created a content API for the new website that allowed different departments to authenticate and post data to the website using the tools they had at their disposal. The new website that went live in early 2011 featured a faceted search based on Drupal's Apache Solr implementation and many departments can now post and retrieve live data using the backend content API. What is more, the agency can now open its data to other consumers to combine the data in new and interesting ways. A few of those ways are posted on the website (`http://www.fcc.gov/`) if you care to see more.

Imagine a world where interacting with the government was as easy as getting and posting requests to a predictable API. As a programmer/developer/UI enthusiast, how much innovation would you be able to bring to the user interactions? You don't have to dream too much to understand the power of a published API and what that can do to change your business and its customers.

The smart guys at Seabourne have been kind enough to open source the module they created for the website. The ContentAPI module allows a direct interface with the content of your website, and using this module you can open your site data to other programs reusing every node on the site in other contexts.

What does that have to do with us? Well, what we have done with this **Services** module is create an API to this website's data. APIs turn data into a common format, which is globally accessible by any application. Even if your website is just content, you can benefit by enabling an API to your website. Facebook's API enabled the success of games such as **Farmville** and **Words with friends**. Much of Google's success can be attributed to its open APIs and self-service style framework.

Using this new services framework we can create a mobile web interface that uses JSON calls to navigate to the content of the site.

However, before we do that, in the last example we created a custom module that when requested, returned a snippet of rendered HTML when we clicked on a marker on the map. Let us change that module to work with the **Services** module.

Customized services

Services are a great way of exposing data to the web through an API. However, what if it is not the type of data that we need, or what if we need node data, but we need it in a rendered form? In a few short lines of code, we can create a customized service that will return rendered nodes in a specified build mode.

In order to interact with the REST server we can basically use two hooks:

- `hook_rest_server_request_parsers_alter`
- `hook_rest_server_response_formatters_alter`

The `parsers_alter` hook modifies data going into the REST server, and the `formatters_alter` hook allows you to change data coming out of the REST server. Let us now write a formatter.

Time for action – custom REST service formatter

This formatter can be genericized to return the node in any format that you choose.

1. Make a new folder in your website's `sites/all/modules/custom` folder. If the custom folder does not exist then create one. Call the new folder `rendered_node`. Create three empty text files inside the `rendered_node` folder named as: `rendered_node.info`, `rendered_node.module` and `rendered_node.inc`. You can do this from the command line with the command touch, or you can do it with your favorite text editor.

2. Add the following lines to `rendered_node.info`:

```
name = Rendered Node
description = Service to return rendered nodes
package = Services
core = 7.x
php = 5.x

files[] = rendered_node.inc
```

3. Add the following lines to `rendered_node.module`:

```
<?php

function rendered_node_rest_server_response_formatters_
alter(&$formatters) {

  // Add a html response format.
  $formatters['html'] = array(
    'mime types' => array('text/html'),
    'view' => 'RESTRenderedNode',
    'view arguments' => array('format' => 'html'),
  'file' => 'rendered_node.inc'
  );

}
```

4. Add the following lines to `rendered_node.inc`:

```
<?php

classRESTRenderedNode extends RESTServerView {

  public function render() {
  switch ($this->arguments['format']) {
  case 'html':
  return $this->render_html($this->model);
      }
  return '';
    }

  public function render_html($data) {
    if (isset($_REQUEST['build'])) {
      $build = $_REQUEST['build'];
    }
    else {
      $build = "full";
    }
    echo drupal_render(node_view($data, $build));
  }
}
```

5. Clear the site's cache and enable the module.

6. Navigate to **Structure | Services | List** and under the active REST server, choose **Edit Resources**.

7. Under the **Server** configuration there should be a new **html** response formatter. Make sure it is enabled.

8. Using the node from the previous example, navigate to `http://dpk.local/rest/node/NODE_ID.html` substituting the node ID wherever appropriate:

What just happened?

We created a standard module with the `.info` file and a `.module` file. We used the `rest_server_response_formatters_alter` hook to add an HTML formatter for the REST server. The `.inc` file does the heavy lifting for the formatter. It is a responder class based on the `RESTServerView` class. Basically, it retrieves the node data from the argument and renders it based on the build. Default build is a "full" build, but we can use `?build=BUILDTYPE` in the URL to change the display suite build type.

Using these tools, you should be able to construct services for even the most complex of native applications, but in many cases, as in the case of the Pizza Kitchen, the cost of developing a native application is out of reach. In these situations, it may be helpful to construct a website that functions like a native application. That is where the jQuery Mobile comes in.

Have a go hero – creating REST service formatter

Create a REST service formatter that returns the `node_export` code to push the site content from one site to another using the Feeds module. Create a Feeds reader that will import the content into another site.

jQuery Mobile

More than a year ago, a project to bring jQuery into the touch-screen world had begun. This was the beginning of what is now known as the **jQuery Mobile** (**jQM**). jQM provides a touch-optimized interface for modern touch-screen devices at the same time limiting the amount of bandwidth that is needed to refresh the page. jQM also provides default theming that mimics a mobile device's native application look-and-feel and makes form elements more "touch-screen friendly" in the handheld's native browser.

Tim Cosgrove and Brian McMurray undertook an effort to bring this project into the Drupal space. The result is a module and base theme, **jQuery Mobile** and **jqm** respectively, that are fantastic starting points for using jQuery Mobile on your Drupal site.

jQM makes extensive use of the `data-*` attributes in HTML 5 markup to work its dark magic. HTML5 spec states that any attribute that begins with the word "`data-`" will be ignored by the browser. The browser does not try to interpret the data for layout or styling. jQM uses these attributes to define parts of the page for markup and then provides app-like transitions between loaded "page"s.

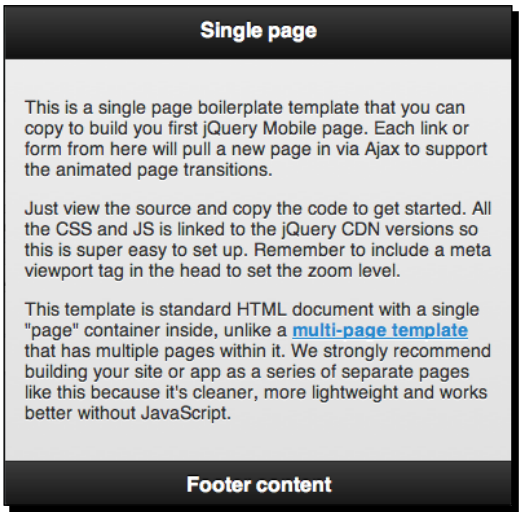
Default behavior for standard jQM links it to fetch the page via AJAX, search the page for the `data-role="page"` element and append that to the current HTML document's body. Using this method, if you have a JavaScript on one page that does not appear on the home page or if your `Drupal.settings` variable has different values on the new page, then they will be ignored. For those pages that load their own JavaScript and CSS, we will need to link them with the `data-ajax="false"` property set. This will keep jQuery from trying to load the page without all of its components in place.

The sample core of a jQM page looks something like this:

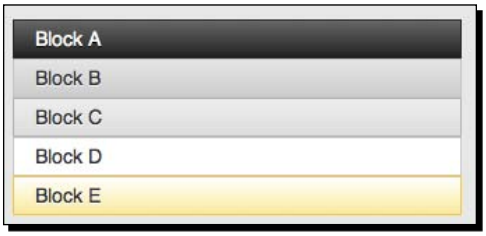
```
<div data-role="page">
  <div data-role="header">Single page</div>
  <div data-role="content">…</div>
```

```
    <div data-role="footer">Footer content</div>
</div>
```

Using this markup you will get a screen similar to the following screenshot:



jQM has five default themes (A–E). You can use any of the five in combination with each other in dialogs, headers, footers, and body content:



Time for action – using jQM as our base theme

jQuery mobile is a theme and a module. We will need to install both.

1. Open a terminal and enter the following commands:

```
cd ~/Sites/dpk
drush dl jqm jquerymobile
drush pm-enable jquerymobile
cd sites/all/libraries
git clone git://github.com/jquery/jquery-mobile.git
```

```
cd jquerymobile
make
export VER=`cat version.txt`
cp compiled/jquery.mobile.js "jquery.mobile-${VER}.js"
cp -R compiled/images .
cp -R themes/default/images .
drush vset jquerymobile_current_version `cat version.txt`
ln -s `pwd` ../../modules/contrib/jquerymobile
```

Carat v/s single quote v/s double quote

In shell scripting, the carat mark (the key typically placed to the left of the number 1 on your keyboard), the single quote (to the right of the semi-colon), and the double quote are not used interchangeably. The carat mark is used as "execute the command enclosed and return the result". In the preceding export command, `` `cat version.txt` `` reads the file named as `version.txt` and returns the contents of the file to the screen, which is captured by the carat-quote and returned into the VER variable. On the fourth-last line, the double quote does a variable substitution on the `${VER}` variable and returns the value with the string to rename the file with the version number. The single quote is used to quote items without variable and control-key substitutions. Double quote is used on items where there are escape or variable substitutions. Be careful while using the quote when you are typing these commands.

2. At this point **Drush** will ask you if you are sure that you want to set this variable. Choose **1** to indicate "yes". Your folder should now look similar to the following screenshot:

```
-rw-r--r--   1 stovak  staff     122 Dec 25 11:24 .gitignore
-rw-r--r--   1 stovak  staff   15099 Dec 25 11:24 GPL-LICENSE.txt
-rw-r--r--   1 stovak  staff     193 Dec 25 11:24 LICENSE-INFO.txt
-rw-r--r--   1 stovak  staff    1074 Dec 25 11:24 MIT-LICENSE.txt
-rw-r--r--   1 stovak  staff    7775 Dec 25 11:24 Makefile
-rw-r--r--   1 stovak  staff    4743 Dec 25 11:24 README.md
drwxr-xr-x   4 stovak  staff     136 Dec 25 11:17 build
-rw-r--r--   1 stovak  staff     530 Dec 25 11:24 combine.php
drwxr-xr-x  10 stovak  staff     340 Dec 25 11:36 compiled
drwxr-xr-x   4 stovak  staff     136 Dec 25 11:17 css
drwxr-xr-x  13 stovak  staff     442 Dec 25 11:17 docs
drwxr-xr-x   7 stovak  staff     238 Dec 25 11:17 experiments
drwxr-xr-x   4 stovak  staff     136 Dec 25 11:17 external
drwxr-xr-x   7 stovak  staff     238 Dec 25 11:44 images
-rwxr-xr-x   1 stovak  staff    2751 Dec 25 11:24 index.html
-rw-r--r--   1 stovak  staff   82750 Dec 25 11:44 jquery.mobile-1.0.css
-rw-r--r--   1 stovak  staff  214923 Dec 25 11:44 jquery.mobile-1.0.js
-rw-r--r--   1 stovak  staff   49152 Dec 25 11:44 jquery.mobile.min.css
-rw-r--r--   1 stovak  staff   81903 Dec 25 11:44 jquery.mobile.min.js
-rw-r--r--   1 stovak  staff   31765 Dec 25 11:44 jquery.mobile.structure.css
-rw-r--r--   1 stovak  staff   25268 Dec 25 11:44 jquery.mobile.structure.min.css
-rw-r--r--   1 stovak  staff  132121 Dec 25 11:44 jquery.mobile.zip
drwxr-xr-x  39 stovak  staff    1326 Dec 25 11:17 js
drwxr-xr-x   6 stovak  staff     204 Dec 25 11:17 tests
drwxr-xr-x   6 stovak  staff     204 Dec 25 11:17 tools
-rw-r--r--   1 stovak  staff       3 Dec 25 11:24 version.txt
i7:jquerymobile stovak$
```

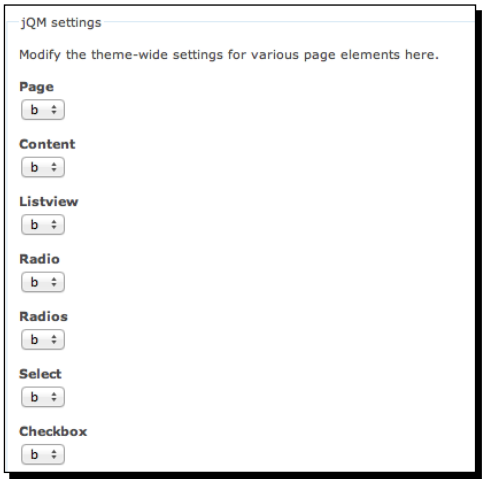
3. Navigate to **Configure | Appearance | List** and enable the **jQM** theme.

4. Add the following line to `sites/all/themes/dpk_mobile/dpk_mobile.info`:

```
base theme = jqm
```

5. If you look in the new **jqm** theme, there is a folder called `templates`. Copy this folder to your `dpk_mobile` theme folder. `page.tpl` and `html.tpl` already exist in the theme. Move them into the templates folder overwriting the files that we just copied. You should have your previous `page.tpl` and `html.tpl` with several new templates for node, comment, and block.

6. On the full-site, navigate to **Admin** | **Appearance** | **Settings** | **Themes** | **DPK Mobile**. Under the item **jQM settings**, change all the options to **b** and save the theme settings:



7. Delete everything in `page.tpl.php` and replace it with the following lines of code (leaving out my explanation lines):

```
<?php
// $Id:$
global $user;
?>
<div data-role="page" id="<?php print $jqm_page_id ?>" data-
  theme="<?php print $page_data_theme ?>" >
```

Notice the `data-role` and `data-theme` attributes. `data-role` attribute enables every page to act like a jQM "Page". `data-theme` takes its value from the settings of the theme. We will discuss this more in a minute:

```
<header data-role="header" data-position-"inline">
```

I would like an iPhone-style header across the top. The jQM role for this is simply header:

```
<?php if (!$is_front): ?>
    <?=l("home", "<front>", array("attributes" => array("data-
role"
      => "button", "data-icon" => "back", "class" => "ui-btn-
left")));?>
<?phpendif; ?>
```

If we are not on the home page, add a **Home** button. Notice that the use of `data-role` and `data-icon` are combined with the jQuery UI styling class of `ui-btn-left`, which will float the button to the left-side of the header:

```
<h1><?php print $title ? $title : $site_name;  ?></h1>
  <?php if ($user->uid == 0) { ?>
    <ahref="#login" data-role="button" data-icon="gear" class="ui-
btn-right" data-rel="dialog">Login</a>
  <?php } else { ?>
    <ahref="/user/logout" data-role="button" data-icon="gear"
class="ui-btn-right"  data-rel="dialog" data-ajax="false">Logout</
a>
  <?php } ?>
</header><!-- /data-role="header" -->
```

If the user is not logged in, add a **Login** button. If the user is logged in, add a **Logout** button. The Drupal logout page is a simple cookie that is destroyed, and then redirected back to the home page. We will need to specifically prohibit this page from loading via AJAX. In order to do that, we have added a `data-ajax="false"` property:

```
<?php if (isset($tabs) && $tabs): ?>
<nav data-role="navbar">
<?php print render($tabs); ?>
</nav><!-- /navbar -->
<?phpendif; ?>
```

For standard Drupal editing tabs, render them as a `data-role="navbar"`:

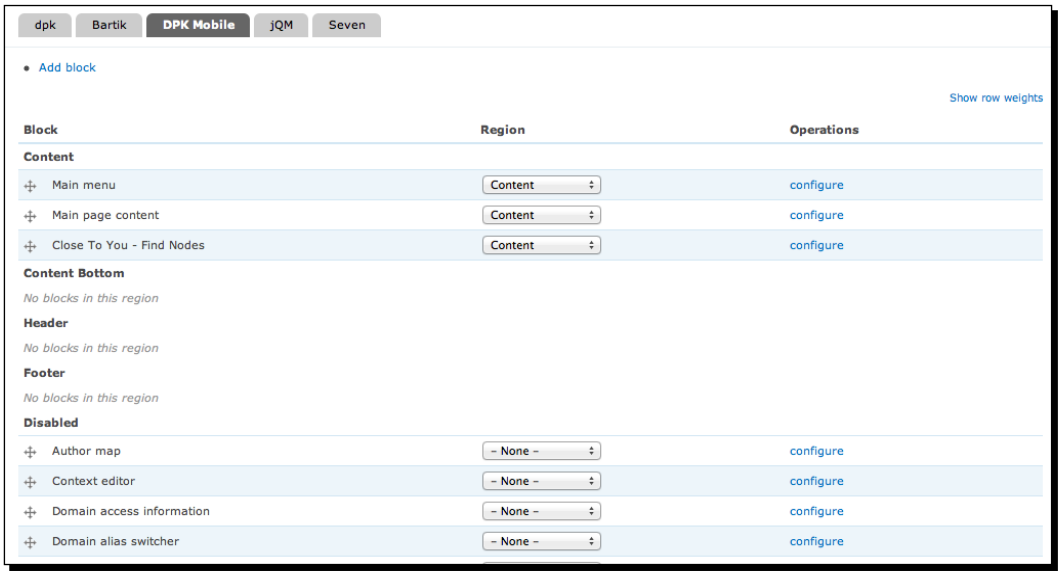
```
<article data-role="content" data-theme="<?php print $jqm_content_
data_theme ?>">
<?php if ($show_messages&& $messages): ?>
  <div class="ui-body ui-body-e">
    <?php print $messages; ?>
  </div>
<?phpendif; ?>
```

Theming the messages pane with `ui-body-e` will give it a yellow "Post-It" look:

```
<?php print render($page['content']) ?>
<?php print render($page['content_bottom']); ?>
</article><!-- /data-role="content" -->

<footer data-role="footer" class="footer">
<h1>Footer Content</h1>
<?php print render($page['footer']) ?>

</footer><!-- /data-role="footer" -->

</div><!-- /data-role="page" -->

<!-- /page.tpl.boundary -->
```

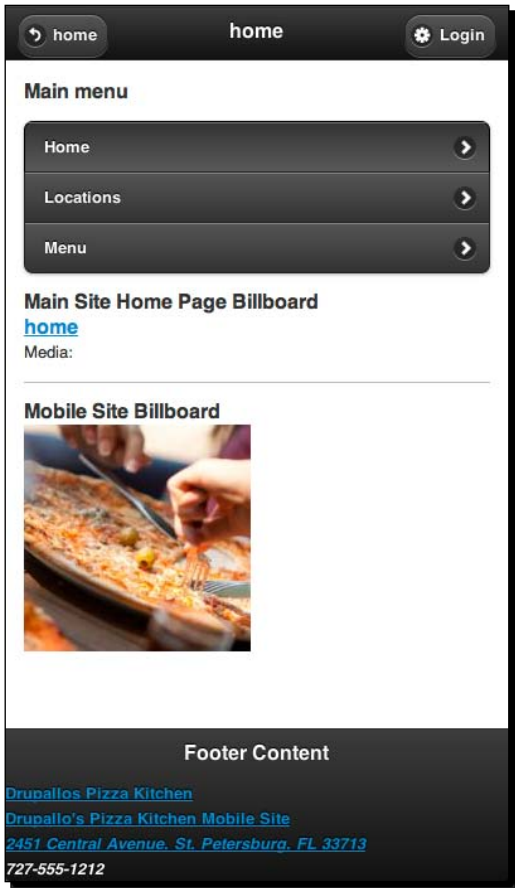
Finish the standard page and close the tags.

8. Navigate to **Configure** | **Blocks** and then click on the **DPK Mobile** theme. Make the **Main menu** active in the **Content** region. Click on the **Save** button to save the block configuration:



9. Navigate to **Configure** | **Development** | **Performance** and clear the Drupal Site cache.

10. Navigate to the mobile site and your page should look similar to the following screenshot:



What just happened?

We edited the `page.tpl.php` file of our mobile theme and added the jQuery Mobile functionality—first, by making the jQM theme our base theme, then by adding elements to the `page.tpl` that allows the jQuery Mobile to make sense to the page. jQuery Mobile scans the page and looks for `data-*` properties or elements to let it know what to do with the elements that it finds. We also added a login page to the bottom of the main page template. On AJAX-loaded sub-pages this login request will be ignored. We added a link to that login form in the top header. After the user is logged in, the link will show up as a **Logout** link.

This is a great improvement in the look and feel for our page, but it has still got some problems. Click on the login form and try to log in. Once you submit the login form, Drupal tries to redirect you back to the home page for a successful log in and jQuery mobile doesn't understand the redirect. Let us use what we learned about logging in via a Services AJAX call to submit this login form. We will then transit the user back to the current page.

Have a go hero

On another site you are working on making a JavaScript call that retrieves content from this site and re-uses it on the second site. As the DPK site is local to your computer, it should only appear when you view the site locally. Make sure there is no visible JavaScript error when it fails for everyone else.

jQuery Mobile JavaScript Events

jQuery Mobile adds jQuery's legendary ease of use in crafting event listeners to the touch-screen space. With jQM on the page, we get all of the gestures mobile users are familiar with added to jQuery's event binding. Gestures such as tap, swipe, and tap-hold can be used to trigger user events on the page. In addition, we can bind them to accelerometer events such orientation change. Understanding that each HTML page can have multiple jQuery Mobile `data-role="page"` elements, we can use jQM's events to bind actions before those "pages" show, after they are shown, on page `init`, and so on.

What we need to do, is lasso that login form, submit it via AJAX, and redirect the user back to the front page. Let us get started.

Time for action – the AJAX login form

Add the following lines to the following files:

1. To the bottom of the `sites/all/themes/dpk_mobile/templates/page.tpl.php` file, add the following lines:

```
<?php if ($user->uid == 0) { ?>
<div data-role="page" id="login" data-theme="<?php print $page_
data_theme ?>">
  <header data-role="header" data-position-"inline">
    <h1>User Login</h1>
  </header>
  <div data-role="content" data-theme="<?php print $page_data_
theme ?>">
        <?php $form = drupal_get_form("user_login");
$form['#attributes']['data-ajax'] = "false"; echo drupal_
render($form); ?>
```

```
    <div class="ui-body ui-body-e" style="display:none;" id="form-
errors"></div>
  </div>
</div>
<?php } ?>
```

2. Edit the `sites/all/themes/dpk_mobile/js/global.js` file by adding the following lines:

```
Drupal.behaviors.jQMPageInit = {
    attach: function(context) {
      $("#user-login").not(".jQMPageInit-processed").
submit(function(evt){
        if (evt) { evt.preventDefault(); }
        toSubmit = {
          "username": $(this).find("#edit-name").val(),
          "password": $(this).find("#edit-pass").val()
        }
        $.ajax({
          url: "/rest/user/login.json",
          dataType: "json",
          data: toSubmit,
          type: "post",
          error: function(jqXHR, textStatus, errorThrown) {
            $("#form-errors").html("There was an error logging you
in").show();
          },
          complete: function(jqXHR, textStatus){
            document.location.href = "/";
          }
        });
      }).addClass("jQMPageInit-processed");
      $("#user-logout").not(".jQMPageInit-processed").
click(function(evt) {
        if (evt) { evt.preventDefault(); }
        $.mobile.showPageLoadingMsg();
        document.location.href = "/user/logout";


      }).addClass("jQMPageInit-processed");
    },
    detach: function(context){
    }
  }
```

3. Navigate to `http://m.dpk.local`. The login/logout process should work as expected.

What just happened?

Most of the normal Drupal interaction is circumvented by the jQuery Mobile in favor of AJAX loads. In this example, we are taking the normal login process and making it AJAX-ified. We create a login form that is a jQuery Mobile `role=dialog` at the bottom of the page. It will pop-up when the user clicks on the **Login** button. We intercept the form's normal interaction by telling jQuery mobile `ajax=false`. We then construct a `Drupal.behavior` JavaScript method and attach it to our form's submit action.

The login method grabs the username and password from the form and submits it via an AJAX call. If there is an error, we have added an **errors** area to the bottom of the form and themed it with the Post-It style jQM Yellow theme.

We do the same interception with the logout form. We use the `$.mobile.showPageLoadingMsg()` method to throw up a "**this computer is doing something**" spinner for those times when an impatient user meets a slow server and does a simple page redirect to the user logout URL.

Pop quiz

1. The module that provides a REST interface to Drupal's core content is called:
    a. Services
    b. Views
    c. Displays
    d. jQuery update
2. API stands for
    a. All Pieces Included
    b. Application Programming Interface
    c. Alter Public Instantiation
    d. None of the above
3. The REST programming pattern
    a. Provides a stateless interface with the web server's content
    b. Is the foundation of modern mobile applications
    c. Uses the get, put, post, and delete verbs acting on server-based nouns
    d. All of the above

4. HTML attributes that begin with the word "`data-`" are:
    a. Ignored
    b. The value is readable by a jQuery call
    c. Used to power jQuery Mobile
    d. All of the above

5. In order to read an attribute of a tag with the class name selected with a jQuery, we use the line:
    a. `jQuery(".selector").val();`
    b. `jQuery(".selector").attr("attribute");`
    c. `jQuery(".selector").parent();`
    d. `jQuery(".selector").load();`

Summary

In this chapter we enabled the **Services** module and began using Drupal's services via methods other than the standard loading of a page. This process can be used as the backend to create powerful native applications for iPhones and Androids using standard messaging protocols.

Sometimes the expense of creating and maintaining a native application is beyond the reach of the client. For those instances, we started creating a jQuery Mobile site based on the Drupal theme and module.

Using this foundation, we can build a native application based on open standards and freely-available software. In the next chapter, we will work on some fit-and-finish items for your mobile web app.

9
Putting it Together

Having just the vision's no solution
Everything depends on execution
The art of making art
is putting it together…
-Stephen Sondheim from "Sunday in the Park with George."

In the last few chapters we saw a lot of tools that help in making a really great mobile site. However, now that we have the tools to make it great, how do we go about putting it all together?

In this chapter, we are going to start putting the disparate pieces into a real, live, workable mobile site. In this chapter we will cover:

- Analyzing the way a working Display Suite site is set up
- Understanding build modes
- Creating a new build mode
- Creating a custom field
- Customizing properties of menu items
- Creating a customized build mode for our content API
- Adding custom web-downloadable fonts to our website

First, let's discuss a module that we have briefly touched on, but let's now go a little deeper into Display Suite.

Display Suite

Display Suite (**DS**) is the module that made Drupal more powerful than any other previous content management suite for me. We have talked a little about it in the previous chapters and even written some code for it, but let me go into a little more detail here.

Build mode: When a node's information is retrieved by Drupal, the node's display is "built" by Drupal's rendering engine. The build process wraps the viewable elements in HTML tags. This way of building a node's display is called the build mode. Out of the box, Drupal comes with two build modes: Full and Teaser.

In the early days of Drupal, nodes were visible basically in two ways: Teaser and Full views. CCK gave you control over what was visible in that view on the "display" screen. You could hide CCK elements in one view and have them visible in another. If your field referenced a photo, you could use **ImageMagick** or **GD** to resize the photo on the fly to the correct dimensions. However, this screen was ultimately limited to just the two modes of display. DS changed that. It allows you to have unlimited build modes for every content type. In addition, you could create a build mode for comments and user objects. With Drupal 7 and the new Entity API, DS allows you to theme data entities. We will discuss more on entities later in the chapter. For now, let's take a look at how DS works its magic.

