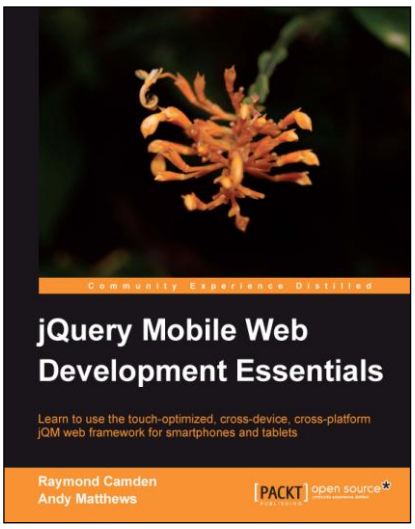
[PACKT] open source
community experience distilled
PUBLISHING

jQuery Mobile Web Development Essentials

Raymond Camden
Andy Matthews



Chapter No. 7
"Creating Modal Dialogs, Grids, and Collapsible Blocks"

In this package, you will find:

A Biography of the authors of the book

A preview chapter from the book, Chapter NO.7 "Creating Modal Dialogs, Grids, and Collapsible Blocks"

A synopsis of the book's content

Information on where to buy this book

About the Authors

Raymond Camden is a Developer Evangelist for Adobe focusing on web standards and mobile development. He is a contributing author to numerous technical books including the best selling *ColdFusion Web Application Construction Kit*, published by *Adobe Press*. He has spoken at conferences around the world and maintains many popular ColdFusion community websites. He is the manager of `www.RIAForge.org`, `www.CFLib.org`, and writes at his blog `www.raymondcamden.com`. Raymond is happily married and a proud father to three kids and is somewhat of a Star Wars nut.

I'd like to thank everyone on the jQuery and jQuery Mobile teams for making tools that have changed my life. Without your hard work and dedication, the web would be less awesome. Thank you Andy, for coming on board and helping to make this book better.

Andy Matthews has been working as a web and application developer for 13 years, with an experience in a wide range of industries, and has a skill set which includes graphic design, programming, business strategy and planning, and marketing. Throughout his career he has been privileged to work on projects which interfaced with industry giants such as Craigslist, written code that allowed Enterprise level sales teams to quickly and efficiently build presentations for their clients. He stays up-to-date with current trends in the marketplace by helping previous employers transition to newer, more effective, coding habits and standards. He is a frequent speaker at conferences around the country. He has also developed software for the open source community, and he currently works for a social networking startup `Goba.mobi` in Nashville, TN.

I'd like to thank my wife Jaime, and my children Noelle, Evan, and Mason for their patience and grace in letting me pursue my passion. Most of all, thank you God for giving me the desire to learn, the ability to pick things up quickly, and the perseverance to apply the knowledge I've gained throughout the years.

jQuery Mobile Web Development Essentials

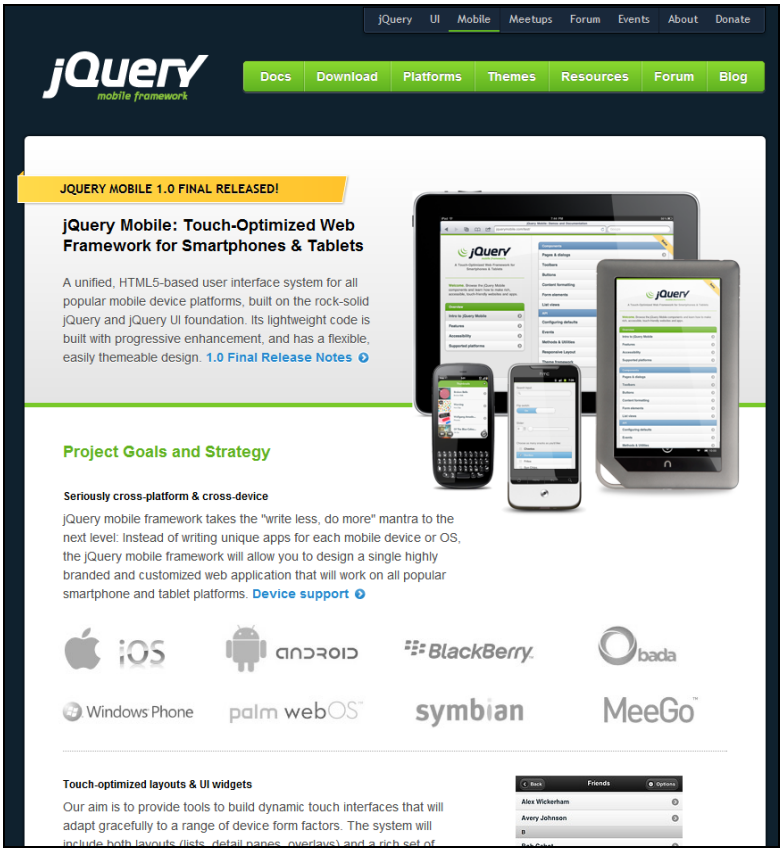
What is jQuery Mobile?

On August 11, 2010, nearly two years ago, John Resig (creator of jQuery) announced the jQuery Mobile project. While focused on the UI framework, it was also a recognition of jQuery itself as a tool for mobile sites and that work would be done to the core framework itself, to make it work better on devices. Release after release, the jQuery Mobile project evolved into a powerful framework encompassing more platforms, more features, and better performance with every update.

But what do we mean when we say a *UI framework?* What does it mean for developers and designers? jQuery Mobile provides a way to turn regular HTML (and CSS) into mobile friendly sites. As you will see soon in the first chapter, you can take a regular HTML page, add in the required bits for jQuery Mobile (essentially five lines of HTML), and find your page transformed into a mobile-friendly version instantly.

Unlike other frameworks, jQuery Mobile is focused on HTML. In fact, for a framework tied to jQuery, you can do a heck of a lot of work without writing one line of JavaScript. It's a powerful, practical way of creating mobile websites that any existing HTML developer can pick up and adapt within a few hours. Compare this to other frameworks, such as Sencha Touch. Sencha Touch is also a powerful framework, but its approach is radically different, using JavaScript to help define and layout pages. jQuery Mobile is much friendlier to people who are more familiar with HTML, as opposed to JavaScript. jQuery Mobile is *touch friendly*, which will make sense to anyone who has used a smart phone and struggled to click the exact right spot on a website with tiny text and hard to spot links. It will make sense to anyone who accidentally clicked on a Reset button instead of Submit. jQuery Mobile will enhance your content to help solve these issues. Regular buttons become large, fat, and easy to hit buttons. Links can be turned into list based navigation systems. Content can be split into virtual pages with smooth transitions. You will be surprised just how jQuery Mobile works without writing much code at all.

jQuery Mobile has some very big sponsors. They include Nokia, Blackberry, Adobe, and other large companies. These companies have put in money, hardware, and developer resources to help ensure the success of the project:



What's the cost?

Ah, the million dollar question. Luckily this one is easy to answer: nothing. jQuery Mobile, like jQuery itself, is completely free to use for any purpose. Not only that, it's completely open source. Don't like how something works? You can change it. Want something not supported by the framework? You can add it. To be fair, digging deep into the code base is probably something most folks will not be comfortable doing. However, the fact that you can if you need to, and the fact that other people can, leads to a product that will be open to development by the community at large.

What do you need to know?

Finally, along with not paying a dime to get, and work with, jQuery Mobile, the best thing is that you probably already have all the skills necessary to work with the framework. As you will see in the upcoming chapters, jQuery Mobile is an HTML based framework. If you know HTML, even just simple HTML, you can use the jQuery Mobile framework. Knowledge of CSS and JavaScript is a plus, but not entirely required. (While jQuery Mobile uses a lot of CSS and JavaScript behind the scenes, you don't actually have to write any of this yourself!)

What about native apps?

jQuery Mobile does not create native applications. You'll see later in the book how you can combine jQuery Mobile with *wrapper* technologies such as PhoneGap to create native apps but, in general, jQuery Mobile is for building websites. The question on whether to develop a website or a mobile app is not something this book can answer. You need to look at your business needs and see what will satisfy them. Because we are not building mobile apps themselves, you do not have to worry about setting up any accounts with Google or Apple, or paying any fees for the marketplace. Any user with a mobile device that includes a browser will be able to view your mobile-optimized sites.

Again – if you want to develop true mobile apps with jQuery Mobile, it's definitely an option.

Help!

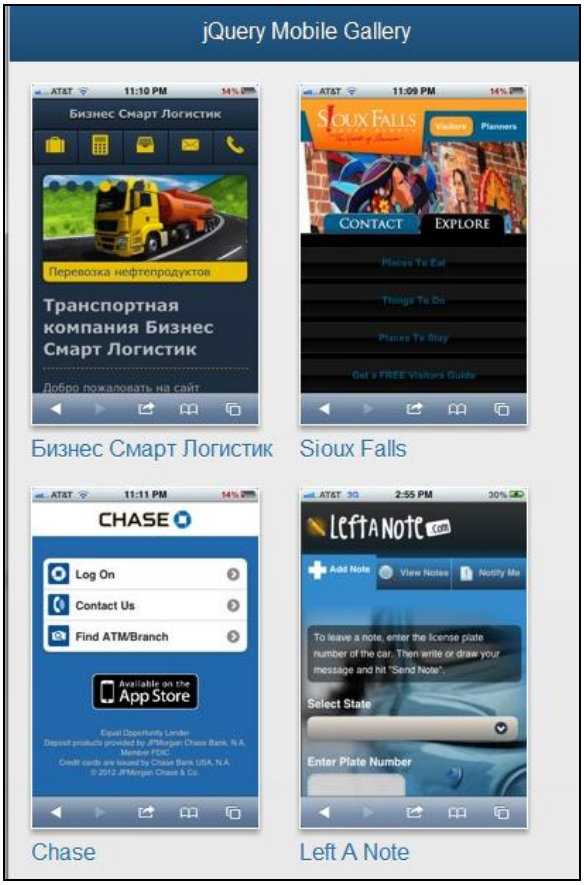
While we'd like to think that this book will cover every single possible topic you would need for all your jQuery Mobile needs, most likely there will be things we can't cover. If you need help, there are a couple of places you can try.

First, the jQuery Mobile docs (`http://jquerymobile.com/demos/1.0/`), cover syntax, features, and development in general, much like this book. While the material may cover some of the same ground, if you find something confusing here, try the official docs. Sometimes a second explanation can really help.

Second, the jQuery Mobile forum (`http://forum.jquery.com/jquery-mobile`), is an open ended discussion list for jQuery Mobile topics. This is the perfect place to ask questions. Also, it's a good place to learn about problems other people are having. You may even be able to help them. One of the best ways to learn a new topic is by helping others.

Examples

Want to see jQuery Mobile in action? There's a site for that. JQM Gallery (`http://www.jqmgallery.com/`), is a collection of sites submitted by users, built using jQuery Mobile. Not surprisingly, this website too uses jQuery Mobile, which makes it yet another way to sample jQuery Mobile:



Бизнес Смарт Логистик | Sioux Falls

Chase | Left A Note

What This Book Covers

Chapter 1, Preparing your First jQuery Mobile Project, works you through your first jQuery Mobile project. It details what must be added to your project's directory and source code.

Chapter 2, Working with jQuery Mobile Pages, continues the work in the previous chapter and introduces the concept of jQuery Mobile pages.

Chapter 3, Enhancing Pages with Headers, Footers, and Toolbars, explains how to enhance your pages with nicely formatted headers and footers.

Chapter 4, Working with Lists, describes how to create jQuery Mobile list views. These are mobile optimized lists which are especially great for navigation.

Chapter 5, Getting Practical – Building a Simple Hotel Mobile Site, walks you through creating your first "real" (albeit simple) jQuery Mobile application.

Chapter 6, Working with Forms and jQuery Mobile, explains the process of using jQuery Mobile optimized forms. Layout and special form features are covered in detail.

Chapter 7, Creating Modal Dialogs, Grids, and Collapsible Blocks, walks you through special jQuery Mobile user interface items for creating grid based layouts, dialogs, and collapsible content areas.

Chapter 8, jQuery Mobile Configuration, Utilities, and JavaScript methods, describes the various JavaScript-based utilities your code may have need of.

Chapter 9, Working with Events, details the events thrown by various jQuery Mobile-related features, like pages loading and unloading.

Chapter 10, Moving Further with the Notekeeper Mobile Application, walks you through the process of creating another website, an HTML5-enhanced note taking application.

Chapter 11, Enhancing jQuery Mobile, demonstrates how to change the default appearance of your jQuery Mobile sites by selecting and creating unique themes.

Chapter 12, Creating Native Applications, takes what you've learned previously and shows how to use the open source PhoneGap project to create real native applications.

Chapter 13, Becoming an expert Build an RSS Reader application, expands upon the previous chapter by creating an application that lets you add and read RSS feeds on mobile devices.

7
Creating Modal Dialogs, Grids, and Collapsible Blocks

In this chapter, we will look at dialogs, grids, and collapsible blocks. In the previous chapters we've dealt with pages, buttons, and form controls. While jQuery Mobile provides great support for them, there are even more UI controls you get within the framework.

In this chapter, we will:

- Discuss how to link to and create dialogs – also how to handle leaving them
- Demonstrate grids and how you can add them to your pages
- Show how collapsible blocks allow you to pack a lot of information in a small amount of space

Creating dialogs

Dialogs: at least under the jQuery Mobile framework: are small windows that cover an existing page. They typically provide a short message or question for the user. They will also typically include a button that allows the user to dismiss the dialog and return back to the site. Creating a dialog in jQuery Mobile is done by simply adding a simple attribute to a link: `data-rel="dialog"`. The following listing demonstrates an example:

```
Listing 7-1: test1.html
<!DOCTYPE html>
<html>
  <head>
    <title>Dialog Test</title>
    <meta name="viewport" content="width=device-width, initial-
      scale=1">
```

```
    <link rel="stylesheet" href="http://code.jquery.com/mobile/
      latest/jquery.mobile.min.css" />
    <script src="http://code.jquery.com/jquery-
      1.7.1.min.js"></script>
    <script src="http://code.jquery.com/mobile/latest/
      jquery.mobile.min.js"></script>
  </head>
  <body>
    <div data-role="page" id="first">
      <div data-role="header">
        <h1>Dialog Test</h1>
      </div>
      <div data-role="content">
        <p>
          <a href="#page2">Another Page (normal)</a>
        </p>
        <p>
          <a href="#page3" data-rel="dialog">A Dialog (dialog)</a>
        </p>
        </div>
      </div>
      <div data-role="page" id="page2">
        <div data-role="header">
          <h1>The Second</h1>
        </div>
        <div data-role="content">
          <p>
            This is the Second
          </p>
        </div>
      </div>
      <div data-role="page" id="page3">
        <div data-role="header">
          <h1>The Third</h1>
        </div>
        <div data-role="content">
        <p>
          This is the Third
        </p>
      </div>
    </div>
  </body>
</html>
```

This is a simple, multi-page jQuery Mobile site. Notice how we link to the second and third page. The first link is typical. The second link, though, includes the `data-rel` attribute mentioned earlier. Notice that both the second and third page are defined in the usual manner. So the only change we have here is in the link. When that second link is clicked, the page is rendered completely differently:

Remember, that page wasn't defined differently. The change you see in the previous screenshot is driven by the change to the link itself. That's it! Clicking the little **X** button will hide the dialog and return the user back to the original page.

Any link within the page will handle closing the dialog as well. If you wish to add a cancel type button, or link, you can do so using `data-rel="back"` in the link. The target of the link should be to the page that launched the dialog. `Listing 7-2` shows a modified version of the earlier template. In this one, we've simply added two buttons to the dialog. The first button will launch the second page, while the second one will act as a **Cancel** action.

```
Listing 7-2: test2.html
<!DOCTYPE html>
<html>
  <head>
    <title>Dialog Test (2)</title>
    <meta name="viewport" content="width=device-width, initial-
      scale=1">
    <link rel="stylesheet" href="http://code.jquery.com/mobile/
      latest/jquery.mobile.min.css" />
    <script src="http://code.jquery.com/jquery-
      1.7.1.min.js"></script>
```

```
    <script src="http://code.jquery.com/mobile/
      latest/jquery.mobile.min.js"></script>
  </head>
  <body>
    <div data-role="page" id="first">
      <div data-role="header">
        <h1>Dialog Test</h1>
      </div>
      <div data-role="content">
        <p>
          <a href="#page2">Another Page (normal)</a>
        </p>
        <p>
          <a href="#page3" data-rel="dialog">A Dialog (dialog)</a>
        </p>
      </div>
    </div>
    <div data-role="page" id="page2">
      <div data-role="header">
        <h1>The Second</h1>
      </div>
    <div data-role="content">
      <p>
        This is the Second
      </p>
    </div>
  </div>
  <div data-role="page" id="page3">
    <div data-role="header">
      <h1>The Third</h1>
    </div>
    <div data-role="content">
      <p>
        This is the Third
      </p>
        <a href="#page2" data-role="button">Page 2</a>
        <a href="#first" data-role="button" data-
          rel="back">Cancel</a>
      </div>
    </div>
  </body>
</html>
```

The major change in this template is the addition of the buttons in the dialog, contained within `page3 div`. Notice the first link is turned into a button, but outside of that is a simple link. The second button includes the addition of the `data-rel="back"` attribute. This will handle simply dismissing the dialog. The following screenshot shows how the dialog looks with the buttons added:

Laying out content with grids

Grids are one of the few features of jQuery Mobile that do not make use of particular data attributes. Instead, you work with grids simply by specifying CSS classes for your content.

Grids come in four flavors: Two column, Three column, Four column, and Five column. (You will probably not want to use the five column on a phone device. Save that for a tablet instead.)

You begin a grid with a `div` block that makes use of the class `ui-grid-X`, where `X` will be either `a`, `b`, `c`, or `d`. `ui-grid-a` represents a two column grid. `ui-grid-b` is a three column grid. You can probably guess what `c` and `d` create.

So to begin a two column grid, you would wrap your content with the following:

```
<div class="ui-grid-a">
  Content
</div>
```

Within the `div` tag, you then use a `div` for each "cell" of the content. The class for grid calls begins with `ui-block-X`, where `X` goes from `a` to `d`. `ui-block-a` would be used for the first cell, `ui-block-b` for the next, and so on. This works much like HTML tables.

Putting it together, the following code snippet demonstrates a simple two column grid with two cells of content:

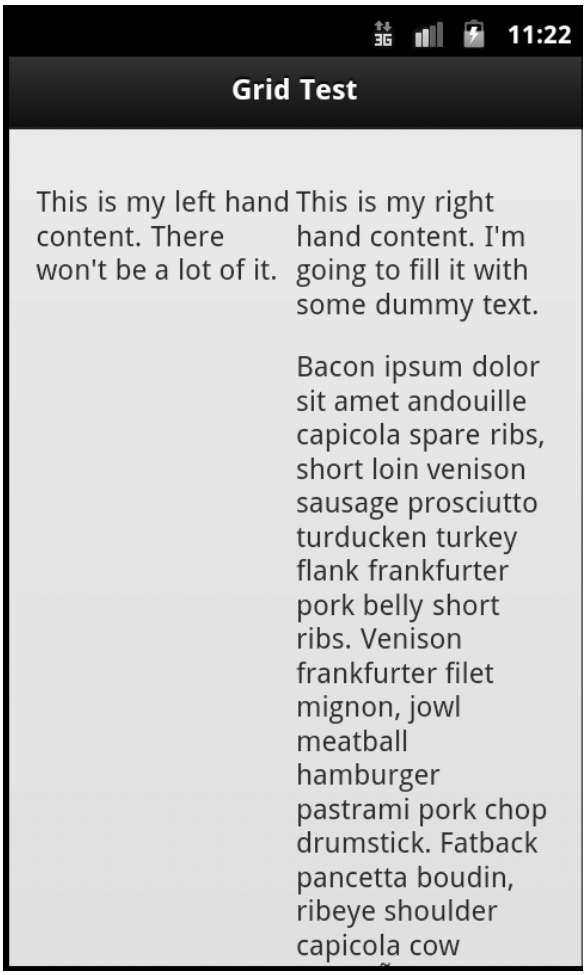
```
<div class="ui-grid-a">
  <div class="ui-block-a">Left</div>
  <div class="ui-block-b">Right</div>
</div>
```

Text within a cell will automatically wrap. `Listing 7-3` demonstrates a simple grid with a large amount of text in one of the columns:

```
Listing 7-3: test3.html
<!DOCTYPE html>
<html>
  <head>
    <title>Grid Test</title>
    <meta name="viewport" content="width=device-width, initial-
      scale=1">
    <link rel="stylesheet" href="http://code.jquery.com/mobile/
      latest/jquery.mobile.min.css" />
    <script src="http://code.jquery.com/jquery-
      1.7.1.min.js"></script>
    <script src="http://code.jquery.com/mobile/
      latest/jquery.mobile.min.js"></script>
  </head>
  <body>
    <div data-role="page" id="first">
      <div data-role="header">
        <h1>Grid Test</h1>
      </div>
      <div data-role="content">
        <div class="ui-grid-a">
          <div class="ui-block-a">
            <p>
              This is my left hand content. There won't be a lot of
                it.
```

```
            </p>
          </div>
          <div class="ui-block-b">
            <p>
              This is my right hand content. I'm going to fill it
              with some dummy text.
            </p>
            <p>
              Bacon ipsum dolor sit amet andouille capicola spare
              ribs, short loin venison sausage prosciutto turducken
              turkey flank frankfurter pork belly short ribs. Venison
              frankfurter filet mignon, jowl meatball hamburger
              pastrami pork chop drumstick. Fatback pancetta boudin,
              ribeye shoulder capicola cow leberkäse bresaola spare
              ribs prosciutto venison ball tip jowl andouille. Beef
              ribs t-bone swine, tail capicola turkey pork belly
              leberkäse frankfurter jowl. Shankle ball tip sirloin
              frankfurter bacon beef ribs. Tenderloin beef ribs pork
              chop, pancetta turkey bacon short ribs ham flank chuck
              pork belly. Tongue strip steak short ribs tail swine.
            </p>
          </div>
        </div>
      </div>
    </div>
  </body>
</html>
```

In the mobile browser, you can clearly see the two columns:



Working with other types of grids then is simply a matter of switching to the other classes. For example, a four column grid would be set up similar to the following code snippet:

```
<div class="ui-grid-c">
  <div class="ui-block-a">1st cell</div>
  <div class="ui-block-b">2nd cell</div>
  <div class="ui-block-c">3rd cell</div>
</div>
```

Again, keep in mind your target audience. Anything over two columns may be too thin on a mobile phone.