Hooks, styles, and build modes

What is DS? At its core, it consists of two main things. First, it takes node, user, or comment properties and wraps them in DIV's with specific class names and patterns. In the first chapter, we mentioned that Drupal's modules extend the core by implementing a series of "hooks" that allow you to name a function correctly and have that function executed at a certain time with certain data. For instance, a hook that you can probably get familiar with right away would be `hook_preprocess_page`. This hook allows you to alter variables on your `page.tpl` before the final output is rendered. DS uses some node, user, and comment hooks to take the data from the database, wrap it in DIVs with predefined classes, and substitute it for what is normally returned as the rendered node. DS also adds a CSS file to the Drupal base CSS that has some default layouts for those predefined classes. Let's take a look at some nodes that have been themed with DS and the markup that it generates. `Performance.gov` is a Drupal-based government site that is built on Drupal 6, but will help illustrate the powerful way in which DS lays out the nodes. Consider the following page:

Ensuring Effective Oversight of Government Finances

The Federal government has a fundamental responsibility to be effective stewards of the taxpayers' money. We must be responsible with money that comes in to the government, money that is spent, and money that is used in running the government itself. Decision makers and the public must have confidence in financial management in order to make informed decisions about managing government programs and implementing policy. Since the passage of the Chief Financial Officers Act (CFO Act) of 1990, the financial community has made important strides in instilling strong accounting and financial management practices. Over the past 20 years, an increasing number of agencies have sustained disciplined financial operations, implemented effective internal controls, and integrated transaction processing and accounting records. This year, 21 of the 24 CFO Act agencies received a clean audit opinion—the highest number of clean opinions the Federal Government has achieved since the passage of the CFO Act. Agencies achieved this accomplishment in an increasingly complex reporting environment and stricter audit standards. However, challenges remain. The initiatives below discuss some of our key efforts to continue improving the management of the government's finances.

Featured Story: Campaign to Cut Waste

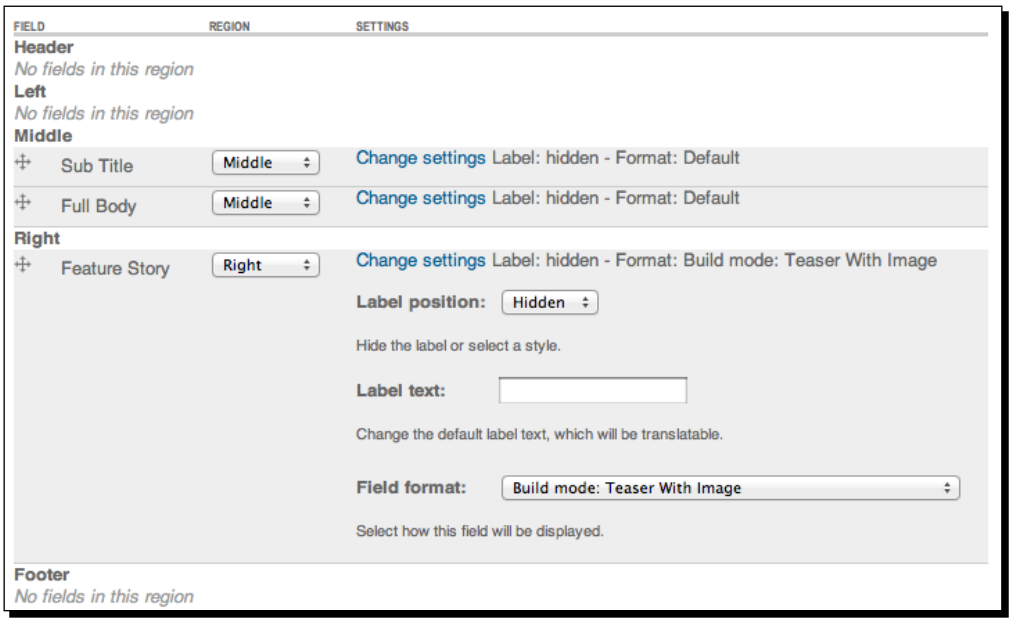
As part of the President and Vice President's Campaign to Cut Waste, Federal agencies are identifying innovative ways to cut costs and improve efficiencies. From eliminating excess properties to cracking down on unnecessary travel expenses, agency Chief Financial Officers are leading the charge to ensure that funds are not wasted. Learn more.

This is a **full** display of the overview content type. Overview is used as the `<front>` page for each of the site's different areas of focus. There are nine areas of focus that are split into hostnames with the Domains module as we did in *Chapter 3*. Each domain has its own overview. The overview data seems deceptively simple with a title, a body, and a featured story node reference field.

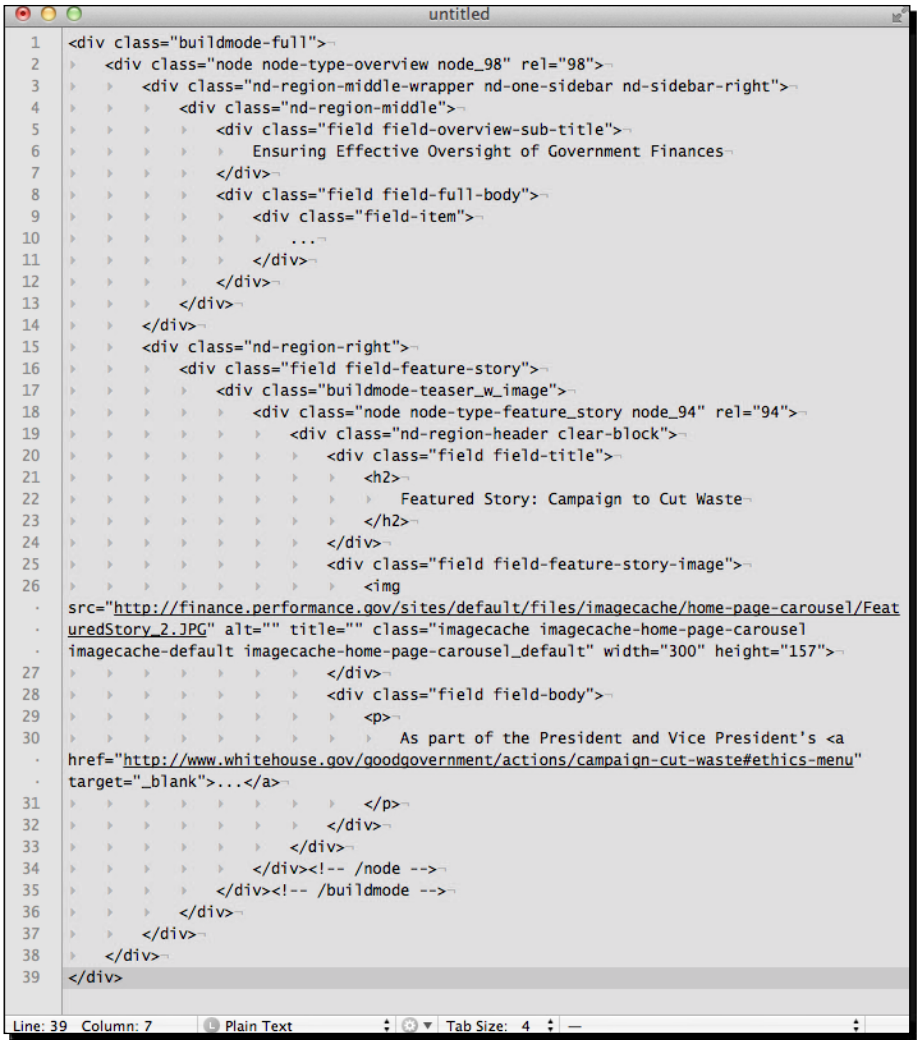
Note that this is a Drupal 6 website. The screen looks slightly different in Drupal 7, but the principles are the same.

Label	Name	Type	Operations
Title	Node module form.		
Sub Title	field_overview_sub_title	Text	Configure Remove
Body	Node module form.		
Feature Story	field_feature_story	Node reference	Configure Remove
Menu settings	Menu module form.		
Domain access	Domain-specific settings for posts.		
Meta tags	Meta tags fieldset.		
Authoring information	Node module form.		
Revision information	Node module form.		
Publishing options	Node module form.		
Comment settings	Comment module form.		
Path settings	Path module form.		
File attachments	Upload module form.		
URL redirects	Path redirect module listing		
Custom Breadcrumbs	Custom Breadcrumbs module form.		

The **Feature Story** field points to one or more of the featured story content types. Featured stories are published across all domains, which allow different areas of focus to share the featured story content. The DS layout also seems deceptively simple, which is shown in the following screenshot:

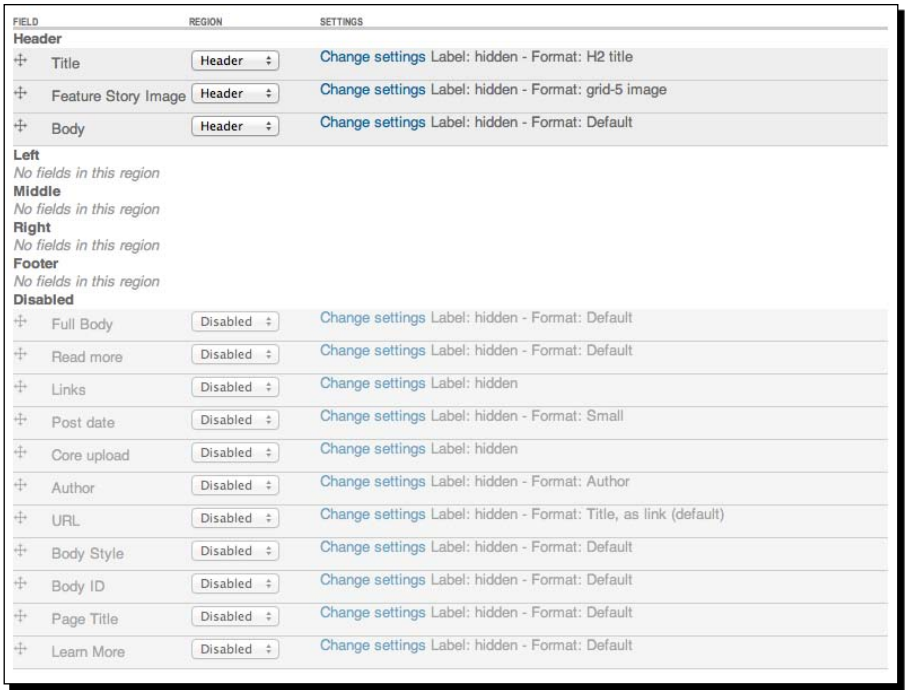


As you can see, the **Subtitle** and **Full Body** CCK fields are in the **Middle** region and the **Feature Story** is in the **Right** region. We have also created a customized build mode for featured stories called **Teaser with Image**. We have told DS that wherever the overview is displayed, display the **Feature Story** in the **Teaser With Image** mode and put it in the right region. The generated markup looks similar to the following screenshot:



In this case, DS has wrapped the build in a `buildmode-full` container. Node data is wrapped up with a container DIV that has a generic `node` class, the content type `node-type-overview`, and a specific node-id class `node_98`. The middle region has a wrapper and container DIV, `nd-region-middle-wrapper`, as well as two classes that give it space for the right-hand sidebar `nd-one-sidebar` and `nd-sidebar-right`. The middle region's container DIV has two `field` wrapper DIV's. If there had been multiple entries in the field, DS would have wrapped those entries in a `field-items` container DIV. **Sub Title** and **Full Body** just have one database entry. Now, notice the `nd-region-right` DIV.

Inside that there is a `field-feature-story` container with the `featured story` in the `buildmode` we described in the DS setting: `Teaser_with_image`. Featured Story's `teaser_with_image` `buildmode` looks similar to the following screenshot:



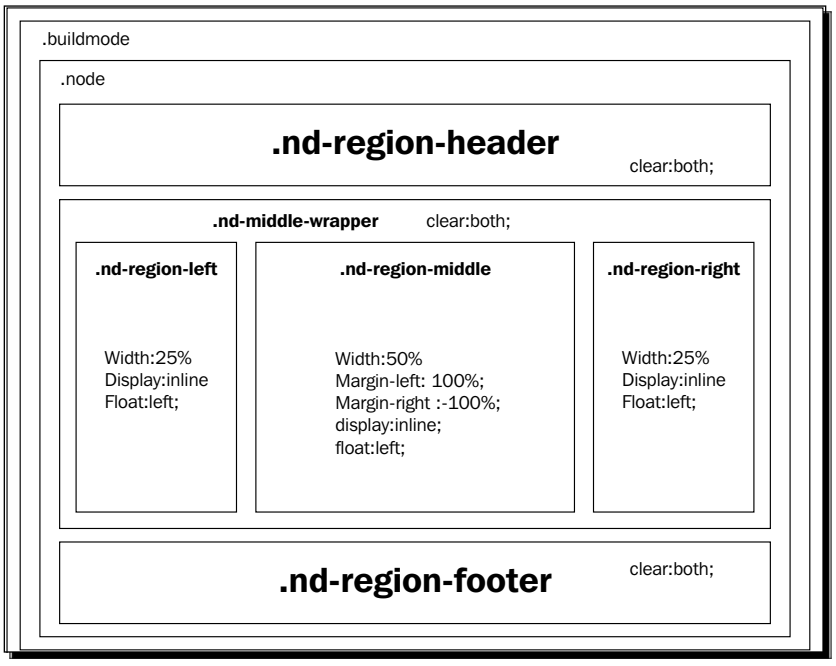
FIELD	REGION	SETTINGS
Header		
Title	Header	Change settings Label: hidden - Format: H2 title
Feature Story Image	Header	Change settings Label: hidden - Format: grid-5 image
Body	Header	Change settings Label: hidden - Format: Default
Left		
No fields in this region		
Middle		
No fields in this region		
Right		
No fields in this region		
Footer		
No fields in this region		
Disabled		
Full Body	Disabled	Change settings Label: hidden - Format: Default
Read more	Disabled	Change settings Label: hidden - Format: Default
Links	Disabled	Change settings Label: hidden
Post date	Disabled	Change settings Label: hidden - Format: Small
Core upload	Disabled	Change settings Label: hidden
Author	Disabled	Change settings Label: hidden - Format: Author
URL	Disabled	Change settings Label: hidden - Format: Title, as link (default)
Body Style	Disabled	Change settings Label: hidden - Format: Default
Body ID	Disabled	Change settings Label: hidden - Format: Default
Page Title	Disabled	Change settings Label: hidden - Format: Default
Learn More	Disabled	Change settings Label: hidden - Format: Default

We have put the node title, the feature story image, and body properties in the **Header** region, which results in line numbers 19-32 of the markup. All of the fields are stacked in the **Header** region. We also have places in the site that display the feature story in a teaser without an image and link it directly to the full version of feature stories—all with different layouts, and without writing any markup or CSS. We estimated that by using DS we eliminated 40% of the CSS that we would normally have to write for a project. DS is a fantastic way to rapidly layout nodes with little or no custom code. Let's take a look at some of the default CSS that is present in the DS. This is the Drupal 6 version that powers the preceding example page:

```
/* $Id:nd_regions.css,v 1.1.2.3 2010/05/31 12:16:09 swentelExp $ */
.nd-region-header{clear:both;}
.nd-region-left{display:inline;float:left;}
.nd-region-middle-wrapper
{display:inline;float:left;width:100%;margin-right:-100%;}
.nd-region-right{display:inline;float:right;}
.nd-region-footer{clear:both;}
```

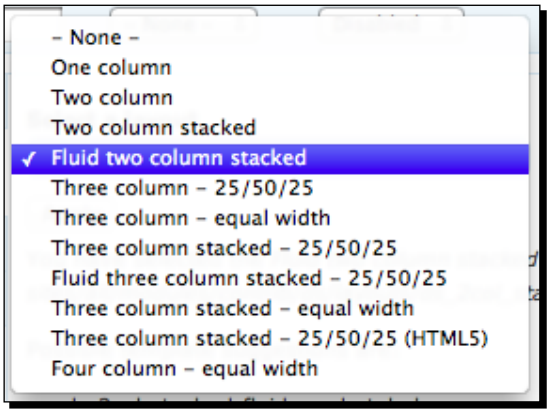
```
/* default region dimensions */
.nd-region-left
{width:25%;}
.nd-region-right
{width:25%;}

/* middle region */
.nd-no-sidebars .nd-region-middle
{}
.nd-sidebar-left .nd-region-middle
{margin-left:25%;}
.nd-sidebar-right .nd-region-middle
{margin-right:25%;}
.nd-two-sidebars .nd-region-middle
{margin-left:25%;margin-right:25%;}

/* Fix for IE */
.nd-no-sidebars{display:block;float:none;margin-right:0;width:auto;}
.nd-sidebar-right{*display:inline;*float:right;*width:100%;*margin-
left:-100%;}
```

How does it work? Well, in Drupal 6, DS assumes a five region node with header, left, middle, right, and footer.



The middle wrapper's margins are 100 percent on its sides to eliminate problems with older versions of Internet Explorer. Default behavior of Display Suite is to not render any region that does not have content. This can be overridden with the **Render all regions** plugin.

In Drupal 7, there are multiple layouts that one can choose from:



The word **stacked** indicates a header on top and footer at the bottom. Hence in Drupal 7 we have the choice from one to four columns, equal or 50/25 width, and stacked or not stacked. There is also a three column option with an HTML5 markup that will use header, section, and footer elements instead of DIVs. Honestly, I don't know if I agree with the use of the HTML5 header and footer elements used this way, but this option is available if you care to take advantage of it.

Time for action – retheming nodes for our jQuery mobile theme

We need to re-do the menu so that it looks good with our new jQuery mobile theme. Let's get started:

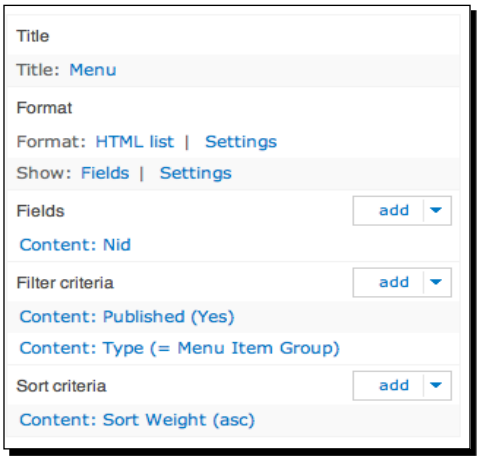
1. Create a new folder in the **dpk** theme. Call it **templates**. Move `page.tpl.php` and `html.tpl.php` into the folder. Create a new file in the folder called `views-view-list--menu--page.tpl.php` with the following code as its contents:

```
<div class="menu-item-list">
<?php if (!empty($title)) : ?>
<h3><?php print $title; ?></h3>
<?php endif; ?>
<?php foreach ($view->result as $id => $row){
    echo drupal_render(node_view(node_load($row->nid), "full"));
    } ?>
</div>
```

2. Create a new file in the **dpk_mobile** theme's **templates** folder called `views-view-list--menu--page.tpl.php` with the following code as its contents:

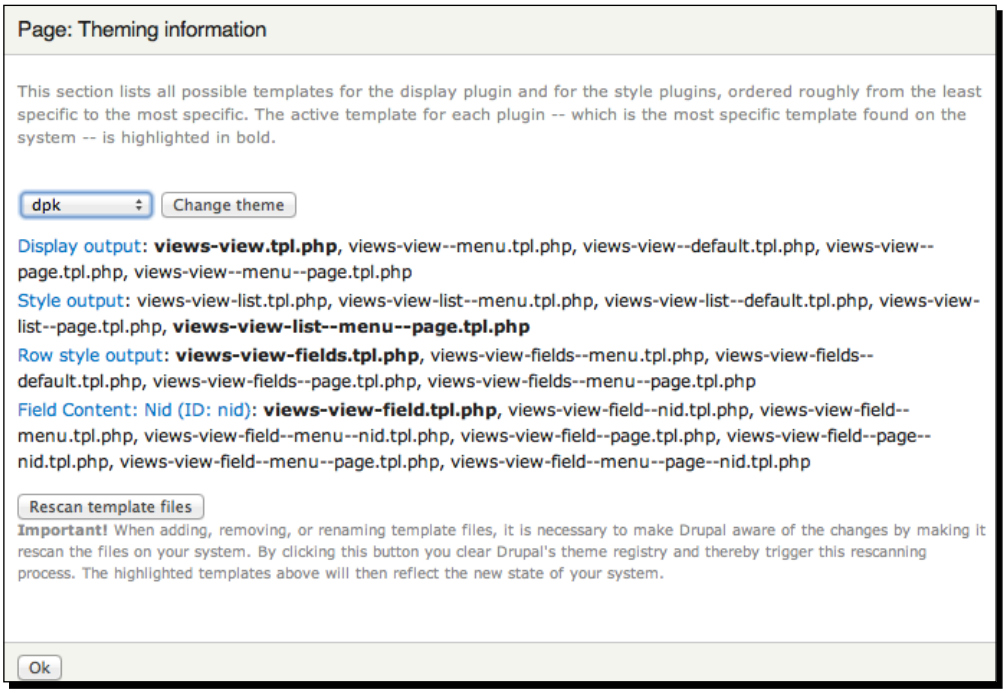
```
<div class="menu-item-list">
<?php if (!empty($title)) : ?>
<h3><?php print $title; ?></h3>
<?php endif; ?>
<ul data-role="listview" data-theme="c" data-inset="true">
<?php foreach ($view->result as $id => $row){
    $group = node_load($row->nid); ?>
    <li data-role="list-divider" role='heading'>
      <h3><?php print($group->title); ?></h3>
    </li>
    <?php foreach($group->field_items['und'] as $ref){ ?>
      <li data-role="list-item" role='heading'>
        <a href="/rest/node/<?php echo $ref['nid'];
?>?build=mobile" data-rel="dialog" data-transition="pop">
        <?php print(node_load($ref['nid'])->title); ?>
        </a>
      </li>
    <?php } ?>
  <?php } ?>
</ul>
</div>
```

3. Navigate to **Structure | Views** and edit the **Menu View** that creates our list of menu item groups. Under the **Show** item, click on **Content** and change it to **Fields**. Under the **Fields** list, add **Content: NID** and remove any other components that might be there. Your view's first column should look similar to the following screenshot:

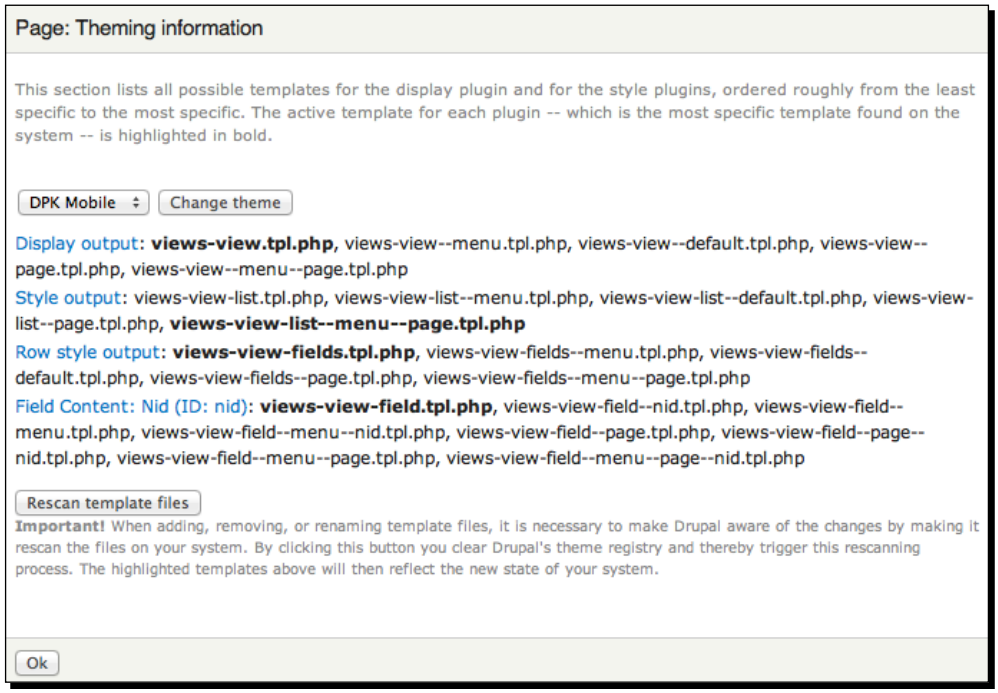


4. In the third column of the page display is a tab called **Advanced**. Click on the **Advanced** tab, and in the resulting panel, click on the **Theme Information** button.

5. In the resulting overlay, there are four theming levels. Under the **Style output** level, the first item might be in bold. What we want is the fourth item to be in bold, which also coincides with the file that you have just created in the **dpk** theme. If this filename does not appear here then click on the **Rescan template files** button. The views should find the new template that you have added to the theme. If the fourth item is not in bold, go back and double check the name of the files that you created in step 1. They should be exact for views to find them:

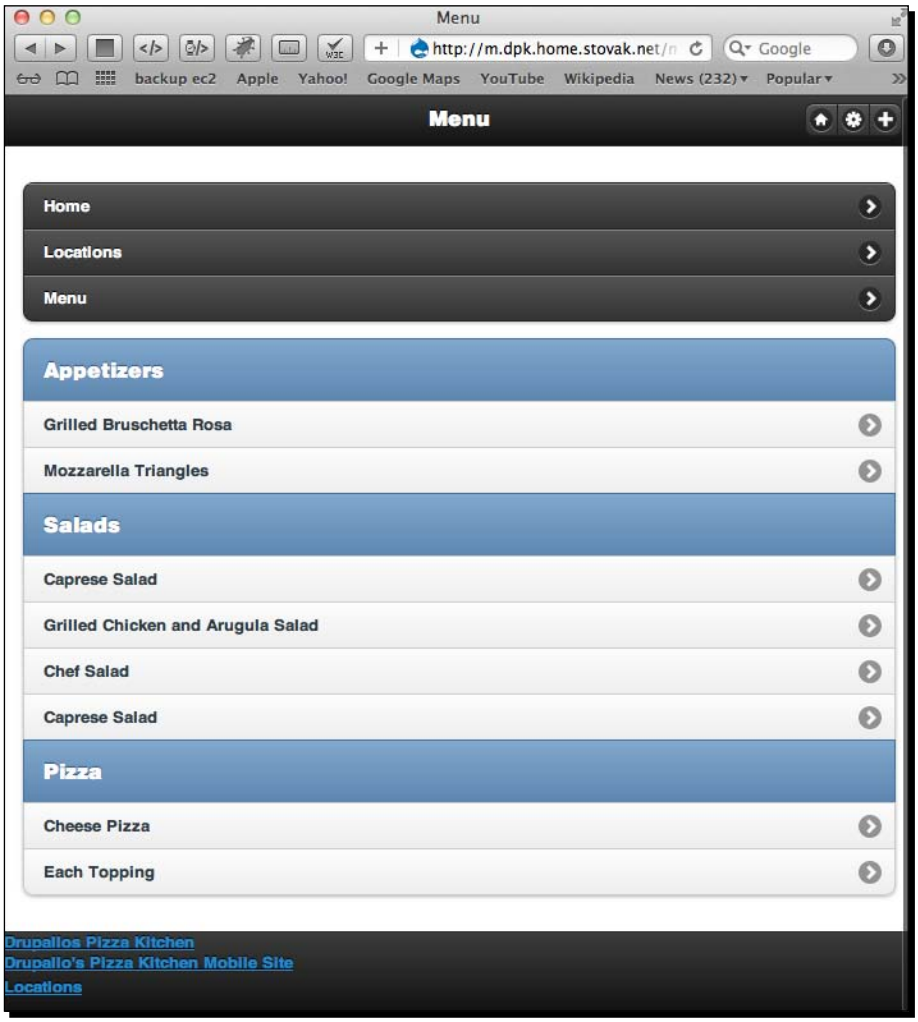


6. Once the DPK theme has found the new template, switch to the **DPK Mobile** theme and rescan to make sure views has found the template for that theme. Once both the themes have been discovered in the new template, click on the **OK** button in the overlay and then **Save** your view in the top right-hand corner of the page:



7. Navigate to **Configuration | Development | Performance** and click on the **Clear all Caches** button.

8. In one browser window navigate to the desktop site's menu page at `http://dpk.local/menu`. In another window, navigate to the mobile site's menu page at `http://m.dpk.local/menu`. The desktop site should be unchanged, while the mobile site should look something similar to the following screenshot:



Wait! Your menu items look different from mine! We will discuss more about this in a minute. For now, let's just get this page working.

What just happened?

In our desktop site, we are using Display Suite to theme our view results. We want to keep doing that for the desktop site, but add some markup to the mobile site so that the jQuery mobile can transform it into a pretty list. View templates have very specific naming conventions for very specific theming levels. We themed and named the files so that the resulting code would theme the list of results. The first view theming template iterates through the results of the view and themes them with a standard full view. In that template, we have combined several actions into a single line:

```
echo drupal_render(node_view(node_load($row->nid), "full"));
```

The Iteration `$ref` holds a single row of view results. We changed our view to only give us the node ID in the view results. We are using `node_load` to load the view from the node ID, `node_view` to allow Display Suite to theme the resulting node as a full build, and `drupal_render` to render the HTML for that build, all with a single line of PHP. However, the mobile template works a little differently.

In our mobile template, we first create the UL tag with `data-role="listview"`. Remember our view returns `menu_group` nodes, and on that group node there is a field called **Items** that lists the menu items for the group. We loaded the group node and made its title a list item with `data-role="list-divider"`. Then, we added list items underneath the divider for each referenced node. Notice each node's link URL: `/rest/node/<?php echo $ref['nid']; ?>?build=mobile`. We want to link to the REST services' node endpoint and grab the node data with JSON. We have again consolidated commands with the statement:

```
print(node_load($ref['nid'])->title);
```

What this statement does is it loads the node with the given node ID and from the resulting object just prints the title. Also, notice that we have made the links for `data-rel="dialog" data-transition="pop"`. This will pop-up a dialog with the resulting node in it when clicked, or rather touched.

However, when you click on the menu items, nothing shows up in the pop-up. Why is that happening? Well, jQuery Mobile looks for a `data-role="page"` structure on the resulting page to show in the dialog. We do not have a `data-role="page"` element in our rendered node service that we created in the last chapter. Let's add this element to the module.

Have a go hero – adding a CSS3-based page transactions

Add CSS3-based page transitions to the jQuery mobile theme.

Time for action – adding theming to the rendered node

Let's create a function to return the rendered nodes as a service.

1. Edit **sites/all/modules/custom/rendered_node/rendered_node.module** and add the highlighted lines:

```
<?php

function rendered_node_theme() {
  return array(
    "rendered_node_content" => array(
      "arguments" => array("node" => null, "build" => null),
      "template" => "rendered_node_content"
    )

  );
}


function rendered_node_rest_server_response_formatters_
alter(&$formatters) {

  // Add an html response format.
  $formatters['html'] = array(
    'mime types' => array('text/html'),
    'view' => 'RESTRenderedNode',
    'view arguments' => array('format' => 'html'),
  'file' => 'rendered_node.inc'
  );

}
```

2. Edit `sites/all/modules/custom/rendered_node/rendered_node.inc`. Delete the "render_html" function entirely and change the render function as highlighted.

```
class RESTRenderedNode extends RESTServerView {

  public function render() {
  switch ($this->arguments['format']) {
  case 'html':
    if (isset($_REQUEST['build'])) {
      $build = $_REQUEST['build'];
    } else {
```

```
      $build = "full";
    }
  return theme("rendered_node_content", array("node" => $this-
>model,
    "build" => $build));
      }
  return '';
    }

}
```

3. Create a new file:

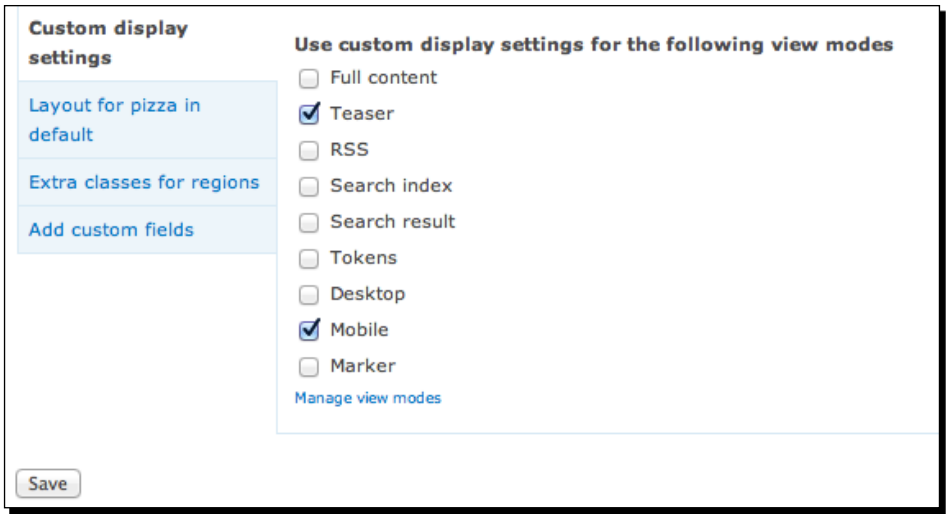
   sites/all/modules/custom/rendered_node/rendered_node_content.tpl.php with the following contents:

```
<div data-role="page" data-theme="b">
  <header data-role="header" role="banner" data-theme="b">
    <h1><?php echo $node->title; ?></h1>
  </header>
  <article data-role="content" role="content" data-theme="d">
    <?php echo drupal_render(node_view($node, $build)); ?>
  </article>
</div>
```

4. Navigate to **Configuration | Development | Performance** and click on the **Clear all Caches** button.

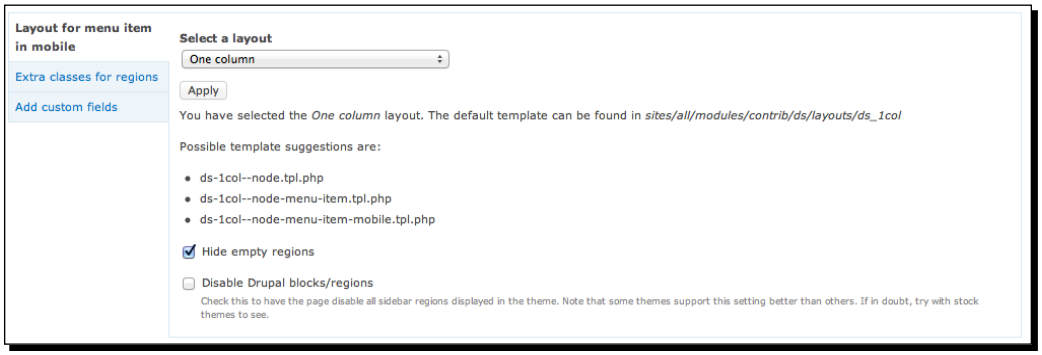
5. Navigate to **Admin | Structure | Layout | Display Suite**. On the line that says **Menu item**, click on the **Manage Display** option.

6. Scroll to the bottom of the page. In the bottom tabs region there is a tab called **Custom display settings**. Check the box labeled **Mobile** and click on the **Save** button, as shown in the following screenshot:

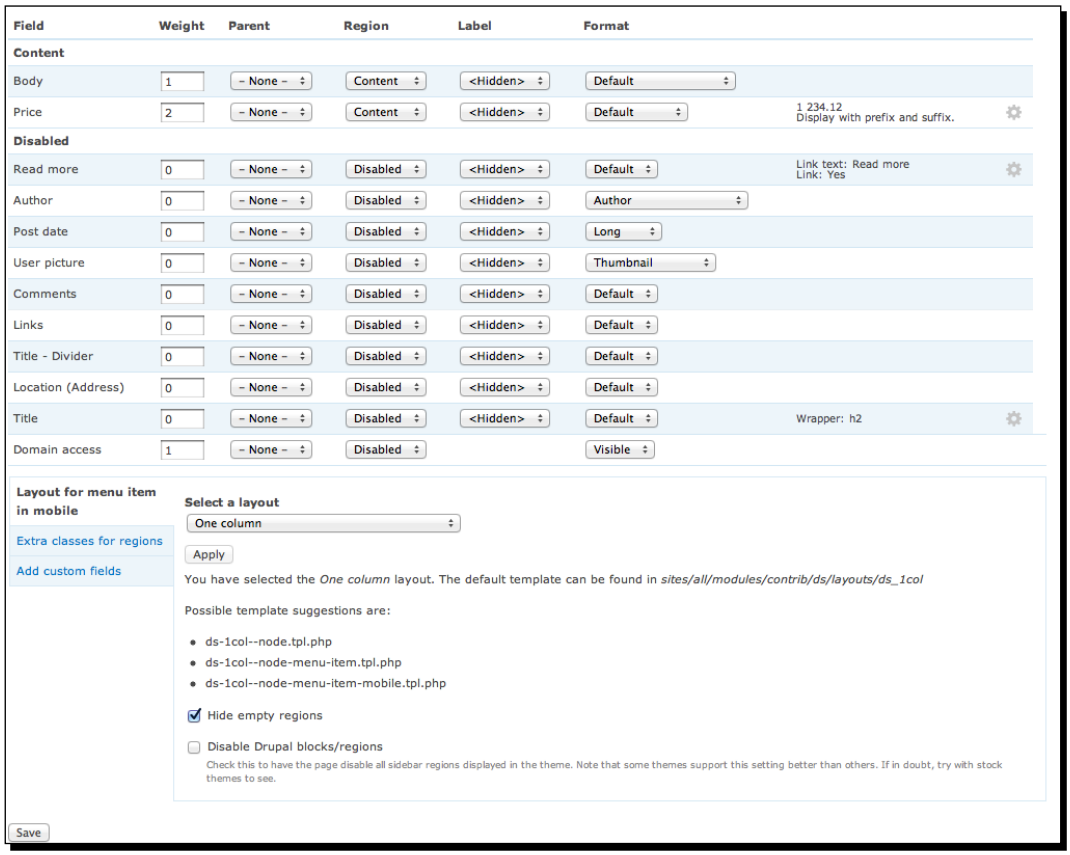


7. At the top of the page underneath the **Manage Display** tab, you should now see a **Mobile** Option. Click on the **Mobile** option.

8. Again, scroll to the bottom tab and click on the tab labeled **Layout** for the menu item in **Mobile**. Choose **One column** and click on the **Apply** button. If **Apply** is not available, then click on the **Save** button:

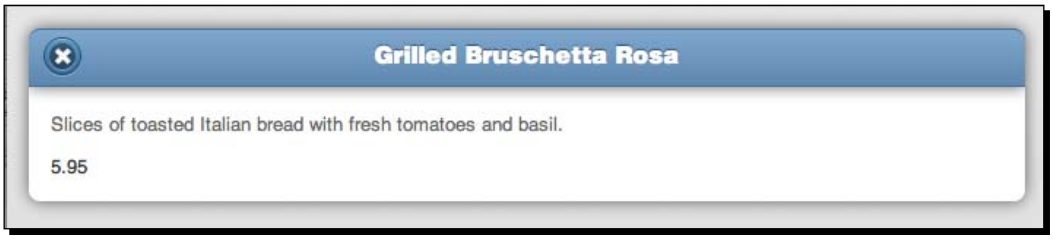


9. On the lines labeled **Body** and **Price**, change their **Region** from **Disabled** to **Content**. Click on the **Save** button to save the new settings.



10. Repeat steps 5 to 8 for content types Pizza, Sandwich, and Soup. For items where the **price** is in alternative fields, make sure you add all the price fields.

11. Refresh the menu page and you will find that, when you click on a menu item, the resulting pop-up looks something similar to the following screenshot:



What just happened?

The code that we have written for the new view template causes jQuery Mobile to make AJAX calls to the Services module's REST endpoint asking for a node that is themed in HTML. We created a theming function for our Services endpoint that adds some markup to allow jQuery mobile to load the node into a jQM dialog. We then created `mobile` builds of all our menu item nodes to make sure that the title is not duplicated on the themed node. We have put the node title in our View template and will not need it in the node's build.

We have come a long way in our mobile theme in a very short time, but we need to make sure that the **Locations** page does not load with jQuery mobile's AJAX API. In order to accomplish this, we will have to write some code that alters how menu items are written.

Have a go hero – adding an API hook

Add an API hook to correctly render a User's homepage (e.g. user/USERNAME/view).

Beyond core menu items

Drupal's menu structure did not get a major renovation since version 5.0. I am hoping that this will change with Drupal 8, but I am not holding my breath. Until then, we will have to put up with a witches brew of add-on menu functionality. One such module is **Menu Attributes**. How many times have you wanted each menu item to have a unique ID? Menu attributes solve that problem. How many times have you wanted to add extra classes to menu items? Menu Attributes solve this problem as well. What Menu Attributes does is extend the functionality of menu items to allow a series of standard markup choices to be changed on a per-menu-item basis. However, it is much better than just that.

Menu Attributes adds its own hooks, so that you can add your own attributes to menu attributes' configuration options. Consider that at the beginning of this book you probably did not know that the `data-role` and `data-theme` properties exist and as little as a year ago, they did not exist at all. This is fantastic news! We can add some of our `data-*` properties to Menu Attributes with a little bit of hook magic. Let's get started!

Time for action – customized menu attributes

Let's install the Menu Attributes module and create a custom module with a new hook. We will explain how and why this works:

1. Open a command-line terminal and enter the following commands:

```
cd ~/Sites/dpk
drush dl menu_attributes
```

```
drush pm-enable menu_attributes
mkdir sites/all/modules/custom/jqm_menu_hooks
touch sites/all/modules/custom/jqm_menu_hooks/jqm_menu_hooks.
module
touch sites/all/modules/custom/jqm_menu_hooks/jqm_menu_hooks.info
```

2. Add the following lines to the empty file at `sites/all/modules/custom/jqm_menu_hooks/jqm_menu_hooks.module`:

```
<?php


functionjqm_menu_hooks_menu_attribute_info() {
  $info = array();

  $info['data-ajax'] = array(
     'label' => t('jQuery Mobile data-ajax=FALSE'),
       'description' => t('Prevent the menu item from loading via
        ajax in the jQuery mobile environment. Default is true.
          Checking this box makes the value false.'),
    "form" => array(
      '#type' => "checkbox",
      '#default_value' => false,
      "#return_value" => "false"
    )
  );

  return $info;
}
```

3. Add the following lines to the empty file at: `sites/all/modules/custom/jqm_menu_hooks/jqm_menu_hooks.info`:

```
name = jQueryMobile Menu Hooks
description = "Adds options to menu items related to jQuery
Mobile."
version = VERSION
core = 7.x
version = "7.x-1.x-dev"
project = "jquerymobile"
dependencies[] = jquerymobile
```

4. Hop back over to the terminal window and type the following commands:

```
drush pm-enable jqm_menu_hooks
drush cc all
```

5. Navigate to **Structure | Menus | Main Menu | List Links** and edit the **Locations** link. There should be a new area called **Menu Item Attributes**. In that area, click on the checkbox for **data-ajax=false**. Click on the **Save** button to save the menu item.

6. If you refresh the mobile site and view the source, the **Locations** menu item will now have the `data-ajax="false"` property. When you click the **Locations** item in the menu, the **Locations** page will load and all the JavaScript will work.

What just happened?

The Menu Attributes module creates a Drupal hook, namely `hook_menu_attribute_info`. If you look in the Menu Attributes module, you will see that in `menu_attributes_info` there is a call to `module_invoke`. What `module_invoke` does is, it goes out to every installed Drupal module and finds out which ones have functions with the name the function invokes. It then sends the given data to those module's functions. This is how hooks are created. Returning a form item with that hook will add that option to the options that Menu Attributes tracks for menu items. We do not have to worry about storing the values or anything else because Menu Attributes takes care of the data after that. If you ever see **Nick Schoonens** at **DrupalCon** make sure that you buy him a beer for a very well written module with properly implemented module hooks.

There are just a few more minor details we need to clean up before we can call this mobile theme done.

Fonts

Any discussion of modern custom theming would not be complete without a discussion of fonts. About 10 years ago I had to break it to designer after designer, that when you do a web design, you could either have the font you wanted in a GIF, or you could have editable text, but you could not have both. More recently, that has changed.

Starting in about 2000, web browser coders started attempting to create a vision for how to make downloadable fonts more sensible. It was a balancing act. You didn't want to put your nice new font out on the web for just anyone to download, but at the same time, you wanted the browser to be able to download a typeface, or a subset of a typeface, on the fly and render text in the font. There were a lot of competing standards. Adobe advocated their PostScript format, but the file sizes were very large. Microsoft extended the TrueType font standard to allow embedding in Word documents and in web pages through Internet Explorer. TrueType, however, was a licensed format.

Open source browser creators like Mozilla and kHTML/WebKit embraced more open formats, none of which ever really took off. About 2007 when everyone sort of "decided" on **OpenType** as the standard going forward, the font manufacturers began licensing commercial fonts for "web embedding". The URL's on which the downloadable fonts could be used would be programmed in the font itself making downloading it and attempting to use it on a desktop machine useless. In the meantime, older browsers (for example, Internet Explorer) can make use of existing technology by using downloadable TrueType fonts. The long and the short of it is that web page font embedding is now a reality. There is even a format that is supported in Mobile Safari and Mobile Webkit on Android.

Browser	**Font Standard Supported**
Internet Explorer	Embedded OpenType (EOT)
Firefox	Truetype, Opentype, and .WOFF
Webkit/Safari/Chrome	Truetype, Opentype and .WOFF
Mobile Safari/Mobile Webkit	SVG

However, here is the thing: for as large as images are, fonts are sometimes even larger. Certain languages of some fonts can be as much as 500KB+ and to download three or four weights for a web page would make it impossibly slow for some users.

Hence, my advice to you is to be judicious with the fonts that you embed and to make sure you have obtained the proper licensing. Google has a series of web fonts that are open source and using them incurs no licensing fees. However, most designers want to use their favorite version of Future or Helvetica, which has this certain look. In that case, there are several services that will license fonts for embedding in a very cost-effective way. **Fonts.com Web fonts** (`webfonts.fonts.com)` and **Typekit** (`https://typekit.com/)` are two very viable options. Typekit is owned by Adobe and Fonts.com is owned by Monotype. Between the two of them, they cover most of the named fonts that designers love to use and the great thing about the Drupal module is that you can use them by themselves, with each other, or in any combination. Your only limit is what your users are willing to tolerate in terms of download times.

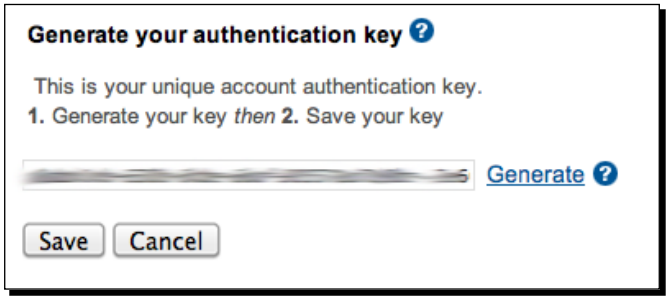
The Drupal module that enables font-embedding with a myriad of different services is called **Font Your Face**. Let's install it and dress up some of our headlines.

Time for action – adding fonts

We will be using `webfonts.fonts.com` for our font provider. Google has some free fonts that you can use. Also, the Drupal module supports several different providers:

1. Navigate to `http://webfonts.fonts.com` and sign up for a free account. You will need to put in your e-mail address and they will send you an e-mail to verify your e-mail address.

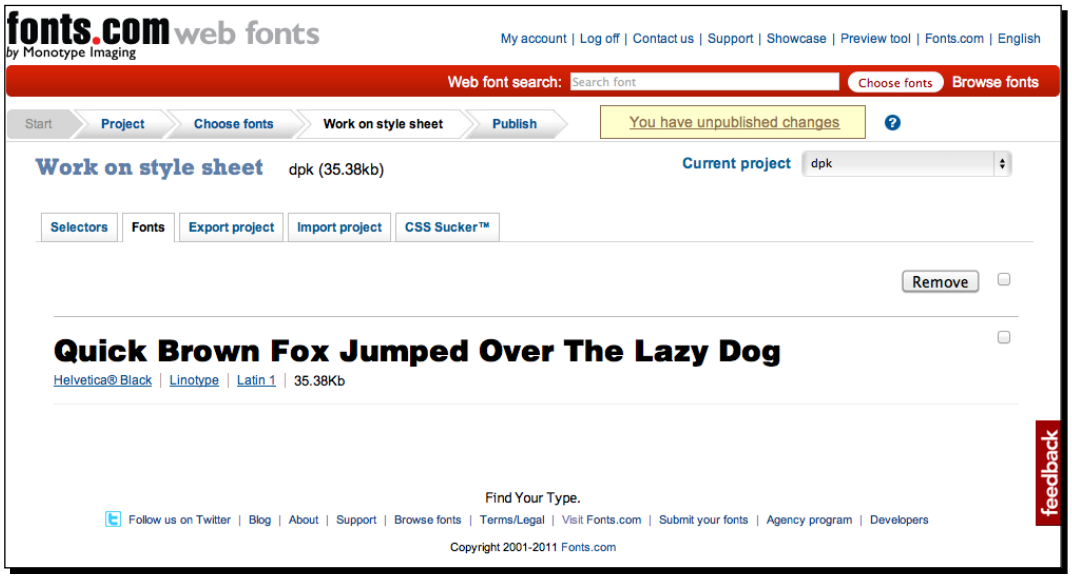
2. Once your account e-mail address is verified, navigate to **MyAccount | Account Summary**. There is a panel called **Get your authentication** key (shown in the following screenshot). Click on the **Generate** link to generate the key and click on the **Save** button to save it. Copy it to the clipboard and save it. You will need it once we get to the Drupal portion.



3. Create a **New Project** at `http://webfonts.fonts.com` with your DPK URLs as being the URLs of the website: `dpk.local` and `m.dpk.local`.

You use the pattern matching in the URLs for anything except your primary domain. Your primary domain must be listed with no wildcards, otherwise your fonts will not import.

4. Add a font to the project. I have added **Helvetica Black** to make the headlines stand out. It does not matter which one you add, just do not forget the name. Once added, at the top of the page, you will see a small dialog that says **You have unpublished changes to this project**. Click on it and publish the changes. Anytime you add or remove fonts from your project, you will need to publish the changes before you are able to use them in Drupal:

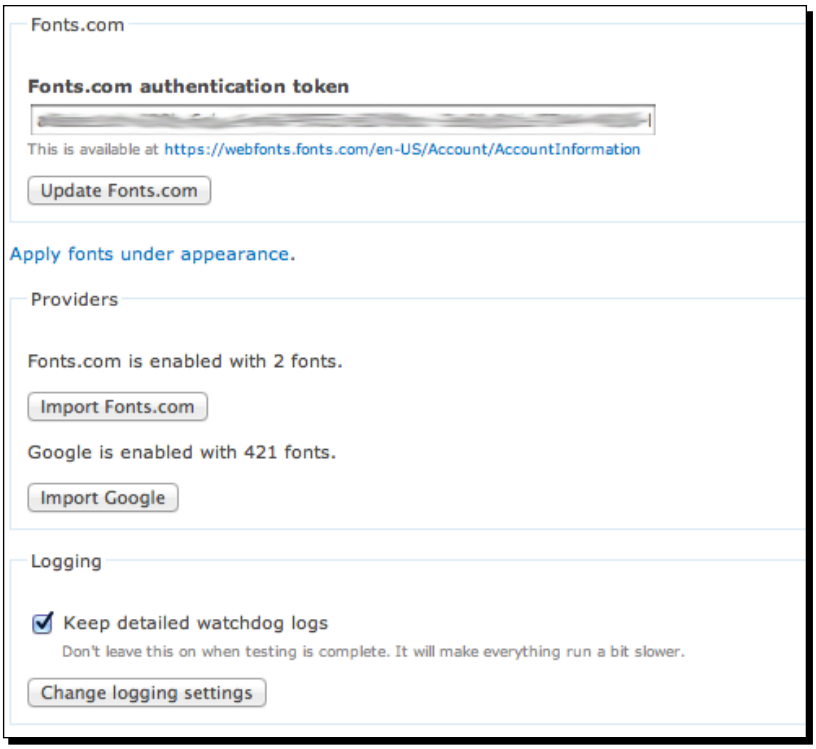


5. Open a terminal and enter the following commands:

```
cd ~/Sites/dpk
drush dl fontyourface
drush pm-enable fontyourface fonts_com google_fonts_api
```

6. In the desktop version of the DPK site, navigate to **Configure** | **User Interface** | **Font Your Face** settings. Enter the Fonts.com user interface token in the space provided. Click on the **Update Fonts.com** button. Now, click on the **Import Fonts.com** button. If you have published your font project on Fonts.com, it should import, however many fonts you have added to the project.

7. Click on the **Import Google** button. It will import about a bajillion Google fonts. It doesn't hurt to have the fonts "imported." They only show up on the front end once they are embedded:



8. Navigate to **Appearance | Font Your Face** and click on the **Browse fonts to embed** button. You can choose a few Google fonts if you have low standards or you can search for the one that you added from Fonts.com:

9. Once found, click on the font name, select **Embed**, and click on the **Save** button. In the English-speaking world, you always want to import the Latin version of your fonts. Cyrillic and Greek fonts only need be imported if you need them for those languages.