To create multiple rows in a grid, you simply repeat blocks. The following code snippet demonstrates a simple example of a grid with two rows of cells:

```
<div class="ui-grid-a">
  <div class="ui-block-a">Left Top</div>
  <div class="ui-block-b">Right Top</div>
  <div class="ui-block-a">Left Bottom</div>
  <div class="ui-block-b">Right Bottom</div>
</div>
```

Notice that there isn't any concept of a row. jQuery Mobile handles knowing that it should create a new row when the block starts over with the one marked `ui-block-a`. The following code snippet, `Listing 7-4` is a simple example:

```
Listing 7-4:test4.html
<!DOCTYPE html>
<html>
  <head>
    <title>Grid Test (2)</title>
    <meta name="viewport" content="width=device-width, initial-
      scale=1">
    <link rel="stylesheet" href="http://code.jquery.com/mobile/
      latest/jquery.mobile.min.css" />
    <script src="http://code.jquery.com/jquery-
      1.7.1.min.js"></script>
    <script src="http://code.jquery.com/mobile/latest/
      jquery.mobile.min.js"></script>
  </head>
  <body>
    <div data-role="page" id="first">
      <div data-role="header">
        <h1>Grid Test</h1>
      </div>
      <div data-role="content">
        <div class="ui-grid-a">
          <div class="ui-block-a">
            <p>
              <img src="ray.png">
            </p>
          </div>
          <div class="ui-block-b">
            <p>
              This is Raymond Camden. Here is some text about him. It
              may wrap or it may not but jQuery Mobile will make it
              look good. Unlike Ray!
            </p>
```

```
            </div>
            <div class="ui-block-a">
              <p>
                This is Scott Stroz. Scott Stroz is a guy who plays
                golf and is really good at FPS video games.
              </p>
            </div>
            <div class="ui-block-b">
              <p>
                <img src="scott.png">
              </p>
            </div>
          </div>
        </div>
      </div>
    </body>
  </html>
```

The following screenshot shows the result:

Working with collapsible content

The final widget we will look at in this chapter supports collapsible content. This is simply content that can be collapsed and expanded. Creating a collapsible content widget is as simple as wrapping it in a div, adding `data-role="collapsible"`, and including a title for the content. Consider the following simple example:

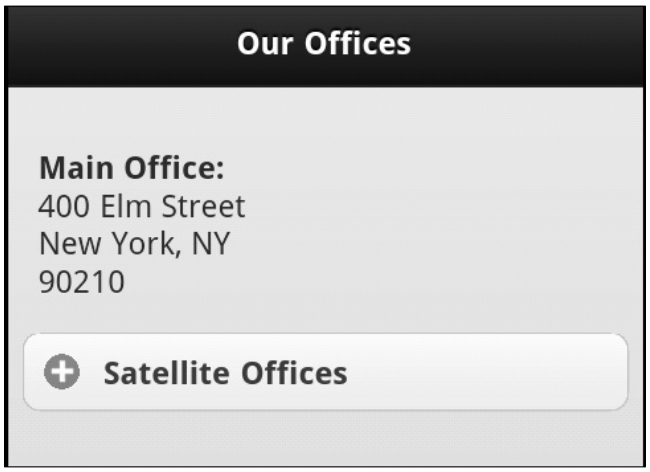
```
<div data-role="collapsible">
  <h1>My News</h1>
  <p>This is the latest news about me…
</div>
```

Upon rendering, jQuery Mobile will turn the title into a clickable banner that can expand and collapse the content within. Let's look at a real example. Imagine you want to share the location of your company's primary address. You also want to include satellite offices. Since most people won't care about the other offices, we can use a simple collapsible content widget to hide the content by default. The following code snippet, `Listing 7-5` demonstrates an example of this:

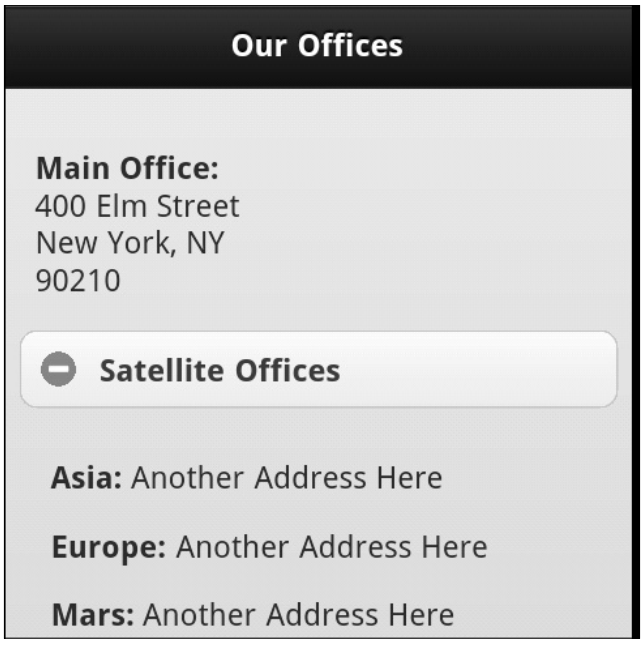
```
Listing 7-5: test5.html
<!DOCTYPE html>
<html>
  <head>
    <title>Collapsible Content</title>
    <meta name="viewport" content="width=device-width, initial-
      scale=1">
    <link rel="stylesheet" href="http://code.jquery.com/mobile/
      latest/jquery.mobile.min.css" />
    <script src="http://code.jquery.com/jquery-
      1.7.1.min.js"></script>
    <script src="http://code.jquery.com/mobile/latest/
      jquery.mobile.min.js"></script>
  </head>
  <body>
    <div data-role="page" id="first">
      <div data-role="header">
        <h1>Our Offices</h1>
      </div>
      <div data-role="content">
        <p>
          <strong>Main Office:</strong><br/>
          400 Elm Street<br/>
          New York, NY<br/>
          90210
        </p>
```

```
            <div data-role="collapsible">
              <h3>Satellite Offices</h3>
                <p>
                  <strong>Asia:</strong>
                  Another Address Here
                </p>
                <p>
                  <strong>Europe:</strong>
                  Another Address Here
                </p>
                <p>
                  <strong>Mars:</strong>
                  Another Address Here
                </p>
            </div>
          </div>
        </div>
      </body>
    </html>
```

You can see that the other offices are all wrapped in the `div` tag using the new collapsible content role. When viewed, notice that they are hidden:



Clicking the **+** next to the title opens it, and can be clicked again to reclose it:



By default, jQuery Mobile will collapse and hide the content. You can, of course, tell jQuery Mobile to initialize the block open instead of closed. To do so, simply add `data-collapsed="false"` to the initial `div` tag. For example:

```
<div data-role="collapsible" data-collapsed="false">
  <h1>My News</h1>
  <p>This is the latest news about me…
</div>
```

This region will still have the ability to collapse and open, but will simply default to being opened initially.

Another option for collapsible content blocks is the ability to theme the content of the area that is collapsed. By providing a `data-content-theme` attribute, you can specify a background color that makes the region a bit more cohesive. Theming is covered in *Chapter 11*, *Theming jQuery Mobile*, but we can take a look at a quick example. In the following screenshot, the first region does not make use of the feature, while the second one does:

Notice that the icon is also shifted to the right. This demonstrates another option, `data-iconpos`. The following code snippet, found in the code folder as `test5-2.html`, demonstrates these options:

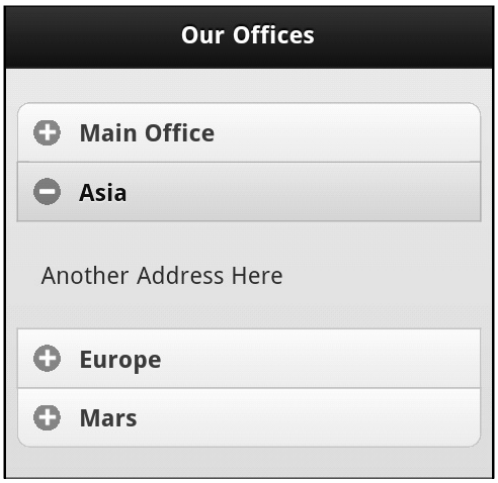
```
<div data-role="collapsible">
  <h3>First</h3>
  <p>
    Hello World...
  </p>
</div>
<div data-role="collapsible" data-content-theme="c" data-
  iconpos="right">
  <h3>First</h3>
  <p>
    Hello World again...
  </p>
 </div>
```

Finally, you can take multiple collapsible regions and combine them into one called an accordion. This is done by simply taking multiple collapsible blocks and wrapping them in a new `div` tag. This `div` tag makes use of `data-role="collapsible-set"` to make the inner blocks as one unit. `Listing 7-6` demonstrates an example of this. It takes the earlier office address example and uses a collapsible set for each unique address:

```
Listing 7-6: test6.html
<!DOCTYPE html>
<html>
  <head>
    <title>Collapsible Content</title>
    <meta name="viewport" content="width=device-width, initial-
      scale=1">
    <link rel="stylesheet" href="http://code.jquery.com/mobile/
      latest/jquery.mobile.min.css" />
    <script src="http://code.jquery.com/jquery-
      1.7.1.min.js"></script>
    <script src="http://code.jquery.com/mobile/
      latest/jquery.mobile.min.js"></script>
  </head>
  <body>
    <div data-role="page" id="first">
      <div data-role="header">
        <h1>Our Offices</h1>
      </div>
      <div data-role="content">
        <div data-role="collapsible-set">
          <div data-role="collapsible">
            <h3>Main Office</h3>
              <p>
                400 Elm Street<br/>
                New York, NY<br/>
                90210
              </p>
          </div>
          <div data-role="collapsible">
            <h3>Asia</h3>
              <p>
                Another Address Here
              </p>
          </div>
          <div data-role="collapsible">
            <h3>Europe</h3>
              <p>
                Another Address Here
              </p>
```

```
            </div>
            <div data-role="collapsible">
              <h3>Mars</h3>
                <p>
                  Another Address Here
                </p>
            </div>
          </div>
        </div>
      </div>
    </body>
  </html>
```

In `listing 7-6`, we simply wrap four collapsible blocks with a `div` tag that makes use of a collapsible set. Once done, jQuery Mobile will group them together and automatically close one once another is open:



Summary

In this chapter, we learned more about how jQuery Mobile enhances basic HTML to provide additional layout controls to our mobile pages. With dialogs, we learned how to provide a basic, quick, modal message to users. With grids, we learned a new way to easily layout content in columns. Finally, with the collapsible content blocks, we learned a cool way to share additional content without taking up as much screen space.

In the next chapter, we demonstrate a full, real example that creates a basic Note Tracker. It makes use of additional HTML5 features, as well as some of the UI tips you've learned over the past few chapters.

Where to buy this book

You can buy jQuery Mobile Web Development Essentials from the Packt Publishing website: `http://www.packtpub.com/jquery-mobile-web-development-essentials/book`.

Free shipping to the US, UK, Europe and selected Asian countries. For more information, please read our shipping policy.

Alternatively, you can buy the book from Amazon, BN.com, Computer Manuals and most internet book retailers.

www.PacktPub.com

For More Information:
www.packtpub.com/jquery-mobile-web-development-essentials/book

[PACKT] PUBLISHING

HTML5 iPhone Web Application Development

Alvin Crespo

Chapter No. 1
"Application Architecture"

In this package, you will find:

A Biography of the author of the book

A preview chapter from the book, Chapter NO.1 "Application Architecture"

A synopsis of the book's content

Information on where to buy this book

About the Author

Alvin Crespo is a creative technologist strongly focused on delivering compelling user experiences through the use of frontend technologies. Utilizing the latest industry standards, he strives to move the Web forward promoting open source technologies. Having worked in startup and agency environments, he has helped build and architect complex applications for both medium and large-sized companies.

First and foremost, I would like to thank my lovely wife, Janice Smith, for helping me produce this book. This has only been possible through the love and support you have given me. To my friends and family who have been there throughout the process, my love and endless thanks cannot express how awesome you all are.

HTML5 iPhone Web Application Development

Web applications have come a long way since the 90s, where static HTML pages were delivered to the client. In those days, web pages had a strict client-server model, where much of the processing was done on the server and the client only presented the information with little to no interaction. Such information was only accessible through desktop computers on very slow networks.

Those days have passed, and we are now connected in ways that were impossible before. From cellphones that can still make calls on the subway, to tablets presenting the latest article from your favorite newspaper 30,000 feet in the air; we are now in a digital age where information is easily accessible through innovative technologies. Yet, we still struggle to create a seamless interaction between technology and the physical world.

Although we have devices that are sensitive to our touch, can detect our location, and have the ability to monitor our vital signals, it is still up to us to make applications that will change the world. The hard work that goes into creating these applications usually requires large teams, complex business roles, and costly expenses.

For a brief period of time, developing these applications presented a challenge to many entrepreneurs who were looking to drive change. A staggered mobile market, which continues to this day, contributed to limited development resources. What we saw was an increase in the advancement of these technologies, but very few people who understood or were even interested in learning all the languages felt the necessity to create a cross-platform application.

However, it was only a matter of time until a single platform would arrive and change the world forever. HTML5, and its implementation across devices, helped drive the force necessary to deliver a platform that allowed developers to innovate and change the world. Leveraging this technology in our applications allows us to push the limit of the hardware while creating something that many users can enjoy, no matter what device they prefer to use.

Over the years, I have come to realize that device agnostic applications will become the norm. We have seen competitors adopt these standards with little to no impact on their success; in fact, it can be argued that it has done the opposite. For these reasons, this book was written to provide you with the techniques to create applications, based on open standards, and for the successful creation of device agnostic software.

What This Book Covers

Chapter 1, Application Architecture, helps you to learn how to create a standard architecture for iPhone web application development. We will customize the standard HTML5 Mobile Boilerplate for our needs throughout the book.

Chapter 2, Integrating HTML5 Video, helps you to learn the basics of implementing an HTML5 video player within your web application. We'll review the specification and implement an exposed API to tap into.

Chapter 3, HTML5 Audio, explains an implementation of the HTML5 Audio API. We'll create an audio player that utilizes the same principles from Chapter 2 to create a reusable component.

Chapter 4, Touch and Gestures, helps you to learn about touch and gesture events, including the similarities and differences. We'll go over a couple of examples and more importantly, the specification to properly integrate our application's user experience.

Chapter 5, Understanding HTML5 Forms, explains the new features in HTML5 forms, ultimately understanding its uses for our iOS web applications. We'll review the new inputs, their interactions, and the behaviors expected from the iOS operating system.

Chapter 6, Location-aware Applications, will have Geolocation as the key point, from the specification to the full implementation in the Safari iOS browser. We'll create an example that utilizes this feature and demonstrate how we can utilize it in our own applications.

Chapter 7, One-page Applications, is jam-packed with information on how to create a seamless experience in your application. We'll go over the principles of the MVC design pattern and create an example that utilizes its full potential.

Chapter 8, Offline Applications, will cover key topics such as Caching, History, and local storage. The essentials will be covered and the details exposed in order for us to create true offline applications.

Chapter 9, Principles of Clean and Optimized Code, will have us sidestepping the development process to refine our craftsmanship. We'll go over best practices, industry supported techniques, and ways to improve our code for the overall benefit of our applications.

Chapter 10, Creating a Native iPhone Web Application, reviews how we can create the native application we have learned previously. Applying the same techniques we'll create native applications based on open standards

1
Application Architecture

In this chapter, we will create a standard architecture for our iPhone application. We will base it on the HTML5 Mobile Boilerplate and customize it for the needs of the several projects in this book. From marking up our content in HTML5 to creating a JavaScript framework, we'll create static pages that help us focus on the foundations of iPhone Web Application development.

In this chapter, we will cover:

- Implementing the HTML5 Mobile Boilerplate
- Creating a preliminary architecture
- Customizing our framework
- Creating semantic markup
- Structuring our stylesheets
- Responsive design principles
- Establishing our JavaScript architecture
- Routing to a mobile site
- Home screen icons
- Introducing our build script
- Deploying our project

Implementing the HTML5 Mobile Boilerplate

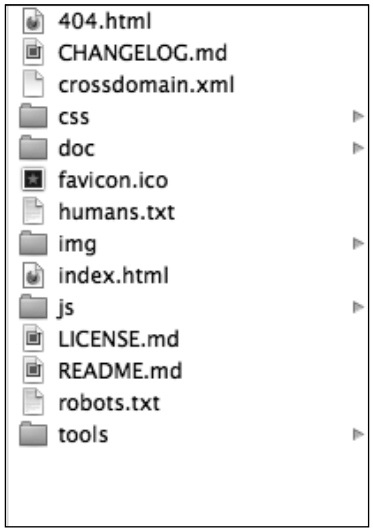
When you begin development, it is always critical to start with a basic framework that can be molded to the needs of your project. In many cases, we develop these frameworks in-house where we work, or perhaps for our own personal projects. However, the open source community has presented us with a great framework we can use in our projects—the HTML5 Mobile Boilerplate. This framework is based on the well-known HTML5 Boilerplate, and has been optimized for mobile including a lean HTML template; the utilization of `Zepto`, and use of tools and helpers optimized for mobile.

Downloading and installing the HTML5 Mobile Boilerplate

The first step we need to take is to download the HTML5 Mobile Boilerplate, which is located here:

```
http://html5boilerplate.com/mobile/
```

Once the boilerplate is downloaded, you should see the following structure from the unzipped archive file:



The Preliminary Directory Structure

The next step is to take these files and place them in the directory of your choice. For example, I have placed my files in the following directory on my Mac:

```
/Users/alvincrespo/Sites/html5iphonewebapp
```

Next, we'll want to use a build system that helps us create multiple environments, ease the deployment process, and overall make things easier when we want to optimize our site for testing and/or production.

According to the documentation for the HTML5 Mobile Boilerplate, there are two different types of build system, such as the Node Build script and the Ant Build script. In this book, we'll be using the Ant Build script. I would recommend using the Ant Build script since it has been around for a while and has the appropriate features that I use in my projects, including CSS Split, which will help split up the main CSS file that comes with the boilerplate.

Integrating the build script

To download the Ant Build script, go to the following link:

```
https://github.com/h5bp/ant-build-script
```

Then, download the zip file by clicking on the **Download as zip** button. When you have downloaded the Ant Build script, copy the folder and its contents to your project.

Once your Ant Build script directory is fully transferred over to your project, rename the directory containing the build script to `build`. At this point, you should have your project completely set up for the rest of the applications in this book. We will cover how to utilize the build script later on in this chapter.

Creating our application framework

With every project, it's important to create a framework that adjusts to your project's needs. It's critical to think about every aspect of the project. From the required document to the team's strengths and weaknesses, it's important we establish a solid foundation that helps us build and adjust accordingly.

Modifying the boilerplate

We'll now modify our boilerplate for the needs of the projects we will be building. For simplicity, we'll remove the following items from the folder:

- `CHANGELOG.md`
- `crossdomain.xml`
- `README.md`
- `/doc (Directory)`

Now that the directory has been cleaned up, it's time to take a look at some of the boilerplate code and customize it for the needs of the projects in this book.

Customizing our markup

First, open up the application in your favorite text editor. Once we've opened up the application in the editor of our choice, let's look at `index.html`.

The index file needs to be cleaned up in order to focus on iPhone Web Application development, and also unused items such as Google Analytics need to be removed. So let's remove some code that is not necessary for us.

Look for the following code:

```
<!DOCTYPE html>
<!--[if IEMobile 7 ]>    <html class="no-js iem7"> <![endif]-->
<!--[if (gt IEMobile 7)|!(IEMobile)]><!--> <html class="no-js">
<!--<![endif]-->
```

Downloading the example code

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

And modify it to this:

```
<!DOCTYPE html>
<html class="no-js">
```

What we've done here is removed detection for IE Mobile. Although this may be helpful for other projects, for us it doesn't really help in creating a fully compatible application just for the iPhone. However, we also need to remove an `IEMobile` specific meta tag:

```
<meta http-equiv="cleartype" content="on">
```

The previous meta tag turns on `cleartype` (a utility that assists with the rendering of fonts) for the IE mobile. This isn't necessary for us and is not a requirement for our applications.

Now that we've removed some unnecessary markup from our page, we can go ahead and start enabling features that will enhance our application. Look for the following meta tags and enable them, by removing the comments surrounding them:

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

These directives inform our application that it can run in fullscreen and they set the status bar to black.

We can also remove the following code from the `<head>` of the document:

```
<!-- This script prevents links from opening in Mobile Safari.
https://gist.github.com/1042026 -->
<!--
        <script>(function(a,b,c){if(c in b&&b[c]){var d,e=a.
location,f=/^(a|html)$/i;a.addEventListener("click",function(a){d=a.
target;while(!f.test(d.nodeName))d=d.parentNode;"href"in d&&(d.href.
indexOf("http")||~d.href.indexOf(e.host))&&(a.preventDefault(),e.
href=d.href)},!1)}})(document,window.navigator,"standalone")</script>
-->
```

Once we've removed the previous script, your markup should now look like the following:

```
<!DOCTYPE html>
<head>
    <meta charset="utf-8">
    <title></title>
    <meta name="description" content="">
    <meta name="HandheldFriendly" content="True">
    <meta name="MobileOptimized" content="320">
    <meta name="viewport" content="width=device-width">
    <link rel="apple-touch-icon-precomposed" sizes="144x144"
href="img/touch/apple-touch-icon-144x144-precomposed.png">
```

```
    <link rel="apple-touch-icon-precomposed" sizes="114x114"
href="img/touch/apple-touch-icon-114x114-precomposed.png">
    <link rel="apple-touch-icon-precomposed" sizes="72x72" href="img/
touch/apple-touch-icon-72x72-precomposed.png">
    <link rel="apple-touch-icon-precomposed" href="img/touch/apple-
touch-icon-57x57-precomposed.png">
    <link rel="shortcut icon" href="img/touch/apple-touch-icon.png">
    <meta name="apple-mobile-web-app-capable" content="yes">
    <meta name="apple-mobile-web-app-status-bar-style"
content="black">
    <link rel="stylesheet" href="css/normalize.css">
    <link rel="stylesheet" href="css/main.css">
    <script src="js/vendor/modernizr-2.6.1.min.js"></script>
</head>
```

Now, we can focus on cleaning up our body. Lucky for us, we only need to remove one thing—Google Analytics, since we will not be focusing on tracking for iPhone Web Apps.

To do this, find the following code and remove it:

```
<!-- Google Analytics: change UA-XXXXX-X to be your site's ID. -->
<script>
    var _gaq=[["_setAccount","UA-XXXXX-X"],["_trackPageview"]];
    (function(d,t){var g=d.createElement(t),s=d.
getElementsByTagName(t)[0];g.async=1;
    g.src=("https:"==location.protocol?"//ssl":"//www")+".google-
analytics.com/ga.js";
    s.parentNode.insertBefore(g,s)}(document,"script"));
</script>
```

The only scripts that you should have on the page should be the following:

```
<script src="js/vendor/zepto.min.js"></script>
<script src="js/helper.js"></script>
```

Once we've completed the previous steps, our markup should be clean and simple as follows:

```
<!DOCTYPE html>
<html class="no-js">
<head>
    <meta charset="utf-8">
    <title></title>
    <meta name="description" content="">
    <meta name="HandheldFriendly" content="True">
```

```
    <meta name="MobileOptimized" content="320">
    <meta name="viewport" content="width=device-width">

    <link rel="apple-touch-icon-precomposed" sizes="144x144"
href="img/touch/apple-touch-icon-144x144-precomposed.png">
    <link rel="apple-touch-icon-precomposed" sizes="114x114"
href="img/touch/apple-touch-icon-114x114-precomposed.png">
    <link rel="apple-touch-icon-precomposed" sizes="72x72" href="img/
touch/apple-touch-icon-72x72-precomposed.png">
    <link rel="apple-touch-icon-precomposed" href="img/touch/apple-
touch-icon-57x57-precomposed.png">
    <link rel="shortcut icon" href="img/touch/apple-touch-icon.png">

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

    <link rel="stylesheet" href="css/normalize.css">
    <link rel="stylesheet" href="css/main.css">
    <script src="js/vendor/modernizr-2.6.1.min.js"></script>
</head>
    <body>

        <!-- Add your site or application content here -->

        <script src="js/vendor/zepto.min.js"></script>
        <script src="js/helper.js"></script>
    </body>
</html>
```

From here, we should examine our stylesheets and scripts for every project and optimize it as much as we can prior to beginning a project. However, this boilerplate that we will be using has been optimized by the community and continuously enhanced with support from many developers, and for our use here, both styles and scripts are good to go. If you are curious, I encourage you to look at the `normalize.css` file, which contains excellent directives for resetting a page. It would also be beneficial to review the `main.css` file that has been enhanced with this boilerplate to support mobile devices.

Now, we'll move on to establishing our framework.

Customizing our framework

It's critical for developers to establish a framework for each project they are working on, no matter how small or big the project may be. Of course, your framework should adjust to the requirements that the project demands as well. In this section, we'll establish a simple framework that we can work with throughout the use of this book.

We've gone through and cleaned up the boilerplate for our needs, now we'll go through and expand upon the boilerplate to include the files that are critical to the applications we will build.

The first application will be based on the HTML5 Video specification (`http://dev.w3.org/html5/spec-author-view/video.html`). In that application we'll create a specific functionality for our video player that includes play, pause, and fullscreen functionalities. So let's create a directory specific to this application; we'll call this directory `video`.

In this directory, we'll create an `index.html` file and copy the contents from the homepage of the `index.html` file.

Now that we have our video section created, let's create the `video.css` file inside of our `css` directory.

Then, create an `App` directory within our `/js` folder. Within the `/js/App` directory, let's create an `App.js` file. Later, we'll explain in detail what this file is, but for now it will be our main application namespace that will essentially encapsulate global functionality for our application.

Finally, let's create an `App.Video.js` file that will contain our video application functionality within the `/js/App` directory.

You will now repeat the previous steps for each of our applications; including Video, Audio, Touch, Forms, Location, Single Page, and Offline. In the end, your directory structure should have the following new directories and files:

```
/audio
    index.html
/css
    audio.css
    forms.css
    location.css
    main.css
    normalize.css
    singlepage.css
```

```
        touch.css
        video.css
    /forms
        index.html
    /js
        /App/App.Audio.js
        /App/App.Forms.js
        /App/App.js
        /App/App.Location.js
        /App/App.SinglePage.js
        /App/App.Touch.js
        /App/App.Video.js
    /location
        index.html
    /offline
        index.html
    /singlepage
        index.html
    /touch
        index.html
    /video
        .index.html
```

At this point, we should fix the references to our dependencies, such as our JavaScript and stylesheet. So let's open up `/video/index.html`.

Let's modify the following lines:

```
<link rel="stylesheet" href="css/normalize.css">
<link rel="stylesheet" href="css/main.css">
<script src="js/vendor/modernizr-2.6.1.min.js"></script>
```

Change the previous markup to the following:

```
<link rel="stylesheet" href="../css/normalize.css">
<link rel="stylesheet" href="../css/main.css">
<script src="../js/vendor/modernizr-2.6.1.min.js"></script>
```

Note that we add `../` to each dependency. This is essentially telling the page to go up one level and retrieve the appropriate files. We also need to do this for the apple-touch-icon-precomposed links, shortcut icon, and the scripts at the bottom of the page.

Our framework is now almost complete, except that they aren't connected yet. So now that we've got everything organized, let's start hooking up everything to one another. It won't look pretty, but at least it will be working and moving towards a fully functional application.