10. If you view the source on the DPK pages now, you will see a new Javascript file:

```
<script type="text/javascript" src="http://fast.fonts.com/jsapi/
YOUR_PROJECT_ID.js"></script>
```

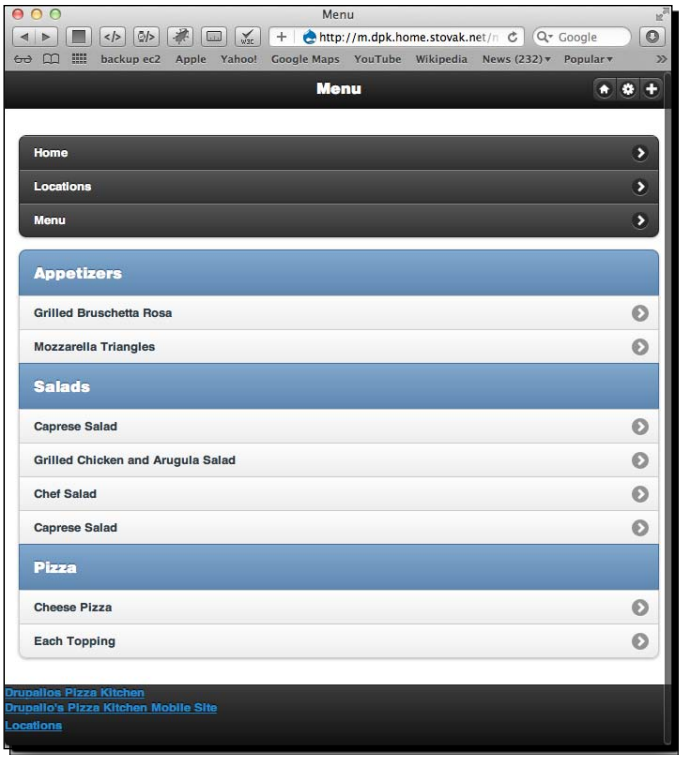
11. Edit `sites/all/themes/dpk/css/styles.css`. Change the text as highlighted on line number 38. Make sure the font-family property matches the font-family property in the List in Drupal.

```
h1,
h2,
h3,
h4,
h5,
h6 {
margin: 0;
padding: 0;
font-weight: normal;
font-family: "Helvetica W01 Blk", Helvetica, Arial, sans-serif;
}
```

12. Edit `sites/all/themes/dpk_mobile/css/styles.css` adding the highlighted lines, again, taking care that your font-family name matches the Drupal list item.

```
body { font: normal 10pt/12pt Helvetica, Arial, Sans-Serif; color:
#666; }
p { font: normal 10pt/12pt Helvetica, Arial, Sans-Serif; color:
#666; margin: .25em 0;}
h1 { font-family:"Helvetica W01 Blk", sans-serif; font-size: 2em;
}
h2 { font-family:"Helvetica W01 Blk", sans-serif; font-size:
1.5em; }
h3 { font-family:"Helvetica W01 Blk", sans-serif; font-size:
1.25em; }
```

13. Navigate to the mobile site and it should look similar to the following screenshot. Notice that the H1, H2, and H3 items will be in the new font face:



What just happened?

The Font-Your-Face module adds an interface to Drupal with several major font-embedding sources. We added the module and connected it to an account at Fonts.com. The module embeds code that downloads fonts for all web browsers and renders the fonts in the browser on the fly with the page. The fonts work on mobiles as well as desktop browsers.

Again, I feel the need to warn you. Be judicious with your font downloading. It can add to a page's download time significantly.

Pop Quiz

1. Display Suite adds the following functionality to Drupal:
    a. The ability to have an unlimited number of node build modes
    b. The ability of content types to vary layout and add code-based fields
    c. The ability of node reference fields to display nodes in a specific build mode when referenced
    d. The ability to a Allow modes to have multi-column layouts with little or no templating
    e. All of the above
2. jQuery Mobile library:
    a. Adds touch events to the jQuery library
    b. Adds iPhone-style theming to the page
    c. Both 1 & 2
    d. Neither 1 nor 2
3. In order to Add `data-*` properties to menu items:
    a. It cannot be done
    b. You must install the custom menu attributes module and write a hook
    c. You must reconfigure the views module
    d. You must clear the Drupal menu cache
4. Which web font technology does Firefox support?
    a. EOT
    b. WOFF
    c. SVG
    d. OpenType
    e. Both 2 and 4
    f. None of the above

5. Which web font technology does Internet Explorer support?
    a. EOT
    b. WOFF
    c. SVG
    d. OpenType
    e. Both 2 and 4
    f. None of the above
6. Which web font technology does iPhone support?
    a. EOT
    b. WOFF
    c. SVG
    d. OpenType
    e. Both 2 and 4
    f. None of the above

Summary

In this chapter, we pulled together different elements of the mobile theme to extend a unified jQuery Mobile look and feel to our site. We learned more about how Display Suite themes its nodes and outputs generic DIV tags with classes in order to allow column grids with little or no CSS creation on our part.

We changed our menu page view, so that we could use what we had learned about the Display Suite and to put into practice the great AJAX features that jQuery mobile has to offer.

We fixed the **Location** menu item so that it would load in the browser correctly without using AJAX.

Lastly, we added some fonts beyond the five web-safe fonts, which we are all used to using. Finally, our jQuery mobile site looks good now!

10
Tabula Rasa: Nurturing your Site for Tablets

Contrary to popular belief, tablet computing didn't begin with the iPad. In 1983, Radio Shack introduced the TRS-80 Model 100. It was a white, slate-like, all-in-one computer with an 8-line LCD screen and a full QWERTY keyboard, and it cost about 1,000 USD. Apple introduced the Newton and its laptop-like cousin, the eMate, in the mid 90s and it had an instant cult following. The handwriting-recognition feature was revolutionary for the time, although still lacking in its ability to be used day-to-day.

Palm had several experimental Palm Pilots that were larger than the standard handheld, but never made the technology a shipping product until HP bought the company and they released the HP Touchpad to compete with the iPad. Hewlett Packard and Toshiba had been selling tablet style Windows-based machines for several years before the iPad premiered. Windows-based tablets were sold to vertical markets such as hospitals and medical clinics for specific pieces of software. It's interesting to note that those software packages are now struggling to remake their software for Android and iOS tablets.

In this chapter, we'll learn about the tablet revolution. We'll look at how the tablet is changing both desktop and mobile computing and making us re-think some of our choices for multiple designs and multiple URLs for our website.

In this chapter, we'll:

- Examine touch events and go over the differences between touch events and mouse-click events
- Learn to add touch events to our jQuery cycle on the home page
- Take a look at the adaptive web page designs and begin the process of adapting a design for three layouts—phone, tablets, and desktop
- Learn to set the viewport with JavaScript

This chapter may seem a little iPad-heavy. However, iPad web traffic is about 50 times that of its nearest competitor. How long it will retain this lead is debatable and only time will tell. No one, however, can argue that the success of the iPad ignited sales and a re-imagination of just what exactly a tablet computer could do and the markets they could address. There is no industry on earth that will not be impacted by lower-cost, tablet-style ubiquitous computing.

The human touch

There's a reason touchscreen interfaces were rarely used before Apple re-invented them in the iPhone. It's because programming them is very difficult. With a mouse-driven interface you have a single point of contact: the mouse's pointer. With a touchscreen, you potentially have ten points of contact, each one with a separate motion. And you also have to deal with limiting spurious input when the user accidentally touches the tablet when they didn't mean to. Does the user's swipe downward mean they want to scroll the page or to drag a single page element? The questions go on to infinity.

With this chapter, we stand on the shoulders of those giants who have done the heavy lifting and given us a JavaScript interface that registers touch and gestures for use in our web pages. Many Bothans died to bring us this information.

To understand the tablet is to understand the touch interface, and in order to understand the touch interface, we need to learn how touch events differ from mouse events. But that begs the question: what is an event?

The event-driven model

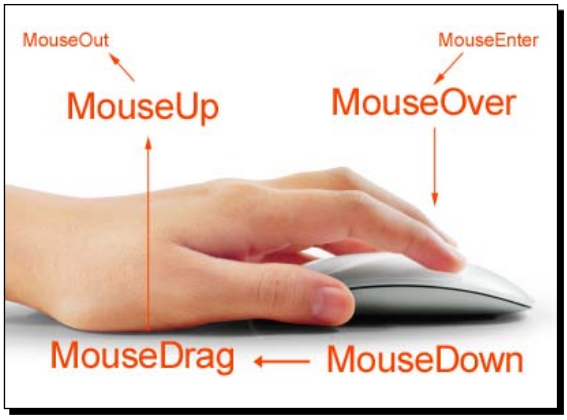
Many developers use JavaScript-based events and have not even the slightest clue as to what they can do or their power. In addition, many developers get into situations where they don't know why their events are misfiring or, worse yet, bubbling to other event handlers and causing a cascade of event activity.

As you may or may not know, an HTML document is made up of a series of tags organized in a hierarchical structure called the **HTML document**. In JavaScript, this document is referred to through the reserved word document. Simple enough, right? Well, what if I want to interact with the tag inside of a document, and not the document as a whole? Well, for that we need a way of addressing nested items inside the main `<html>` tag. For that, we use the **Document Object Model** (**DOM**).

DOM is a cross-platform and language-independent convention for representing and interacting with objects in HTML, XHTML, and XML documents. Aspects of the DOM (such as its elements) may be addressed and manipulated within the syntax of the programming language in use. The public interface of a DOM is specified in its **Application Programming Interface** (**API**). For more details on DOM, refer to the Wikipedia document at:

`http://en.wikipedia.org/wiki/Document_Object_Model`

The body of that document then becomes `document.body`. The head of the document, likewise, becomes `document.head`. Now, what happens when your mouse interacts with this web page? This is said to be a DOM **event**. When you click, the elements that are the receivers of that action are said to *propagate* the event through the DOM. In the early days, Microsoft and Netscape/Firefox had competing ways of handling those events. But they finally gave way to the modern W3C's standard, which unifies the two ways and, even more importantly, jQuery has done a lot to standardize the way we think about events and event handling. In most browsers today, mouse events are pretty standardized, as we are now more than 20 years into the mouse-enabled computing era:



For tablets and touchscreen phones, obviously, there is no mouse. There are only your fingers to serve the purpose of the mouse. And here's where things get simultaneously complicated as well as simple.

Touch and go

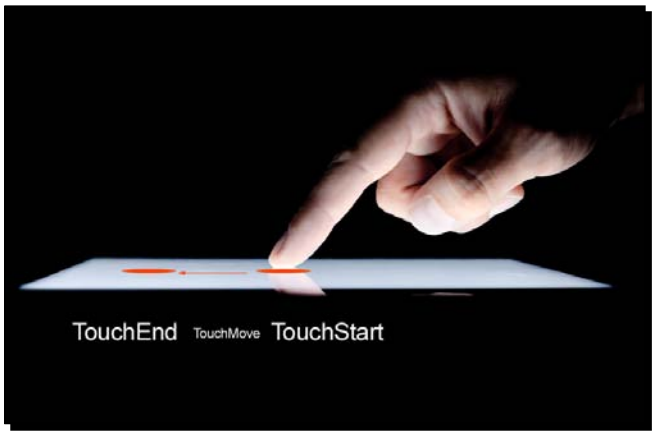
Much of what we talk about as touch interaction is made up of two distinct types of touches—single touches and gestures. A single touch is exactly that. One finger placed on the screen from the start till the end. A gesture is defined as one or more fingers touching the surface of the area and accompanied by a specific motion: Touch + Motion. To open most tablets, you swipe your finger across a specific area. To scroll inside a `div` element, you use two fingers pushing up and down. In fact, scrolling itself is a gesture and tablets only respond to the scroll event once it's over. We will cover more on that later.

Gestures have redefined user interaction. I wonder how long it took for someone to figure out that the zoom in and zoom out is best accomplished with a pinch of the fingers? It seems so obvious once you do it and it immediately becomes second nature. My mom was pinching to zoom on her iPhone within the first 5 minutes of owning it.

Touch events are very similar to multiple mouse events without a hover state. There is no response from the device when a finger is over the device but has not pressed down. There is an effort on the part of many mobile OS makers to simulate the hover event by allowing the hover event to trigger with the first click, and the click event to trigger with the second click on the same object. I would advise against using it for any meaningful user interaction as it is inconsistently implemented, and many times the single click triggers the link as well as the hover-reveal in drop-down menus.

Not using the hover event to guide users through navigation changes the way we interact with a web page. Much of the work we've done to guide users through our pages is based on the hover-response event model to clue users in on where links are. We have to get beyond that.

Drop-down menus quickly become frustrating at the second and third levels, especially if the click and hover events were incorrectly implemented in the desktop browser. Forward and back buttons are rendered obsolete by a forward and backwards swipe gesture.



The main event

There are basically three touch events—`touchstart`, `touchmove`, and `touchend`. Gesture events are, likewise: `gesturestart`, `gesturemove`, and `gestureend`. All gestures register a touch event but not all touch events register gestures. Gestures are registered when multiple fingers make contact with the touch surface and register significant location change in a concerted effort, such as two or more fingers swiping, a pinch action, and so on.

In general, I've found it a good practice to use touch events to register finger actions; but it is required to return null on a touch event when there are multiple fingers involved and to handle such events with gestures.

jQuery mobile has a nice suite of touch events built into its core that we can hook into. But jQuery and jQuery mobile sometimes fall short of the interaction we want to have for our users, so we'll outline best practices for adding customized user touch events to both the full and mobile version of the demo site.

Let's get started…

Time for action – adding a swipe advance to the home page

The JavaScript to handle touch events is a little tricky; so, pay attention:

1. Add the following lines to both `sites/all/themes/dpk/js/global.js` and `sites/all/themes/dpk_mobile/js/global.js`:

```
Drupal.settings.isTouchDevice = function() {
  return "ontouchstart" in window;
}

if (Drupal.settings.isTouchDevice() ) {
  Drupal.behaviors.jQueryMobileSlideShowTouchAdvance = {
    attach: function(context, settings) {
      self = Drupal.behaviors.jQueryMobileSlideShowTouchAdvance;
jQuery.each(jQuery(".views_slideshow_cycle_main.
viewsSlideshowCycle-processed"), function(idx, value) {
          value.addEventListener("touchstart", self.
handleTouchStart);
          jQuery(value).addClass("views-slideshow-mobile-
processed");
        })
        jQuery(self).bind("swipe", self.handleSwipe);
    },
    detach: function() { }, original: { x: 0, y: 0}, changed: { x:
0, y: 0}, direction: { x: "", y: "" }, fired: false,
```

```
    handleTouchStart: function(evt) {
      self = Drupal.behaviors.jQueryMobileSlideShowTouchAdvance;
      if (evt.touches) {
        if (evt.targetTouches.length != 1) { return false; }
        if (evt.touches.length) { evt.preventDefault(); evt.
stopPropagation() }
          self.original = { x: evt.touches[0].clientX, y: evt.
touches[0].clientY }
        self.target = jQuery(this).attr("id").replace("views_
slideshow_cycle_main_", "");
        Drupal.viewsSlideshow.action({ "action": "pause",
"slideshowID": self.target });
        evt.target.addEventListener("touchmove", self.
handleTouchMove);
        evt.target.addEventListener("touchend", self.
handleTouchEnd);
      }
      },
    handleTouchMove: function(evt) {
      self = Drupal.behaviors.jQueryMobileSlideShowTouchAdvance;
      self.changed = {
                x: (evt.touches.length) ? evt.touches[0].clientX:
evt.changedTouches[0].clientX,
                y: (evt.touches.length) ? evt.touches[0].clientY:
evt.changedTouches[0].clientY
            };
      h = parseInt(self.original.x - self.changed.x), v =
parseInt(self.original.y - self.changed.y);
      if (h !== 0) { self.direction.x = (h < 0) ? "right":"left";
}
      if (v !== 0) { self.direction.y = (v < 0) ? "up": "down"; }
      jQuery(self).trigger("swipe");
    },
    handleTouchEnd: function(evt) {
      self = Drupal.behaviors.jQueryMobileSlideShowTouchAdvance;
      evt.target.removeEventListener("touchmove", self.
handleTouchMove);
      evt.target.removeEventListener("touchend", self.
handleTouchEnd);
      self.fired = false;
    },
    handleSwipe: function(evt) {
   self = Drupal.behaviors.jQueryMobileSlideShowTouchAdvance;
   if (evt != undefined && self.fired == false) {
    Drupal.viewsSlideshow.action({ "action":  (self.direction.x
== "left")?"nextSlide":"previousSlide", "slideshowID": self.target
```

```
});
    self.fired = true; //only fire advance once per touch
   }
  }
 }
}
```

2. Clear Drupal's cache by either navigating to **Configuration** | **Performance** and clicking on the **Clear cache** button or entering these lines in a terminal:

```
cd ~/sites/dpk/
drush cc all
```

3. Navigate to either home page with a touch-enabled device and you should be able to advance the home page slideshow with your fingers.

What just happened?

Let's take a look at how this code works. First, we have a function, `isTouchDevice`. This function returns `true`/`false` values if touch events are enabled on the browser. We use an `if` statement to wall off the touchscreen code, so browsers that aren't capable don't register an error. The Drupal behavior `jQueryMobileSlideShowTouchAdvance` has the `attach` and `detach` functions to satisfy the Drupal behavior API. In each function, we locally assign the `self` variable with the value of the entire object. We'll use this in place of the `this` keyword. In the Drupal `behavior` object, `this` can sometimes ambiguously refer to the entire object, or to the current sub-object. In this case, we want the reference to be to just the sub-object so we assign it to `self`. The `attach` function grabs all `slideshow_cycle` `div` elements in a jQuery `each` loop. The iteration of the loop adds an event listener to the `div` tag. It's important to note that the event listener is not bound with jQuery event binding. jQuery event binding does not yet support touch events. There's an effort to add them, but they are not in the general release that is used with Drupal 7. We must then add them with the browser native function, `AddEventListener`. We use the `handleTouchStart` method to respond to the `touchstart` event. We will add `touchend` and `touchmove` events after the `touchstart` is triggered.

The other event that we're adding listens to this object for the `swipe` event. This is a custom event we will create that will be triggered when a swipe action happens. We will cover more on that shortly.

The `detach` function is used to add cleanup to items when they are removed from the DOM. Currently, we have no interaction that removes items from the DOM and therefore no cleanup that's necessary for that removal to take place.

Next, we add some defaults—`original`, `changed`, `direction`, and `fired`. We'll use those properties in our event response methods.

`HandleTouchStart` event is fired when the finger first touches the surface. We make sure the `evt.touches` object has value and is only one touch. We want to disregard touches that are gestures. Also, we use `preventDefault` and `stopPropagation` on the event to keep it from bubbling up to other items in the DOM. `self.original` is the variable that will hold the touch's original coordinates. We store the values for `touch[0]`. We also name the target by getting the DOM ID of the cycle containing the `div` element. We can use string transforms on that ID to obtain the ID of the jQuery cycle being touched and will use that value when we send messages to the slideshow, based on the touch actions, like we do in the next line. We tell the slideshow to pause normal activity while we figure out what the user wants. To figure that out, we add `touchmove` and `touchend` events listening to the `div` element. `handleTouchMove` figures out the changed touch value. It does so by looking at the `ClientX` and `ClientY` values in the touch event.

Some browsers support the `changedTouches` value which will do some calculations on how much the touch has changed since the last event was triggered. If it's available, we use it, or we use the value of the X and Y coordinates in the touch event's `touches` array. We do some subtraction against the original touch to find out how much the touch has changed and in what direction. We use `self.direction` to store the direction of the change. We store the direction in and tell the world that a swipe has begun on our `div` element by triggering a custom event on our `self` object.

If you remember correctly, we used the `handleSwipe` method to respond to the `swipe` event. In `handleSwipe` we make sure the event has not already fired. If it hasn't, we use that `swipe` event to trigger a `next` or `previous` action on our jQuery cycle slideshow. Once we've fired the event, we change the `self.fired` to true so it will only fire once per touch.

In the `touchend` responder, `HandleTouchEnd`, we remove both the `touchmove` and `touchend` responders and reset the fired state.

But adding the touch events to both the desktop and the mobile themes begs the question, "Into which category does the table fall?"

Have a go hero – adding a swipe gesture

Add a `swipe` gesture event to the **Menu Item** page that allows you to scroll through menu items.

The changing landscape (or portrait)

So, if you remember, back in *Chapter 4*, *Introduction to a Theme,* we decided to go with two sites, one for mobile one for desktop. This is where that choice breaks down. Are tablets desktops or should they use the standard site? When does a screen size become a tablet versus a mobile phone? And are those distinctions increasingly irrelevant? The answer to the last question is, most assuredly, yes. This is where a new design comes into play.

Responsive web design is a design discipline that believes that the same markup should be used for both desktop and mobile screens, with the browser managing the display of items, rather than the user choosing an experience. If the screen is smaller, the layout adjusts and content emphasis remains.

Conversely, the popularity of Internet-connected game consoles and DVI ports on large screen televisions gives us yet another paradigm for web pages—the large screen. I sit in front of a 72" TV screen and connect it to either my laptop or iPad and I have a browsing experience that is more passive, but completely immersive.

Right now, I bet you're thinking, "So which is it Mr Author, two sites or one?" Well, both, actually. In some cases, with some interactions it will be necessary to do two site themes and maintain them both. In some cases, when you can start from scratch, you can do a design that can work on every browser screen size. Let's start over and put responsive design principals to work with what we already know about media queries and touch interfaces.

"Starting over" or "Everything you know about designing websites is wrong"

Responsive web design forces the designer to start over—to forget the artificial limitations of the size that print imposes and to start with a blank canvas. Once that blank canvas is in place, though, how do you fill it? How do you create "The One True Design" (cue the theme music)?

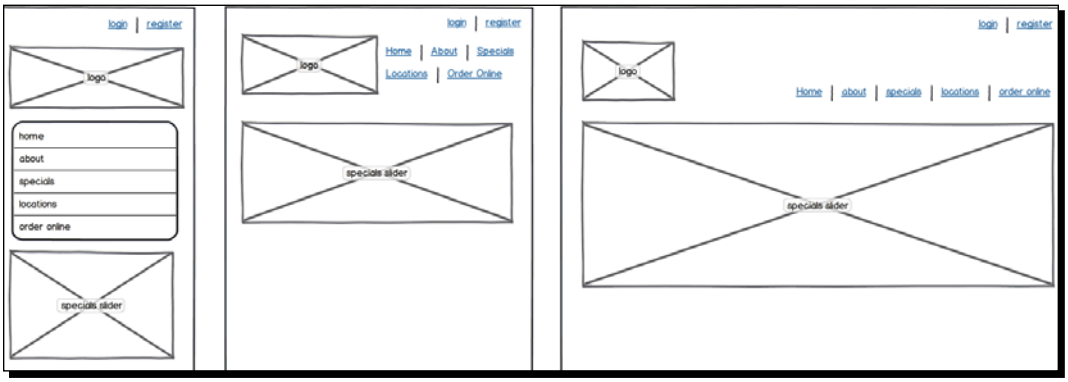
This book is not a treatise on how to create the perfect design. For that, I can recommend *A Book Apart* and anything published by `smashingmagazine.com`. Currently, they are at the forefront of this movement and regularly publish ideas and information that is helpful without too much technical jargon.

No, this book is more about giving you strategies to implement the designs you're given or that you create using Drupal. In point of fact, responsive design, at the time of writing, is in its infancy and will change significantly over the next 10 years, as new technology forces us to rethink our assumptions about what books, television, and movies are and what the Web is.

So suffice to say, it begins with content. Prioritizing content is the job of the designer. Items you want the user to perceive first, second, and third are the organizing structure of your responsive design makeover. In most instances, it's helpful to present the web developer with four views of the website.

Wire framing made easy

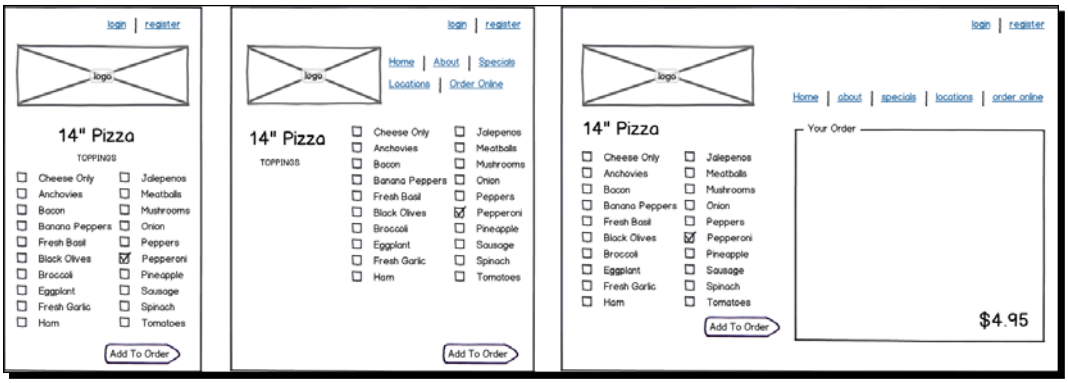
Start with wireframes. A great wire framing tool is called **Balsamiq**. It has a purposefully "rough" look to all of the elements you use. That way, it makes you focus on the elements and leave the design for a later stage. It's also helpful for focusing clients on the elements. Many times the stake holders see a mockup and immediately begin the discussion of "I like blue but I don't like green/I like this font, but don't like that one." It can be difficult to move the stake holders out of this mindset, but presenting them with black-and-white chalk-style drawings of website elements can, in many cases, be helpful. Balsamiq is a great tool for doing just that:



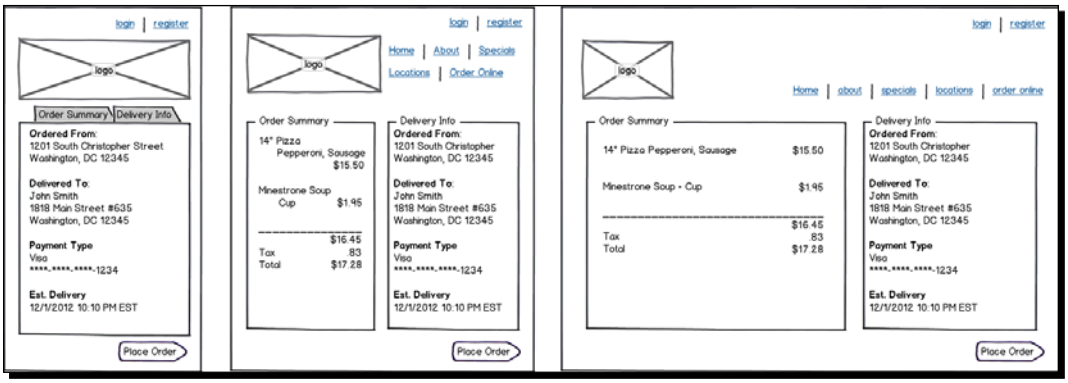
These were created with Balsamiq but could have been created in almost any primitive drawing program. There are many free ones as well as the more specialized pay ones. A simple layout like this is very easy to plan and implement. But very few of the websites you develop will ever be this simple. Let's take for instance that the menu item we have not, as yet, implemented, is for online ordering. How does that work? What do those screens look like? At this point we have a **Menu** page but, as per this mockup, that menu page will become the online ordering section. How do we move these menu items we created in *Chapter 3*, *Selecting the Right Domain for your Mobile Site*, to a place where they can be put in an order and paid for? And more importantly, how does each location know what was ordered from their location?

These are questions that come up in the mockup and requirements phase and whether you are building the site yourself or being given requirements from a superior, or a client, you now have a better idea of the challenges you will face implementing the single design for this site.

Again, this book will focus on the implementation more than the design phase. With that, we've been given these mockups for the new online ordering system. The following mockup diagram is for adding an order:



The following mockup diagram is for placing an order:



We'll implement these mockups using the Drupal 7 **Commerce** module. The **Commerce** module is just a series of customized data entities and views that we can use as the building blocks of our commerce portion. We'll theme the views in the standard Drupal way but with an eye to multi-width screens, lack of hover state, and keeping in mind "hit zones" with fingers on small mobile devices. We'll also add some location awareness to assist with the delivery process. Once an order is placed, an e-mail will need to be sent to the correct franchise notifying them of the pizza order and initiating the process of getting it out the door.

Drupal 7 Commerce module

There are some things you need to know about the **Commerce** module before we begin. There are several steps to creating a product to be sold and then getting that product into a cart to checkout.

The first step is to create the product type. The product type is a kind of a content type for nodes. It defines the basic structure and fields that the item will have.

Once you've created the product type you can create products of that type. Simply to the nodes themselves, products are instances of their product type. Each product may have variations that will affect the price, such as size, and in the case of a pizza, it's toppings.

Once you've created the product type and have products of that type, you can create a cart line item to receive that product into the cart. The cart line item will allow you to show what product options are applied to that specific product instance.

Now that the product portion is created, you need to connect it with the rest of the Drupal universe. That's where product references come in. The product reference field is a field type that can be added to a note to connect it to a product. You can then add a link to the product to add it to the cart, as we will do in the *Time for action – the one true theme* section:

Product type	**Product**	**Cart line item**	**Node product reference field**
Analogous to content types for nodes. Defines the fields and options for a given product group.	Analogous to nodes but with pricing and features that may or may not alter pricing.	Shopping cart container that holds information about what product options were selected by the user as well as pricing based on those options.	Allows nodes to point to product entities. Connects products with the rest of the Drupal nodes.

Confused? It may make more sense when you see it in action. We've got a lot of work to do. Ready to begin?

Time for action – the one true theme

Let's download, install, and configure the Drupal **Commerce** module:

1. Open a terminal window and enter the following command:

```
drush dl rules addressfield commerce commerce_custom_line_items
```

2. It doesn't make sense to enable all of the modules that come with Drupal **Commerce** with Drush, so just navigate to **Sites** | **Modules** and ensure that all of the following modules are enabled: **Cart**, **Checkout**, **Commerce**, **Commerce UI**, **Customer**, **Customer UI**, **Line Item**, **Line Item UI**, **Order**, **Order UI**, **Payment**, **Payment Method Example**, **Payment UI**, **Price**, **Product**, **Product Pricing**, **Product Pricing UI**, **Product Reference**, **Product UI**, **Tax**, **Tax UI**, **Commerce Custom Line Items**.

3. Navigate to **Store** | **Products** | **Product types** | **Add product type**. Name our first product type **Pizza**. Save the product type and then click on **Manage Fields**.

4. Under **Add new field**, add a field called **Size** with the machine name **field_size** and the **List (text)** type displayed with the **Checkboxes/radio buttons** widget. Click on **Save**. On the **Pizza** settings screen, make the **Allowed values** field as follows. Then click on **Save**:

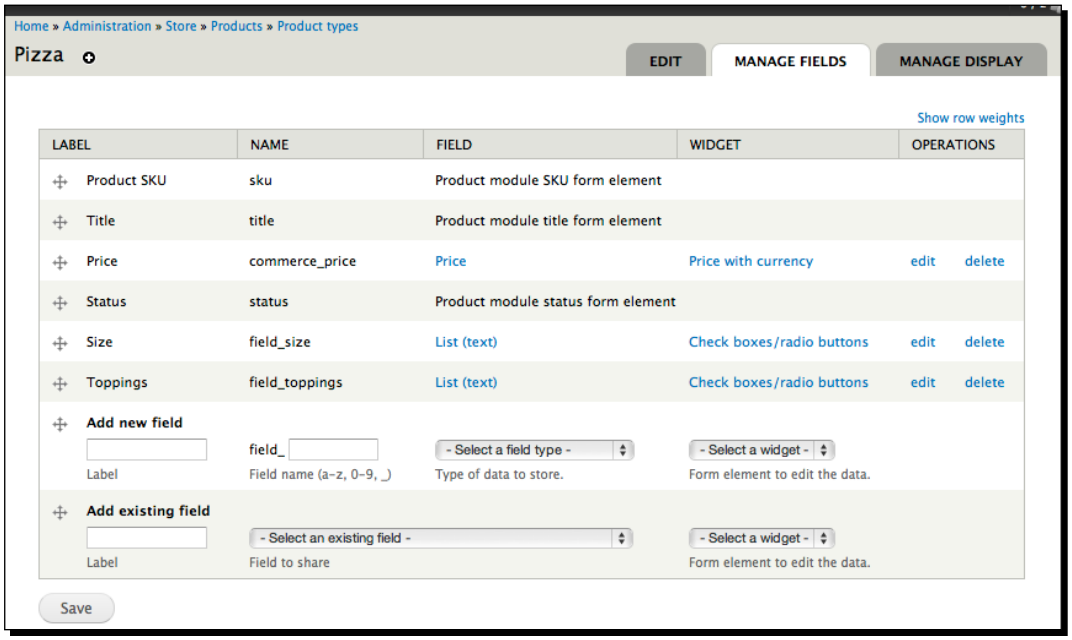
```
0|Large
16|Medium
14|Small
```

5. On the final settings page, make the **Size** field as a required field with the default value of **Large**. Under **Attribute field settings** make sure **Enable this field to function as an attribute field on Add to Cart forms** is checked and use the radio buttons widget. Save the new field.

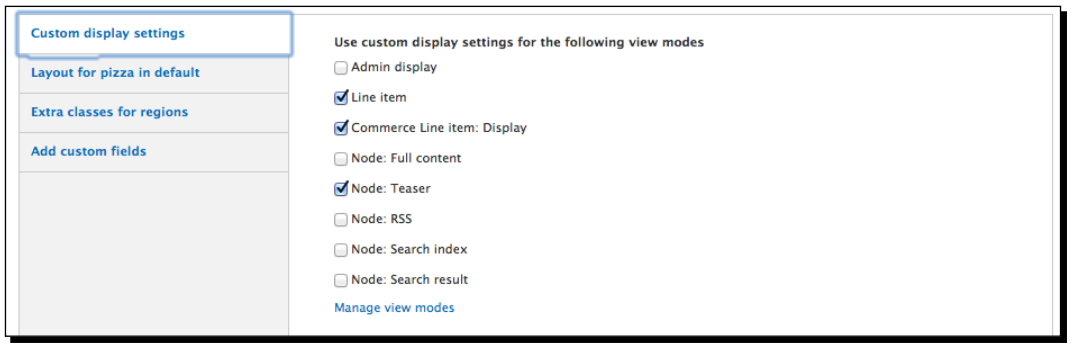
6. Create a new field called **Toppings** with the machine name **field_toppings** and the **List (text)** type displayed with the **Checkboxes/radio buttons** widget. Click on **Save**. Use the following settings for the **Allowed values** field, and then click on **Save**:

```
0|Cheese Only
ANC|Anchovies
BAC|Bacon
BAN|Banana Peppers
BAS|Fresh Basil
BOL|Black Olives
BRO|Broccoli
EGG|Eggplant
GAR|Fresh Garlic
GOL|Green Olives
GPP|Green Peppers
HAM|Ham
JAP|Jalepenos
MBL|Meatballs
MUS|Mushrooms
ONN|Onion
PEP|Pepperoni
PIN|Pineapple
SAU|Sausage
SPN|Spinach
TOM|Tomatoes
```

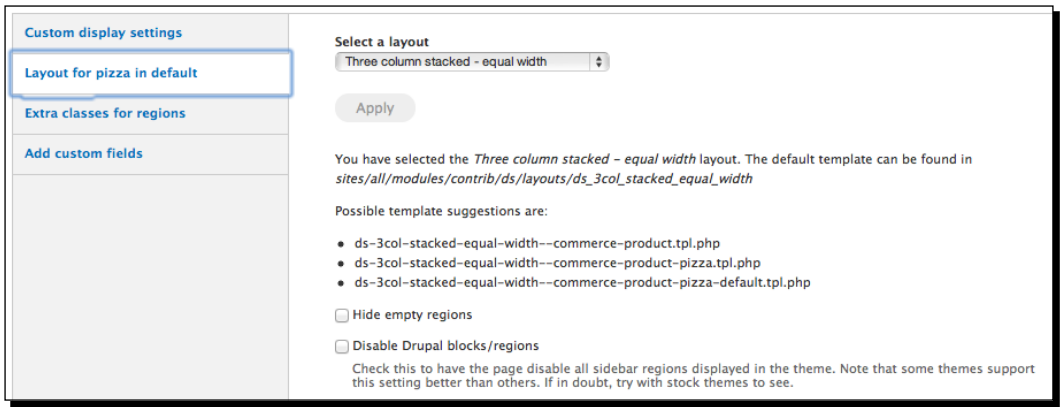
7. On the final settings page, make the **Size** field a required field with the default value of **Large**. Under **Attribute field settings**, make sure **Enable this field to function as an attribute field on Add to Cart forms** is checked and use the radio buttons widget. Save the new field:



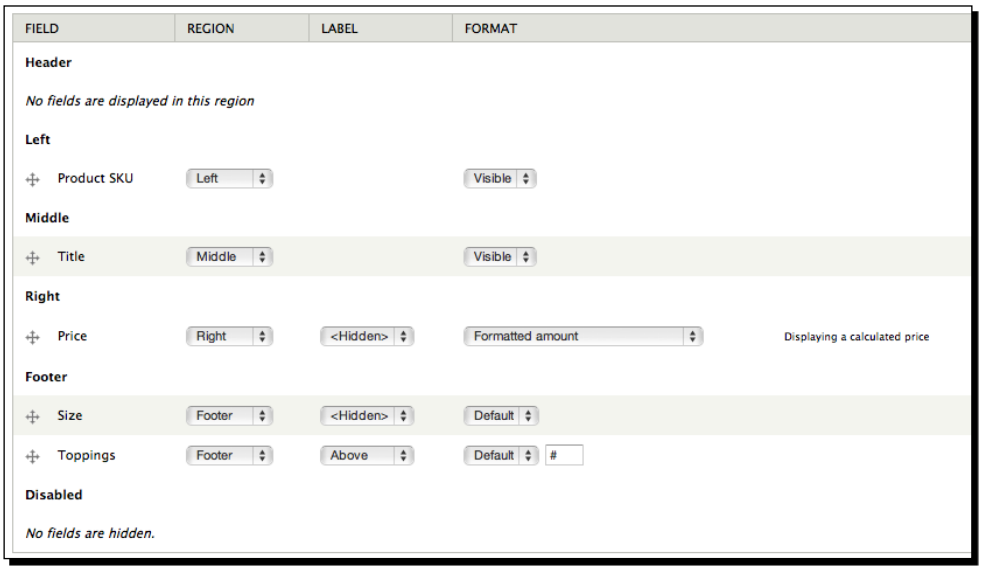
8. In the top right-hand corner, click on **MANAGE DISPLAY**. Underneath this tab are the currently active build modes. You may have just the default mode or you may have several build modes. At the bottom, there's an item called **Layout for pizza in default**. This panel controls which build modes you plan to customize:



9. The second vertical tab controls the layout for the build mode you are currently editing. The default build mode will be used for any display that is not customized:



10. Starting with the default build mode, select **Three column stacked – equal width** under the **Select a layout** section and click on **Save**. The presentation area should now have five regions, **Header**, **Left**, **Middle**, **Right**, and **Footer**. Drag the **Product SKU** to the **Left** region, **Title** to the **Middle** region, and **Price** to the **Right** region. Drag **Size** and **Toppings** to the **Footer** region as shown in the following screenshot, and save the display settings:



Do this for each active build mode. You can worry about further customizations later.

What just happened?

Remember our four-panel grid from the start of the exercise? I'll repeat it again so you don't have to keep looking back:

Product type	Product	Cart line item	Node product reference field
Analogous to content types for nodes. Defines the fields and options for a given product group.	Analogous to nodes but with pricing and features that may or may not alter pricing.	Shopping cart container that holds information about what product options were selected by the user as well as pricing based on those options.	Allows nodes to point to product entities and create an **Add to cart** button/form. Connects products with the rest of the Drupal nodes.

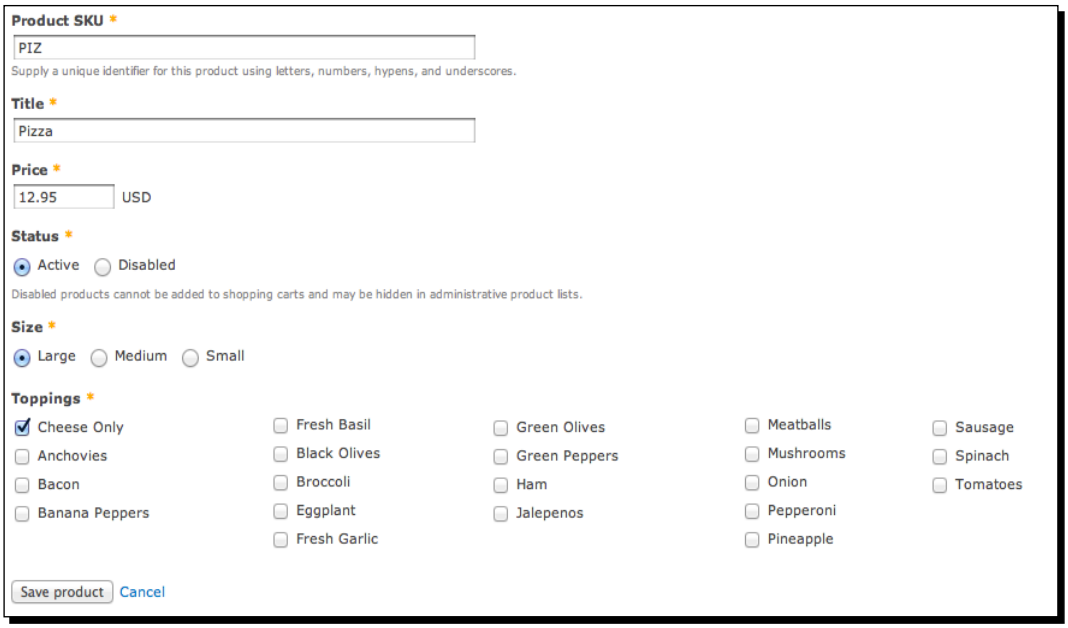
After the modules are installed and enabled, the first thing we did was define a product type. Before you create a node, you have to create a content type. This is the same concept. The product type serves as the prototype for a series of products with similar features. In this case, we basically have two features: **Size** and **Toppings**. We created the size and topping options in the product type. We set up the Display Suite settings for our new product type making sure that we display the size and toppings attributes.

Time for action – creating a product

Now, we need to create a product to be able to show the product on the frontend:

1. Navigate to **Store** | **Products** | **Add product** | **Create Pizza**.
2. Give the new product the following values then click on **Save product**, as shown in the screenshot:

Fields	Values
Product SKU	**PIZ**
Title	**Pizza**
Status	**Active**
Size	**Large**
Toppings	**Cheese Only**

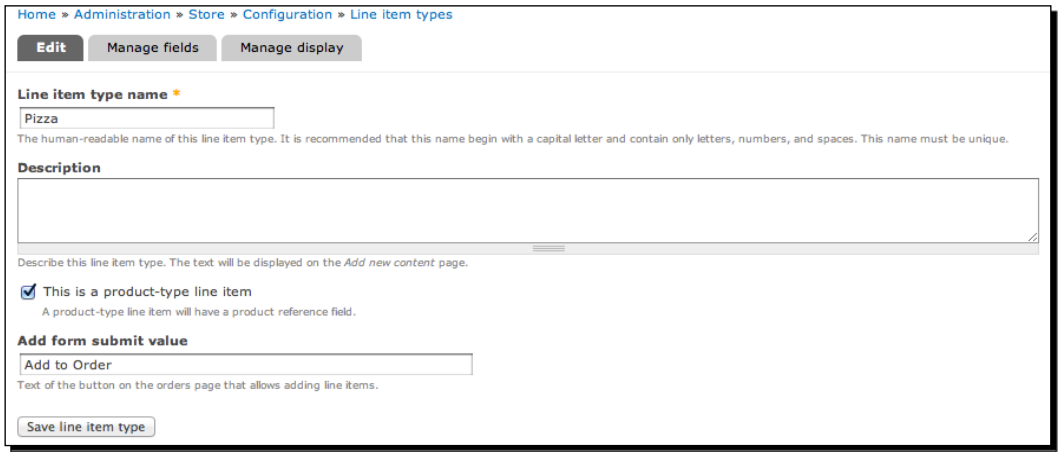


3. Navigate to **Store** | **Configuration** | **Line item types** | **Add line item type**. Name the new line item type **Pizza**, check the **This is a Product-type Line Item** checkbox, and change the **Add form submit value** to **Add to order**. Click on **Save line item type** and then click on **Manage fields**.

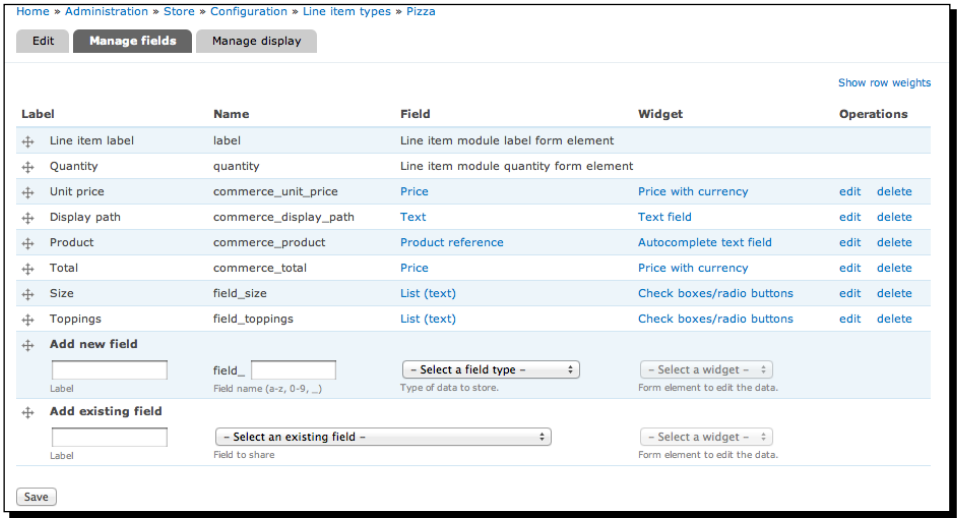
4. Under the **Add an existing field** section, click on the **Select an existing field** pull-down menu and select **field_size**. Add the label **Size** and the form widget as **Check boxes/radio buttons**, then click on **Save**. Make this field required with a default value as **Large**. Check the **Add to cart form settings** checkbox. It can have the value as **1**, and the allowed values should be filled in from the **Product type** listing. They should be the same as in step 4 in the *Time for action – the one true theme* section. Save the field.

5. Under **Add an existing field** section, click on the **Select an existing field** pull-down menu and select **Toppings**. Add the label as **Toppings** and the form widget as **Check boxes/radio buttons** then click on **Save**. Make this field as required with a default value of **Cheese Only**. Check the **Add to cart form** checkbox. It can have unlimited values and the allowed values should be filled in from the **Product type** listing. They should be the same as in step 6 in the *Time for action – the one true theme* section. Save the field.

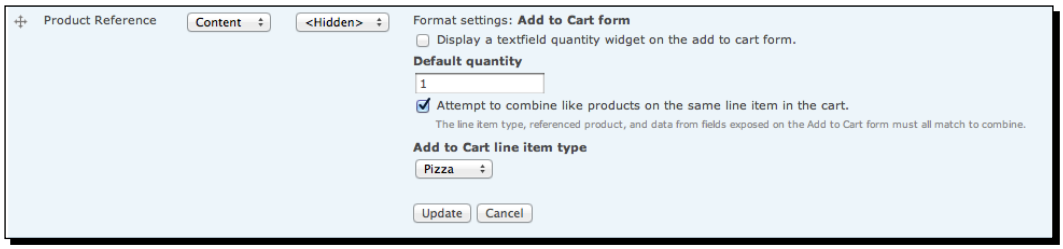
6. Navigate to **Structure** | **Content Types** | **Pizza** | **Manage fields**. Add a new field with the label **Product reference** and a machine name of **field_product_ref** of the type **Product Reference** using the **Autocomplete text field** widget. On the settings page check the **Render fields from the referenced Products when viewing this entity** checkbox. Save settings for the new field:



7. Click on the **Manage display** tab of the **Pizza** content type. Using the handlebars beside the fields, replicate the same build mode settings and layout from steps 9 to 11 from the *Time for action – the one true theme* section—**Three column stacked**, with **Toppings** and **Size** in the **Footer** region. Once you've moved the **Product Reference** up, for **Format**, choose **Add to Cart form**. Move the old pricing fields to the **Disabled** section. Our product pricing will replace them:



8. Click on the line labeled **Product Reference**. On the select menu labeled **Add to Cart line item type**, select **Pizza**, and then click on **Update**:



9. Navigate to **Admin** | **Content** and filter the list so that you see only items of the **Pizza** type. Edit the **Each topping** node and under **Publishing options**, uncheck the **Published** checkbox. Save the node.

10. Edit the **Cheese Pizza** node. Delete the values out of the **Price** field and, in the **Product Reference** field, begin entering **Pizza**. The auto-complete feature will finish your typing with the **Pizza** product you created in Steps 13 and 14 in the earlier exercise. Add the following line to the body of the node:

```
Each toping: add $1.35/$1.65/$1.90 for SM/MD/LG
```

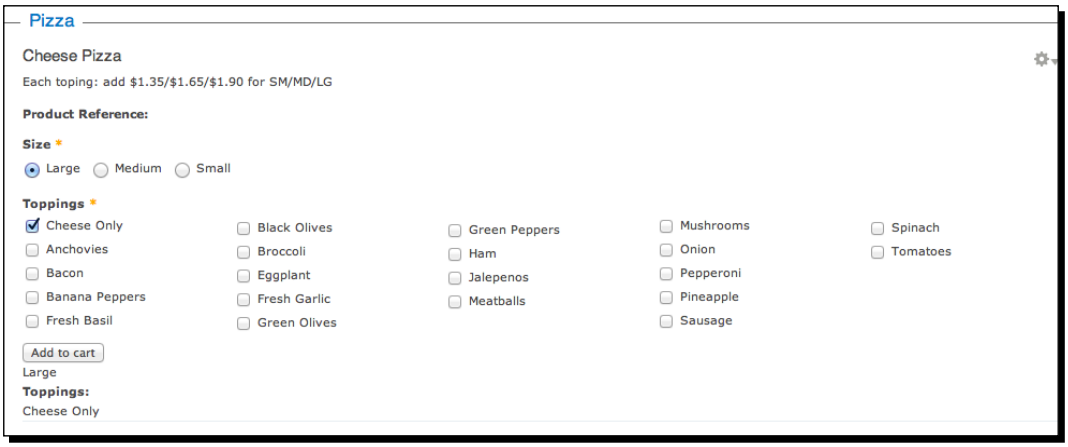
11. Save the node's display settings.

12. Hop over to your text editor and open the `sites/all/themes/dpk/css/styles.css` file. Add the following lines:

```
 .form-item .form-checkboxes {
  -moz-column-count: 5; -moz-column-gap: 1em;
  -webkit-column-count: 5; -webkit-column-gap: 1em;
  column-count: 5; column-gap: 1em; }
 .form-item .form-radios { float:none; clear: both;
overflow:hidden; }
 .form-item .form-radios .form-type-radio { float:left; margin-
left: 10px;   }
 .form-item .form-radios .form-type-radio:first-child { margin-
left: 0px; }
 .node.node-menu-item-group { border: 0px none;}
 .node-menu-item-group > h2 { margin: 0px 0px -15px 20px; }
 .node-menu-item-group > h2 a { background: white; padding: 0px
.5em .5em .5em; }
 .node-menu-item-group .content { padding: 2em; border: 1px solid
#666; }
```

13. Clear the Drupal cache.

14. Navigate to the **Menu** page, and your **Pizza** section should now look similar to the following screenshot:



What just happened?

We created a product for the type, **Pizza**. The **Pizza** product's default size is **Large** with **Cheese Only** as the default topping.

Once we have the product type and the product, we then created the cart line item. This storage bucket holds user-chosen products once they are added to the cart. We created a customized line item just for pizza that will hold the toppings and size that the user has chosen.

Now that we have these three, we need to connect the behind-the-scenes product with the node system where Drupal displays the bulk of its content. We do that through the **Product Reference** field. We added a **Product Reference** field to our content objects that show up on our **Menu** page, in this case, **Cheese Pizza**. We adjusted the Display Suite settings for our **Pizza** node and unpublished the node that explains the pricing structure for extra toppings. Once we have Display Suite set up correctly and the product's backend connected to the node frontend, we are able to see the **Add to Cart** form with all the toppings and sizes.

Now that we have the basics of how to create cart items, let's work out some strategies for implementing the mockups.

Have a go hero – adding a PayPal payment option

Add a PayPal payment option to the checkout process. When the checkout process is finished, add an e-mail to both the customer and the correct franchise that is responsible for delivery.

A room with a viewport

Since their earliest version, mobile browsers have been able to do something desktop browsers cannot and that was scaling the screen. They've had to do this because screen real estate on handhelds is at a premium. Before the iPhone perfected the pinch to zoom, you could use keys on Nokia and Blackberry browsers to zoom into and out of a rendered HTML screen. Most mobile browsers assume that standard HTML documents are meant to have a width of 980 pixels. Fortunately, you can control the width of the rendered screen with a `<meta>` tag. You can set the viewport scale at the optimum width for whatever screen on which its being viewed. I usually set the viewport at the optimum for the 320 by 480 pixels screen and use JavaScript to change the settings if the screen is larger.

Let's take a look at the settings in the `html.tpl.php` file:

```
<meta name="viewport" content="width=760, initial-scale=0.74, minimum-
scale=0.4">
```

We can use JavaScript to adjust this setting when the screen calls for it.