Let's start with the main `index.html` file, `/ourapp/index.html`. Once we've opened up the main `index.html` file, let's create a basic site structure inside our `<body>` element. We'll give it a class of `"site-wrapper"` and put it right below the comment `Add your site or application content here`:

```
<body>
    <!-- Add your site or application content here -->
    <div class="site-wrapper">

    </div>
    <script src="js/vendor/zepto.min.js"></script>
    <script src="js/helper.js"></script>
</body>
```

Within the wrapper containing our site, let's use the new HTML5 `<nav>` element to semantically describe the main navigation bar that will exist across all our apps:

```
<div class="site-wrapper">
<nav>
</nav>
</div>
```

Nothing too special yet, but now we'll go ahead and use the unordered list element and create a navigation bar with no styling:

```
<nav>
    <ul>
        <li>
            <a href="./index.html">Application
Architecture</a>
        </li>
        <li>
            <a href="./video/index.html">HTML5 Video</a>
        </li>
        <li>
            <a href="./audio/index.html">HTML5 Audio</a>
        </li>
        <li>
            <a href="./touch/index.html">Touch and Gesture
Events</a>
```

```
        </li>
        <li>
            <a href="./forms/index.html">HTML5 Forms</a>
        </li>
        <li>
            <a href="./location/index.html">Location Aware
Applications</a>
        </li>
        <li>
            <a href="./singlepage/index.html">Single Page
Applications</a>
        </li>
    </ul>
</nav>
```

If we copy the code that we have created in `/video/index.html` and test the page, you see that it will not work correctly. For all subdirectories, like video and audio, we'll need to change the relative path from `./` to `../` so that we can go up one folder. With this in mind, the `nav` element would look like the following within the other applications:

```
<nav>
    <ul>
        <li>
            <a href="../index.html">Application
Architecture</a>
        </li>
        <li>
            <a href="../video/index.html">HTML5 Video</a>
        </li>
        <li>
            <a href="../audio/index.html">HTML5 Audio</a>
        </li>
        <li>
            <a href="../touch/index.html">Touch and Gesture
Events</a>
        </li>
        <li>
            <a href="../forms/index.html">HTML5 Forms</a>
        </li>
        <li>
            <a href="../location/index.html">Location Aware
Applications</a>
        </li>
```

```
            <li>
                <a href="../singlepage/index.html">Single Page
    Applications</a>
            </li>
        </ul>
    </nav>
```

Now, we can copy the navigation from `/video/index.html` to the rest of the application files or to the `index.html` files we created previously. Once this is done, we will have a single site that now connects well with each other.

Believe it or not, we have a very simple website going on here. Our pages are set up with basic markup and general styles. At this point, we need a navigation that brings our pages together. However, we've barely touched on some important aspects, including semantic markup for applications, which we'll discuss next.

Creating semantic markup

Semantic markup is important for several reasons, including search engine optimization, creating maintainable architectures, making code easily understandable, and meeting accessibility requirements. However, you should be familiar with structuring your page with markup that is related to your content. There are new elements within the HTML5 specification that help to ease this process, including the `<header>`, `<nav>`, `<footer>`, `<section>`, `<article>`, and `<aside>` elements. Each one of these elements helps describe the aspects of a page and easily identifies components of your application. In this section, let's structure our applications, beginning with our Video application.

Creating the header

First, let's start by giving our main index page a title and a header that describes the page we are on. Let's open the main `index.html` file in our application at `/index.html`.

Find the `<title>` tag and enter it in `iPhone Web Application Development - Home`. Note that we use a hyphen here. This is important since it makes it easier for users to scan the content of the page and helps with the ranking for specific keywords.

You should now have the following `<title>` in the `<head>` tag of your document:

```
<title>iPhone Web Application Development - Home</title>
```

Now we want the content of the page to reflect the title as well and alert the user of their progress on our site. What we want to do is create a header that describes the section they are on. In order to achieve this, let's place the following code before the navigation we created previously. Your code should then look like this:

```
<hgroup>
    <h1>iPhone Web Application Development</h1>
    <h2>Home</h2>
</hgroup>
<nav>...</nav>
```

The `<hgroup>` element is used to group multiple headers for a section. The rank of the headers is based on `<h1>` to `<h6>`, with `<h1>` being the highest rank and `<h6>` the lowest. Therefore, the highlighted text places our `<h1>` content higher than our `<h2>`.

Also note that we are not using the `<section>` element yet. However, this page does validate using the W3C Markup Validation Service (`http://validator.w3.org/`).

We can further describe the page by wrapping our `<hgroup>` and `<nav>` elements within a `<header>` element to give the page an introductory aid. Once you do this, you should have the following code:

```
<header>
    <hgroup>... </hgroup>
    <nav>... </nav>
</header>
```

With the previous code, we have finally given our page some structure. We are describing our page with a main header for the site and a sub header for the page. We have also given the page a navigation menu, allowing the user to navigate across applications.

Creating the footer

Now let's add a `<footer>` that contains the name of this book with its copyright date:

```
<footer>
    <p>iPhone Web Application Development &copy; 2013</p>
</footer>
```

The previous code will basically relate to the nearest sectioning ancestor. Thus the footer will relate to the content before it, which we will fill in a bit later. At this point, your content should look like this:

```
<div class="site-wrapper">
    <header>
        <hgroup>...</hgroup>
        <nav>...</nav>
    </header>
    <footer>...</footer>
</div>
```

Clearing up section

You may be wondering why we are not using the `<section>` element right away for the `<div>` element that contains both the `<header>` and `<footer>` element. In this case, it's not necessarily useful since we are not creating a page where the element's contents would be listed in an outline. This is the suggestion by the W3C and is something every developer should be aware of when deciding which element to use, `<div>` or `<section>`. In the end, it comes down to the content itself and the outline the team wishes to create.

Now that we have a basic structure for our pages, we can go ahead and do the same for the rest of our applications. This will be done for you in the code provided with this book in case you wish to review a final version.

With this in mind, we will move forward with our application development, making sure that we use semantic code when and where it makes sense.

Structuring our stylesheets

Styling is extremely important in any application we build, especially since it is the first aspect of any application the user experiences. In this section, we'll start structuring our styles appropriately.

Global styling

First, let's open our `main.css` file, located in the `CSS` directory. When you open this file, you'll see default boilerplate styles. At this point, let's skip through these to create our own styles. We'll review those styles as we continue to develop our applications.

Find the following line in `main.css`:

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

It's after this comment that we want to include the global styles for the semantic code we wrote previously.

Start by defining the global site styling such as the background color:

```
html{
    background: #231F20;
    border-top: 10px solid #FDFF3A;
    border-bottom: 5px solid #FDFF3A;
    width: 100%;
}
```

In the previous styling, we are making some stylistic choices like setting our background color and some borders. The important part here is that the width is defined at 100 percent for the HTML element. This will basically allow us to extend to 100 percent of the width of the phone for all our content.

Defining our global fonts

We then have to define overall fonts on the page. This will be basic for now and can continue to extend as design as per our application, but for now take a look at the following styles:

```
h1, h2, p, a {
    font-family: Arial, Helvetica, sans-serif;
    text-decoration: none;
}
```

```
h1, h2 {
    color: #A12E33;
    font-weight: bold;
    margin: 0;
    padding: 0;
}

h1 {
    font-size: 18px;
}

h2 {
    font-size: 14px;
    font-weight: normal;
}

p {
    color: #F15E00;
    font-size: 12px;
}

a,
a:visited {
    color: #F19C28;
}
```

In the previous code, you can see that we are working from a higher level down, the essential understanding of Cascading Style Sheets. We first define our headers, anchors, and paragraphs by using a specific font family and having no decoration.

As we work down the previous styles, we start to define each one more specifically, with headers having no padding or margins and a specific color. Then, when we go down further, we can see that each type of header has a specific font size and we do the same for paragraphs and anchors.

Our page layout

Once we've defined some of our fonts and site styling, we include some basic layout information for the `<div>` element containing our content:

```
.site-wrapper {
    padding: 5px 10px 10px;
}
```

Since our element automatically scales to 100 percent of the width of the screen, we tell the content to have a padding of `5px` at the top, `10px` at the left and right, and `10px` on the bottom. Alternatively, we could have written the following styles:

```
padding-top: 5px;
padding-left: 10px;
padding-right: 10px;
padding-bottom: 10px;
```

The former is known as a shorthand property setting and is considered best practice.

Using content with :before and :after

Since we also want to make sure our second header is differentiated in some form, we can use a CSS3 pseudo class selector and property to define the before and after content, as following:

```
hgroup h2:before,
hgroup h2:after {
    content: " :: ";
}
```

Keep in mind that the `:before` and `:after` pseudo selectors are supported in Safari 3.2 and above.

The previous selector targets the `<h2>` elements within the `<hgroup>` element and appends the content we have defined in the property before and after it, as per the `:before` and `:after` pseudo class selector.

Styling our navigation

Next, let's style our navigation to look and feel a bit more useable.

```
nav ul {
    padding: 0;
}

nav li {
    list-style: none;
}
```

```
nav a {
    display: block;
    font-size: 12px;
    padding: 5px 0;
}
```

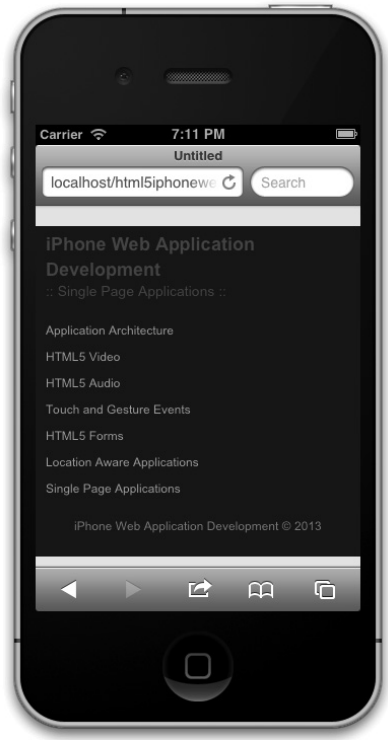
Here we remove the padding off the `<ul>` element and then remove the default styling option from each list element. Finally, we make sure each anchor is displayed correctly by setting the font size to `12px` and add padding to the top and bottom of each anchor to allow for easy selection on the iPhone.

Finally, we'll add some styling to our footer.

```
footer p {
    text-align: center;
}
```

Very simply, we're aligning the paragraph within the footer to center. Since we've defined the default styles for our paragraph in our fonts section, the styling gets picked.

When the previous styles are applied properly, your result should be similar to the following display:



Responsive design principles

Responsive design is the key to our mobile applications. Given the fact that many mobile experiences now surpass those viewed on desktop, it is essential we create applications that fit our evolving technological landscape. Lucky for us, the HTML5 Mobile Boilerplate comes with preliminary styles that we can modify.

Media queries to the rescue

First, let's open up our `main.css` file in our `css` directory.

Next, scroll down to the bottom of the file and you should see the following styling:

```
/* ==========================================================================
========
   EXAMPLE Media Queries for Responsive Design.
   Theses examples override the primary ('mobile first') styles.
   Modify as content requires.
==========================================================================
===== */

@media only screen and (min-width: 800px) {
}

@media only screen and (-webkit-min-device-pixel-ratio: 1.5),
       only screen and (min-resolution: 144dpi) {}
```

Although this styling gets us off the ground, for iPhone development, we need some more customization. The first media query is specific for tablet devices, and the second media query helps us by targeting devices with higher resolution, such as the iPhone 4.

What we want to do is make this a bit simpler. Since we are only targeting iPhones, this is what we can replace the previous code with:

```
/* iPhone 4 and 5 Styles*/
@media only screen and (-webkit-min-device-pixel-ratio: 2) { }
```

The previous code will target both the iPhone 4 and 5. We specifically target these two devices by checking the `-webkit-min-device-pixel-ratio` property on the device, and if it is true it means we can serve high definition graphics.

Another aspect we want to check is our viewport settings in the `index.html` pages we've set up. Luckily, we cleaned this up earlier and it should have the following:

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

The previous code snippet will basically resize our content based on the width of the device.

At this point, we should be set for implementing responsive styling later on in our applications. Now that our styling is set for our applications and is general enough to expand upon, let's start adding the framework behind the scripts.

Responsive images

Images are an extremely important part of any application. It helps showcase the features of a product and exemplifies information you want the user to understand. However, today's varying amount of devices require content to respond correctly. On top of that, we need to be able to deliver content that is appropriate for the experience, meaning we need to tailor to higher resolution devices so that the highest quality content reaches that audience.

There are multiple techniques for delivering the appropriate content. However, the one you choose depends on the requirements of your project. In this part, we'll review the traditional responsive web design principle of resizing an image according to its content and/or container.

Fluid images

In this technique, the developer sets all the images to a maximum width of 100 percent. We then define the container of the image to adjust accordingly.

Fluid width images

To achieve full width images, we can do the following:

```
<body>
<img src="img/batman.jpeg" alt="Its Batman!">
</body>
```

The markup is pretty simple, we essentially wrap an image into an element that extends the full width of what we need. In this case, the body will extend 100 percent in width.

Next, we'll define the style of the image as follows:

```
img {
    max-width: 100%;
}
```

With this simple CSS declaration, we are telling our images to have their maximum width set to 100 percent of the containing content. This will automatically resize the image as the device's width changes, which is essential if we want to make sites responsive to the user's device.

Full width images

In this case, we want the image to stay its full width, but we also need it to cut off accordingly.

To achieve this, we can start by simply creating a `div` with a `class`, in this case we add a class of `overflow`:

```
<div class="overflow"></div>
```

We can then create the styling that keeps the image at full width and cuts off based on the resizing of the content:

```
overflow {
    background: transparent url('img/somgimg.jpg') no-repeat 50% 0;
    height: 500px;
    width: 100%;
}
```

This is a bit complex, but essentially we attach the image with a `background` property. The key here is to make sure we center it using 50 percent. The height property is just to show the image, and the width tells the container to be 100 percent related to its content.

These are the two techniques we use when implementing a traditional responsive design. We'll be implementing these techniques much later when we create the video and image galleries.

Establishing our JavaScript architecture

When establishing a JavaScript architecture for your application, there's a lot to think about, including possible changes in the near or short term, security, ease of use and implementation, documentation, and more. Once we can answer the various questions we have, we can then decide on the pattern (module, facade and/or mediator, and so on). We also need to know what library or framework would be best suited for us, such as `jQuery`, `Zepto.js`, `Backbone.js`, or `Angular.js`.

Luckily for us, we'll be keeping it plain and simple in order to deliver an effective application on an iPhone. We'll be utilizing `Zepto.js` as our supported library to keep it light. We'll then build upon Zepto by creating a custom JavaScript framework that follows a modular pattern.

Structuring our app functionality

First, let's open up our application directory in our preferred text editor.

Next, open the `App.js` file we created earlier within our JavaScript directory. The `App.js` file should be completely empty, and it shouldn't be included anywhere. This is where we will begin writing our framework.

Namespacing our application

If you're new to JavaScript, you have most likely created most of your code in the global scope—perhaps laying out most of your JavaScript inside of script tags. Although this may achieve some of your goals, when working on large scale applications we want to avoid such practices. Some of the reasons we want to namespace our applications is for maintainability, efficiency, and portability.

Let's start out by checking for the `App` namespace; if it exists we'll use what's there, if it does not exist, then we'll make an empty object. The following code shows how we can achieve this:

```
var App = window.App || {};
```

Immediately Invoked Function Expressions

Great! We are checking for the `App` namespace, now let's define it. Let's include the following code after the check:

```
App = (function(){}());
```

The previous code is doing several things, let's take it one step at a time. First, we're setting the `App` namespace to what is known as an **Immediately Invoked Function Expression** (**IIFE**). We are essentially creating a function that is wrapped by parentheses and immediately invoking it after the closing brace.

When we use the previous technique, or IIFE, we create a new execution context or scope. This helps in creating self-containing code that will hopefully, not impact other code on the site. It protects us and helps us follow the modular pattern efficiently.

Let's extend the previous functionality by passing in the window, document, and Zepto objects, as follows:

```
App = (function(window, document, $){
}(window, document, Zepto));
```

I know that this may be a bit confusing, but let's take a second to think through what we're doing here. First, we are setting some parameters in the function named `window`, `document`, and `$`. Then, we are passing in `window`, `document`, and `Zepto` when we invoke this method. Remember, we discussed previously that this creates a new scope or execution context? Well, this becomes useful to us because we can now pass in references to any object that might be global.

How is this useful to us? Well, imagine if you wanted to use the actual `Zepto` object over and over again it would be kind of tiring. It's not that difficult to type `Zepto`, but you can just namespace it to the dollar sign and keep it simple.

Use strict

Ok, so we've got our module setup. Now let's continue to extend it by including the `use strict` directives:

```
App = (function(window, document, $){
    'use strict';
}(window, document, Zepto));
```

This directive helps us debug our applications by making changes to how JavaScript runs, allowing certain errors to be thrown instead of failing silently.

Default options

Default options are a great way of giving your codebase some extensibility. If, for example, we want to customize or cache an element related to the application itself then following are the defaults we will use:

```
var _defaults = {
'element': document.body,
    'name': 'App',
    'videoOptions': {},
    'audioOptions': {},
    'touchOptions': {},
    'formOptions': {},
    'locationOptions': {},
    'singlePageOptions': {}
};
```

Let's look at these defaults briefly. First we will create a `defaults` variable, which will contain all the defaults for our application(s). Inside it, we have defined a default location to be referenced for our application with the `'element'` default set to `document.body`—which gets our body element in **DOM** (**Document Object Model**). We then create a custom name for our application called `'App'`. After this, we create empty objects for our video, audio, touch, form, location, and single page applications—to be built later. These empty objects will be extended as we continue through the book.

Defining the constructor

Now we need to define our constructor after the `use strict` directive. This constructor will take a single parameter named `options`. We will then extend the defaults with the parameter `options` and store these settings that can be retrieved later, if needed. We will then ultimately cache the `'element'` option as a `Zepto` object.

```
function App(options) {
    this.options = $.extend({}, _defaults, options);
    this.$element = $(this.options.element);
}
```

Here is what the previous code is accomplishing. First, we are using the keyword `this`, which is a reference to what will be, an instance of App itself. Thus, `this` is the context of the object itself. Hopefully, this is not too confusing and will become clear as we go on. In this case, we are using `this` to define an object `options`, which will contain the merged contents of `_defaults` and any custom options we pass into the constructor.

Note, when we pass an empty object, or `{}` into `$.extend()` as the first parameter, we are telling `Zepto` to merge `_defaults` and `options` into a new object, thus not overwriting the `_defaults` object. This is useful when we need to do some sort of check in the future with the default options.

Once we've defined the options, we then cache the element with `this.$element`, where `$` in front of `element` is just for my reference so that I immediately recognize a Zepto object versus a plain JavaScript object.

The prototype

Ok, so we've created our `App` namespace, constructed an IIFE to contain our code and defined our constructor. Now, let's start creating some public methods that can be accessed to make this a bit modular. But before we do that, let's try to understand JavaScript's `prototype`.

Think of `prototype` as a live object that can be accessed, modified, and updated whenever and however you like. It can also be thought of as a pointer, because JavaScript will continue to go down the chain until it finds the object or it will return `undefined`. The prototype is simply a way of extending functionality to any non-plain object.

To make things a bit more confusing, I mentioned that non-plain objects have prototypes. These non-plain objects would be Arrays, Strings, Numbers, and so on. A plain object is one where we simple declare an empty object as follows:

```
var x = {};
```

The `x` variable does not have a prototype, it is simply there as a key/value storage similar to our `_defaults` object.

If you haven't yet understood the prototype, don't worry, it's all about getting your hands dirty and getting some experience. So, let's keep moving and getting our applications to work.

At this point, your `App.js` file should look like the following:

```
var App = window.App || {};
App = (function(window, document, $){
    'use strict';
    var _defaults = {
        'element': document.body,
        'name': 'App',
        // Configurable Options for each other class
        'videoOptions': {},
        'audioOptions': {},
        'touchOptions': {},
        'formOptions': {},
        'locationOptions': {},
        'singlePageOptions': {}
    };
    function App(options) {
        this.options = $.extend({}, _defaults, options);
        this.$element = $(this.options.element);
    }
}(window, document, Zepto));
```

Defining public methods

Now we need to create some public methods by typing into the prototype. We'll create a `getDefaults` method, which returns our default options; `toString` will overwrite the native `toString` method so we can return a custom name. Then we'll create initialization methods to create our other applications, and we'll name these `initVideo`, `initAudio`, `initLocalization`, `initTouch`, `initForms`, and `initSinglePage` respectively.

```
App.prototype.getDefaults = function() {
    return _defaults;
};

App.prototype.toString = function() {
    return '[ ' + (this.options.name || 'App') + ' ]';
};

App.prototype.initVideo = function() {
    App.Video.init(this.options.videoOptions);
    return this;
};

App.prototype.initAudio = function() {
    App.Audio.init(this.options.audioOptions);
    return this;
};

App.prototype.initLocalization = function() {
    App.Location.init(this.options.locationOptions);
    return this;
};

App.prototype.initTouch = function() {
    App.Touch.init(this.options.touchOptions);
    return this;
};

App.prototype.initForms = function() {
    App.Forms.init(this.options.formOptions);
    return this;
};

App.prototype.initSinglePage = function() {
    App.SinglePage.init(this.options.singlePageOptions);
    return this;
};
```

At this point we have several methods we can access publicly when we create an instance of `App`. First, let's review the code we implemented previously, specifically this line that gets duplicated, but customized based on the `init` method:

```
App.Touch.init(this.options.touchOptions);
```

For every `init` method we have created a call to the appropriate application, for example, `App.Touch`, `App.Forms`, `App.Video`, and so on. Then we pass it the options we've defined in the constructor that merged our defaults, for example, `this.options.touchOptions`, `this.options.formOptions`, `this.options.videoOptions`, and so on.

Note, we haven't created these classes yet for Video, Forms, Touch, and others, but we will be creating these soon.

Returning our constructor/function

The last thing we need to do in `App.js` includes returning the constructor. So, after all the public methods defined previously, include the following code:

```
return App;
```

This bit of code, although simple, is extremely important. Let's look at a stripped down version of `App.js` to better understand what's going on:

```
App = (function(){
    function App() {}
    return App;
}());
```

As mentioned earlier, we are creating an `App` namespace that gets set to the immediately invoked function expression. When we do this, we create a new scope inside this function.

This is why we can have a function or constructor with the name `App` as well and have no conflicts or errors. But if you recall, our function `App` is also an object, just like everything in JavaScript is an object. This is why, when we return our function `App` the `App` namespace gets set to the constructor. This then allows you to create multiple instances of `App`, while centralizing your code inside of a new scope that is untouchable.

Integrating a custom module template

Now, to get the rest of our architecture together we need to open up every other `App` file in the JavaScript directory we are in (`/js/App`).

When we have these files open, we need to paste the following template, which is based on the script we've written for `App.js`:

```
var App = window.App || {};

App.Module = (function(window, document, $){
    'use strict';

    var _defaults = {
        'name': 'Module'
    };

    function Module(options) {
        this.options = $.extend({}, _defaults, options);

        this.$element = $(this.options.element);
    }

    Module.prototype.getDefaults = function() {
        return _defaults;
    };

    Module.prototype.toString = function() {
        return '[ ' + (this.options.name || 'Module') + ' ]';
    };

    Module.prototype.init = function() {

        return this;
    };

    return Module;

}(window, document, Zepto));
```

When we have each template in, we must then change `Module` to the appropriate type, that is Video, Audio, Location, and so on.

Once you are done with pasting in the section and changing the names, you should be all set with the basic JavaScript architecture.

Including our scripts

One of the last items you will need to take care of is including this basic architecture into each `index.html` file. In order to do this, you will need to paste the following code at the bottom of the page, right after the inclusion of `helper.js`:

```
<script src="js/App/App.js"></script>
<script src="js/App/App.Audio.js"></script>
<script src="js/App/App.Forms.js"></script>
<script src="js/App/App.Location.js"></script>
<script src="js/App/App.SinglePage.js"></script>
<script src="js/App/App.Touch.js"></script>
<script src="js/App/App.Video.js"></script>
<script src="js/main.js"></script>
```

We are basically including each script of the framework. What's important here is to always include `App.js` first. The reason for this is that `App.js` creates the `App` object and directly modifies it. If you include it after all the other scripts, then `App.js` will overwrite the other scripts because it's directly affecting the `App` object.

Initializing our framework

The last item we need to take care of is `main.js`, which includes the initialization of our application. We do this by wrapping our code in IIFE and then exposing the instance to the `window` object. We do this with the following code:

```
(function(window, document) {
    'use strict';

    var app = new App({
        'element': document.querySelector('.site-wrapper')
    });

    window.app = app;

}(window, document));
```

What we've seen earlier is an IIFE being assigned to an object. Here we don't see that because it's not necessary. We just want to make sure our code would not affect the rest of the code, which in most cases would not happen because of the simplicity of this project. However, as a best practice I try to self contain my code in most cases.

The difference in the previous code is that we see the initialization of our framework here:

```
var app = new App({
    'element': document.querySelector('.site-wrapper')
});
```

We do that by using the `new` keyword, creating a new instance of `App`, and then passing it an object, which will be merged into our default options we previously wrote.

`querySelector` is a JavaScript method that is attached to the document object. This method accepts a selector that we would normally use in CSS, parse DOM, and find the appropriate element. In this case, we are telling our application to self contain itself to the element with the `site-wrapper` class.

When we finally initialize our application, we then attach `app` to the `window` object:

```
window.app = app;
```

This basically makes it accessible anywhere in our application by attaching it to the `window` object.