Time for action – setting the viewport with JavaScript

Edit the `sites/all/themes/dpk/js/global.js` file. Add the following lines:

```
Drupal.behaviors.browsers = {
 attach: function(context, settings) {
  if (Drupal.settings.isTouchDevice()) {
            // default is iPad settings, if phone, swap it out
            if ($(window).width() <= 480 ) {
                $("meta[name=viewport]").attr("content",
"width=device-width, initial-scale=1, maximum-scale=1")
            }
        }
},
detach: function(context,settings){}
}
```

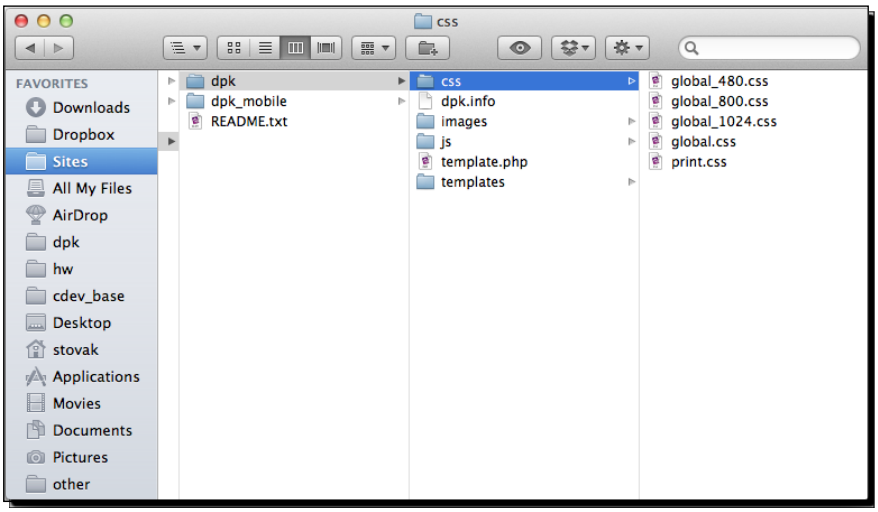
What just happened?

So the first item sets the `<meta>` tag, `viewport` with settings that are complimentary to an iPad or large-format tablet. We set the `width` to `760` which will make the page visible on the tablet and is squarely in the center of our middle width. If the screen size is less than 480 pixels, the JavaScript behavior trades out the `viewport` setting for the one that sets the width of the `viewport` at the device's width.

Now, let's add back the media queried style sheets and implement the wireframe designs.

Time for action – advanced media queries for tablets

1. Rename `sites/all/themes/dpk/css/styles.css` to `sites/all/themes/dpk/css/global.css`. As shown in the following screenshot, add three more files in that folder—`global_480.css`, `global_800.css`, and `global_1024.css`.



2. Edit the file, `sites/all/themes/dpk/dpk.info`. Add the highlighted lines in the following code snippet:

```
stylesheets[all][] = css/global.css
stylesheets[print][] = css/print.css
stylesheets[screen and (max-width: 480px)][] = css/global_480.css
stylesheets[screen and (min-width: 481px) and (max-width: 1024px)]
[] = css/global_800.css
stylesheets[screen and (min-width: 1024px)][] = css/global_1024.
css
```

3. Edit the file, `sites/all/themes/dpk/css/global.css`. Change the lines as shown in the following code snippet. Delete the `min-width` lines and add the highlighted content:

```
/* With 3 columns, require a minimum width of 1000px to ensure
there is enough horizontal space. */
body.two-sidebars {
}
/* With 2 columns, require a minimum width of 800px. */
body.sidebar-first,
body.sidebar-second {
}
```

```
/* We must define 100% width to avoid the body being too narrow
for near-empty pages */
#wrapper #container #center {
  float: left; /* LTR */
  width: 100%;
}

/* So we move the #center container over the sidebars to
compensate */
body.sidebar-first #center {
  margin-left: -15%; /* LTR */
}
body.sidebar-second #center {
  margin-right: -15%; /* LTR */
}
body.two-sidebars #center {
  margin: 0 -15%;
}

/* And add blanks left and right for the sidebars to fill */
body.sidebar-first #squeeze {
  margin-left: 15%; /* LTR */
}
body.sidebar-second #squeeze {
  margin-right: 15%; /* LTR */
}
body.two-sidebars #squeeze {
  margin: 0 15%;
}

footer { height: 100px; background: #333; text-align: center;
color: white; position: fixed; bottom: 0px; left: 0px; width:
100%; }
```

4. Add the following lines to `sites/all/themes/dpk/css/global_480`:

```
#views_slideshow_cycle_main_home-page { display:none; }
#main-header { height: 65px; }
#main-header .container { height: 65px; }
.search-form { top: 10px; right: 0px; }
#main-header h1 a { width: 150px; height: 55px;
  background: transparent url(../images/logo_sm.png) no-repeat
center center;
  }
```

```
#commerce-checkout-form-checkout fieldset.cart_contents {
float:left; }
#commerce-checkout-form-checkout fieldset.customer_profile_billing
{ }
#commerce-checkout-form-checkout fieldset.customer_profile_billing
.fieldset-wrapper { display:none; }
#commerce-checkout-form-checkout fieldset.checkout-buttons {clear:
both; }

footer { position: fixed; bottom: 0px; left: 0px; width: 100%;
height: 40px; }
```

5. Open a WebKit-based browser or Firefox and visit `http://dpk.local/menu`. Add a menu item to the cart. If you size the window up and down, the layout now responds to size change, by making both the header and footer logos smaller and in fixed position.

What just happened?

This is where the importance of semantic markup comes into play. Notice how the logo is not an image, it's an `<h1>` and `<a>` tag with a background image, width, and height applied by CSS rule. Well, because there's no hard image on the page as we size up and down we can swap out the width and height on the `<a>` tag. Tricks such as these are the basis of responsive design. We use CSS rules to describe the three base layouts of our site—phone, tablet, and desktop.

Pop quiz

1. The DOM is:
    a. Cross-platform
    b. Cross-language
    c. A programmatic way of addressing HTML tags
    d. None of the above
2. Commonly used mouse events are:
    a. `mouseenter`, `mouseleave`,`mouseup`, `mousedown`
    b. `mousemove`, `mousestart`, `mouseend`
    c. `mouseback`, `mouseforward`, `mouseadvance`
    d. None of the above

3. Commonly used touch events are:
    a. `touchenter` and `touchleave`
    b. `touchup` and `touchdown`
    c. `touchback` and `touchforward`
    d. `touchstart` and `touchend` and `touchmove`
4. Responsive design:
    a. Seeks to have a single design for all screen sizes
    b. Depends on media queries to serve the correct CSS
    c. Both a and b
    d. Neither a nor b
5. Drupal Commerce product types:
    a. Are analogous to content types for nodes
    b. Define the fields and options for a given product group
    c. Both a and b
    d. Neither a nor b
6. Drupal Commerce products are:
    a. Analogous to nodes
    b. Feature pricing
    c. Features that may or may not alter pricing
    d. All of the above
7. Drupal Commerce cart line items:
    a. Add a shopping cart container that holds information about products
    b. Understand which options were chosen by the user
    c. Calculate pricing based on features and options
    d. All of the above
8. Node product reference fields:
    a. Point to products
    b. Associate nodes with product entities
    c. Are able to display an **Add to Cart** button
    d. All of the above

Summary

In this chapter, we examined touch events and went over the differences between touch events and mouse-click events. We added touch events to our jQuery cycle on the home page so that the user can advance the slideshow with their fingers.

We took a second look at adaptive web page designs and began the process of adapting a design for three layouts. We added CSS for each layout and began the process of defining the look and feel of each of the three use cases.

In the next chapter, we'll go over staging and then deploying a Drupal site in a way where the web developer doesn't spend hours on configuring the server.

11
A Home in the Clouds

At this point in the writing of this book, the earlier chapters of the book have gone on sale in the raw form, and I am starting to get some user feedback. Most of the newbies that have read, and used, the book have had a similar reaction: "There is too much command line stuff. Why do we have to learn Drush? Do we really need all the command line stuff to make the website work? Why can't you make it a simple, easy-to-use web interface?"

Well, there is a reason why Drush and drush_make are powerful tools, and you are going to see in this chapter how it all fits together. Also, you will see how using a drush_make-based workflow will free you from many of the boring repetitive tasks that occupy your day so many times, and impede development as well as progress.

In this chapter, we will cover the following topics:

- Understanding some of the problems with our current system of development
- Setting up an Amazon virtual hosting account
- Setting up an account with the RightSpace virtual host management
- Creating a server pattern for our RightSpace host manager to clone the servers off it
- Cloning a development server
- Installing Jenkins on that server
- Using Jenkins to deploy our code from GitHub
- Using Jenkins to backup or restore our databases
- Using Jenkins to sync our sandbox server to our production server

Problems introduced by modern websites

Once upon a time, web pages were a single HTML document. When you made them live, you logged into a server you were sharing with 100 other websites, and uploaded the files through the **File Transfer Protocol** (**FTP**). For each web page, you had maybe six, eight, or ten images that needed to be uploaded to the images folder along with the web page. It was a simpler time then.

But then technologies such as PHP, JavaScript**,** and CSS changed all that. It was no more that a web page was confined to one page. Instead, you had multiple pages, and several files that were shared across the multiple web pages. There were tons of image files for each web page, and you had to take care to name each image so that it did not conflict with the naming of the other images for other pages. You tried to establish commonalities among the pages. You may have put your header and footer in a single file to aid with the organization. Your pages became a mound of spaghetti code with inclusions from all over your document tree.

Then along came Drupal. It allowed you to concentrate on a single code tree, and share a theming layer among all the pages. If you built custom functionality into the modules, you could compartmentalize so that the look and feel of the site was consistent.

But now we have another series of problems:

- Upgrading Drupal, and its contributed and core modules, when security updates become available.
- Staging your site for development and production so that the development site is off limits to the search engines, prying eyes, and hackers, and the live site never gets the experimental code that might bring it to its knees.
- Testing functionality on devices like the phones and tablets as you are developing your site on your local machine.
- Syncing the code and database between the live site and the "staging" site and multiple other environments.
- You have seven clients, that is, all Drupal installations. All of them have servers on different hosting providers with different configuration options on each server, but you are just a single web developer. How do you manage that mess without an entire operations department at your service?

All these problems have been solved in multiple ways by different developers all over the world with multiple open source tools, and we should learn from their mistakes, and create a development workflow that works for the project. We are going to suggest some strategies that we hope will be helpful for using the latest in Cloud services, more specifically, RightScale and Amazon. We will set up some accounts on Amazon or Rackspace and RightScale to begin this process, and go through a sample server and code deployment. But first, let's get some of our jargons out of the way.

Amazon Web Services (AWS)

Amazon has been a sort of retail clearing house that allowed other retailers to list products on their site. Over the years, they created some **Application Programming** I**nterfaces** (**APIs**) to allow access to other retailers to list the products, relay shipping, and fulfillment information, so that customers of Amazon had a single frontend to glean all the information about their orders.

Sometime around late 2001, Amazon took the business decision to take some of the web services they were using in-house and make them available to the retailers, and to the general public for sale. Based on this ideology, **Amazon Web Services** (**AWS**) was born.

AWS is an umbrella term that encompasses all their electronic offerings related to storage, hosting, and content delivery over the web. Underneath the AWS banner, there are many other acronyms for the various products they offer:

- **Simple Storage Service** (**S3**): Think of this service as an elastic hard drive in the cloud. It is a place to store backups from the servers and the files that need to be made available to multiple locations safely and securely.
- **Elastic Computing Cloud** (**EC2**): Recall the time when a server was a huge rack of computers sitting in a closet in the office of your company somewhere? How many times have there been issues where someone goes into the closet and turns off the machine, or the air conditioner leaks and kills the server, or some other hard drive failure, and the server dies only to leave the website down and the information technology of your business stranded. Amazon has thousands of servers in server farms all over the world and you can log in, select an operating system, and boot a virtual server instance as you would boot any other server. We are going to do that in just a minute, but we will cover a few more acronyms before we begin.
- **Electronic Billing Solutions**（**EBS**) **storage**: When you create a virtual server instance, storage of it is virtual, as well as lives and dies with the server. When the server is terminated, the storage disappears into the cloud, as if it never existed. Sometimes, we need to make the storage available to those servers that do not disappear when the instance is terminated. For this, Amazon created EBS storage. It is built on their S3 storage product, but made available to the server instances easily, to allow the database backups and files to be copied from one server instance to another.
- **Virtual Private Cloud**: It allows server instances that are available only through a secure **Virtual Private Network** (**VPN**) connection. This service is for those applications that contain the data, and want to limit the interactions only to the members of their virtual network.

Amazon is only one of many virtual server providers. We are using them for this example but you could easily use Rackspace or Slicehost to create one or more small virtual servers for development. The tool we are going to use, that is RightScale, will work with many different virtual server providers. However, since most people these days have an Amazon account, I figured that it would be the "path of least resistance" to add the AWS service to your Amazon account for the purpose of this exercise, and you could do it without incurring a large signup fee which brings us to the big disclaimer.

You will be charged if you use Amazon to create a server as instructed in this example, as follows:

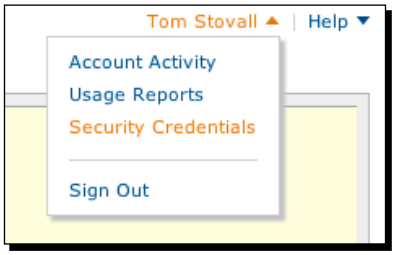
Amazon EC2 instances are all pay-per-use. You pay only for the time the server is running. Pricing of Amazon, based on the examples we create here, is about two dollars per day in the US. If you create the example, and immediately terminate all the servers, you should be charged less than five dollars for the exercise. If you do not want to spend the money to do the exercise, it is easy enough to follow without actually doing anything, or incurring any charges. I will make it clear as to the point from where you will be charged.

So if you are ready, let's set up your Amazon Web Services account. It is free. You will only be charged when you start launching the server instances. Ready? Here we go:

RightScale versus Puppet: RightScale is a commercial service that allows you to create, clone, and manage multiple virtual servers in several different commercial virtual server farms. RightScale is a feature with a limited free version, and I believe they are planning to phase-in a limitation on the number of servers that you can control as well. There is also an open source product that will do similar things. The software is called **Puppet** (`http://puppetlabs.com`). **Puppet Labs** is truly an open source software (OSS), that is, the software is free, and the company makes money from the consulting and training. I would not advise a deep dive into Puppet unless you have experienced some training from the company. Puppet is primarily for experienced server operations engineers. If you know people who are experienced in Puppet, you are more than welcome to consult about this exercise with them, and they can probably show you much better ways to do this using Puppet. RightScale was selected for this exercise because it is very easy to use. If you continue to use RightScale, you will eventually come up against a limitation of the free product, and you can make a decision at that point as to whether to get a staff member trained in Puppet, or spend the money on the commercial version of RightScale. Good luck with either.

Time for action – setting up AWS and RightScale

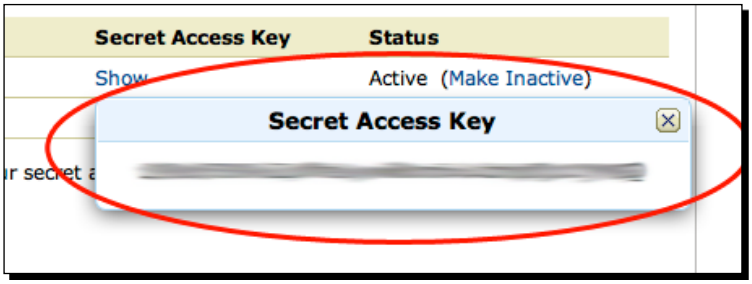
1. Navigate to `http://aws.amazon.com`, and in the top right-hand corner, click on the **Create an AWS Account** link. You will be asked to sign in to your Amazon account, and indicate a form of payment for any charges you incur. You would not be charged at this time. You will be charged when you launch the server instances, later in the chapter. Amazon will dial the phone number connected to your Amazon account, and ask for a **Personal Identification Number** (**PIN**) that will be displayed on the screen. After the account is completely set up, you will get an e-mail notification that it is ready to use:



2. Once your account is ready to use, select **Sign in to the management console**, and then in the top right-hand corner, click on the name on the account, and then select **Security Credentials**. There are three numbers on this page that you need. The first is your **Account Number**.

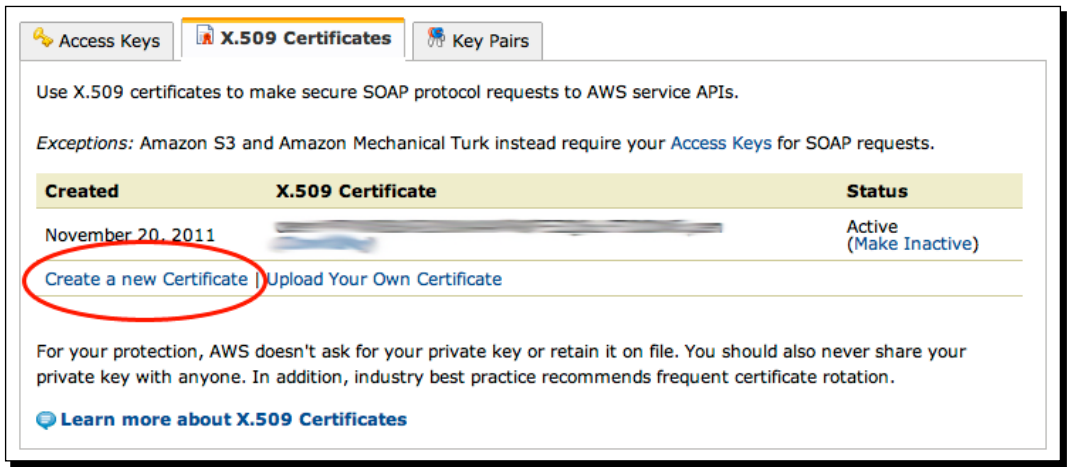
3. The second is your **Access Key ID**. It is on a tab named **Access Keys** close to the center of the page, as shown in the following screenshot:

4. The third is your **Secret Access Key**. In the area following the **Secret Access Key,** there is a link that simply says **Show**. Click on the link, and your access key will appear in a pop up. Copy these three numbers to some easily-accessible place:

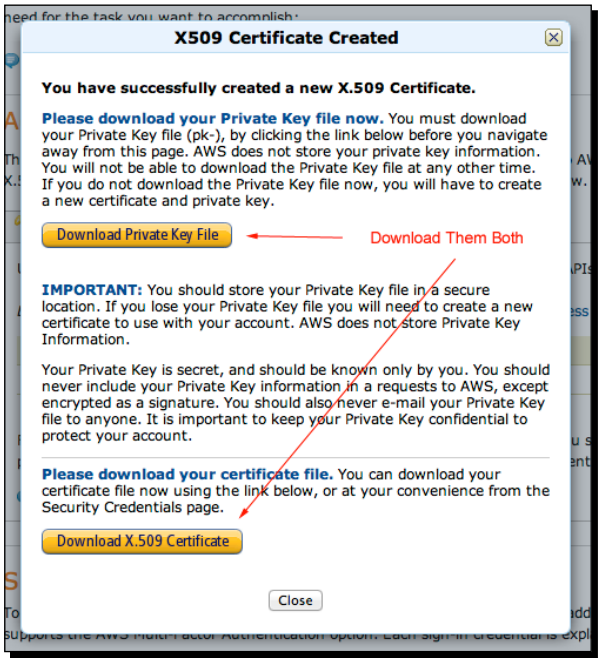


You will need them while signing up for RightScale.

5. The other thing you will need while signing up for RightScale is an **X.509 Certificate**:



6. The second tab on the tab group where you got the secret key is named as **X.509 Certificates**. When you click on **Create**, Amazon will give you two files to download, which will be the keys to get into your virtual machines, and to change the settings in your Amazon account. Treat these files like credit card numbers. Save them somewhere as "important":

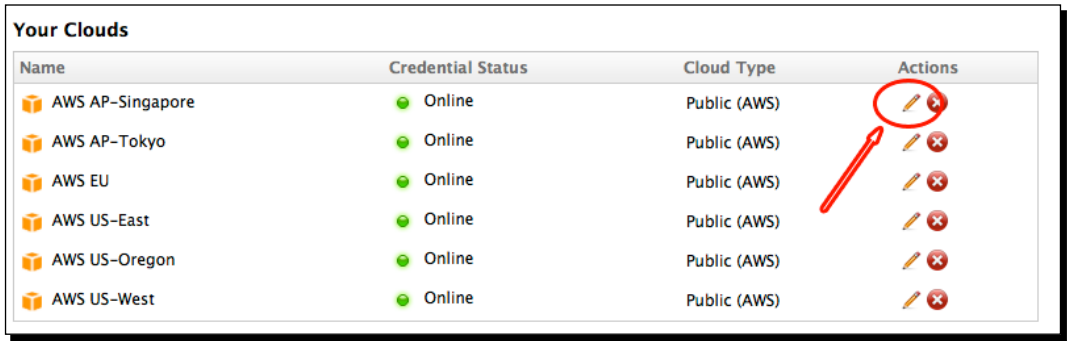


7. Once you have all the three numbers and both the certificates, navigate to `http://rightscale.com`, and choose **Create a Free Account** in the top right-hand corner. The account setup is straightforward. Once you have created an account and logged in, you are presented with a dashboard. On the front of the dashboard are two large buttons, one of which reads **Connect to Cloud**. Click on **Connect to Cloud,** and there will be a place to add an Amazon or Rackspace account. Click on the green plus beside AWS. In the first step, enter the Amazon Account Number, and the keys using the information you got from the security credentials in the first step, as shown in the following screenshot:

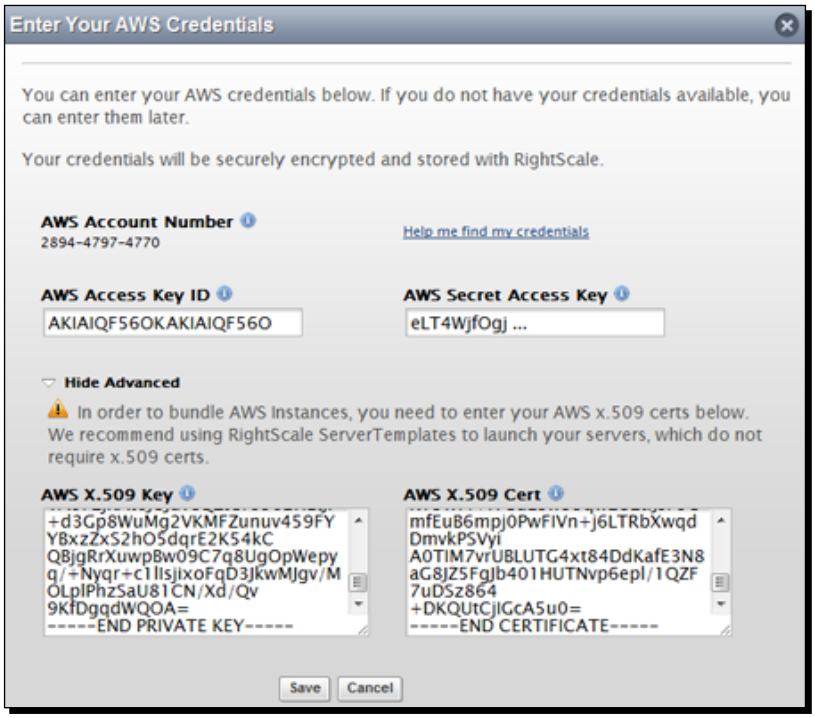
8. Next, Rightscale will create a new **Secure Shell Key** (**SSH**) pair, that is, **SSH Key Pair**. An SSH key pair consists of a private and a public key that allows you to access the servers created by RightScale. We will use this to log into the server later. Create one using a logical name. Click on **Continue**:

9. Next, create a security group for your servers. Just go with the default ports. Click on **Continue**:

10. In the resulting screen, you should see **Your Clouds** with all the the Amazon regions flashing green. Click on the pencil beside one of the regions, as shown in the following screenshot:



11. The next screenshot is the part where we upload the two X.509 certificates. Expand the **Show Advanced** area. Open each one of them individually with a text editor, and copy or paste the contents into the appropriate text area. Save the information, and you will be returned to the clouds page with all the green light besides the Amazon regions. Rightscale now has access to all your Amazon credentials, as shown in the following screenshot:



What just happened?

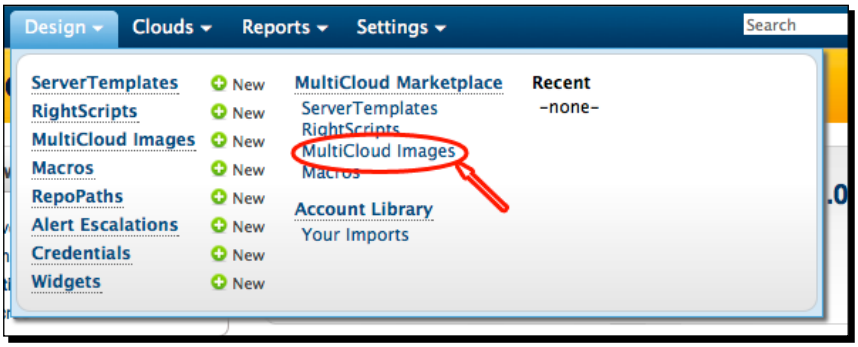
We set up an account at Amazon Web Services, and then at RightScale, and gave RightScale the access to our Amazon account with the ability to configure and launch server instances. We do that through a system of X.509 Keys.

We all know that when you use FTP, SFTP, or SSH to log into a server, you can do so with a username and password. But passwords have some very obvious problems. Firstly, people forget or lose them. Secondly, people pick those passwords that are easy to remember, eventually turning out easy for other people to guess or figure out. So when you manage more than ten servers, different environments, and different sites, the passwords start to get unmanageable.

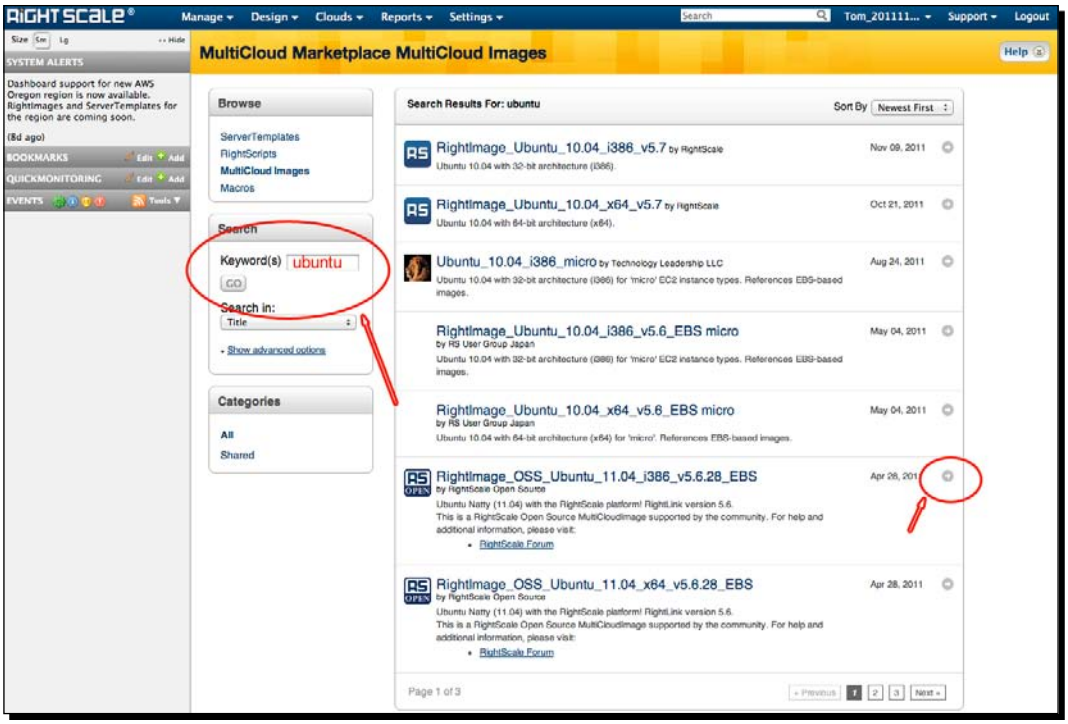
X.509 keys solve this problem by putting a key on your computer that will allow access to the servers and accounts. Amazon uses these keys to identify the users and to authorize the applications for users through their APIs. This key has two parts, a certificate and a key. Think of them as identifying the client and server. If you have both, you can become both the client and the server at your will. We gave RightScale both certificate and key so as to enable a two-way communication from our RightScale account.

Time for action – using an AMI to create a server

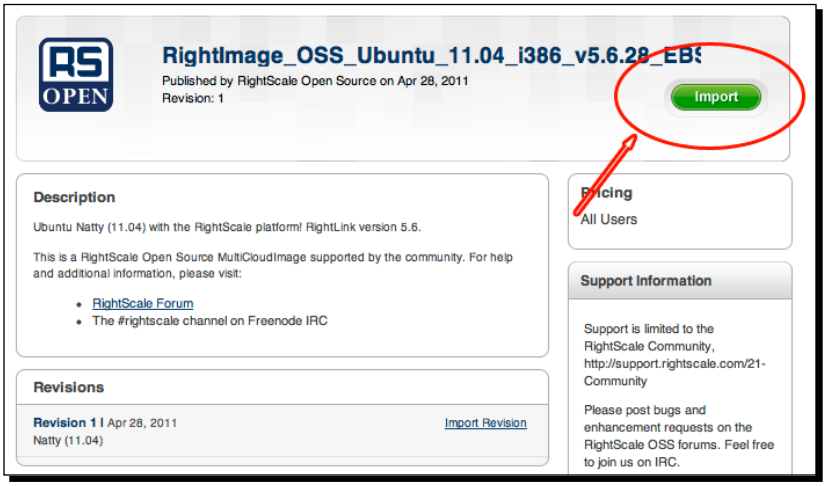
Now that we have set up our account, let's create some servers: from the top menu on the RightScale website, choose **Design I MultiCloud Marketplace | MultiCloud Images**, as shown in the following screenshot:



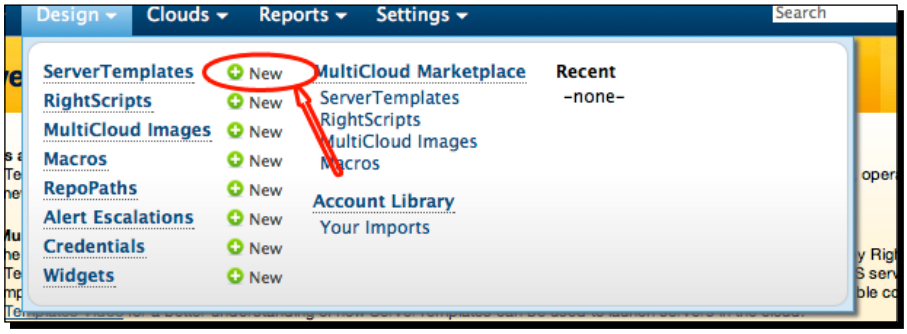
1. In the resulting page, search for **ubuntu**. There are two Ubuntu 11.X images, one for i386, and the other for x64. Select the 11.X for i386, as shown in the following screenshot:



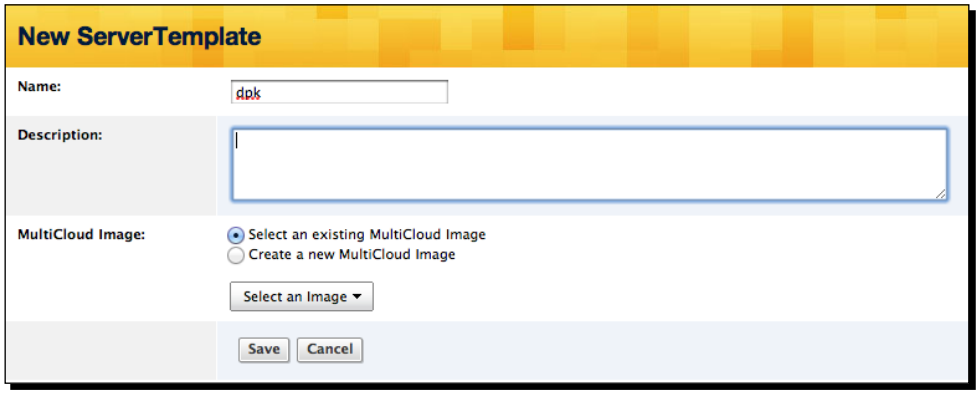
2. Click on **Import** to import the Ubuntu image into your usable templates, as follows:



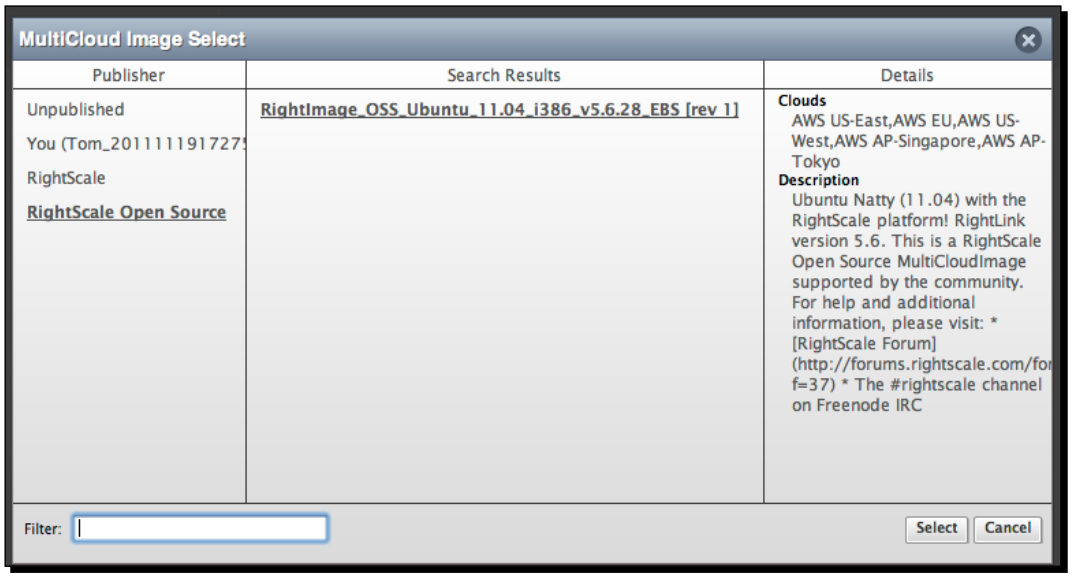
From the top menu on the RightScale website, choose **Design | ServerTemplates | New:**



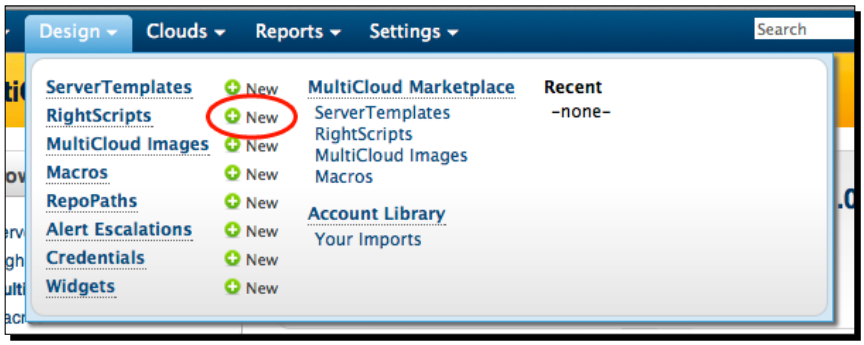
3. Name the new template **dpk**:



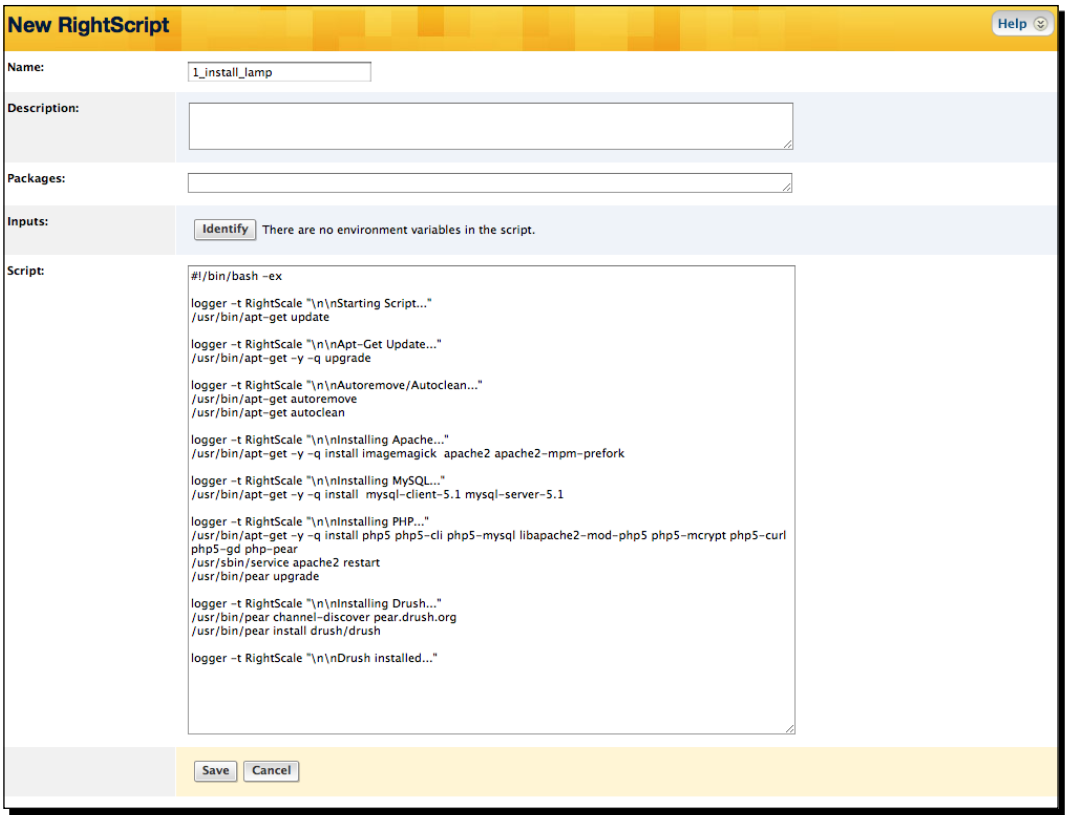
4. Click on **Select an Image** and select the image you just imported in the previous step. Once selected, click on **Save** to save the new template:



5. From the **Design** Menu, select **RightScripts | New**:

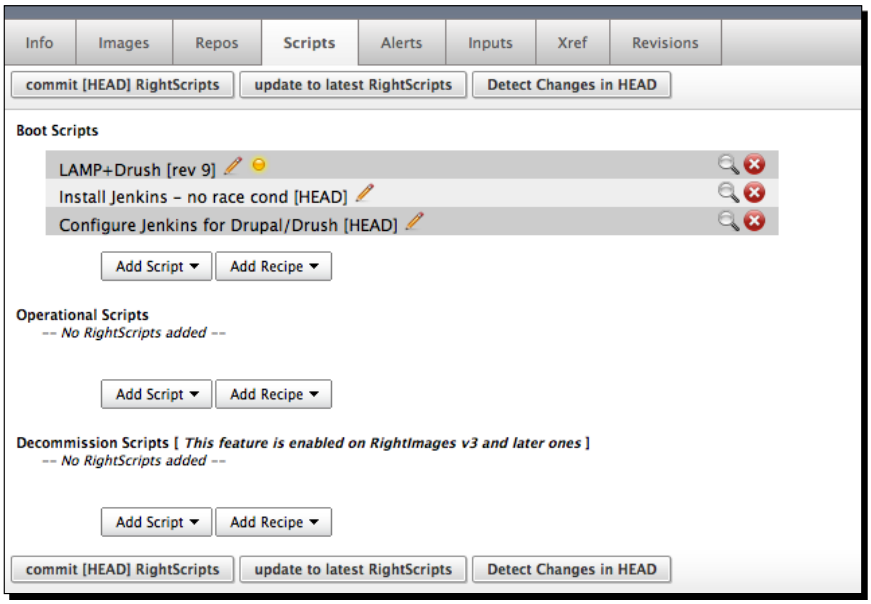


6. In your text editor, open the file at the following link: `sites/private/shell_scripts/1_install_lamp.sh` from the code base. Copy the contents of this file into the **Script** field of the **New RightScript**. Click on the **Identify** button to identify the custom variables used by the script, as shown in the following screenshot:

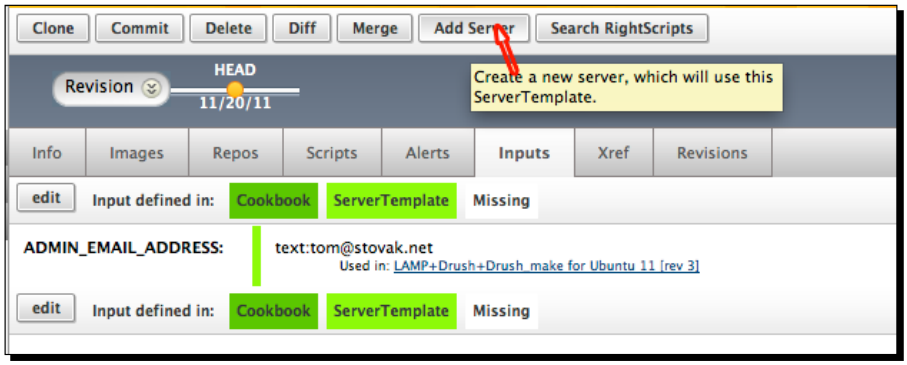


7. Create a new RightScript for each of the other two items in that folder, **2_install_jenkins** and **3_configure_install.sh**:

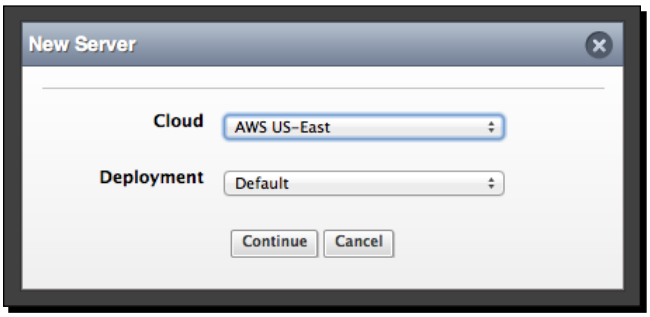
8. Select **Design** | **Server Templates**. In the resulting screen, click on your dpk server template to edit it. Click on the **Scripts** tab, and then click on **Add Script**. Find and select the three scripts you just created. Add them in the order of their numbers as 1, 2, and 3. They must run in this order for them to work, as shown in the following screenshot:



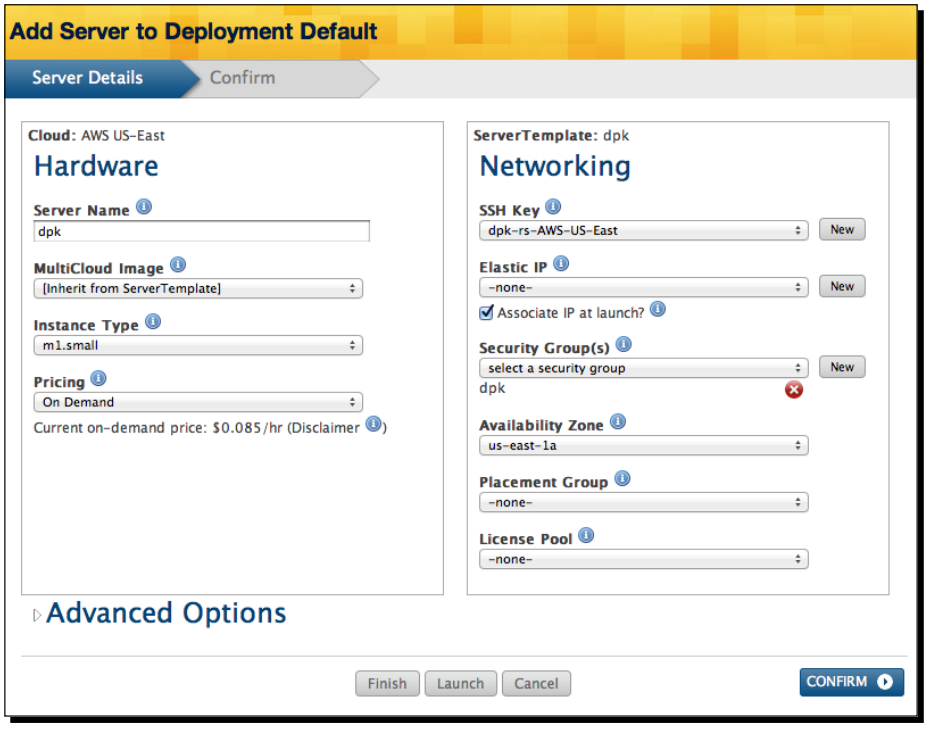
9. Click on **Add Server**, as shown in the following screenshot:



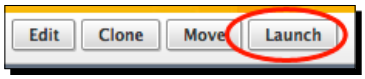
10. Select **AWS US-East** and the deployment group **Default**, as shown in the following screenshot:



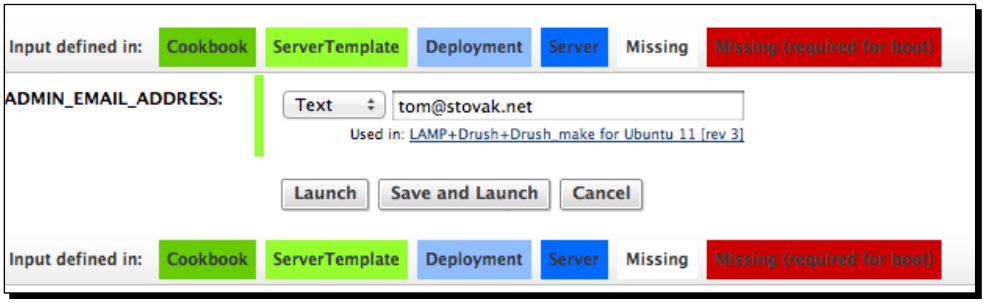
11. Confirm the details of your server instance. Create the server on one of the regions close to your location. The **SSH Key** and **Security Group(s)** will be the ones you created when you created your account on RightScale. If you do not have one, create one here. Click on **CONFIRM**. Then on the next screen, click on **Finish**:



12. At the top of the next screen, there is a **Launch** button. This is the point at which you begin being charged. From here onwards, charges will be incurred on your Amazon account. If you are willing to proceed, click on **Launch** to launch the instance:

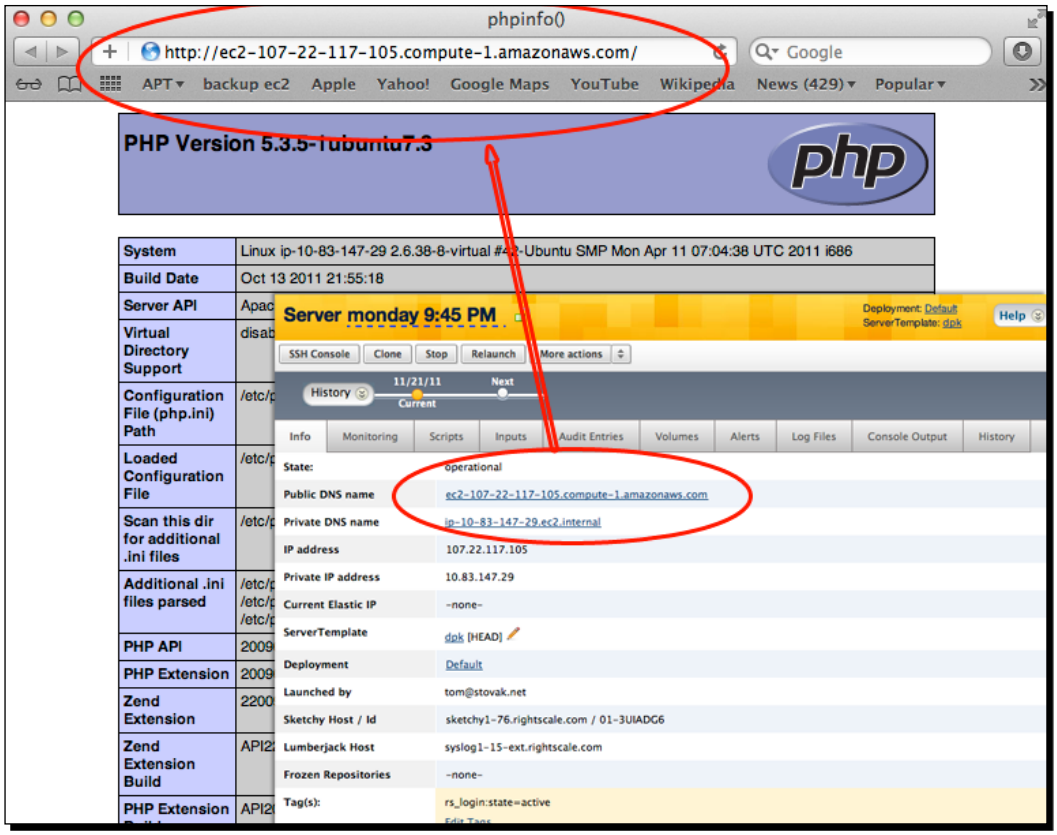


13. Confirm the e-mail address of your GitHub account, and then click on **Launch**:



14. For the next few minutes, RightScale will launch the server, and configure it according to the script we have created. Once the server is operational, you will get an e-mail from RightScale. This process could take up to an hour but usually takes about ten to fifteen minutes.

15. Once the instance is running, navigate to **Manage** | **Servers**. You should see the server you launched, and the state of this server should be **operational**. Click on the name of the server, and then you will select the **Info** panel of this server. The first value on the **Info** panel is the **Public DNS name** value. Copy this value, and paste it into a browser:



What just happened?

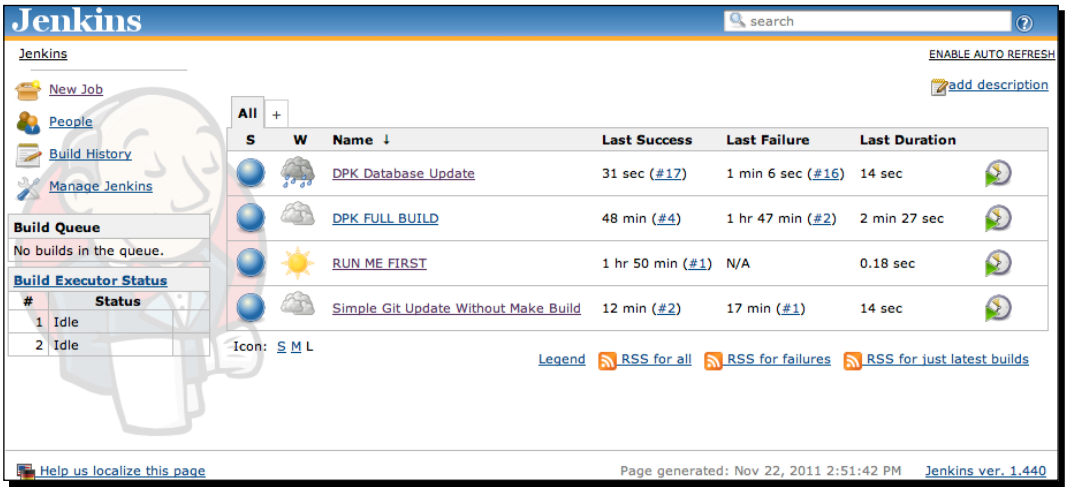
Once we had set up the account, we created a server template from an Ubuntu Linux image, and a series of three scripts that do the heavy lifting of installing all the relevant open source software. The first script installed the web server (called **Apache**), the database software, MySQL, and the PHP scripting language and Drush. The second script installed a software development product called Jenkins. Jenkins checked our code out of GitHub, copied it to the web server, and ran drush_make for us. The third script configured Jenkins, and added drush_make to its user.