We are now done with the framework for our application. Although we don't have anything being manipulated on the page, or have attached any events that correlate with a user's input, we now have a solid foundation for coding that follows best practices, is effective, efficient, and easily accessible.

Routing to a mobile site

Unless we are making a completely responsive site where the styles of the site shift based on the dimensions of the device, we most likely will need to do some sort of redirect to a mobile friendly version of our site.

Lucky for us, this can easily be achieved in several ways. Although I won't cover in detail the ways in which we can achieve this, here are a few techniques that might help out when deciding how to move forward.

Since this book is geared towards the frontend, routing to a mobile site will be briefly covered with PHP and htaccess. We can always perform this process on the frontend, but it should be avoided for SEO and page-ranking purposes.

Redirecting via PHP

In PHP we could do the following type of redirect:

```
<?php
    $iphone = strpos($_SERVER['HTTP_USER_AGENT'], "iPhone");
    if ($iphone) {
        header('Location: http://mobile.site.com/');
    }
?>
```

In this example we are creating a variable, `$iPhone`, and giving it a Boolean value of true or false. If `iPhone` is found in the user agent, which may or may not be the best technique to use, then we tell the page to redirect using the `header()` method in PHP.

Again, there are other ways of achieving this, but this will get you off the ground and running.

Redirecting via htaccess

We can also detect the iPhone and redirect it by putting these instructions on the server using an `htaccess` file:

```
RewriteEngine on
RewriteCond %{HTTP_USER_AGENT} iPhone
RewriteRule .* http://mobile.example.com/ [R]
```

In this example, we are turning on the rewrite engine, creating a rewrite condition that checks for the `iPhone` text in the user agent, and then creates a rewrite rule if the condition is met.

In essence, if we want to redirect to a mobile version of our site, we need to be able to detect the type of device, not its dimensions, and then redirect appropriately.

Home screen icons

If you're creating an application that should mimic the feeling of being a native application, or to simply increase the experience of a web app—it is a good idea to have bookmark icons that represent your application.

At the moment, we do support this feature with the following markup in our `index.html` files:

```
<link rel="apple-touch-icon-precomposed" sizes="144x144"
href="img/touch/apple-touch-icon-144x144-precomposed.png">
<link rel="apple-touch-icon-precomposed" sizes="114x114"
href="img/touch/apple-touch-icon-114x114-precomposed.png">
<link rel="apple-touch-icon-precomposed" sizes="72x72"
href="img/touch/apple-touch-icon-72x72-precomposed.png">
<link rel="apple-touch-icon-precomposed" href="img/touch/apple-
touch-icon-57x57-precomposed.png">
<link rel="shortcut icon" href="img/touch/apple-touch-icon.png">
```

These directives inform Safari that we have home screen icons for the appropriate devices. Starting from top to bottom we are supporting retina display, first-generation iPad and non-Retina iPhone, iPad Touch, and even Android 2.1+.

To put it simply, we have an application that users can bookmark to their home screen, allowing them to instantly access the web application from their home screen.

Introducing our build script

Earlier, we installed our build script along with the HTML5 Mobile Boilerplate. We'll now explore the build script a bit further by customizing it for our purposes. We'll need to make sure our styles, scripts, images, and markup are optimized for deployment. It will also be necessary for us to set up multiple environments to test our application thoroughly.

Configuring our build script

Let's start by configuring the build script for our needs, this way we'll have a custom build script that works for us and gets us going immediately.

Minifying and concatenating scripts

First, let's make sure our scripts get concatenated and minified. So let's open all our `index.html` files and wrap all our scripts at the bottom of the page with the following comments:

```
<!-- scripts concatenated and minified via ant build script-->
<script src="path/to/script.js"></script>
<!-- end scripts-->
```

The previous comments are used by the `ant` task, or build script, to find all JavaScript files being used, concatenate, and minify them. The process will also use a timestamp for the newly optimized JavaScript file in order to bust caching on the server.

Minifying and concatenating styles

By default, the Ant Build script minifies and concatenates our styles. However, if we want to retain comments that identify a particular section of our app, such as the video or audio section, then we need to do something that will keep those comments.

The comments can be used to identify a section, and it can be written as follows:

```
/*!
  Video Styling
*/
```

Write the previous comments for each stylesheet.

We then need to add each of our stylesheets to the project properties so that each can be minified by the YUI compressor. To do this, we need to open up the `project.properties` file located in `/build/config`.

Then find the following line:

```
file.stylesheets  =
```

Once we've found that line, let's add all our `css` files as follows:

```
file.stylesheets  = audio.css,forms.css,location.css,singlepage.
css,touch.css,video.css
```

Note, that there are no spaces after each file. This is necessary for the build script to process.

This is all we need to do at the moment for optimizing our styles.

Creating multiple environments

Typically a project will run on a development, test, and production environment. The test environment should be closest to production in terms of configuration, allowing us to effectively reproduce any issues that might come up.

In order to build our environments correctly, let's go through the process of building our project. First, let's open up `Terminal`, a program that allows you to interact with the operating system of any Unix style computer through a command-line interface.

Navigating our directories

Once the terminal is up and running, we have to navigate to our project. Here are a couple of commands that will help you navigate:

```
cd /somesite
```

The previous command means we are changing our directory from the current directory to the `somesite` directory, relative to where you're now.

```
cd ../somesite
```

This command tells us to change the directory, but going up a level with `../` and then going into the `somesite` directory.

As an easier example to understand, my project exists in `/Sites/html5iphonewebapp`. So what I can do is use the following command to enter my project:

```
cd /Users/somuser/Sites/html5iphonewebapp
```

This changes the directory for me to the project where I am developing this application.

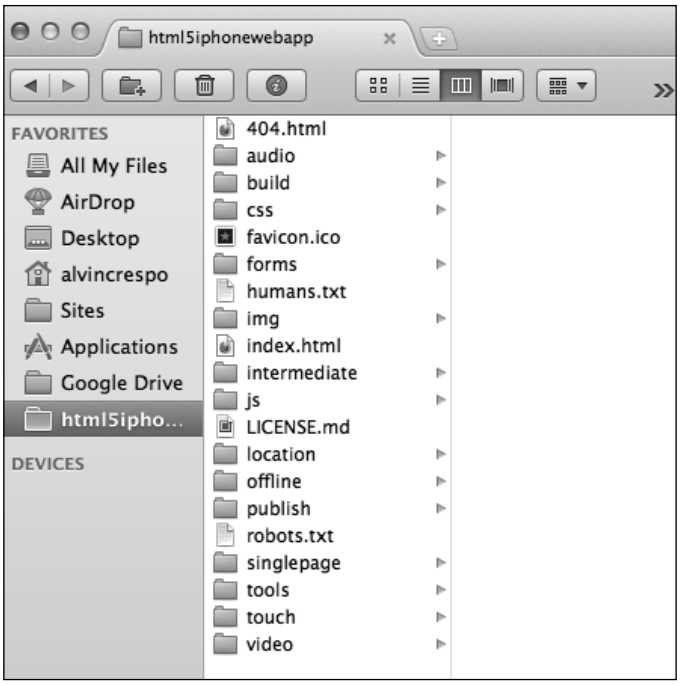
Building our project

Once we've entered the project directory, we can start building our project. By default, the Ant Build script creates a production environment, which optimizes all parts of the process.

```
ant build
```

This command tells us to build our project, and as explained creates our production version in a directory labeled `publish`. You will also notice that when you run that command, your terminal will update, letting you know what step in the process the build is in.

Once the build is complete, your directory structure should look similar to the following screenshot:



The `publish` directory represents the production environment. You will also see that an intermediate directory has been created; this is your test environment.

However, let's say you wanted to have full control of the build and wanted to create your environments manually, then one can do the following in the terminal:

```
ant build -Denv=dev
```

This command, `ant build -Denv=`, lets us define which environment we want to build and does it accordingly.

We now have a project that is ready to be built upon. There were many steps in this process, so I encourage you to practice this process in order to develop a good architecture and deployment process that works for you and/or your team.

Summary

In this chapter, we saw how to use the HTML5 Mobile Boilerplate for our projects, from downloading the default package to customizing it for our needs. We also took a couple of simple steps to establish a solid architecture for our JavaScript, CSS, and HTML. As a bonus, we went over including a build process and customizing it for our project. We then quickly reviewed best practices for JavaScript applications and gave a couple of tips on how to direct users to a separate mobile site. We are now prepared for in-depth development of the mobile web applications.

Where to buy this book

You can buy HTML5 iPhone Web Application Development from the Packt Publishing website: `http://www.packtpub.com/html5-iphone-web-application-development/book`.

Free shipping to the US, UK, Europe and selected Asian countries. For more information, please read our shipping policy.

Alternatively, you can buy the book from Amazon, BN.com, Computer Manuals and most internet book retailers.

www.PacktPub.com

For More Information:
www.packtpub.com/html5-iphone-web-application-development/book

[PACKT] open source
community experience distilled
PUBLISHING

Creating Mobile Apps with jQuery Mobile

Shane Gliser

Chapter No. 5

"Client-side Templating, JSON APIs, and HTML5 Web Storage"

In this package, you will find:

A Biography of the author of the book

A preview chapter from the book, Chapter NO.5 "Client-side Templating, JSON APIs, and HTML5 Web Storage"

A synopsis of the book’s content

Information on where to buy this book

About the Author

Shane Gliser graduated from Washburn University in 2001, specializing in Java development. Over the next several years, he developed a love of web development and taught himself HTML, CSS, and JavaScript. Having shifted his focus again, Shane's primary passions are user experience and the mobile web.

Shane began working with jQuery Mobile while it was still in the Alpha 2 phase and deployed American Century Investments' mobile site while the framework was still in Beta 2. Since then, he has rebranded and re-launched his own personal business, Roughly Brilliant Digital Studios (`http://roughlybrilliant.com`), as a place where he could start blogging tips about using jQuery Mobile.

Major thanks go to Todd Parker, Scott Jehl, and the rest of the crew at Filament Group and the many other volunteers who have given their time and talent to creating jQuery Mobile.

Jim Tharp, thank you for being my mobile partner-in-crime and for your continuous, epic sense of humor.

To the leadership team at American Century Investments, thank you for believing in my little two-week demo and trusting us to march down this unknown path.

Creating Mobile Apps with jQuery Mobile

Can we build it? Yes, we can!

Mobile is the fastest growing technology sector in existence. It is a wave of change that has shattered all analysts' expectations. You have the choice to harness that wave or to be swept under. In *Creating Mobile Apps with jQuery Mobile*, we'll take you through several projects of increasing complexity across a variety of industries. At the same time, we'll tackle several mobile usability and experience issues that are common to all mobile implementations, not just jQuery Mobile.

By the end you will have all the skills necessary to take jQuery Mobile and a host of other technologies and techniques to create truly unique offerings. This will be fun. It will be challenging, and by the end, you will be quoting Bob the Builder, "Can we build it? Yes we can!"

What This Book Covers

Chapter 1, Prototyping jQuery Mobile, harnesses the power of rapid prototyping before you start coding. Come to a quicker, better, and shared understanding with your clients.

Chapter 2, A Mom-and-Pop Mobile Website, implements the prototypes from *Chapter 1*. The design is unique and begins to establish a base server-side template.

Chapter 3, Analytics, Long Forms, and Front-end Validation, takes the casual implementation of *Chapter 2* and adds in Google Analytics, the jQuery Validate framework, and a technique for dealing with long forms.

Chapter 4, QR Codes, Geolocation, Google Maps API, and HTML5 Video, will have you implement a site for a movie theater chain.

Chapter 5, Client-side Templating, JSON APIs, and HTML5 Web Storage, creates a social news nexus, tapping into the API powers of Twitter, Flickr, and the Google Feeds API.

Chapter 6, HTML5 Audio, takes HTML5 audio and progressive enhancement to turn a very basic web audio player page into a musical artist's showcase.

Chapter 7, Fully Responsive Photography, explores the user of jQuery Mobile as a mobile-first, **responsive web design** (**RWD**) platform. We also take a very quick look at typography as it applies to RWD.

Chapter 8, Integrating jQuery Mobile into Existing Sites, explores the methods of building jQuery Mobile sites for clients who want their pages mobilized but don't have a **content management system** (**CMS**). We also dig deep into mobile detection methods including client-side, server-side, and a combination of the two.

Chapter 9, Content Management Systems and jQM, teaches us how to integrate jQM into WordPress and Drupal.

Chapter 10, Putting it all together – Flood.FM, builds on the knowledge of the previous chapters and creates, adds a little more, and considers compilation using PhoneGap Build.

5

Client-side Templating, JSON APIs, and HTML5 Web Storage

We've come a long way already and we've got some pretty hefty default templates and boilerplates for business. In this chapter, we're going to simplify and focus on some other things. We are going to create an aggregating news site based off social media. Until now, we've paid close attention to progressive enhancement. For this chapter, we leave that behind. This will require JavaScript.

In this chapter you will learn the following:

- Client-side templating options
- JsRender
- Patching into JSON API (Twitter)
- Programmatically changing pages
- Generated pages and DOM weight management
- Leveraging RSS feeds (natively)
- HTML5 Web Storage
- Leveraging the Google Feeds API

Client-side templating

(In a grumpy old man's voice) Back in my day, we rendered all the pages on the server, and we liked it! LOL! Times are changing and we are seeing a massive ground swell in client-side templating frameworks. At their heart, they're pretty much all the same in that they take JSON data and apply an HTML-based template contained within a script tag.

If you know what **JSON** is, skip this paragraph. I spent a little time last chapter discussing this, but just in case you skipped ahead and don't know, JSON is JavaScript written in such a way that it can be used as a data exchange format. It's more efficient than XML and is instantly interpretable by the browser in an object-oriented fashion. JSON can request data even across domains using JSONP. For more on JSON, read `http://en.wikipedia.org/wiki/JSON`. For more on JSONP, read `http://en.wikipedia.org/wiki/JSONP`.

All these client-side libraries have some sort of notation in them to show where the data goes and gives ways to implement looping and conditionals. Some are "logic-less" and operate on the philosophy that there should be as little logic as possible. If you subscribe to this wonderfully academic approach, good for you.

Honestly, from a purely pragmatic perspective, I believe that the template is the perfect place for code. The more flexible, the better. JSON holds the data and the templates are used to transform it. To draw a parallel, XML is the data format and XSL templates are used to transform. Nobody whines about logic in XSL; so, I don't see why it should be a problem in JS templates. But, all of this discussion is purely academic. In the end, they'll pretty much all do what you're looking for. If you're more of a designer than a coder, you may want to look more at the logic-less templates.

Following is a fairly exhaustive list of client-side templating frameworks. I'll probably miss a few and there will inevitably be more by the time this book gets published, but it's a start.

- doT
- dust.js
- Eco
- EJS
- Google Closure Templates
- handlebars

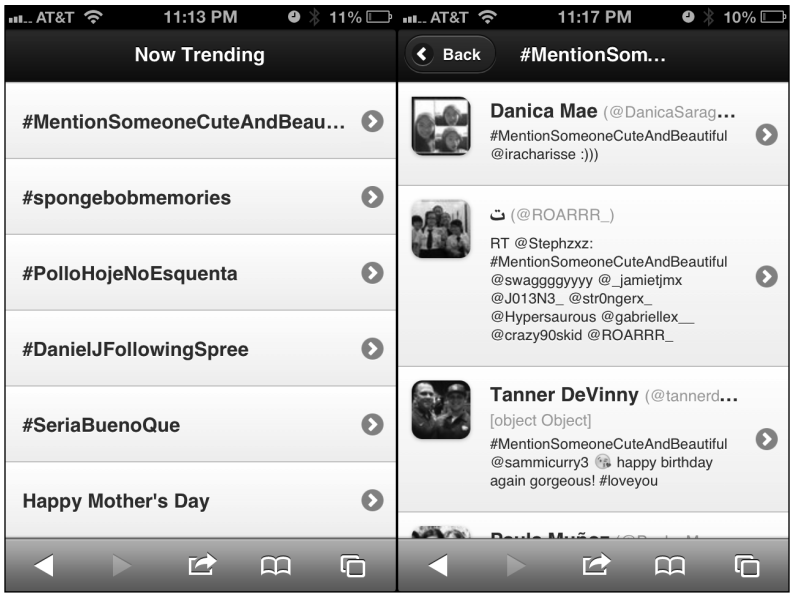
- haml-js
- kite
- Jade
- jQote2
- jQuery templates (discontinued)
- jsRender / jsView
- Parrot
- node-asyncEJS
- Nun
- Mu
- mustache
- montage
- Stencil
- underscore.js

Now, call me a fanboy, but, if it's officially jQuery, I love it. So, the first thing I tried was **jQuery Templates**. Sadly, shortly after learning to love it, the jQuery team abandoned the project and pointed people to **JsRender** as the continuation of the project. Whether that will be the continued direction in the future is another question, but, in the mean time, the features and power of JsRender make it a compelling offering and the basis for template work for the rest of this chapter. Not to mention, it's only 14k minified and fast as lightning. You can download the latest edition from `https://github.com/BorisMoore/jsrender`.

If you're looking for help to make the decision on the right template framework for you, Andy Matthews was kind enough to offer the following link during the review process for this chapter: `http://garann.github.com/template-chooser/`. It discusses the merits of several frameworks to help you make an informed choice. Thanks, Andy!

Patching into JSON APIs (Twitter)

It's always fun to watch the trending topics on Twitter. It, like so many other popular online destinations, has a JSON API. Let's have some fun. Here's what we're going to build. You can see the listview on the left-side and the search view on the right-side.



At this point, I'm going to dispense with the academically correct practice of separating the CSS and JS from the HTML. Aside from the libraries, all page-specific code (HTML, CSS, and JS) will be located within the single page. The following code is our starting base page. It is `twitter.html` in the code bundle for the chapter:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-
scale=1, maximum-scale=1.0, user-scalable=no">
    <title>Chapter 5 - News</title>
    <link rel="stylesheet" href="http://code.jquery.com/mobile/1.3.0/
jquery.mobile-1.3.0.min.css" />
    <script src="http://code.jquery.com/jquery-1.8.2.min.js"></script>
    <script src="js/jsrender.min.js" type="text/javascript"></script>
  <script src="http://code.jquery.com/mobile/1.3.0/jquery.mobile-
1.3.0.min.js"></script>
```

This next bit of styling will help our Twitter results look more Twitter-ish:

```
    <style type="text/css">
      .twitterItem .ui-li-has-thumb .ui-btn-inner a.ui-link-inherit,
#results .ui-li-static.ui-li-has-thumb{
         min-height: 50px;
         padding-left: 70px;
      }
      .twitterItem .ui-li-thumb, #results .ui-listview .ui-li-icon,
#results .ui-li-content{
         margin-top: 10px;
         margin-left: 10px;
      }
      .twitterItem .ui-li-desc{
         white-space:normal;
         margin-left:-25px;
      }
      .twitterItem .handle{
        font-size:80%;
        font-weight:normal;
        color:#aaa;
      }
      .twitterItem .ui-li-heading{
        margin: 0 0 .6em -25px;
      }
    </style>
  </head>
  <body>
```

This page is pretty much just a placeholder that will be filled in once we get back results from hitting the Twitter API:

```
  <div id="home_page" data-role="page">
    <div data-role="header"><h1>Now Trending</h1></div>
    <div data-role="content">
      <ul id="results" data-role="listview" data-dividertheme="b">
      </ul>
    </div>
  </div>
```

The following script is the processing core of the page.

```
  <script type="text/javascript">
    $(document).on("pagebeforeshow", "#home_page",  function(){

      //before we show the page, go get the trending topics
```

```
        //from twitter
      $.ajax({
        url:"https://api.twitter.com/1/trends/daily.json",
          dataType:"jsonp",
          success: function(data) {
            var keys = Object.keys(data.trends);

            //Invoke jsRender on the template and pass in
            //the data to be used in the rendering.
            var content = $("#twitterTendingTemplate")
             .render(data.trends[keys[0]]);

            //Inject the rendered content into the results area
            //and refresh the listview
            $("#results").html( content ).listview("refresh");
          }
        })
        .error(function(jqXHR, textStatus, errorThrown){
          alert(textStatus+" - "+errorThrown);
        });
      });

      $(document).on('click', 'a.twitterSearch', function(){
        var searchTerm = $(this).attr("data-search");

        //take the search term from the clicked element and
        //do a search with the Twitter API
        $.ajax({
          url:"http://search.twitter.com/search.json?q="+escape
  (searchTerm),
          dataType:"jsonp",
          success: function(data){

            //create a unique page ID based on the search term
            data.pageId = searchTerm.replace(/[# ]*/g,"");
            //add the search term to the data object
            data.searchTerm = searchTerm;

            //render the template with JsRender and the data
            var content = $("#twitterSearchPageTemplate").render(data);

            //The rendered content is a full jQuery Mobile
            //page with a unique ID.  Append it directly to the
            //body element
```

```
            $(document.body).append(content);

            //switch to the newly injected page
            $.mobile.changePage("#"+data.pageId);
          }
        })
        .error(function(jqXHR, textStatus, errorThrown){
          alert(textStatus+" - "+errorThrown);
        });
      });
    </script>
```

Following are the two JsRender templates:

```
    <script id="twitterTendingTemplate" type="text/x-jsrender">
      <li class="trendingItem">
        <a href="javascript://" class="twitterSearch" data-
search="{{>name}}">
          <h3>{{>name}}</h3>
        </a>
      </li>
    </script>

    <script id="twitterSearchPageTemplate" type="text/x-jsrender">
      <div id="{{>pageId}}" data-role="page" data-add-back-btn="true">
        <div data-role="header">
          <h1>{{>searchTerm}}</h1>
        </div>
        <div data-role="content">
          <ul id="results" data-role="listview" data-dividertheme="b">
            {{for results}}
              <li class="twitterItem">
              <a href="http://twitter.com/{{>from_user}}">
                <img src="{{>profile_image_url}}" alt="{{>from_user_
name}}" class="ui-shadow ui-corner-all" />
                <h3>{{>from_user_name}}
                  <span class="handle">
                    (@{{>from_user}})<br/>
                    {{>location}}
                    {{if geo}}
                      {{>geo}}
                    {{/if}}
                  </span>
                </h3>
                <p>{{>text}}</p>
```

```
            </a>
          </li>
        {{/for}}
      </ul>
    </div>
  </div>
</script>
</body>
</html>
```

OK, that was a lot of code to throw at you all at once, but most of it should look pretty familiar at this point. Let's start with explaining some of the newest stuff.

Normally, to pull data into a web page, you are subject to the same-domain policy even when you're pulling JSON. However, if it's coming from another domain, you'll need to bypass the same-domain policy. To bypass the some-domain policy, you could use some sort of server-side proxy such as PHP's **cURL** (`http://php.net/manual/en/book.curl.php`) or the Apache **HTTP Core Components** (`http://hc.apache.org/`) in the Java world.

Let's just keep things simple and use **JSONP** (also known as **JSON with Padding**). JSONP does not use a normal Ajax request to pull information. Despite the fact that the configuration options are for the `$.ajax` command, behind the scenes, it will execute the call for the data as a standalone script tag as follows:

```
 <script type="text/javascript" src=" https://
api.twitter.com/1/trends/daily.json?callback=jQue
ry17200315623809583485 1_1345608708562&_=1345608708657"></script>
```

It is worth noting that JSONP is called using a GET request. That means that you can't use it to pass sensitive data, because it would be instantly viewable through network traffic scanning or simply looking at a browser's request history. So, no logging in over JSONP or passing anything sensitive. Got it?!

Before the actual request is made, jQuery will create a semi-random function name that will be executed once the response is received from the server. By appending that function name as the callback within the URL, we are telling Twitter to wrap their response to us with this function call. So, instead of receiving JSON script like `{"trends": …}`, we have a script written to our page that starts as follows:

```
jQuery17200315623809583485 1_1345608708562({"trends": …}).
```

The reason this works is because the same-domain policy does not exist for scripts. Handy, yes? After the script is loaded and the callback has processed, we will have the data in JSON format. In the end, the execution under the sheets is vastly different, but the results are the same as you would get with regular `getJSON` requests from your own domain.

Here is a slice of the response back from Twitter:

```
jQuery17200264253814239054 92_1345774796764({
  "as_of": 1345774741,
  "trends": {
    "2012-08-23 05:20": [
       {
         "events": null,
         "name": "#ThingsISayTooMuch",
         "query": "#ThingsISayTooMuch",
         "promoted_content": null
       },
       {
         "events": null,
         "name": "#QuieroUnBesoDe",
         "query": "#QuieroUnBesoDe",
         "promoted_content": null
       },
       {
          "events": null,
          "name": "#ASongIKnowAllTheLyricsTo",
          "query": "#ASongIKnowAllTheLyricsTo",
          "promoted_content": null
       },
```

Next, we whittle down the response to only the part we want (the very latest set of trending topics) and pass that array into JsRender for … well… rendering. It may seem more simple to just loop through the JSON and use string concatenation to build your output but take a look at the following template and tell me that's not going to be a lot cleaner to read and maintain:

```
<script id="twitterTendingTemplate" type="text/x-jsrender">
  <li class="trendingItem">
    <a href="javascript://" class="twitterSearch" data-
search="{{>name}}">
      <h3>{{>name}}</h3>
    </a>
  </li>
</script>
```

The `text/x-jsrender` type on the script will ensure that the page does not try to parse the inner contents as JavaScript. Since we passed in an array to JsRender, the template will be written for every object in the array. Now that is simple! Granted, we're only pulling the name out of the data object, but you get the idea of how this works.

Let's look at the next significant block of JavaScript:

```
$(document).on('click', "a.twitterSearch", function(){
  //grab the search term off the link
  var searchTerm = $(this).attr("data-search");

  //do a Twitter search based on that term
  $.ajax({       url:"http://search.twitter.com/search.json?q=
"+escape(searchTerm),
   dataType:"jsonp",
   success: function(data){
     //create the pageID by stripping
     //all non-alphanumeric data
     var pageId = searchTerm.replace(/[^a-zA-Z0-9]+/g,"");
     //throw the pageId and original search term
     //into the data that we'll be sending to JSRenderdata.pageId =
pageId;
     data.searchTerm = searchTerm;

     //render the page and append it to the document body
$(document.body).append($("#twitterSearchPageTemplate")
       .render(data));

     //set the page to remove itself once left
     $("#"+pageId).attr( "data-" + $.mobile.ns
       + "external-page", true )
       .one( 'pagecreate', $.mobile._bindPageRemove );
     //switch to the new page
     $.mobile.changePage("#"+data.pageId);
    }
  })
  .error(function(jqXHR, textStatus, errorThrown){
    //If anything goes wrong, at least we'll know.
    alert(textStatus+" - "+errorThrown);
  });
});
```

First, we pull the search term from the attribute on the link itself. The search term itself is somewhat inappropriate as an `id` attribute for dynamically rendered pages; so, we'll strip out any spaces and non-alphanumeric content. We then append the `pageId` and `searchTerm` attribute to the JSON object we received back from Twitter. Following is a sample of returned data from this call:

```
jQuery17200264253814239054 92_1345774796765({
    "completed_in": 0.02,
    "max_id": 238829616129777665,
    "max_id_str": "238829616129777665",
```

```
    "next_page": "?page=2&max_id=238829616129777665&q=%23ThingsISay
TooMuch",
    "page": 1,
    "query": "%23ThingsISayToMuch",
    "refresh_url": "?since_id=238829616129777665&q=%23ThingsISay
TooMuch",
    "results": [
        {
            "created_at": "Fri, 24 Aug 2012 02:46:24 +0000",
            "from_user": "MichelleEspra",
            "from_user_id": 183194730,
            "from_user_id_str": "183194730",
            "from_user_name": "Michelle Espranita",
            "geo": null,
            "id": 238829583808483328,
            "id_str": "238829583808483328",
            "iso_language_code": "en",
            "metadata": {
                "result_type": "recent"
            },
            "profile_image_url": "http:\/\/a0.twimg.com\/profile_
images\/2315127236\/Photo_20on_202012-03-03_20at_2001.39_20_232_
normal.jpg",
            "profile_image_url_https": "https:\/\/si0.
twimg.com\/profile_images\/2315127236\/Photo_20on_202012-03-
03_20at_2001.39_20_232_normal.jpg",
            "source": "&lt;a href=&quot;http:\/\/twitter.
com\/&quot;&gt;web&lt;\/a&gt;",
            "text": "RT @MuchOfficial: @MichelleEspra I'd be the
aforementioned Much! #ThingsISayTooMuch",
            "to_user": null,
            "to_user_id": 0,
            "to_user_id_str": "0",
            "to_user_name": null,
            "in_reply_to_status_id": 238518389595840512,
            "in_reply_to_status_id_str": "238518389595840512"
        }

}
```

So, we'll take this response and pass it into the renderer to be transformed against `twitterSearchPageTemplate`:

```
<script id="twitterSearchPageTemplate" type="text/x-jsrender">
    <div id="{{>pageId}}" data-role="page" data-add-back-btn="true">
      <div data-role="header">
        <h1>{{>searchTerm}}</h1>
```

```
        </div>
        <div data-role="content">
          <ul id="results" data-role="listview" data-dividertheme="b">
            {{for results}}
              <li class="twitterItem">
              <a href="http://twitter.com/{{>from_user}}">
                <img src="{{>profile_image_url}}" alt="{{>from_user_
name}}" class="ui-shadow ui-corner-all" />
                <h3>{{>from_user_name}}
                  <span class="handle">
                    (@{{>from_user}})<br/>
                    {{>location}}
                      {{if geo}}
                        {{>geo}}
                      {{/if}}
                  </span>
                </h3>
                <p>{{>text}}</p>
              </a>
            </li>
          {{/for}}
        </ul>
      </div>
    </div>
  </script>
```

These are simple implementations. The examples on GitHub show many more options that are worth exploring. Check out `http://borismoore.github.com/jsrender/demos/` for details on creating more complex templates. This is a rapidly changing library (most client-side templating libraries are). So don't be surprised if, by the time you read this, there are a lot more options and slightly changed syntax.

Once we have the results of the transformation, we'll be ready to append the new page's source to the document's body and then programmatically change to this new page.

Programmatically changing pages

There are two ways to programmatically change pages in jQuery Mobile, and the differences are subtle:

- Call `$.mobile.changePage` and pass in a selector to the ID of the page you want to go to. This works the same way with URLs. Either way will yield the same results as if the user had clicked on a link. The page is inserted into the browser's history as one might expect. Following is the example code:

  ```
  $.mobile.changePage("#"+data.pageId);
  ```

- Create a jQuery object by selecting the page you want to change to first. Then, pass that jQuery object into the `$.mobile.changePage` function. The result is that the page is shown but the URL never updates, and, thus, it does not exist in the browser's history. This might be useful in situations where, if the user refreshes the page, you would want them to start the process over at the first screen. It prevents deep linking through bookmarks into other pages in a multipage layout. Following is an example:

  ```
  var $newPage = $("#"+data.pageId);
  $.mobile.changePage($newPage);
  ```

Generated pages and DOM weight management

In the normal course of events while surfing traditional mobile sites, jQuery Mobile will mark each page as `external-page`, which will cause the page to be removed from the DOM once the user navigates away from that page. The idea behind this is that it will manage DOM weight because "budget" (crappy) devices may not have as much memory to dedicate to their browsers. External pages will likely still be in the device cache for quick recall. So reloading them should be lightning fast. If you want to learn more about how jQuery Mobile handles this behavior, check out `http://jquerymobile.com/demos/1.3.0/docs/pages/page-cache.html`.

jQuery Mobile has done a great job at managing DOM weight through normal means. However, when we dynamically create pages, they are not automatically deleted from the DOM on exit. This can become especially overwhelming if there are a lot of them. We could easily overwhelm the miserable browsers on dumb phones and even some of the early-model or budget smartphones. If a dynamically-created page is likely to be viewed again within a session, then it may well be worth leaving in the DOM. However, since we're generating it in the browser to begin with, it's probably safer and faster to just re-render the page.

You can mark a page for deletion using this line of code *after* the page has been rendered but *before* the page is initialized:

```
$("#"+pageId).attr( "data-" + $.mobile.ns + "external-page", true
).one( 'pagecreate', $.mobile._bindPageRemove );
```

WARNING: This line of code comes almost verbatim from the library code itself. This is how they do it behind the scenes. Please note that `$.mobile._bindPageRemove` begins with an underscore. We are not dealing with a public method here.

This particular code is an undocumented and unofficial part of the API, which means that it could be changed on any given release. As central as this is to the framework, I doubt they'll change it; however, anytime you start introducing code that relies on continued presence of non-public APIs, you run the risk of an upgrade breaking your code without any warning in the release notes. Use freely, but thoroughly test each library upgrade.

Leveraging RSS feeds

What can I say? My editors made me do it. I hadn't initially planned on building anything around RSS. I'm glad they did because after looking around, there's a lot more information out there being fed by RSS than by JSON feeds. I figured the digital world had advanced a little more than it really had. So, Usha, thank you for making me include this.

First things first, if we don't use a server-side proxy, we will crash right into the unforgiving wall of the same-original policy. Examples include cURL in PHP systems, Apache HTTP Core Components in Java, or something like HttpWebRequest on .Net.

Following is the page I created in PHP to leverage cURL to grab the Ars Technica feed. The source for this file is in `ars.php` in the chapter code bundle.

```
<?PHP

//based on original example from…
//http://www.jonasjohn.de/snippets/php/curl-example.htm

//is cURL installed yet?
if (!function_exists('curl_init')){
  die('Sorry cURL is not installed!');
}
```

```
// OK cool. Then, let's create a new cURL resource handle
$ch = curl_init();

// Now set some options (most are optional)
// Set URL to download
curl_setopt($ch, CURLOPT_URL, "http://feeds.arstechnica.com/
arstechnica/index?format=xml");

// Set a referer
curl_setopt($ch, CURLOPT_REFERER, "http://bookexample/chapter5");

// User agent
curl_setopt($ch, CURLOPT_USERAGENT, "BookExampleCurl/1.0");

// Include header in result? (0 = yes, 1 = no)
curl_setopt($ch, CURLOPT_HEADER, 0);

// Should cURL return or print out the data?
// (true = return, false = print)
curl_setopt($ch, CURLOPT_RETURNTRANSFER, true);

// Timeout in seconds
curl_setopt($ch, CURLOPT_TIMEOUT, 10);

// Download the given URL, and return output
$output = curl_exec($ch);

// Close the cURL resource, and free system resources curl_close($ch);

echo $output;
?>
```

WARNING: cURL and other server-side proxy libraries are very powerful and, thus, very dangerous tools. Do *not* parameterize the URL that you intend to hit with this page. Hard code the URL. If you must take a parameter from the calling URL to build your destination, then *you must escape all parameters*. If you do not, you can rest assured that someday a hacker is going to have a lot of fun with your site with cross-site scripting (`https://www.owasp.org/index.php/Cross-site_Scripting_(XSS)`).

Next, let's add some buttons to the top. One for our Twitter feed, and one for Ars Technica. The final source for this next part will be in the file `index.html` in the code bundle for the chapter:

```
<div data-role="header">
  <h1>News</h1>
</div>
<div data-role="footer">
  <div data-role="navbar">
    <ul>
      <li><a id="twitter" href="#" class="ui-btn-active">Twitter</a></
li>
      <li><a id="ars" href="#">Feed</a></li>
    </ul>
  </div>
</div>
<div data-role="content">
  <ul id="results" data-role="listview" data-dividertheme="b"></ul>
</div>
```

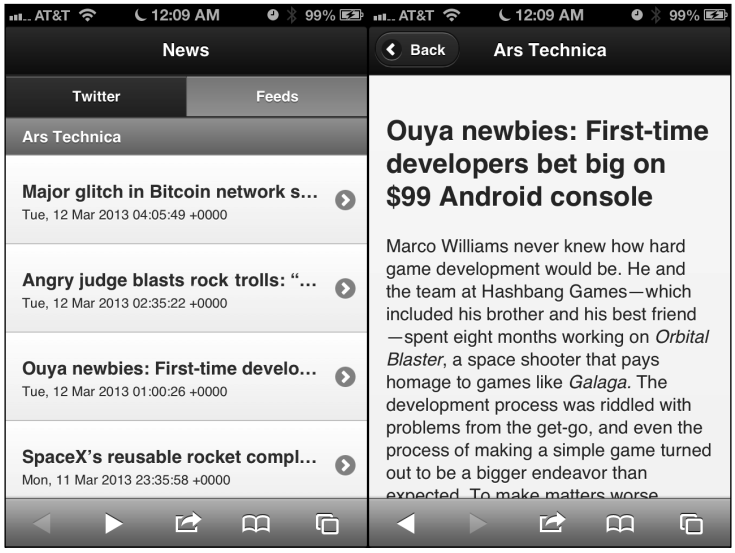
Next, let's add to our scripts to load the feed:

```
function loadArs(){
  //scroll back up to the top
  $.mobile.silentScroll(0);

  //Go get the Ars Technica feed content
  $.ajax({
    url:"ars.php",
    dataType:"xml",
    success: function(data, textStatus, jqXHR) {

      //Store the response for later use
      localStorage.setItem("ars", jqXHR.responseText);
      //prepare the content for use
      var $feed = $(data);

      //prepare a list divider with the title of the feed.
         var listView = "<li data-role='list-divider'>"+$feed.
find("channel>title").text()+"</li>";
     //loop through every feed item and
     //create a listview element.
      $feed.find("channel>item").each(function(index){
var $item = $(this);
        listView += "<li><a href='javascript://' "
```

```
            +"data-storyIndex='"+index
            +"' class='arsFeed'><h3>"
            +$item.find("title").text()
            +"</h3><p>"+$item.find("pubDate").text()
            +"</p></a></li>";
        });

        //put the new listview in the main display
        $("#results").html(listView);

        //refresh the listview so it looks right
        $("#results").listview("refresh");

       //place hooks on the newly created links
        //so they trigger the display of the
        //story when clicked
        $("#results a.arsFeed").click(function(){

          //get the feed content back out of storage
var arsData = localStorage.getItem("ars");
          //figure out which story was clicked and
          //pull that story's content from the item
var storyIndex = $(this).attr("data-storyIndex");
          var $item =
            $(arsData).find("channel>item:eq("+storyIndex+")");
          //create a new page with the story content
var storyPage = "<div id='ars"+storyIndex+"' "
            +"data-role='page' data-add-back-btn='true'>"
            +"<div data-role='header'><h1>Ars Technica</h1>"
            +"</div><div data-role='content'><h2>"
            +$item.find('title').text()+"</h2>"
            +$item.find('content\\:encoded').html()
            +"</div></div>";

          //append the story page to the body
          $("body").append(storyPage);
          //find all the images in the newly
          //created page.
          $("#ars"+storyIndex+" img").each(function(index, element) {
            var $img = $(element);
            //figure out its currentWidth
            var currentWidth = Number($img.attr("width"));
            //if it has a width and it's large
            if(!isNaN(currentWidth) && currentWidth > 300){
```

```
            //remove the explicit width and height
$img.removeAttr("width").removeAttr("height");
            //make the image scale to the width
//of it's container but never to be
//larger than its original size
            $img.css({"max-width":currentWidth
              +"px","width":"100%"});
          }
        });

        //switch to the new page
        $.mobile.changePage("#ars"+storyIndex);
      });
    }
  });
}

$("#ars").click(loadArs);
```

Here's what our new feed reader looks like!



Forcing responsive images

When you're importing from a page where you have no control over the images embedded in the content, you may have to tweak them to get it to look right in mobile. As in the previous example, I've found it's best to remove the explicit width and heights on the image itself and use CSS to make it fill 100 percent of its current container. Then, use a CSS `max-width` property to ensure the image is never scaled beyond it's original intended sizes.

While not truly being responsive in terms of loading a different size of the image that is appropriate for the resolution based on media queries, we've accomplished the same visible effect with the limited resources at our disposal for cases like this.

HTML5 Web Storage

HTML5 Web Storage is ridiculously simple if you haven't messed with it already. If you have, skip to the next paragraph. There are really only two forms of web storage: `localStorage`, and `sessionStorage`. `localStorage` will keep the information indefinitely. `sessionStorage` will store only for the length of a single session. It's a simple key/value paired system. Everything is string-based. So you'll need to convert the values to other formats as needed, once you've extracted them back out of storage. Check out `http://www.w3schools.com/html5/html5_webstorage.asp` for more information.

Now, this gets interesting with the definition of session. *Do not confuse* the session on your server with the browser session. The user session on your server might be set to expire within 20 minutes or so. However, just because your server session has expired, doesn't mean that your browser knows anything about that. *HTML5 session storage will persist until the browser is actually closed.*

This gets especially tricky on mobile browsers. In both Android and iOS, when you switch tasks or press the home button, the browser doesn't actually close. In both cases, you have to actually use the task killer functions to completely close the browsers. This is something that the end user might not actually do on their own.

But what's the big deal about web storage? Why not just use cookies to store information on the client? After all, it will work with everyone, right? Yes, cookies will work for everyone. However, they were never meant to store massive amounts of data like we're using in this example, and there is a soft limit to the number of cookies you can even store per domain (anywhere from 20-50 depending on the browser). The worst part about trying to use cookies for client-side storage is that they are sent back to the server as part of the request *for every single asset served from that domain*. That means that every CSS, JS, image, and page/Ajax request will carry every cookie with its payload. You can see how this could quickly start to degrade your performance. Adding one cookie could result in that data's transmission many times just to render a single page.

Browser-based databases (a work in progress)

Browser-based databases are in a state of extreme flux right now. There are actually two different standards available at the moment. The first is **Web SQL Database** (`http://www.w3.org/TR/webdatabase/`). You could use it, but, according to the W3C, this spec is no longer active. Many browsers have implemented Web SQL Database, but how long will it be around?

The W3C has, instead, stated that the direction for database on the browser will be **Indexed Database** (`http://www.w3.org/TR/IndexedDB/`). The working draft has editors from Microsoft, Google, and Mozilla; so, we can expect broad support in the future. The problem here is that the working draft was published May 24, 2012. As of the time of writing this chapter, only Firefox, Chrome, and Internet Explorer 10 are supporting IndexedDB (`http://en.wikipedia.org/wiki/Indexed_Database_API`).

JSON to the rescue

For now, we find ourselves in a terrible position of either using a doomed database or waiting for everyone to catch up with the new spec. Web Storage looks like the only safe bet in the near future. So, how can we best leverage that? With JSON, of course! All major browsers support JSON natively.

Think about the way we've always had to deal with relational databases in the past. As object-oriented programmers, we've always done our query and then taken the results data and turned it into an object in memory. We can do almost the exact same thing by simply storing JSON directly to a key in Web Storage by using the `JSON.stringify` method.

Here is an example to test if your system natively supports JSON. The source is `jsonTest.html` in the chapter code bundle:

```
<!DOCTYPE html>
<html>
<head>
  <title>JSON Test</title>
</head>
<body>
<script type="text/javascript">

  var myFeedList = {
    "lastUpdated":"whenever",
    "feeds":[
    {
       "name":"ars",
    "url":"http://feeds.arstechnica.com/arstechnica/
index?format=xml"
    },
    {
      "name":"rbds",
      "url":"http://roughlybrilliant.com/rss.xml"
    }
    ]
  }

myFeedList.lastUpdated = new Date();

localStorage.feedList = JSON.stringify(myFeedList);

var myFeedListRetrieved = JSON.parse(localStorage.feedList);
alert(myFeedListRetrieved.lastUpdated);
</script>
</body>
</html>
```

If all is well, you'll see an alert containing a timestamp.

If, for some reason, you find yourself in the unlucky position of having to support some massively out-of-date system (Windows Phone 7 and BlackBerry 5 or 6, I'm looking at you), go get `json2.js` from `https://github.com/douglascrockford/JSON-js` and include it with your other scripts. Then, you'll be able to stringify and parse JSON.

Leveraging the Google Feeds API

So, we've seen how to natively pull in a normal RSS feed, parse, and build out the pages using normal, tedious string concatenation. Now, let's consider an alternative that I had no idea even existed when I first started writing this chapter. Thanks go to Raymond Camden and Andy Matthews for pointing this out in their book, *jQuery Mobile Web Development Essentials*. You need to follow those two on Twitter at `@cfjedimaster` and `@commadelimited`.

The Google Feeds API can be fed several options, but at its core, it's a way to specify an RSS or ATOM feed and get back a JSON representation. Naturally, this opens up a few more interesting doors in this chapter. If we can now pull in multiple feeds of different types without having to have any kind of server-side proxy, we can greatly simplify our lives. Client-side templates are back in the picture! No more string concatenation! Since they're all in a unified format (including the publish date), we can pull them all together into one master view with all feed stories sorted by date.

Sorting objects by their properties is actually pretty simple. You just have to pass a function to do the comparison. The following code is what we'll use for the date:

```
function compareDates(a,b) {
  var aPubDate = Date.parse(a.publishedDate);
  var bPubDate = Date.parse(b.publishedDate);
  if ( aPubDate < bPubDate) return 1;
  if (aPubDate > bPubDate)  return -1;
  return 0;
}
```

Now, let's specify a JSON object to store the feeds we want to use:

```
var allFeeds = {

  //all the feeds we want to pull in
  "sources":[
"http://feeds.arstechnica.com/arstechnica/index?format=xml",
"http://rss.slashdot.org/Slashdot/slashdot",
"http://www.theregister.co.uk/headlines.atom"
],

  //How many of the feeds have responded?  Once all have
  //responded, we'll finish our processing.
  "sourcesReporting":0,

  //This is where we will store the returned stories.
  "entries":[]
};
```

Next we'll use our processor function to handle the stories as they come in:

```
function assimilateFeed(data){

  //Mark another feed as having reported back
  allFeeds.sourcesReporting++;

  //Grab the title of this feed
  var feedTitle = data.responseData.feed.title;

  //Loop through every entry returned and add the feed title
  //as the source for the story
  for(x = 0; x < data.responseData.feed.entries.length;
    data.responseData.feed.entries[x++].source=feedTitle);

  //Join this field's entries with whatever entries might have
  //already been loaded
  allFeeds.entries = allFeeds.entries.concat(data.responseData.feed.
entries);

  //If all the feeds have reported back, it's time to process
  if(allFeeds.sourcesReporting == allFeeds.sources.length){

    //Sort all the stories by date
    allFeeds.entries.sort(compareDates);

   //Take the results that have now all been combined and
    //sorted by date and use jsRender
    $("#results").html($("#googleFeedTemplate")
      .render(allFeeds)).listview("refresh");
  }
}
```

Here's our JsRender template:

```
<script type="text/x-jsrender" id="googleFeedTemplate">
  {{for entries}}
    <li>
      <a href="{{:link}}" target="_blank">
        <h3>{{:title}}</h3>
        <p><strong>{{:source}}</strong> - {{:publishedDate}}
          <br/>{{:contentSnippet}}
        </p>
      </a>
    </li>
  {{/for}}
</script>
```

And finally, here is the function that will kick off the whole thing:

```
$("#feeds").click( function() {

  //Reset the number of received feeds
  allFeeds.sourcesReporting = 0;

  //Get back to the top of the page
  $.mobile.silentScroll(0);

  //Loop through all the feeds
  for(var x = 0; x < allFeeds.sources.length; x++){
    $.ajax({