Once we had defined the template, we launched our server instance. The scripts started working and we had a working web server. We can now launch multiple versions of this web instance with ease. All of them will be configured exactly the same as the other. It is a perfect way to develop sandbox websites so that you have a sandbox server, one for the **User Acceptance Testing** (**UAT**) and the other for the live site. Now let's actually populate the server with the project we have been working on. Jenkins makes it easy. Let's get started:

Time for action – Jenkins builds our site

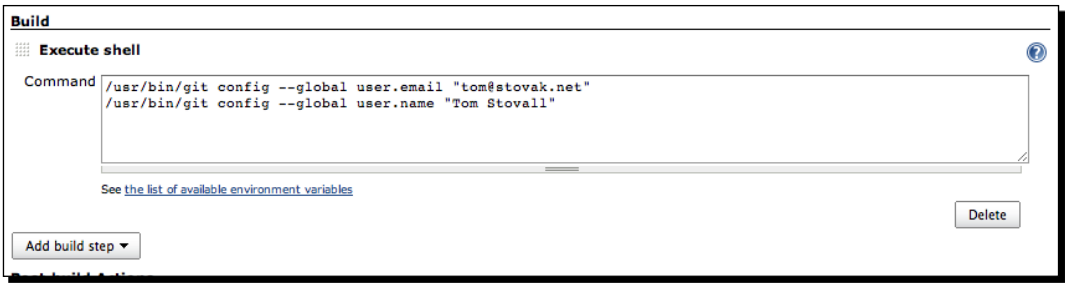
Now let's install Jenkins to handle our build process, as follows:

1. Use the public DNS value to navigate to the new server. Add or build to the URL. If the URL is `http://ec2-107-22-117-105.compute-1.amazonaws.com/`, it becomes `http://ec2-107-22-117-105.compute-1.amazonaws.com/build/`, and the page will look something like this:



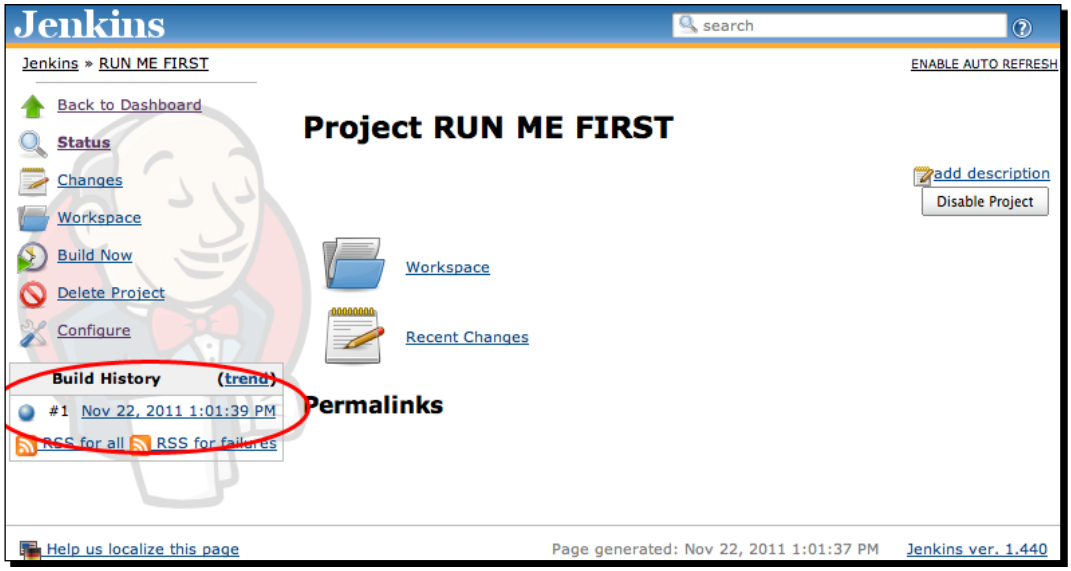
2. There is a job called **RUN ME FIRST** that sets the defaults for the other jobs to run correctly. Click on the name, then in the job actions menu on the left-hand side of the screen, click on **Configure**. About halfway down the page is a **Build** step followed by the commands. Change the address and username values of your GitHub e-mail address and your name, as follows:

```
git config --global user.email "YOUR_EMAIL@GOES.HERE"
git config --global user.name "Your Name"
```



3. Scroll to the bottom of the page and save the job. In the job actions menu on the left-hand side of the screen, click on **Build Now**.

4. Once you are done with the build, there will be a new build in the **Build History**. If the build is successful, the dot beside the build will be blue. If the build is unsuccessful, the build will be red:

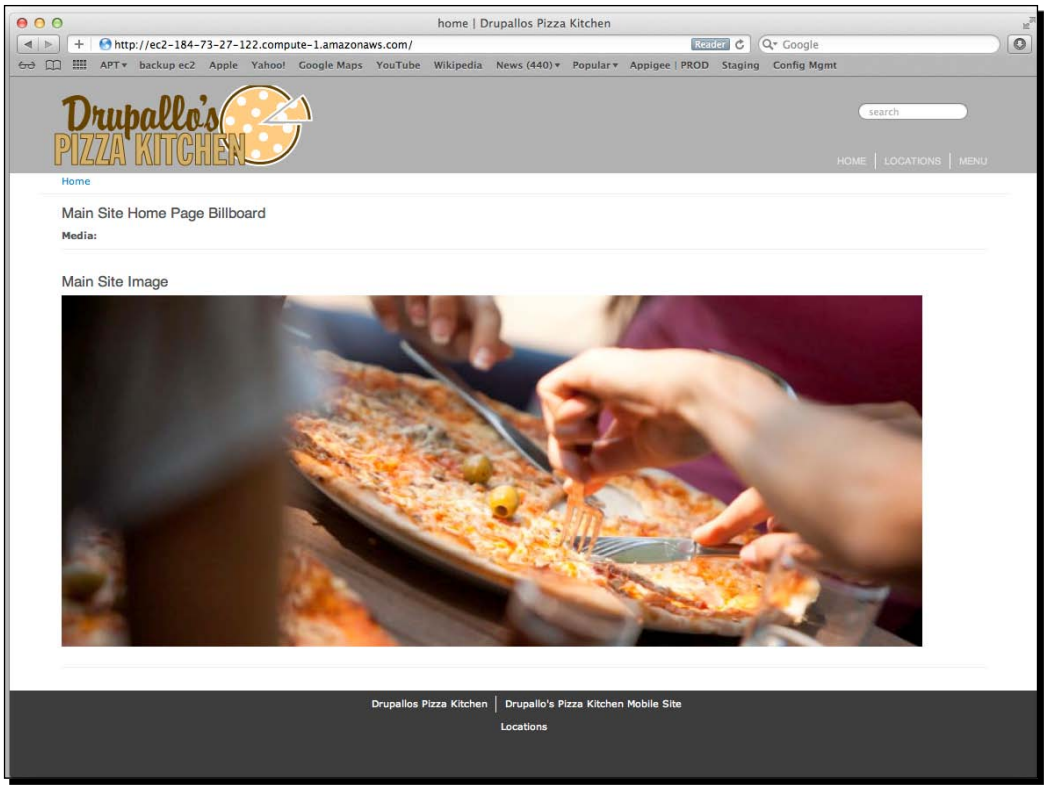


5. In the job list, there is a job called **DPK FULL BUILD**. Click on the name of the build, and then click on **Build Now**. As the building starts up, click on the **Build History** link of this build. In the resulting page, click on **Console Output**. You can see the build as it happens in the console output link:

6. After the build process is completed, go back to the main job list, and click on the **DPK Database Update** job.

7. Click on **Build Now**. The database job will ask you for a URL of the database backup. Use the default value. I have set the default value to be a database backup I know to be in the file tree. Then click on **Build**.

8. Navigate to the public hostname of the server, and you should be able to see the clone of a full DPK site:



What just happened?

When the RightScale scripts built your server, I defined several jobs for Jenkins that were included in the build. The first job we ran, RUN ME FIRST, had set the username for Jenkins to be able to use GIT, and check out projects from the Version Control. The second job was basically an automation of what we did in *Chapter 3*, *Selecting the Right Domain for your Mobile Site*. We checked out the code, and did a drush_make build on the MAKE file in the root directory. The third job was the bane of the existence of everyone, that is, database restoration and backups. Newer developers seem to have a very difficult time with the database connections, and the database server you are connecting to, making the connection correct. This takes all of the guesswork out of it. It allows you to supply a database dump out of the `backup_migrate` backups folder, and restore it. There is no mess and no fuss.

There is one fair warning. Jenkins is a build server for development. It should not be installed on a production server. You can set up a Jenkins job to resync your development server to a remote production server. I have provided a sample one with this chapter in the Jenkins job list. Instructions on how to set it up are in the description of the job. Your production server should be set up, and secured, by an experienced operation server admin, as it is beyond the scope of this book to do this.

Pop quiz

1. What are the features of Cloud Servers like Amazon EC2 and Rackspace?
    a. They allow you to have your own server without owning any hardware
    b. They are primarily "pay per use"
    c. They can be created on demand
    d. They can be all of the above
    e. They are none of the above
2. RightScale is a product that can create, clone, and manage your Amazon and Rackspace virtual server instances:
    a. True
    b. False
3. RightScale uses shell scripts to set up servers, and install software after launch:
    a. True
    b. False
4. What is Jenkins?
    a. It is an open source software project
    b. It is a server that you can use to "build" projects
    c. It is a direct interface with the core kernel of the server
    d. It can be both 1 and 2
    e. It are none of the above

Summary

In the final chapter of our journey together, we set up a workflow that allows the developer to focus on development, and lets the development server create builds from the source control.

We learned how hosting on the virtual servers can give us the power of having our own server at a cost that is very affordable to the average client.

We used RightScale to quickly create a virtual server instance from a server template, and had a working version of our site in a few minutes.

We used Jenkins to grab our changes from source control, and deploy them onto the server quickly, as well as restore database backups from the Drupal `backup_migrate` manual backups directory.

Setting up these tools to do the work for us frees us to work locally, and push the updates out as they are tested and stable.

Pop Quiz Answers

Chapter 2: Setting up a Local Development Environment

1	2	3	4	5
c	d	d	a	b
6	7	8	9	10
d	d	a	b	d

Chapter 3: Selecting the Right Domain for your Mobile Site

1	2	3	4	5	6
d	a	a	d	c	b

Chapter 4: Introduction to a Theme

1	2	3	4
c	d	a	e
5	6	7	8
e	b	c	a

Chapter 5: A Home with a View

1	2	3	4	5
b	e	d	e	a

Chapter 6: The Elephant in the Room: Audio, Video, and Flash Media

1	2	3	4
d	a	a	c

Chapter 7: Location, Location, Location

1	2	3	4	5	6
b	b	a	b	c	c

Chapter 8: Services with a Smile

1	2	3	4	5
a	b	d	d	b

Chapter 9: Putting it Together

1	2	3	4	5	6
e	c	b	e	a	c

Chapter 10: Tabula Rasa: Nurturing your Site for Tablets

1	2	3	4	5	6	7	8
c	a	d	c	c	d	d	d

Chapter 11: A Home in the Clouds

1	2	3	4
e	a	a	d

Index

Symbols

$cookie_domain variable 110
$.mobile.showPageLoadingMsg() method 221
$page_bottom variable 91
$page_top variable 91
<audio> tag 151
** tags 87**
<header> tag 91
<html> tag 255
<meta> tag 93, 273
<video> tag 151
.info file 73
.make file 82
.mobi domain 63
.mobi site 63
.mobi top-level domain 63
.module file 72
.psd file 60
_preprocess_search_block_form hook 100
5.2.x
versus 5.3.x 37

A

add command 139
AddEventListener function 259
AJAX 200
Amazon 282
Amazon Web Services. *See* **AWS**
AMI
used, to create server 289-296
Android Virtual Device. *See* **AVD**
Apache 39, 297
API 201
Application Programming Interface. *See* **API**
Asynchronous JavaScript and XML. *See* **AJAX**
attach function 259
attach method 183
AVD 15
AWS
about 281
setting up 283

B

Backup and Migrate module
about 81
used, to create local backup 80
Balsamiq
about 262
mockup diagram 262, 263
billboard 118
boostrap.php 67
bootstrapping, domain access module
about 67-69
boostrap.php 67
E editor icon 67
browscap redirect 102
browsers 159
Build History 299
buildmode 228
buildmode-full container 227
built-in blocks system
issues 116

C

carat quote 214
CCK 130
changedTouches value 260
charts
about 152
Highcharts 152
Chrome browser 243
close2u_find function 186
close2u module
about 177
downloading 178-190
enabling 178-190
Commerce module, for Drupal 7
about 263
one true theme 264
PayPal payment option, adding 272
product, creating 268-272
product type, creating 264
conditional statements 91
Connect to Cloud 286
Console Output 300
Content Construction Kit. *See* **CCK**
context module
about 116
context 123
context layouts 123
context UI 123
cookies 110
core menu items 240
Create a free Account 286
Create an AWS Account link 283
CRON task 101
ctools 73
customized services
about 209
custom REST service formatter 209-212
Cygwin
about 32
development environment, installing 28-32

D

D7, versus D6 theming
default mobile theme, installing 93, 94
data-* attributes 212
database
local backup, creating 80
desktop site 234
detach function 259
development environment installation
Cygwin, using 28-32
Windows, using 28-32
Display Suite (DS)
about 224
build mode 224
buildmode-full container 227
fonts 242, 243
fonts, adding 244-248
footer node 229
header node 229
hooks 224
left node 229
middle node 229
right node 229
Teaser With Image mode 226
working 224
display suite module 116, 135
display suite settings mode 120
distribution (distro) 24
DOCTYPE 87
Document Object Model. *See* **DOM**
DOM 106, 255
Document Type Definition. *See* **DOCTYPE**
domain access module
about 63
bootstrapping 67-69
commands 63
domain module 64
domain views module 64
installing, for Mac 64
installing, for Windows 65, 67
installing, steps 64
DOM event 255
double quote 214
DPK Database Update job 300
DPK FULL BUILD 300
DPK html.tpl.php file 89
DPK Mobile 233
dpk_mobile.info file 97
dpk_mobile_preprocess_page function 99

DPK website
about 60
URL 60
Drupallo's Pizza Kitchen (DPK) 22
Drupal
about 8
new REST service, testing 203-207
REST service, creating 201-203
using 200
Drupal 6
Embedded Media 143
Drupal 7 theming
versus Drupal 6 theming 89
Drupal 7
html.tpl.php 89
node.tpl.php 90
page.tpl.php 89
Drupal behavior object 259
Drupal behaviors
about 107, 108
redirection, with cookie to remember state 108-110
Drupal Commerce module. *See* **also Commerce module, for Drupal 7**
Drupal Commerce module
configuring 264-267
downloading 264
installing 264
drupallos_homepage module 81
drupal_render 235
Drupal-style variables 128, 129
Drupal website
building, Drush Make website used 47-49
building, from make file 50, 51
Drush
about 24, 45
installing, for Mac OS X 45
installing, for Windows 46
Drush Make
about 24, 45
installing, for Mac OS X 45
using, for Drupal website building 47-49
drush_make command 47
drush make dpk.make command 82
dumb phones 8, 9

E

EBS 281
EC2 281
E editor icon 67
Elastic Computing Cloud. *See* **EC2**
Electronic Billing Solutions. *See* **EBS**
Embedded Media 143
E-TextEditor 33
evt.touches 260
eXtensible Markup Language. *See* **XML**

F

features[ctools] property 128
features.inc file 73
features module
.info file 73
.module file 72
about 70
add command 139
changes, bundling into package 136-139
creating, steps 70-72
ctools 73
features-based app stores 77
features.inc file 73
hooks, adding 77
jQuery Cycle Plugin 72
Strongarm 70
updating 81
updating, with new settings 76
views_default.inc file 74
XML Update URL 77
features-update command 127
field-items container DIV 227
File Transfer Protocol. *See* **FTP**
Firefox browser 243
Flash
iOS 141, 142
fonts
adding 244-248
Fonts.com Web fonts
URL 243
Font Your Face 243
fragmentation 16
FTP 280

G

Geocode Update 175
geocoding
about 175
node's location data 176
geolocation 160
gesture events
gestureend 257
gesturemove 257
gesturestart 257
GIT 24 81
global.js file
creating 102
graph
creating 153-156
graphic formats
SVG 153
VML 153

H

handhelds
designing issues 60, 61
handleSwipe method 260
HandleTouchEnd 260
handleTouchMove 260
HandleTouchStart event 260
handleTouchStart method 259
Highcharts
homepage 152
home page
downloaded feature 127
features[ctools] property 128
features-update command 127
hook_context_default_contexts 129
mobile friendly home page, creating 117-124
original feature 127
updated features, downloading 125-127
home Page context 126
hook_context_default_contexts 129
hook_menu_attribute_info 242
hook_preprocess_page 224
hooks
adding, to features module 77
HTML 255
HTML5
about 87
URL 87
HTML5 <footer> element 93
HTML5 <header> element 91
HTML5 semantic elements 89
HTML5 Shim
URL 89
HTML document 255
html_head_altar hook 100
html.tpl.php 89, 90, 230, 273

I

ICANN 63
Identify button 292
ImageMagick 224
Import button 53
Info panel 296
installation
WAMP 41- 44
interactive web 208
Internet Committee for assigned Names and Numbers. *See* **ICANN**
Internet Explorer browser 243
Internet Explorer (IE) 86
iOS
about 16
Flash 141, 142
Mac OS developer's package, installing 17, 19
iOS Simulator 18
isTouchDevice function 259

J

Java Development Kit. *See* **JDK**
JavaScript Object Notation. *See* **JSON**
JavaScript redirection
writing, for theme 102-104
JDK 14
Jenkins
Build History 299
Build Now 299
Console Output 300
DPK Database Update job 300
DPK FULL BUILD 300
installing, to handle build process 298
RUN ME FIRST job 298

jqm 212
jQM
about 212, 213
using, as base theme 213-219
jQuery Cycle Plugin 72
jQuery Mobile. *See* **jQM**
jQuery Mobile JavaScript Events
about 219
AJAX login form 219-221
jQueryMobileSlideShowTouchAdvance 259
jQuery mobile theme
nodes, retheming 230-234
jQuery onDocumentReady script 105
JSON 200
JSONP 200
JSON with Padding. *See* **JSONP**

K

KHTML 12

L

LAMP 33, 34
Launch button 296
Layout for content type in default tab 132
Linux, Apache, MySQL, and PHP/Perl. *See* **LAMP**
local database
about 51
creating 51-55
locationFail function 191

M

Mac
domain access module, installing 64
Macintosh, Apache, MySQL, and PHP. *See* **MAMP**
Mac OS X
text editors 33
main event
about 257
swipe advance, adding to home page 257-259
make file
Drupal install, building 50, 51
updating 78, 80
MAMP
about 34
default configuration, changing 39-41
local web server, configuring 35-39
media queries, for tablets 274, 275
media query
about 96, 97
mobile theme, theming 97-100
using 96
menu attributes
about 240, 242
customizing 240, 242
menu_attributes_info 242
menu_group 235
menu item content type 131
menu item group content type
fields 133
menu item group type 135
menu module
about 129
content importing, via command line 133
content importing, via Drupal GUI 134
Layout for content type in default tab 132
menu content types, creating 130-132
menu item content type 131
menu item group content type, fields 133
menu item group type 135
node reference 135
Microsoft Windows
Cygwin 28
GIT 28
text editors 33
mobile
about 13
Android development package, installing 14
mobile-friendly home page, creating 117-124
stimulators, setting up 14-16
mobile clients
JavaScript redirection, writing for theme 102-104
redirecting 101, 102
Mobile-ize me 12
mobile mode 124
mobile stimulators
Android development package, installing 14-16

mobile theme
about 95
personalizing 97-100
mobile theme project 98
mobile_tools module 102
Mobile Webkit browser 243
mobile website 59
module_invoke 242
multisite installs 63

N

navigator.geolocation object
about 160
geocoding 175
location data, adding to nodes 161-174
nd-one-sidebar 227
nd-region-middle-wrapper 227
nd-sidebar-right 227
node_load 235
node reference 135
nodes
retheming, for jQuery mobile theme 230-234
node's location data
geocoding 176, 177
node.tpl.php 90
node_view 235

O

onDocumentReady process 105
One Design myth 13
open source software (OSS) 282
OpenType 243

P

page finishing
steps 190-197
page.tpl.php 89, 91, 230
Performance.gov
about 224
demo 225
Featured Story field points 226
Personal Identification Number. *See* **PIN**
Photoshop psd file 87
PIN 283
platforms
about 24, 25
SCM client for Mac, downloading 25-27
POSTER 199
product reference field 264
progressive enhancement
about 86
DOCTYPE 87
HTML5 87
HTML5 semantic elements 89
Public DNS name value 296
puppet
about 282
versus RightScale 282
puppet labs 282

Q

Quirks Mode 88

R

radians 187
RDF 88
redirection flow 105
render all regions plugin 230
rendered node
theming, adding 236-239
Representational State Transfer. *See* **REST**
request 101
Resource Description Framework. *See* **RDF**
responsive web design
about 261
starting over 261
REST 200
RightScale
Secure Shell Key (SSH) 286
setting up 285, 286
Show Advanced area 288
versus puppet 282
rsync command 81

S

S3 281
Safari browser 243
Sandbox
project, installing 175

saveOrigin function 192
Scalable Vector Graphics. *See* **SVG**
Scripts tab 294
SDK 15
Secret Access Key 284
Secure Shell Key. *See* **SSH**
Security Credentials 283
Security Group(s) 295
Select an Image 292
semantic web 88
server
  Add server 294
  creating, AMI used 289-296
server alias 64
Show Advanced area 288
Sign in to the management console 283
Simple Storage Service. *See* **S3**
single quote 214
slideshow_cycle div elements 259
Smart-er phones 9
smart phones 10, 11
Software Development Kit. *See* **SDK**
Source Code Management. *See* **SCM**
SSH 286
SSH Key 295
SSH Key Pair 286
Strongarm 70
Style output level 232
Subversion (SVN) 23
SVG 153
swipe event 259
swipe gesture
  adding 260

T

tablet
  about 11
  designing issues 60, 61
  event-driven model 254
  main event 257
  touch events 256
  touchscreen interfaces 254
  wireframing 262
teaser_with_image buildmode 228
text editors
  for Mac OS X 33
  for Microsoft Windows 33
TextMate 33
theming
  adding, to rendered node 236-240
touchend event 260
touch events
  about 256
  touchend 257
  touchmove 257
  touchstart 257
touchmove event 260
Typekit
  about 243
  URL 243

U

UAT
  hostnames, adding to domain access 82
  instance, creating 80
  server 82
ubuntu 290
User Acceptance Testing environment. *See* **UAT**
userEnterLocationHandlerfunction 191

V

Vector Markup Language. *See* **VML**
videos, incorporating into web content
  about 142, 143
  content, adding 147-150
  encoding process 150-152
  media files, embedding 143-147
viewport
  about 273
  media queries, for tablets 274, 275
  setting, with JavaScript 273
views_default.inc file 74
views-view-list--menu--page.tpl.php 231
Virtual Private Cloud 281
Virtual Private Network. *See* **VPN**
VML 152, 153
VPN 281

W

W3C RDF Primer document
URL 89
WAMP
about 34, 41
installing 41-44
WAP 8
web content
videos, incorporating 142, 143
web embedding 243
WebKit 12
WebM 151
website
issues 280
Windows
domain access module, installing 65, 67
Windows, Apache, MySQL, and PHP. *See* **WAMP**
Wireless Application Protocol. *See* **WAP**

X

X.509 Certificate 285
X.509 key 289
XHTML 255
XML 200, 255
XML-based Remote Procedure Call. *See* **XML-RPC**
XML-RPC 201
XML Update URL 77

Y

Your Clouds 287

Thank you for buying
Drupal 7 Mobile Web Development Beginner's Guide

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

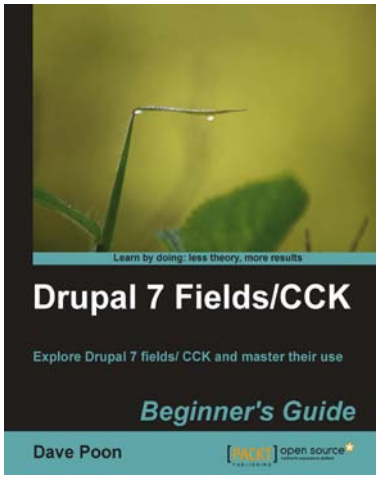
In 2010, Packt launched two new brands, Packt Open Source and Packt Enterprise, in order to continue its focus on specialization. This book is part of the Packt Open Source brand, home to books published on software built around Open Source licences, and offering information to anybody from advanced 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.

[PACKT] open source
community experience distilled
PUBLISHING



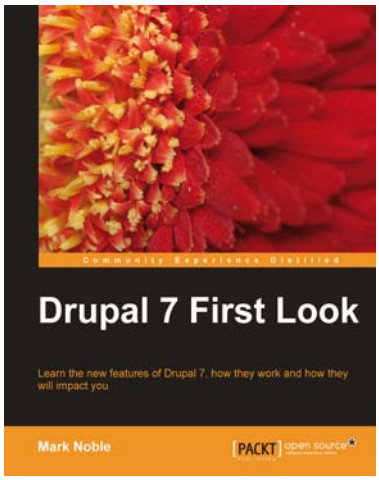
Drupal 7 Fields/CCK Beginner's Guide

ISBN: 978-1-84951-478-1 Paperback: 288 pages

Explore Drupal 7 fields/CCK and master their use

1. Step-by-step guide to building your own Drupal 7 website using the Drupal 7 fields system
2. Specifically written for Drupal 7 development and site building
3. In-depth coverage of theming fields in Drupal 7
4. Discover the new fields system from the database perspective

---



Drupal 7 First Look

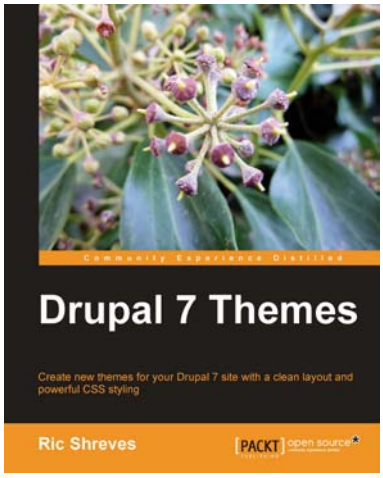
ISBN: 978-1-84951-122-3 Paperback: 288 pages

Learn the new features of Drupal 7, how they work and how they will impact you

1. Get to grips with all of the new features in Drupal 7
2. Upgrade your Drupal 6 site, themes, and modules to Drupal 7
3. Explore the new Drupal 7 administration interface and map your Drupal 6 administration interface to the new Drupal 7 structure
4. Complete coverage of the DBTNG database layer with usage examples and all API changes for both Themes and Modules

Please check **www.PacktPub.com** for information on our titles

[PACKT] open source
community experience distilled
PUBLISHING

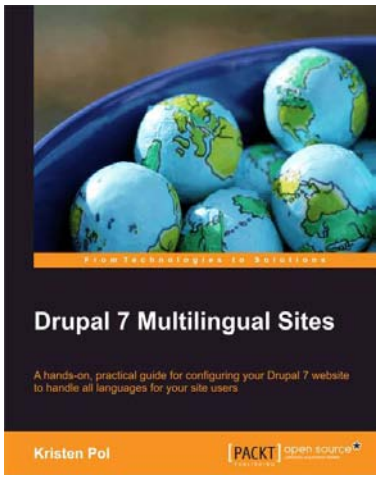


Drupal 7 Themes

ISBN: 978-1-84951-276-3 Paperback: 320 pages

Create new themes for your Drupal 7 site with a clean layout and powerful CSS styling

1. Learn to create new Drupal 7 themes
2. No experience of Drupal theming required
3. Discover techniques and tools for creating and modifying themes
4. The first book to guide you through the new elements and themes available in Drupal 7



Drupal 7 Multilingual Sites

ISBN: 978-1-84951-818-5 Paperback: 100 pages

A hands-on, practical guide for configuring your Drupal 7 website to handle all languages for your site users

1. Prepare your Drupal site to handle content in different languages easily
2. Apply the numerous multilingual modules to your Drupal site and configure it for any number of different languages
3. Organize the multilingual pieces into logical areas for easier handling