//Call to Google's Feed API with the URL encoded
url:"https://ajax.googleapis.com/ajax/services/feed/load?v=1.0&output=
json&q="+escape(allFeeds.sources[x]),
      dataType:"jsonp",
      success:assimilateFeed
    });
  }
});
```

I've included this as part of my functional example in the `challenge.html` file, but the source goes much deeper than that. The source of `challenge.html` has several hidden gems for you to find as well. I tossed in Reddit, Flickr, and a local search of Twitter while I was at it.

Summary

You have been presented with a very wide array of choices for client-side templating. At this point, you now know how to leverage JSON and JSONP and combine them effectively to create pages on the fly. RSS should present no real challenge to you at this point either, since you can do it either natively or using Google Feeds.

In the next chapter, we'll combine some of these techniques as we continue to build our technical tool chest and turn our eyes to HTML5 Audio.

Where to buy this book

You can buy Creating Mobile Apps with jQuery Mobile from the Packt Publishing website: `http://www.packtpub.com/creating-mobile-apps-with-jquery/book`.

Free shipping to the US, UK, Europe and selected Asian countries. For more information, please read our shipping policy.

Alternatively, you can buy the book from Amazon, BN.com, Computer Manuals and most internet book retailers.

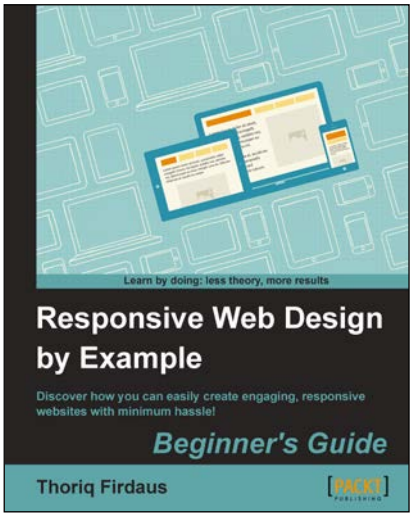
www.PacktPub.com

For More Information:
www.packtpub.com/creating-mobile-apps-with-jquery/book

[PACKT] PUBLISHING

Responsive Web Design by Example Beginner's Guide

Thoriq Firdaus



Chapter No. 2

"Constructing a Responsive Portfolio Page with Skeleton"

In this package, you will find:

A Biography of the author of the book

A preview chapter from the book, Chapter NO.2 "Constructing a Responsive Portfolio Page with Skeleton"

A synopsis of the book’s content

Information on where to buy this book

About the Author

Thoriq Firdaus is a graphic and web designer living in Indonesia. He has been working in web designing projects with several clients from startup to notable companies and organizations worldwide for over five years.

He is very passionate on HTML5 and CSS3 and writes on these subjects regularly at `http://www.hongkiat.com/` and at his own blog `http://creatiface.com/`. Occasionally, he also gives presentations on web design at some local colleges and institutions.

Outside of work, he enjoys watching movies with his family and trying out some good food in a new cafe or restaurant nearby.

First, I would like to thank the team at Packt Publishing for giving me a chance to write this book and also to the editors and reviewers for their help on improving this book with their valuable feedback and comments.

I also thank my friends Arief Bahari (`www.ariefbahari.com`) and Ferina Berliani (`http://nantokaa.tumblr.com/`) for allowing me to use their artwork for this book.

Lastly, I thank my family, especially my wife and daughter, for giving me support during the process of writing this book.

Responsive Web Design by Example Beginner's Guide

Responsive web design is one of the most discussed topics on web, and a very demanding feature for today's websites. It lets the website to adapt in difference viewport sizes nicely. But, if you think that building a responsive website is hard, wait until you have finished this book.

It will also show you how to use some development tools that allow you to build responsive websites faster, more efficiently with lesser number of hurdles.

What This Book Covers

Chapter 1, Responsive Web Design, explains the basics of responsive web design, explores the development tools to build it, and highlights some good examples of a responsive website.

Chapter 2, Constructing a Responsive Portfolio Page with Skeleton, introduces Skeleton, discusses how to use its responsive grid, and starts the first project by constructing the webpage with HTML5.

Chapter 3, Enhancing the Portfolio Website with CSS3, introduces some additional features in CSS3 like Transforms and Transitions, and discusses how to incorporate them to enhance our responsive portfolio website.

Chapter 4, Developing a Product Launch Site with Bootstrap, introduces Bootstrap framework, and explores some of its components to build responsive websites.

Chapter 5, Enhancing the Product Launch Site with CSS3 and LESS, explains several LESS functions to author CSS3, and discusses how to use them to make our responsive Product Launch site look stunning, yet also maintainable. In this chapter, we also test our website to see how it looks in several difference viewport sizes.

Chapter 6, A Responsive Website for Business with Foundation Framework, introduces Foundation framework, and walks through the key features. We also start the third project to build responsive website for business purposes.

Chapter 7, Extending Foundation, explores the Sass CSS preprocessors, SCSS and Compass, and discusses how to extend the website appearance by configuring several Foundation framework variables.

2
Constructing a Responsive Portfolio Page with Skeleton

In our previous chapter, we discussed responsive web design and had a first look at the frameworks that make it possible for us to create a responsive website more quickly.

In this chapter, we will create a simple responsive portfolio website with Skeleton. So, if you are a creative person who wants to showcase your own work on your own website, this could be a perfect chapter to work through.

To sum it up, here is what we will focus on in this chapter:

- Digging into the Skeleton components
- Utilizing the Skeleton components
- Setting up a project with Skeleton
- Preparing the project assets
- Constructing a website with HTML5

So, let's get started.

Getting started with Skeleton

As mentioned in the previous chapter, one of the disadvantages of using a framework is the learning curve; we need to spend some time to learn how to use the framework, particularly if this is the first time using it. So, before we build our responsive portfolio website with Skeleton, it is a good idea to unpack and take a look at what is included in Skeleton.

Time for action – creating a working directory and getting Skeleton

Perform the following steps for creating a working directory and getting Skeleton:

1. First, create a folder named `portfolio`. This should be our working directory for the responsive portfolio website.
2. Under this `portfolio` folder, create two folders named `html` and `psd`.
3. Now it is time to get Skeleton. So, let's go to the Skeleton website (`www.getskeleton.com`).
4. Go to the **Download** section and download the Skeleton package. At the time of writing, the latest version of Skeleton is Version 1.2.
5. Save the downloaded file in the `html` folder.
6. This downloaded file is in the `tar.gz` format. Let's extract it to retrieve the files inside the downloaded file.
7. After extracting, you should find two new folders named `stylesheet` and `images`, and an HTML document named `index.html`. This is optional, but we can now safely remove the `.tar.gz` file.
8. Lastly, from the **Download** section on `www.getskeleton.com`, download the Skeleton PSD template, save it in the `psd` folder, and unpack it.

What just happened?

We have just created a working directory. We have also downloaded the Skeleton package as well as the PSD template, and placed it in the appropriate folder to work on this project.

What is included in Skeleton?

Compared to other frameworks that we have mentioned in this book, Skeleton is the simplest. It is not overstuffed with heavy styles or additional components, such as jQuery plugins, which we may not need for the website. Skeleton comes only with an `index.html` file, a few stylesheets containing the style rules, a few images, and a PSD template. Let's have a look at each of these.

Starter HTML document

Skeleton comes with a starter HTML template named `index.html`, so we don't have to worry about writing the basic HTML document. The author of Skeleton has added the essential elements in this template, including the parts discussed in the following sections.

The viewport meta tag

The viewport meta tag in this HTML starter template is set to `1` for both `initial-scale` and `maximum-scale`, as shown in the following code snippet:

```
<meta name="viewport" content="width=device-width, initial-scale=1,
maximum-scale=1">
```

As we mentioned in the first chapter, setting `initial-scale` to `1` will set the web page to be 100 percent of the viewport size, when we open the web page for the first time.

However, one thing that should be noted when setting `maximum-scale` to `1` is that it will prevent the zooming ability. Thus, it is suggested to ensure that the users, later on, can clearly see the content, text, or images, without zooming the web page.

HTML5 Shim

Since we will be using the HTML5 elements in our document, we need to include the HTML5 Shim JavaScript Library so that Internet Explorer 8 and its earlier versions recognize the new elements from HTML5.

HTML5 Shim, by default, has also been included in the Skeleton starter HTML document; you should find the following line inside the `<head>` section:

```
<!--[if lt IE 9]>
<script src="http://html5shim.googlecode.com/svn/trunk/html5.js"></
script>
<![endif]-->
```

The preceding HTML5 Shim script is wrapped within the conditional comment tag that is designated for Internet Explorer. The comment `<!--[if lt IE 9]>` stated "if less than Internet Explorer 9", which means the script within will only apply to Internet Explorer 8 and its earlier versions where new HTML5 elements are not recognized. Other browsers will simply ignore this comment tag.

You can read a post by Paul Irish (`http://paulirish.com/2011/the-history-of-the-html5-shiv/`) for the history behind HTML5 Shim and about how it was invented and developed.

Responsive Grid

Skeleton is equipped with Responsive Grid to quickly build responsive layout. The Skeleton's grid system is 960 px wide and is made up from sixteen columns of grid that are defined in a very logical naming system.

The columns are defined with the `.columns` class coupled with the respective column numbers `.one`, `.two`, `.three`, `.four`, and so on, to define the column width. These classes can be found in the `skeleton.css` file. The following code snippet shows the definitions of the column numbers and column width in the stylesheet:

```
.container .one.column,
.container .one.columns { width: 40px; }
.container .two.columns { width: 100px; }
.container .three.columns { width: 160px; }
.container .four.columns { width: 220px; }
.container .five.columns { width: 280px; }
.container .six.columns { width: 340px; }
.container .seven.columns { width: 400px; }
.container .eight.columns { width: 460px; }
.container .nine.columns { width: 520px; }
.container .ten.columns { width: 580px; }
.container .eleven.columns { width: 640px; }
.container .twelve.columns { width: 700px; }
.container .thirteen.columns { width: 760px; }
.container .fourteen.columns { width: 820px; }
.container .fifteen.columns { width: 880px; }
.container .sixteen.columns { width: 940px; }
```

If you are not familiar with this practice or you don't know how it works, take a look at the following example.

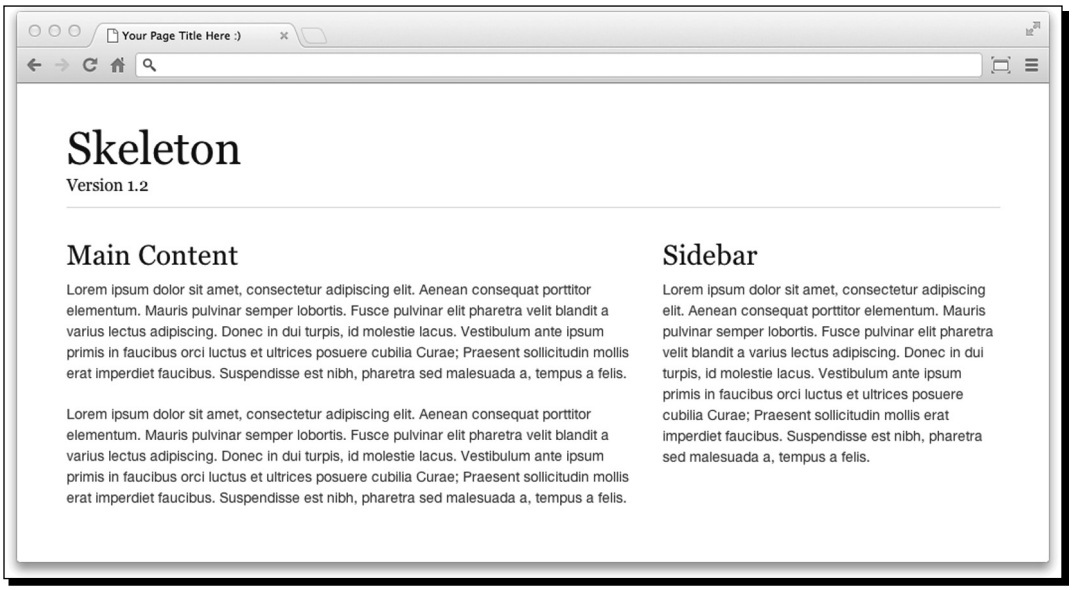
In this example, we have three `div` elements; one of those is for the container. Inside this container, we will have a `div` element to contain a main area and an `aside` element to contain the sidebar area. The following code snippet shows how our markup looks in the code editor:

```
<div>
  <div>
    <h3>Main Content</h3>
    <p>Lorem ipsum dolor sit amet, consectetur adipiscing elit.
    Aenean consequat porttitor elementum. Mauris pulvinar semper
    lobortis. […]</p>
  </div>
  <aside>
    <h3>Sidebar</h3>
    <p>Lorem ipsum dolor sit amet, consectetur adipiscing elit.
    Aenean consequat porttitor elementum. Mauris pulvinar semper
    lobortis.[…]</p>
  </aside>
</div>
```

Since all the styling rules for the columns are predefined, we simply need to add the appropriate classes into these elements, as follows:

```
<div class="container">
  <div class="ten columns">
    <h3>Main Content</h3>
    <p>Lorem ipsum dolor sit amet, consectetur adipiscing elit.
    Aenean consequat porttitor elementum. Mauris pulvinar semper
    lobortis. […] </p>
  </div>
  <aside class="six columns">
    <h3>Sidebar</h3>
    <p>Lorem ipsum dolor sit amet, consectetur adipiscing elit.
    Aenean consequat porttitor elementum. Mauris pulvinar semper
    lobortis. […]</p>
  </aside>
</div>
```

Then, if we view the document in the browser, we will see something as shown in the following screenshot:



Yes, it's that simple. But just to remember, in Skeleton, the columns should be nested inside an element with the `.container` class, otherwise the column styles will not be applied.

Clearing styles

The column's elements are defined by using the CSS float property definition, which causes the column's parent element to collapse. To solve it, Skeleton provides special classes; we can use either the `.row` class or the `.clearfix` class to clear things around the columns. The following code snippet shows the clearing styles' definitions, which can be found in `skeleton.css`:

```
.container:after { content: "\0020"; display: block; height: 0; clear:
both; visibility: hidden; }
.clearfix:before, .clearfix:after,
.row:before,
.row:after { content: '\0020'; display: block; overflow: hidden;
visibility: hidden; width: 0; height: 0; }
.row:after,
.clearfix:after { clear: both; }
.row, .clearfix { zoom: 1; }
.clear { clear: both; display: block; overflow: hidden; visibility:
hidden; width: 0; height: 0; }
```

On the Smashing Magazine website, Louis Lazaris has thoroughly discussed the CSS float property and how it affects the elements around it in the post available at `http://coding.smashingmagazine.com/2009/10/19/the-mystery-of-css-float-property/`.

Media queries

Skeleton has provided CSS3 media queries to apply specific style rules for standard viewport size and also making the grid responsive. For example, the following media query will specify the styles for 959 px viewport size and less:

```
@media only screen and (max-width: 959px) {
  …
}
```

Remember that Skeleton is a 960 grid-based framework, which means the maximum width of the web page would only be 960 px. So when the viewport is 959 px wide or less, in other words, smaller than the base size, the styles under this media query will be applied. The same idea also applies to the other defined media queries for example:

```
/* Tablet Portrait size to standard 960 (devices and browsers) */
@media only screen and (min-width: 768px) and (max-width: 959px) { }
/* All Mobile Sizes (devices and browser) */
@media only screen and (max-width: 767px) { }
/* Mobile Landscape Size to Tablet Portrait (devices and browsers) */
@media only screen and (min-width: 480px) and (max-width: 767px) { }
/* Mobile Portrait Size to Mobile Landscape Size (devices and
browsers) */
@media only screen and (max-width: 479px) { }
```

These media query definitions can be found in the `skeleton.css` and `layout.css` stylesheet.

Referring to our previous example, the web page is already responsive, as the column classes and the styles are predefined under the media queries in the `skeleton.css` stylesheet.

Thus, when we view it in a much smaller viewport with—in this example it as 320 px—we will get the result as shown in the following screenshot:

Typography styles

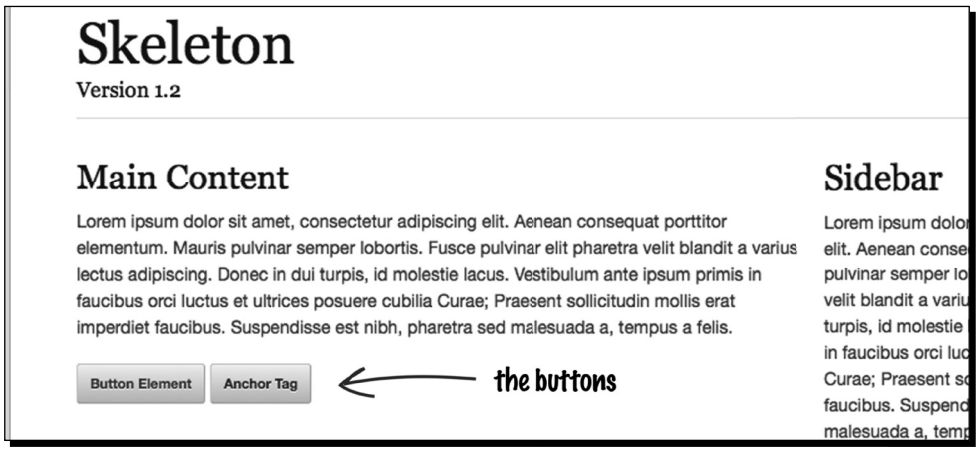
Typography styles have a key role in making a website readable. While the browsers have default styles for typography, Skeleton provides an improvement in this area for some elements, including headings, paragraphs, and pull-quotes. In Skeleton, these typography styles are available in the `base.css` stylesheet.

Button styles

Skeleton provides basic styles for buttons, which are applied by adding the `.button` class to some elements, such as the `<button>` or `<a>` elements, as shown in the following code snippet:

```
<button class="button" type="submit">Button Element</button>
<a href="#" class="button">Anchor Tag</a>
```

The result of the preceding code snippet is rendered, as shown in the following screenshot:

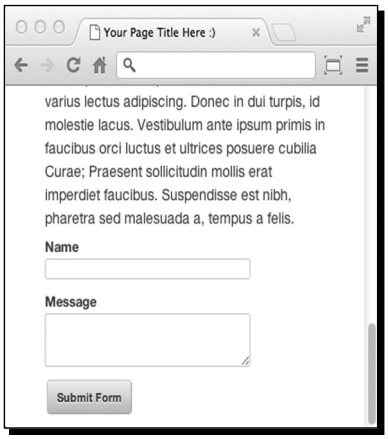


Form styles

Styling form elements can be complicated. But, Skeleton simplifies the process with its default styles. We simply need to structure the markup properly, without adding any special classes, as shown in the following code snippet:

```
<form>
  <label for="name">Name</label>
  <input type="text" id="name">
  <label for="message">Message</label>
  <textarea id="message"></textarea>
  <button type="submit">Submit Form</button>
</form>
```

In the browsers, we will get the result as shown in the following screenshot:



Apple icon devices

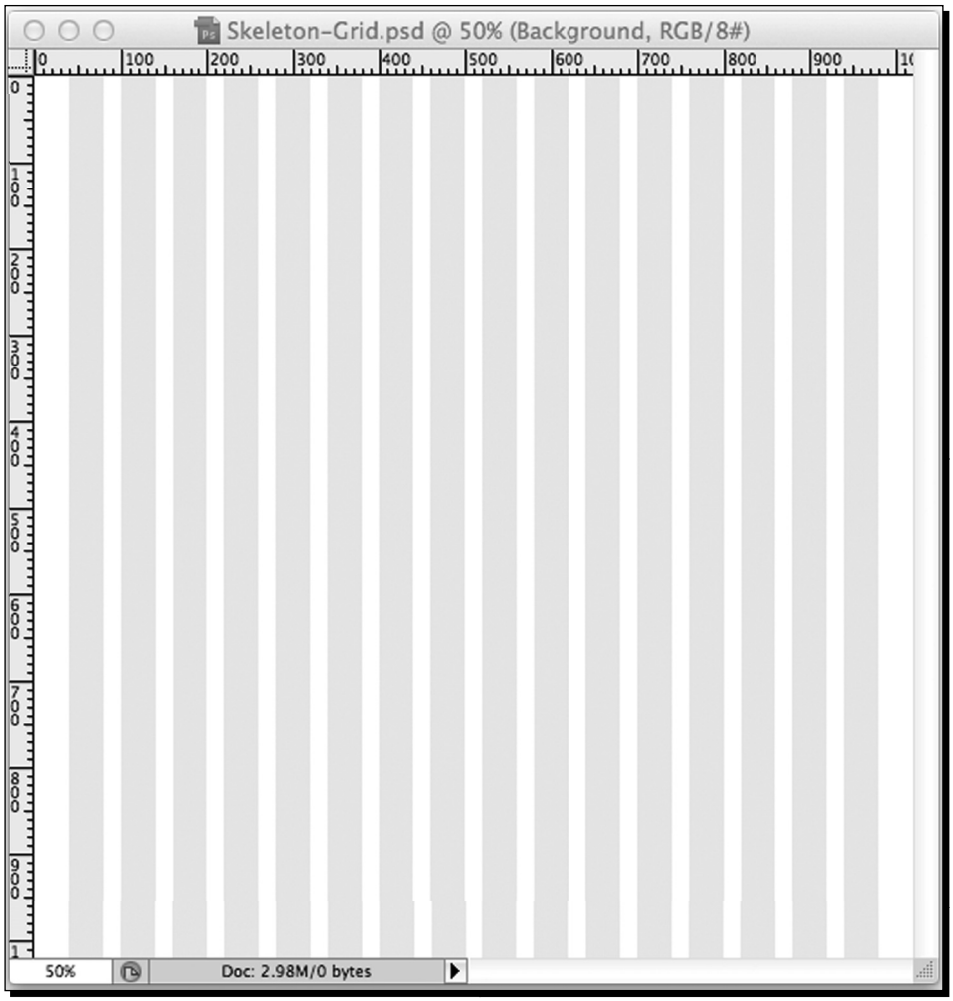
Skeleton comes with favicon and iOS icons, which we can easily replace with our own custom icons, if needed. The following screenshot shows these images in different sizes for different devices and resolutions:

The first one, which is the smallest, is the icon for iPhone. The second one, which is bigger than the first one, is to serve the iPad, while the biggest one will be displayed for Apple devices with higher resolution Retina Display.

You can read the documentation available at Apple Dev Center (`http://developer.apple.com/library/safari/#documentation/AppleApplications/Reference/SafariWebContent/ConfiguringWebApplications/ConfiguringWebApplications.html#//apple_ref/doc/uid/TP40002051-CH3-SW3`) for more details on the use of these icons.

Photoshop template

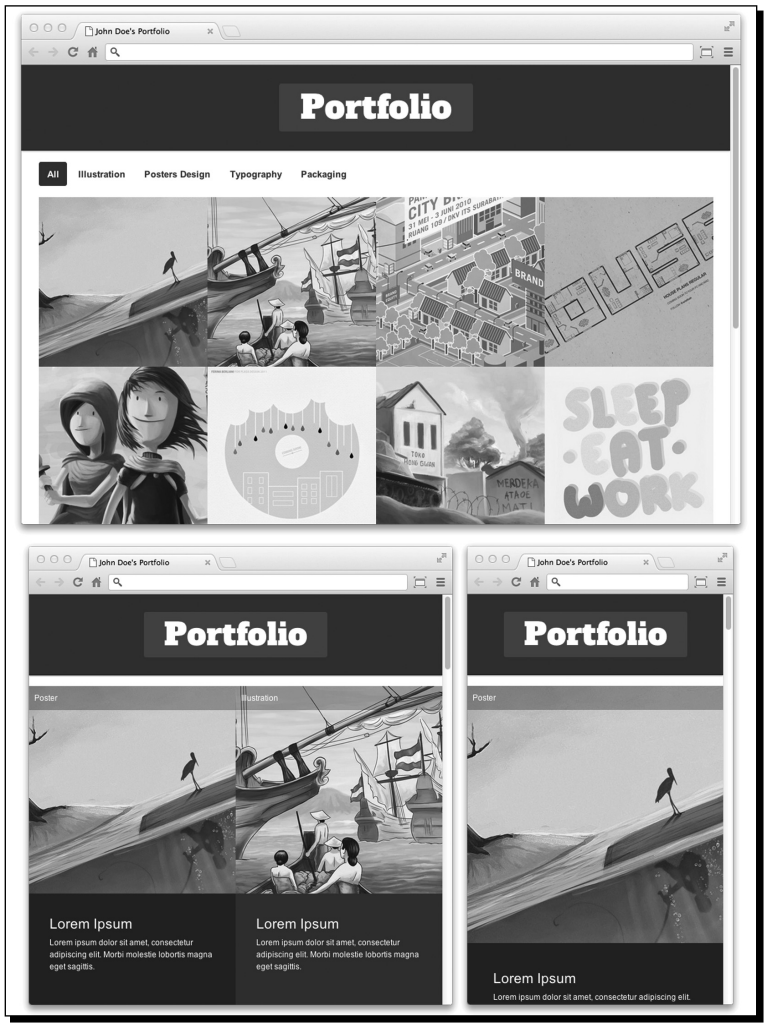
We have downloaded a PSD template earlier in this chapter. This template contains only one extra layer. **Layer** is a semi-transparent overlay showing the 16 columns of the grid, as shown in the following screenshot:



This grid is useful as a visual helper to design the website. So later on, when we translate the design into a web document, we will know the appropriate grid number for the translated elements.

How will the website look?

At this point, you may wonder how our first website will look. It will be really simple with only three sections: the header, the main content area that displays the portfolio, and the footer. The following screenshot shows three different views of the website with respect to the different viewport sizes:

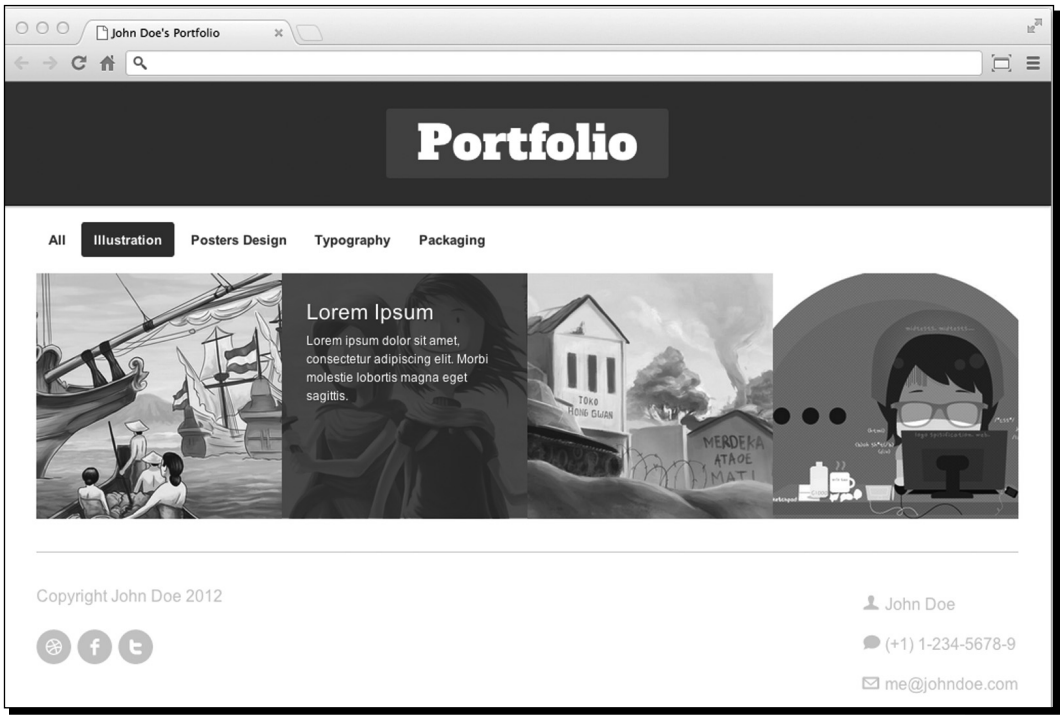


Website navigation

Our website's navigation will be somewhat unusual; rather than being used to move between pages, it will be used to sort the portfolio. We have several categories of portfolios: they are **Illustration**, **Poster Design**, **Typography**, and **Packaging**. The following screenshot shows the result of selecting the **Illustration** category:

Thumbnail hover effect

We will also add a fancy effect to make our website more attractive. When we hover over one of the portfolio thumbnails, the description of that portfolio will be revealed. The following screenshot shows this effect:



Setting up the Skeleton document

Now, it is time to set up the Skeleton document. It is important to note that when we are working on a framework, it is best not to alter the codes in the core files, which are the original files from the downloaded package. If we change these files, it may make our website less maintainable, and our changes may be overwritten if the framework is upgraded later. Thus we need to add a CSS file for our own.

Time for action – adding an extra CSS file

Perform the following steps for adding an extra CSS file:

1. Go to our working directory, `portfolio`.
2. Then go to the `stylesheets` folder and create a new file.

3. Rename this new file as `styles.css`.
4. Open the `index.html` file.
5. Add the following lines inside the `<head>` tag, right after the default Skeleton styles `base.css`, `skeleton.css`, and `layout.css`:

```
<link rel="stylesheet" href="stylesheets/base.css">
<link rel="stylesheet" href="stylesheets/skeleton.css">
<link rel="stylesheet" href="stylesheets/layout.css">
<link rel="stylesheet" href="stylesheets/styles.css">
```

What just happened?

We have just created a new stylesheet named `styles.css`, which we will be using for our own styles apart from the default Skeleton styles. Then, we called this stylesheet in our HTML document so that the styles within this stylesheet show their effect.

The reason we added this stylesheet after the other stylesheet links is because we want our styles to take place over the other style definitions.

You can read about CSS Specificity at `http://coding.smashingmagazine.com/2007/07/27/css-specificity-things-you-should-know/`.

Adding custom fonts

Earlier we were limited to fonts that were installed on a given user's machine, which meant that the only practical fonts were those with a broad installed base, such as Arial, Times, and Georgia. Today, we are able to embed font families for websites apart from the ones in the user's machine.

If you look at our design's header section, you can see that the main Porfolio heading uses an uncommon font—in this case, Alfa Slab One.

There are several options for embedding fonts. For this website we will use Google Web Fonts. In Google Web Fonts, we can find various font types that are allowed to be embedded on websites for free.

Time for action – embedding Google Web Fonts

Perform the following steps for embedding Google Web Fonts:

1. First, go to the Google Web Font website (`http://www.google.com/webfonts`).
2. Find a **Search** box and type `Alfa Slab One`; this is the name of the font that we are going to use for the website logo.

3. Click on the **Quick-use** link, as shown in the following screenshot:

This will direct you to a page that contains some additional information about this font, including how to embed it on a web page.

4. There are three ways to embed a Google font: using the standard way, using the `@import` rule, or using JavaScript.

For this website, we will use the standard way. So, let's copy the following line:

```
<link href='http://fonts.googleapis.com/css?family=Alfa+Slab+One'
rel='stylesheet' type='text/css'>
```

5. Open `index.html` and paste the preceding line inside the `<head>` section directly above the links to other stylesheets, as follows:

```
<link href='http://fonts.googleapis.com/css?family=Alfa+Slab+One'
rel='stylesheet' type='text/css'>
<link rel="stylesheet" href="stylesheets/base.css">
<link rel="stylesheet" href="stylesheets/skeleton.css">
<link rel="stylesheet" href="stylesheets/layout.css">
<link rel="stylesheet" href="stylesheets/styles.css">
```

What just happened?

We have just embedded a new font family in our HTML document from Google Web Fonts.

Alternatively, you can use the `@font-face` rule to embed the font. Font Squirrel provides a handy tool to generate the `@font-face` rule (`http://www.fontsquirrel.com/fontface/generator`). Before embedding the fonts, be sure you agree to the End-users License Agreement of the fonts.

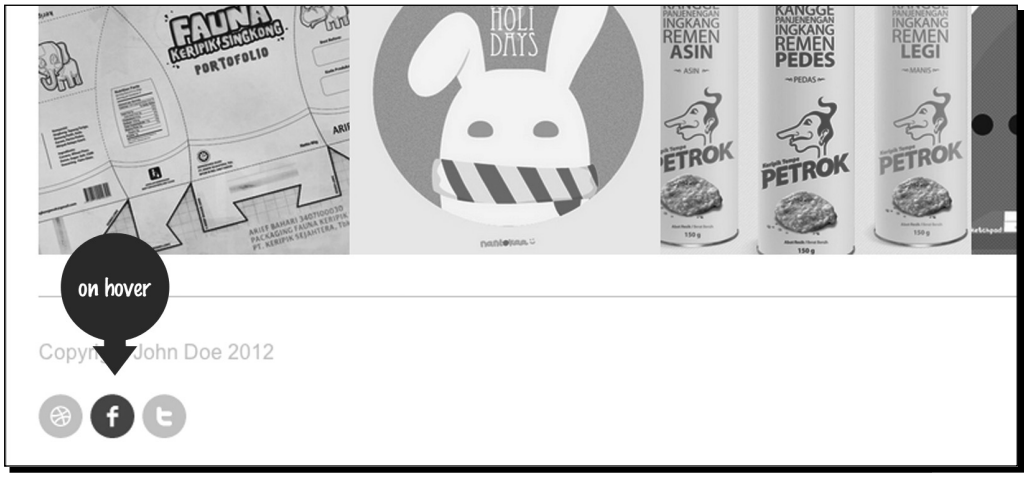
Preparing the images

Since we will be working on a portfolio website, we obviously need some portfolio images to display. I would like to thank two of my artist friends, Ferina Berliani (`http://nantokaa.tumblr.com/`) and Arif Bahari (`http://www.ariefbahari.com`) for letting me use their artwork, and the following images show some of their works that we will be using in this book.

You can use your own images as long as they are sized to a 480 px by 480 px square; you can either use Photoshop or any other image editor of your choice to do so. Then put your images inside the `images` folder under the working directory and name them using this convention: `image-1.jpg`, `image-2.jpg`, `image-3.jpg`, and so on. We have a total of 12 image thumbnails:

Social media icons

In addition, we will place three social media icons in our footer area: one each for Facebook, Twitter, and Dribbble, as shown in the following screenshot:



In the default state, the icons are displayed in grey and then when we hover over these icons, the platform's main brand color will be displayed, such as Facebook's blue and Dribbble's pink. These icons have been provided along with this book.

However, you can substitute with any social media icons that are available on the Internet for free. Just make sure that it is also available in 48 px by 48 px size. These social icons usually come separately. Thus, we will need to concatenate them into one sprite file.

Time for action – sprite images

In the following steps, we will turn these icons into sprite images with a free CSS Sprite Generator Tool (`http://spritegen.website-performance.org/`):

1. Given the icons proper names, such as `twitter.png` and `twitter-hover.png`, as shown in the following screenshot:

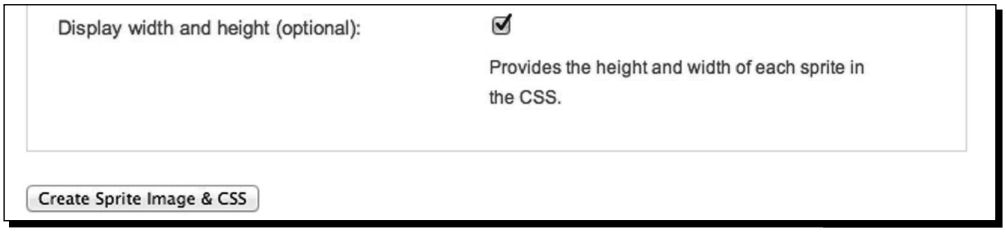
   This naming convention also applies to other icons. You don't have to limit yourself to our example; you can provide more than three icons. After all the images are prepared, add these icons to a ZIP file.

2. Go to the CSS Generator Tool website (`http://spritegen.website-performance.org/`).

3. Upload the ZIP file that we created in Step 2.

4. Under the **Sprite Output Options** section, enter `10` in the **Horizontal Offset** and **Vertical Offset** fields to set them to 10 px:

5. Then, click on the **Create Sprite Image & CSS** button, as shown in the following screenshot:



   This will generate the sprite image as well as the CSS rule to display it.

6. Download the image and save it under the `images` folder in our working directory.

7. Copy the CSS snippet into our `style.css` file. It should resemble the following code snippet:

```
.sprite-dribbble-hover{ background-position: 0 0; width: 48px;
height: 48px; }
.sprite-dribbble{ background-position: 0 -58px; width: 48px;
height: 48px; }
.sprite-facebook-hover{ background-position: 0 -116px; width:
48px; height: 48px; }
.sprite-facebook{ background-position: 0 -174px; width: 48px;
height: 48px; }
.sprite-twitter-hover{ background-position: 0 -232px; width: 48px;
height: 48px; }
.sprite-twitter{ background-position: 0 -290px; width: 48px;
height: 48px; }
```

What just happened?

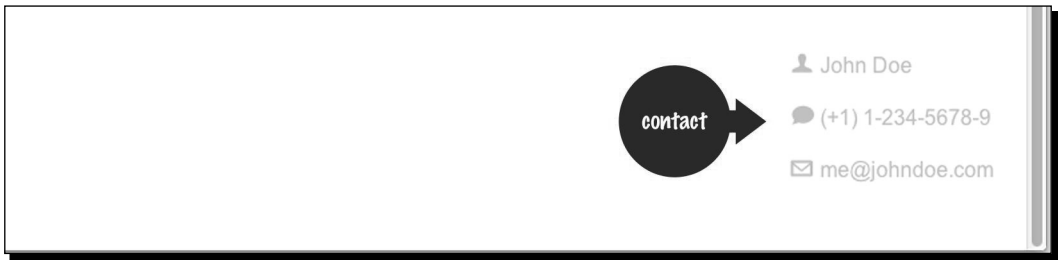
We concatenated the social media icons into one file. We will display these icons on our website using the CSS rule that we have generated. This practice is known as **CSS Sprite**.

Alternatively, you can also follow a screencast by Chris Coyier available at CSS Tricks to create a sprite image in Photoshop (`http://css-tricks.com/video-screencasts/43-how-to-use-css-sprites/`), and as an addition, you can also follow a screencast by Lynda on how to create a sprite grid to help you in positioning sprite images (`http://www.youtube.com/watch?v=Gq7XCMofxcQ`).

Or else, if you are not familiar with CSS Sprites, Dave Shea has discussed this method thoroughly at A List Apart (`http://www.alistapart.com/articles/sprites`).

Contact icons

Our footer will include contact information, such as name, phone number, and e-mail address, each with its own icon as illustrated in the following screenshot:



These icons have been provided along with the code files available with this book, but you can use other icons in 24 px by 24 px size, which are available on the Internet. Similarly, if the icons come separately, you need to concatenate them in one file and generate the CSS rules, as we have demonstrated in the preceding section.

HTML5 elements

HTML5 introduces many new elements and we will use some of them for this website, such as `<header>`, `<section>`, `<figure>`, `<figcaption>`, and `<footer>`.

Element	Discussion
`<header>`	This is used for defining the head of a section. The `<header>` element can be used for the website's header and also the head of other sections where it is reasonable to add it, such as the article's header.
`<footer>`	The `<footer>` element defines the end or the lowest part of a section. Like the `<header>` element, `<footer>` can also be used for the website's footer or the footer part of other sections.
`<section>`	`<section>` can somehow be confusing. But according to the specifications (`http://www.w3.org/html/wg/drafts/html/master/sections.html#the-section-element`), the `<section>` element represents a generic section of a document or application.
`<figure>`	The `<figure>` element is used to represent the document figure, such as an illustration or an image. It can be used with `<figcaption>` to add the caption, if needed.
`<figcaption>`	As mentioned, `<figcaption>` represents the caption of the document's figure. Thus, it should be used along with the `<figure>` element.

Now, let's add these elements to our document.

HTML5 custom data attributes

There are times when developers need to retrieve data within specific elements for further data processing. In the past, some developers used to rely on the `rel` or `class` attributes to store that data, but that way leads to breaking the validity of the document's structure.

To accommodate that situation, HTML5 introduced a new attribute called custom data attribute. We can use this attribute to embed custom data within an HTML element. This attribute is specified with `data-` and followed by the attribute name. For example, an online gaming website can list the top players and use data attributes to store their scores.

```
<ul id="top-players">
  <li class="player-name" data-score="98.9">John Doe</li>
  <li class="player-name" data-score="80.5">Someone Else</li>
  <li class="player-name" data-score="70.2">Friend Someone Else</li>
</ul>
```

It is worth noting that the custom data attribute should only be used when we do not find any applicable or more appropriate attribute for that data. Storing the scores in the `class` attribute as `class="98.9"` is definitely not an applicable approach.

For further reference on data attributes, you can head over to the following pages:

- A documentation on custom data attributes available at `http://www.w3.org/html/wg/drafts/html/master/elements.html`
- *All You Need to Know About the HTML5 Data Attribute* (`http://webdesign.tutsplus.com/tutorials/htmlcss-tutorials/all-you-need-to-know-about-the-html5-data-attribute/`)
- An article on HTML5 data attributes by John Resig (`http://ejohn.org/blog/html-5-data-attributes/`)

Time for action – structuring the HTML document

Perform the following steps for structuring the HTML document:

1. Open the `index.html` file in your working directory.
2. Remove anything present between the `<body>` and `</body>` tags and replace it with the following code snippet to establish the header section. Our website's header is wrapped within the HTML5 `<header>` element and it contains the site logo that is wrapped within a `<div>` element with a class of `logo`.

```
<header class="header">
  <div class="logo">
    <h1>Portfolio</h1>
  </div>
</header>
```

3. Then, put the following `<form>` element with a class of `container` and `clearfix` next to the `<header>` element that we just added. We use this `<div>` to contain the website content.

```
<form class="container clearfix"> </form>
```

The `<form>` element is essentially an element like a `<div>` element. We use `<form>` instead of `<div>` as we will use the HTML form elements `<input>` and `<label>` to construct the website navigation.

You can head over to the article (`http://reference.sitepoint.com/html/elements-form`) from SitePoint to see the complete list of elements that are part of an HTML form.

4. Inside the `<form>` element for a container, we add the HTML structure for the website navigation. As we mentioned earlier, our website navigation is uncommon. We will use the radio button as an input type and each `<input>` element is assigned with a unique ID followed by their respective `<label>` element, as shown in the following code snippet:

```
<input class="nav-menu" id="all" type="radio" name="filter"
checked="checked"/>
<label for="all">All</label>

<input class="nav-menu" id="illustrations" type="radio"
name="filter"/>
<label for="illustrations">Illustration</label>

<input class="nav-menu" id="posters" type="radio" name="filter"/>
<label class="nav-menu" for="posters">Posters Design</label>

<input class="nav-menu" id="typography" type="radio"
name="filter"/>
<label for="typography">Typography</label>

<input class="nav-menu" id="packaging" type="radio"
name="filter"/>
<label for="packaging">Packaging</label>
```

5. Then add an HTML5 `<section>` element with a class of `portfolio` next to those `<input>` and `<label>` elements that we just added.

```
<section class="portfolio"></section>
```

This `<section>` element will be used to contain the portfolio, which includes the image thumbnails and the captions.

6. Inside this `<section>` element, we add the portfolio image thumbnails. Each image thumbnail is wrapped within the HTML5 `<figure>` element.

   We have 12 image thumbnails and we will divide them into four columns. Skeleton has 16 columns of grid and 16 divided by four results in four columns. So, each `<figure>` element is assigned with classes of `four` and `columns` with two additional classes of `all` and its category name.

```
<figure class="four columns all poster">
  <img src="images/image-1.jpg" alt=
  "This is 1st portfolio thumbnail.">
</figure>
```

   The classes of `four` and `columns` are assigned to apply the column styles from Skeleton, while the class of `all` will be used to select the `<figure>` element when we need to apply CSS rules to all `<figure>` elements. We will use the category name class to group the figures and also apply styles to the figures that share the same category.

   We will also provide some text that describes the image with an `alt` attribute. This `alt` attribute is useful for the browser to show alternative information for the users, in case the image fails to load.

7. The image thumbnails are grouped into a category. We assign the category name with the `title` attribute in the `<figure>` element, as follows:

```
<figure class="four columns" title="poster">
  <img src="images/image-1.jpg">
</figure>
```

8. The image thumbnail will have a caption containing the portfolio's description. We will use the HTML5 `<figcaption>` element to contain the description text and place it inside the `<figure>` element, as follows:

```
<figure class="four columns all poster">
  <img src="images/image-1.jpg" alt=
  "This is 1st portfolio thumbnail.">
  <figcaption>
    <h4>Lorem Ipsum</h4>
    <p>Lorem ipsum dolor sit amet, consectetur adipiscing
    elit. Morbi molestie lobortis magna eget sagittis.</p>
  </figcaption>
</figure>
```

9. Then, we will add an HTML5 data attribute to `<figure>` to store the category name where the `<figure>` element is assigned and we simply name this attribute `data-category`.

```
<figure class="four columns all poster">
  <img src="images/image-1.jpg" alt=
  "This is 1st portfolio thumbnail.">
  <figcaption>
    <h4>Lorem Ipsum</h4>
    <p>Lorem ipsum dolor sit amet, consectetur adipiscing
    elit. Morbi molestie lobortis magna eget sagittis.</p>
  </figcaption>
</figure>
```

Now, let's add the rest of the image thumbnails, as follows.

```
<figure class="four columns all illustration" data-
category="illustration">
  <img src="images/image-2.jpg" alt=
  "This is 2nd portfolio thumbnail.">
  <figcaption>
    <h4>Lorem Ipsum</h4>
    <p>Lorem ipsum dolor sit amet, consectetur adipiscing
    elit. Morbi molestie lobortis magna eget sagittis.</p>
  </figcaption>
</figure>
<figure class="four columns all poster" data-category="poster">
  <img src="images/image-3.jpg" alt=
  "This is 3rd portfolio thumbnail.">
  <figcaption>
    <h4>Lorem Ipsum</h4>
    <p>Lorem ipsum dolor sit amet, consectetur adipiscing
    elit. Morbi molestie lobortis magna eget sagittis.</p>
  </figcaption>
</figure>
<figure class="four columns all typography" data-
category="typography">
  <img src="images/image-4.jpg" alt=
  "This is 4th portfolio thumbnail.">
  <figcaption>
    <h4>Lorem Ipsum</h4>
    <p>Lorem ipsum dolor sit amet, consectetur adipiscing
    elit. Morbi molestie lobortis magna eget sagittis.</p>
  </figcaption>
</figure>
<figure class="four columns all illustration" data-
category="illustration">
  <img src="images/image-5.jpg" alt=
  "This is 5th portfolio thumbnail.">
  <figcaption>
```

```
      <h4>Lorem Ipsum</h4>
      <p>Lorem ipsum dolor sit amet, consectetur adipiscing
      elit. Morbi molestie lobortis magna eget sagittis.</p>
    </figcaption>
  </figure>
  <figure class="four columns all poster" data-category="poster">
    <img src="images/image-6.jpg" alt=
    "This is 6th portfolio thumbnail.">
    <figcaption>
      <h4>Lorem Ipsum</h4>
      <p>Lorem ipsum dolor sit amet, consectetur adipiscing
      elit. Morbi molestie lobortis magna eget sagittis.</p>
    </figcaption>
  </figure>
  <figure class="four columns all illustration" data-
  category="illustration">
    <img src="images/image-7.jpg" alt=
    "This is 7th portfolio thumbnail.">
    <figcaption>
      <h4>Lorem Ipsum</h4>
      <p>Lorem ipsum dolor sit amet, consectetur adipiscing
      elit. Morbi molestie lobortis magna eget sagittis.</p>
    </figcaption>
  </figure>
  <figure class="four columns all typography " data-
  category="typography">
    <img src="images/image-8.jpg" alt=
    "This is 8th portfolio thumbnail.">
      <figcaption>
      <h4>Lorem Ipsum</h4>
      <p>Lorem ipsum dolor sit amet, consectetur adipiscing
      elit. Morbi molestie lobortis magna eget sagittis.</p>
    </figcaption>
  </figure>
  <figure class="four columns all package" data-category="package">
    <img src="images/image-9.jpg" alt=
    "This is 8th portfolio thumbnail.">
    <figcaption>
      <h4>Lorem Ipsum</h4>
      <p>Lorem ipsum dolor sit amet, consectetur adipiscing
      elit. Morbi molestie lobortis magna eget sagittis.</p>
    </figcaption>
  </figure>
  <figure class="four columns all poster" data-category="poster">
    <img src="images/image-10.jpg" alt=
    "This is 9th portfolio thumbnail.">
    <figcaption>
      <h4>Lorem Ipsum</h4>
      <p>Lorem ipsum dolor sit amet, consectetur adipiscing
      elit. Morbi molestie lobortis magna eget sagittis.</p>
```

```
    </figcaption>
  </figure>
  <figure class="four columns all package " data-category="package">
    <img src="images/image-11.jpg" alt=
    "This is 10th portfolio thumbnail.">
    <figcaption>
      <h4>Lorem Ipsum</h4>
      <p>Lorem ipsum dolor sit amet, consectetur adipiscing
      elit. Morbi molestie lobortis magna eget sagittis.</p>
    </figcaption>
  </figure>
  <figure class="four columns all illustration "
  title="illustration">
    <img src="images/image-12.jpg" alt=
    "This is 10th portfolio thumbnail.">
    <figcaption>
      <h4>Lorem Ipsum</h4>
      <p>Lorem ipsum dolor sit amet, consectetur adipiscing
      elit. Morbi molestie lobortis magna eget sagittis.</p>
    </figcaption>
  </figure>
```

10. Lastly, for the website footer area, add the following HTML5 `<footer>` element with the `container clearfix` class next to the `<div>` element defined for the container, which we just added in Step 3:

```
<footer class="container clearfix">
  <div class="contact">
    <ul>
      <li class="contact-name">John Doe</li>
      <li class="contact-phone">(+1) 1-234-5678-9</li>
      <li class="contact-email">me@johndoe.com</li>
    </ul>
  </div>
  <div class="social">
    <p class="copyright">Copyright John Doe 2012</p>
    <ul>
      <li class="social-dribbble">
      <a href="#">Dribbble</a></li>
      <li class="social-facebook">
      <a href="#">Facebook</a></li>
      <li class="social-twitter">
      <a href="#">Twitter</a></li>
    </ul>
  </div>
</footer>
```

What just happened?

We have just added the structure for the website to `index.html` using the HTML5 elements and establishing the website header, the image portfolio, and the website footer.

Summary

In this chapter, we started our first project and accomplished the following things:

- Unpacked the Skeleton package and walked through some of the components
- Learned how to use Skeleton responsive grid
- Set up a working directory as well as the project documents
- Prepared the project assets
- Structured the project document with HTML5 elements

Now that the project has been set up, we are going to make up and tweak the website's look with CSS3 in the next chapter.

Where to buy this book

You can buy Responsive Web Design by Example Beginner's Guide from the Packt Publishing website: `http://www.packtpub.com/responsive-web-design-by-example/book`.

Free shipping to the US, UK, Europe and selected Asian countries. For more information, please read our shipping policy.

Alternatively, you can buy the book from Amazon, BN.com, Computer Manuals and most internet book retailers.

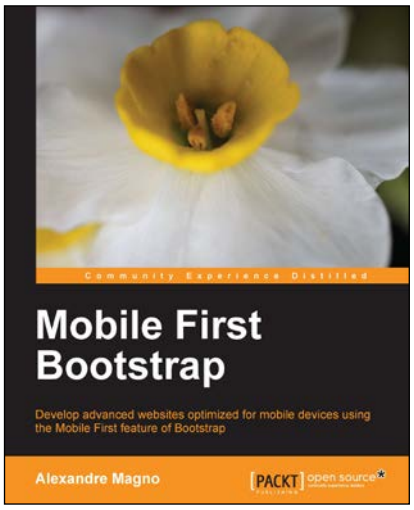
www.PacktPub.com

For More Information:
www.packtpub.com/responsive-web-design-by-example/book

[PACKT] open source
community experience distilled
PUBLISHING

Mobile First Bootstrap

Alexandre Magno



Chapter No. 2
"Designing Stylesheet in Bootstrap 3"

In this package, you will find:

A Biography of the author of the book

A preview chapter from the book, Chapter NO.2 "Designing Stylesheet in Bootstrap 3"

A synopsis of the book’s content

Information on where to buy this book

About the Author

Alexandre Magno has worked for 10 years as a web developer, and is currently working as a software engineer at `globo.com`. He is a very active contributor in the open source community with plenty of projects and acts as a jQuery evangelist and a responsive design missionary. As a multidisciplinary developer, he has strong experience in a wide range of server-side frameworks and CMS such as Ruby on Rails, Django, WordPress, and exploring Node.js. He has developed many libraries that are widely used at `globo.com`, and was one of the first developers to develop mobile websites with a responsive design in the company he worked at. He is an active contributor of Twitter Bootstrap and the creator of one of its branches, the Globo Bootstrap, which is the first translation of Bootstrap to Portuguese, and also developed some components used in `globo.com`.

He is very passionate about web development, and he writes about it in his blog at `alexandremagno.net`. He has already contributed in the publishing of a web magazine about jQuery UI and has even made presentations in some technical events, such as the International Free Software Forum (FISL). Writing this book for him is a great step further, after these achievements.

Besides technology, he is a musician and song writer too. He likes to remember every moment of his life with a music lyric. At this moment, for example, the verse would be "We are the champions, my friend".

Mobile First Bootstrap

We are living in a rapidly changing time because of the way we use the Web. To follow up on this currently changing paradigm, the development cycle finds its way to follow this new scenario. The frontend community is building amazing tools to make it possible to deal with so many changes. In the past, we had to deal with browsers that did not respect the standards. Now, we are dealing with new devices with a wide range of features. The Web is going mobile. With so many limitations on one side, and the power of flexibility on another side, the mobile and tablet just cannot be a substitute for the huge and wide desktop screens. The mobile needs are complementary to the desktop's standard nature. On the other hand, there's a brand new opportunity with its mobile use and simplicity that makes us see an ongoing challenge to expand our business. This way we have a presence in the ever-growing online market. The Web is not only limited to a desktop anymore. We have to think wider.

The most famous frontend frameworks also have to undergo a deep change to meet this demand. They are radical. In the new Bootstrap release, they decided to get their learning from the oldest version and redesign from scratch, but the focus is now on Mobile First as a user environment. Why do you have to focus on the mobile environment now?

Well, in the new paradigm, we will just get a Mobile First framework for simple development like we used to. We have to follow it up and see how powerful an online product cross device with no disturbing complexity can be, even though it sounds complicated.

We know that Bootstrap is not a silver bullet, but it's a great option to start your development cycle for Mobile First. It makes things simple, and we should know how it does it, because this Version 3 is based on a lot of contributions from Bootstrap developers. This is a strong reason for you to follow the same principles.

You may have already used Bootstrap or visited sites that use it. It's not hard to identify Bootstrap's basic template in a lot of websites around here, for example, the documentation APIs of open source projects. Research of the meanpath shows that Bootstrap is present in 1 percent of the 150-million websites worldwide, powering 1 percent of the Web (`http://blog.meanpath.com/twitter-bootstrap-nowpowering-1-percent-of-the-web/`). So, we know the power of its use and the amazing things we can do, but now we can do even more, and you have a chance to explore its new capabilities.

In this book, we will not just look at the Bootstrap changes, but also cover the mindset that makes us think mobile and go through an efficient multidevice development. We will develop a sample short message app that will be called Cochichous, which users can check the nearest messages sent by other users using Twitter API. We will explore HTML5 and JavaScript capabilities, as well the grid and the Bootstrap plugins. Besides that, we will also discover what has changed with this new important release.

What This Book Covers

Chapter 1, Bootstrap 3.0 is Mobile First, introduces you to the Mobile First development, and makes you understand why Bootstrap was redesigned to this new approach and the reasons we should take care of. We will get a little vision about the responsive design and how Mobile First works with it. Then, we will get in contact with Bootstrap documents to start checking the new documentation and make tests to familiarize ourselves with a Mobile First website.

Chapter 2, Designing Stylesheet in Bootstrap 3, gets us started with the CSS structure and the main grid changes, as well as showing us an example on how to make decisions about breakpoints using Bootstrap grid classes. We will get an overview of how the navigation deals with Mobile First, and how to get forms optimized to different devices.

Chapter 3, JavaScript, the Behavior in Mobile First Development, will lay the foundation of stylesheets, and we will get in touch with JavaScript. In Bootstrap, this is represented by JavaScript plugins using jQuery that's almost intact in this version. It's a great opportunity to learn how to adapt the Bootstrap JavaScript plugins to work with a new device's capability and how to explore its pattern to build your own plugins. There's a clear explanation about progressive enhancement that we probably have already heard before, but now we are focused on applying Mobile First techniques.

Chapter 4, Getting it All Together – a Simple Twitter App, will get all the pieces already learned to make a real Mobile First sample and a simple product to allow us to have an overall idea of how to develop a Mobile First new product from scratch. We start to deal with the capability offered by the mobile browser to deliver a better experience on mobiles for our users. Then, we get all the points that surround the decision about a web app or a native app.

Chapter 5, Performance Matters, will cover what needs to be done to have our mobile experience optimized, how this optimization affects the accuracy still present in the desktop, and how the Mobile First development can be a powerful tool to make our website faster. We will learn three main techniques: optimizing images, loading components on demand, and the use of fonts to render icons.

2
Designing Stylesheet in Bootstrap 3

What's new in CSS implementation of this new Bootstrap version? One direct answer: responsiveness is not optional anymore. Responsiveness is now included in a single Bootstrap CSS and aims to be mobile friendly from the start.

In this chapter, we will discover the new features in the Bootstrap stylesheet that bring us a powerful choice to stylize the content in a completely reliable way for the device. With the new grid system, it's possible to define how the resolution will behave at different breakpoints and allows us to think about Mobile First from the scratch. We will discover flexible ways to work with semantic grids to explore the forms attached to the grid system, and make different form layouts adapt easily to take form usability a step further. Also, we will to discover some tricks about the issue with relative units and how to use navigation components for friendly mobile navigation.

This chapter will cover the following topics:

- The grid system
- Forms in different resolutions
- The icon library
- Responsive utilities
- Relative units
- Navigation

The grid system

The Bootstrap Mobile First grid system meets all of the developers demands for flexibility. It's not just a simple grid system, as we knew in Bootstrap 2. It's more flexible, adapted to the mobile mindset with full control of all the responsive flows. This means Bootstrap 3 is an enhancement from the best of the existing grid framework, with full new support to the semantic grid system.

Semantic grids

One of the main drawbacks in Bootstrap 2 is the fact that there's no nice way to make the grid become semantic efficiently. This means that you have to stylize the grid classes into HTML in the following manner:

```
<div class="grid_x"></div>
```

In a semantic grid, you can use declarative markups to transform the grid system as we want, using the `LESS` mixins to define rows, columns, and grid elements, as shown in the following code:

```
footer { .column(12); }
```

LESS is a preprocessed CSS language stylesheet that makes use of variables and mixins for developing dynamic CSS. Mixins are a bunch of CSS declarations that can be used as stylesheet functions.

Before getting into the semantic grid system in Bootstrap 3, you should know a little about LESS and the power of mixins. In the following example, we use a basic mixin to make a `border-radius` property work in Firefox and Chrome for you to get familiar with a real mixin (first part is the `LESS` mixin, the second part is the LESS CSS, and the third part is vanilla CSS):

```
//mixin
.rounded {
  border-radius: 5px;
  -moz-border-radius: 5px;
  -webkit-border-radius: 5px;
}
//CSS
#menu {
  color: gray;
  .rounded;
}
```

```
//Output
.rounded {
  border-radius: 5px;
  -moz-border-radius: 5px;
  -webkit-border-radius: 5px;
}
```

Now, with the LESS mixins in mind, we can imagine a grid system generated by mixins. A great example of semantic grid, is Semantic.gs (`http://semantic.gs/`). You can achieve a semantic grid system from other frameworks too, such as Susy (`http://susy.oddbird.net/`).

With the following HTML (with Semantic.gs):

```
<header>...</header>
<article>...</article>
<aside>...</aside>
```

We can have the related CSS (with LESS):

```
//variables
@column-width: 60;
@gutter-width: 20;
@columns: 12;
//Including mixins
header { .column(12); }
article { .column(9); }
aside { .column(3); }
```

The preceding CSS gives us the freedom to control the responsive flow using the `LESS` mixins to construct the grid structure. All grid placements, are CSS responsibility. Because we use CSS to define grid placements and not a class in HTML to tell how many spaces the column will take, we put everything in its place without inserting extra markups and grid styles into CSS. There's a great advantage in using this for complex grids and not becoming a slave to the markup full of grid classes, because you're not limited to declaring grid classes that tell the dimension. Using a semantic grid, we have decoupled the layout from HTML and moved to CSS.

The semantic Bootstrap version from the same grid could be rewritten as follows:

```
.wrapper {
  .make-row();
}
.header {
  .make-sx-column(12);
}
```

```
.article {
  .make-sx-column(9);
}
.aside {
  .make-sx-column(3);
}
```

The grid framework

In contrast to the semantic grid system, the grid framework is the common grid we have already known in Bootstrap that use classes in HTML to determine grid spaces.

The semantic grid is an extracted mixin of the grid framework. To simplify this, we could use:

```
grid_2 {
 .make-column(2);
}
```

Downloading the example code

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

We can do this repeatedly to create an entire custom grid system. In fact, the semantic grid mixin from the Bootstrap core builds the Bootstrap grid framework.

Even with this flexibility, the Bootstrap is a symmetric grid. This means, it follows the `@column-width`, `@gutter-width` and `@columns` variables for building the whole grid with an equal distribution based on these variables. There are other unlimited ratios that are not symmetrical. These are called asymmetric grids. An asymmetric grid is useful for more complex needs.

We have unlimited grid tools; one of these is Singularity, which explores the power of an asymmetric grid (`http://scottkellum.com/2013/04/05/singularity-a-modern-grid-framework.html`). Singularity is a grid system rather than a grid framework. It has more robustness for exploring different ratios in custom grids. Bootstrap 3 is not just a grid system because the grid component is just one part of an entire framework. To see how it works, check the semantic grid examples in the Bootstrap documentation (`http://getbootstrap.com/css/#grid-less`).

The following table shows the main differences between the grid options in Bootstrap 3:

	Grid frameworks	Semantic grid	Asymmetric grids
Pros	• Simple to use • Predefined grid width as per CSS Classes	• No extra markup • Grid dimensions decoupled from the markup	• Organic grid organization • Does not follow fixed-ratio grid dimensions
Cons	• Has extra markup • Row width defined in HTML	• Defining the width of every grid component is mandatory	• Complex to use, more LESS variables

Breakpoints and completely fluid layouts

Breakpoints set a range of common resolution widths used for making significant changes in the layout. Bootstrap has a naming convention for this: phone, tablet, and desktop.

Sometimes, there are more breakpoint cases and matching criteria than the three main ones. So we have two scenarios: a pixel layout using breakpoints or a completely fluid layout.

Now, Bootstrap has a flat-grid framework that allows us to create more complex mobile grids; you can make use of the grid offsets, as we did for desktop in Bootstrap's previous versions.

Predefined classes to control responsive flow

The grid in Bootstrap 3 now has a unique class for defining a grid. There are no `row-fluid` classes anymore to say that the grid should behave as fluid in our stylesheet. We can use declarative classes to define the orientation for many breakpoints.

In Bootstrap 2, we would do the following for enabling a fluid layout:

```
<div class="container-fluid">
  <div class="row-fluid"> … </div>
</div>
```

In Bootstrap 3, we would do the following:

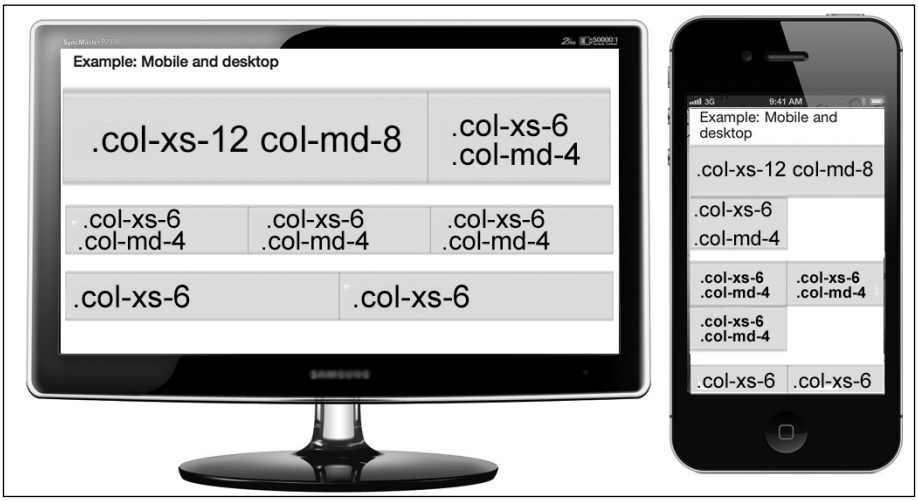
```
<div class="container">
  <div class="row">
    <div class="col-md-4"> … </div>
  </div>
</div>
```

Now, in Version 3, there are grid classes that describe the screen resolution's ranges. This is an intuitive way for enabling us to differentiate between device width and its respective breakpoints.

In the mobile-stacked, one-column version, we already define how Mobile First will behave when a new breakpoint is reached, for example, from a 320-pixel phone to a 768-pixel tablet.

The default column class is `.col-xs-n`, where `n` is a number between 1 and 12, for extra small screens (phones below 768 px). It has 12 columns for mobile, and is prefixed with `lg`, for instance `.col-lg-n` for larger devices, where n is a number between 1 and 12, and we have `.col-sm-x` for tablets that work following the same rationale.

We have the setup, as shown in the following figure. We can compare the grid size of two devices and check out the behavior in wide and small resolutions:



As we can see in the previous figure, **.col-xs-12 .col-md-8** classes indicate 12 columns in mobile, and 8 columns in medium-screen devices such as desktops.

Then, **.col-xs-6 .col-md-4** means that extra small screens (xs) are 6-columns wide and medium screens, such as desktops, are 4-columns wide.

The last column is divided into two columns with the same width: 6 for right, 6 for left, both for mobiles as well as desktops. The extra small screen is the default one, so it will behave in the same way for desktops.

We can have the following variations:

Classes (* is a number between 1 and 12)	Devices
`.col-xs-*`	Below 768 px
`.col-sm-*`	Between 768 px and 992 px
`.col-md-*`	Between 992 px and 1200 px
`.col-lg-*`	Up to 1200 px

You can refer to the Bootstrap *Grid options* section in the documentation at `http://getbootstrap.com/css/#grid-options` for more details.

The Flex grid, a new draft specification for CSS level 3, works in a similar manner, because it uses CSS rules to define breakpoint changes (`http://www.w3.org/TR/css3-flexbox/`). This new box model can optimize the user interface design experience and make it possible for us to really build layouts with CSS in two dimensions. We define how the flex orientation should be, as the Bootstrap 3 does using the preceding classes.

Forms in different resolutions

The form layout still resides in the same grid naming conventions. With the power of Mobile First, you can have forms optimized to match the user's best experience of filling forms.

In the old Bootstrap versions, you use the grid classes for full control on form placement. However, the best practice is to not use classes in the form elements dedicated to make an input column work as a grid. You can wrap the input into a grid container and get its space. By default, the inputs keep your width as 100 percent.

In a form design, we do not need to specify the width in pixels. It seems easier to specify this at the beginning, but it's just a matter of time before you lose the flexibility and feel obligated to design each input size. There's a more convenient way to do this: relative and named sizes.

With named sizes, such as small, medium, and large, granulated names such as big and bigger are better for defining the input sizes.

This convention gives us the power to develop different form designs, without messy layouts in different resolutions, because it is already handled by Bootstrap to behave accordingly in different devices.

If we try to have a fixed width in inputs, we may get into trouble. As the window resizes, the input will keep that size. We may lose the power of the grid framework and also lose a great opportunity to follow the Bootstrap framework to have a flexible grid with little effort. It's really a good practice to use your inputs inside a grid container and leave the control and command with the grid.

We can't forget to mention that using forms semantically is a wise decision for avoiding unnecessary code. With the simple usage of the input in HTML 5 type attributes, we have an easier method for collecting the best user experience for each device. For example, a date-input format can make the date you select in a smart phone look the same as used in apps. Input with the `file` type lets you attach a picture easily and it even lets you use your smart phone camera to take a picture instantly and use for your website.

The icon library

Bootstrap 2 usually uses images to render a basic set of icons as sprites. However, it takes time to load and sometimes cannot be essential thinking in terms of performance for a website page. We always have to import an extra image and sometimes lose flexibility with it when we need optimized sizes in accordance with the device.

In different dimensions, we have to deal with different optimized image sizes, which are optimized for a given device; this results in more image files, extra storage space, more dependencies, and so on.

Glyphicons (`http://glyphicons.getbootstrap.com/`) is an icon library for Bootstrap; part of its core responsibilities is to manage and give utility classes for defining an icon:

```
<i class="icon-camera"></i>
```

This library uses font-face to generate icons instead of sprites, so as an isolated icon library, Glyphicon is getting better and now adapts to flexibility. One of the greatest benefits of this, is that it now uses font-face to font rendering and lets us size the icon according to the font size as well as use colors, animations, and CSS effects, such as shadow. This is a great method to handle icons, but you can't forget that fonts as icons have limitations and we can use them only for monochromatic icons.

Responsive utilities

Bootstrap now has a full set of mixins used for helping you improve the responsive experience. It's time to explore semantic grid mixins and other responsive utilities.

Responsive classes

There are responsive utility classes for making mobile development easier and friendly. With visible and hidden classes, combined with large, small, and medium devices, we have a complete and powerful way to control our device experience to sometimes not display some information in each specific device, as shown in the following table:

Classes	Devices
`.visible-sm`	Small (up to 768 px) visible
`.visible-md`	Medium (768 px to 991 px) visible
`.visible-lg`	Larger (992 px and above) visible
`.hidden-sm`	Small (up to 768 px) hidden
`.hidden-md`	Medium (768 px to 991 px) hidden
`.hidden-lg`	Larger (992 px and above) hidden

Semantic grid variables and functions

Now, we can use a flexible way for defining our grid in CSS (with LESS):

```
.make-row(@gutter: @grid-gutter-width);

.make-sm-column(@columns; @gutter: @grid-gutter-width);

.make-sm-column-offset(@columns);

.make-sm-column-push(@columns);
```

The same applies for the `md` and `xs` prefixes; so, basically, we have the following formula for the semantic grid functions:

```
.make-type-column(@columns; @gutter: @grid-gutter-width);

.make-type-column-offset(@columns);

.make-type-column-push(@columns);
```

The type can be `xs`, `sm`, `md`, and `lg` (extra small, small screen, medium, and large).

With these semantic grid mixins, you can create a grid structure. Use `.make-row()` to create a row to be nested with a composition of column layouts that have a single parameter for defining width. Use `.make-column` to create a column grid that sums up to 12. You can use `.make-type-offset` to give an offset, like we usually do in Bootstrap, as well as while pushing and pulling columns.

Relative units

Which unit of measure should we adapt for our mobile needs? Do we really not need the EM unit anymore? When IE6 did not support font sizing, EM was the only way for font sizing in IE without using JavaScript. For Mobile First, the EM unit has its own advantages.

We need a font unit that allows us to maintain the aspect ratio, as per the screen size. While considering the adequate font unit and while comparing it to the grid, the fonts should follow the grid in a certain ratio. Instead of defining a base font-size in each media-query breakpoint and for each element, redefine the font-sizes. It's better to have a proportional and relative unit ratio, and this can be achieved with an EM or percentage unit.

In this new version, the Bootstrap milestone has a lot of discussions about using the EM unit over pixels, but the limitations with its use don't justify the benefits. They still support IE8 with a pixel fallback, and this approach will generate duplicate code. Nesting the EM unit is historically a headache and a cross browser issue, and sometimes, we need extra math to compute the pixel values.

We can use a mixin to convert the font units, but that's not an ideal solution, and Bootstrap will probably change it in future releases.

Pixel values are still used for font sizing and are controlled by `@font-size-base`, `@font-size-large`, `@font-size-small` that hold fonts for the default size of large screens and small screens respectively.

Navigation

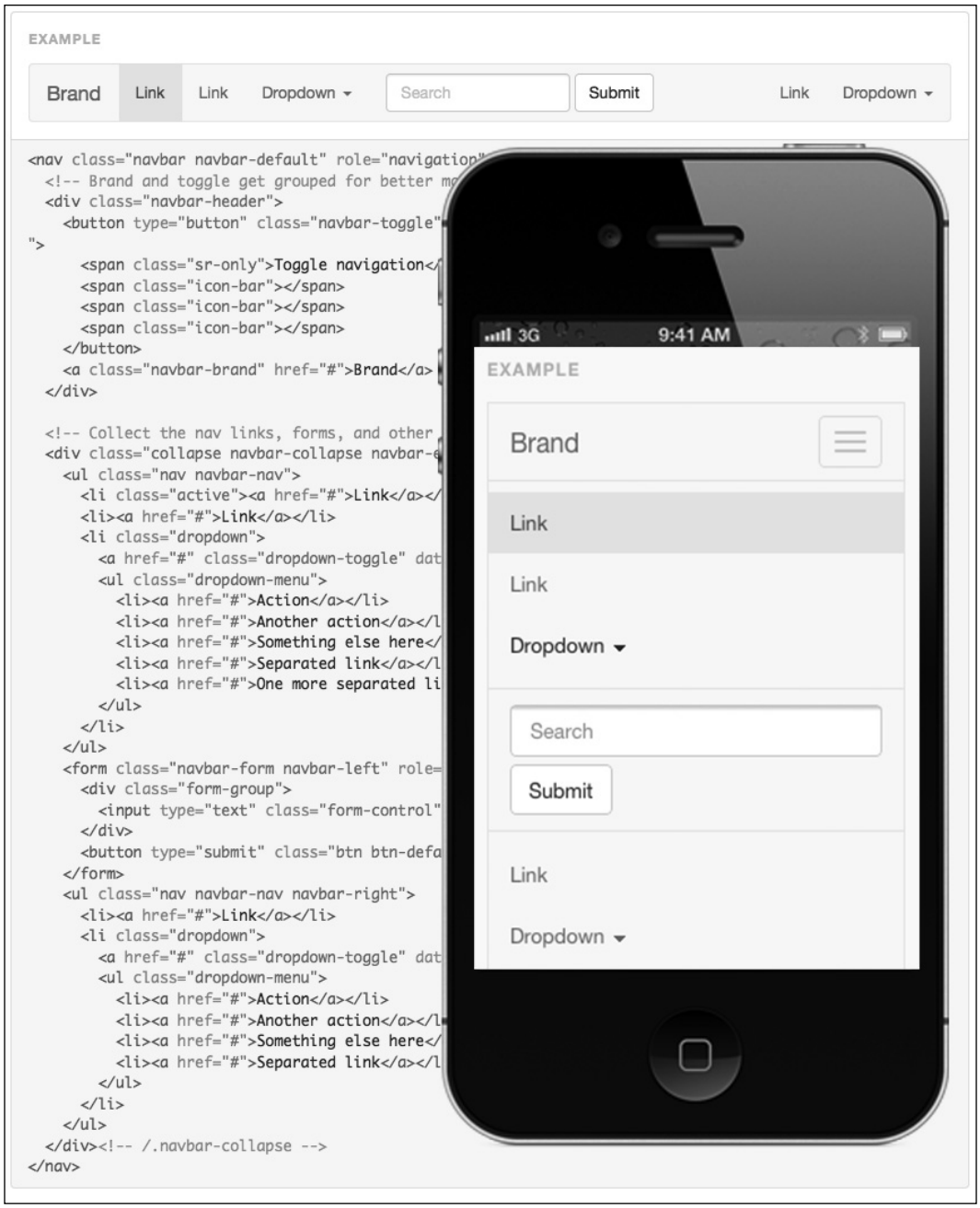
Now, with responsive navigation we don't have more optional CSS files to make your menu fully usable.

The menu now shrinks to a completely adapted menu for all user needs. The famous technique of transforming a full menu into a mobile menu experience is free of any setup. Just add the `navbar` and `navbar-default` classes, and you get a main navigation bar for your website.

The navigation bar uses the JavaScript Collapse plugin (`http://getbootstrap.com/javascript/#collapse`) to have a default responsive navigation bar using just links and lists. Please check out the Bootstrap documentation (`http://getbootstrap.com/components/#navbar`) to see more details about this component.

We can use simple classes such as `navbar` and `navbar-inverse` to have a fully clean experience in different contrasts, so we have `navbar` with a white background and the inverse class when we wish to have `navbar` with a black background.

From a Mobile First point of view, we start with a menu that can be toggled and fully collapsed, which becomes a horizontal menu as the viewport increases. This is shown in the following figure:



Summary

As we could see in this chapter, Bootstrap CSS got cleaner and more powerful in Version 3. We took advantage of Mobile First to get the most well-designed style. With Mobile First in mind, it becomes simple to progress and design our application using the Bootstrap grid and utility classes. Bootstrap was revisited and the Mobile First CSS gave us many approaches for different application needs for mobiles.

You learned different grid conventions adapted for Bootstrap, and got introduced to the semantic grid, and saw how it works with the use of this flexible tool to make Bootstrap grid more convenient for our mobile needs.

We saw how the grid can be a great auxiliary to form design and how powerful it is to keep our form semantic.

We got in touch with the issues with relative units. We saw that Bootstrap tries to explore the EM unit but still has a counterpart that overlaps its benefits, so Bootstrap still has to deal with pixels and use it to define font sizing.

We saw a fully responsive menu, widely used in desktop and mobile applications with a simple snippet of code, and the use of a thin JavaScript for offering simple cross-platform navigation without any tricks.

In the next chapter, we will explore the JavaScript in Mobile First as a behavior layer for the lessons learned from the Bootstrap CSS.

Where to buy this book

You can buy Mobile First Bootstrap from the Packt Publishing website: `http://www.packtpub.com/mobile-first-bootstrap/book`.

Free shipping to the US, UK, Europe and selected Asian countries. For more information, please read our shipping policy.

Alternatively, you can buy the book from Amazon, BN.com, Computer Manuals and most internet book retailers.

www.PacktPub.com

For More Information:
www.packtpub.com/mobile-first-bootstrap/book