Twitter Bootstrap Succinctly

by Peter Shaw

Syncfusion | Technology Resource Portal

Twitter Bootstrap Succinctly

By
Peter Shaw

Foreword by Daniel Jebaraj

Syncfusion®
Deliver innovation with ease®

Important licensing information. Please read.

Technical Reviewer: Zoran Maksimovic, @zoranmax, www.agile-code.com
Copy Editor: Suzanne Kattau
Acquisitions Coordinator: Hillary Bowling, marketing coordinator, Syncfusion, Inc.
Proofreader: Morgan Cartier Weston, content producer, Syncfusion, Inc.

Table of Contents

The Story behind the *Succinctly* Series of Books

Daniel Jebaraj, Vice President
Syncfusion, Inc.

Staying on the cutting edge

As many of you may know, Syncfusion is a provider of software components for the Microsoft platform. This puts us in the exciting but challenging position of always being on the cutting edge.

Whenever platforms or tools are shipping out of Microsoft, which seems to be about every other week these days, we have to educate ourselves, quickly.

Information is plentiful but harder to digest

In reality, this translates into a lot of book orders, blog searches, and Twitter scans.

While more information is becoming available on the Internet and more and more books are being published, even on topics that are relatively new, one aspect that continues to inhibit us is the inability to find concise technology overview books.

We are usually faced with two options: read several 500+ page books or scour the web for relevant blog posts and other articles. Just as everyone else who has a job to do and customers to serve, we find this quite frustrating.

The *Succinctly* series

This frustration translated into a deep desire to produce a series of concise technical books that would be targeted at developers working on the Microsoft platform.

We firmly believe, given the background knowledge such developers have, that most topics can be translated into books that are between 50 and 100 pages.

This is exactly what we resolved to accomplish with the *Succinctly* series. Isn't everything wonderful born out of a deep desire to change things for the better?

The best authors, the best content

Each author was carefully chosen from a pool of talented experts who shared our vision. The book you now hold in your hands, and the others available in this series, are a result of the authors' tireless work. You will find original content that is guaranteed to get you up and running in about the time it takes to drink a few cups of coffee.

Free forever

Syncfusion will be working to produce books on several topics. The books will always be free. Any updates we publish will also be free.

Free? What is the catch?

There is no catch here. Syncfusion has a vested interest in this effort.

As a component vendor, our unique claim has always been that we offer deeper and broader frameworks than anyone else on the market. Developer education greatly helps us market and sell against competing vendors who promise to "enable AJAX support with one click," or "turn the moon to cheese!"

Let us know what you think

If you have any topics of interest, thoughts, or feedback, please feel free to send them to us at succinctly-series@syncfusion.com.

We sincerely hope you enjoy reading this book and that it helps you better understand the topic of study. Thank you for reading.

Please follow us on Twitter and "Like" us on Facebook to help us spread the word about the *Succinctly* series!

About the Author

As an early adopter of IT back in the late 1970s to early 1980s, I started out with a humble little 1KB Sinclair ZX81 home computer.

Within a very short space of time, this small 1KB machine became a 16KB Tandy TRS-80, followed by an Acorn Electron and, eventually, after going through many other different machines, a 4MB, ARM-powered Acorn A5000.

After leaving school and getting involved with DOS-based PCs, I went on to train in many different disciplines in the computer networking and communications industries.

After returning to university in the mid-1990s and gaining a Bachelor of Science in Computing for Industry, I now run my own consulting business in the northeast of England called Digital Solutions Computer Software, Ltd. I advise clients at both a hardware and software level in many different IT disciplines, covering a wide range of domain-specific knowledge—from mobile communications and networks right through to geographic information systems and banking and finance.

With more than 30 years of experience in IT, and with many different platforms and operating systems, I have a lot of knowledge to share.

You can often find me hanging around in the LIDNUG .NET users group on LinkedIn, which I help run. And you can easily find me in places such as Stack Overflow (and its GIS-specific board) and on Twitter as @shawty_ds. I'm now also on Pluralsight, where my various videos are available.

I hope you enjoy the book and get something from it.

Please remember to thank Syncfusion (@Syncfusion on Twitter) for making this book and others possible, allowing people like me to share our knowledge with the .NET community at large. The *Succinctly* Series is a brilliant idea for busy programmers.

Introduction

How many of you consider yourself to be hardcore developers who only do backend coding and wouldn't be caught dead doing UI/UX coding or any of that fancy, flouncy, artsy stuff?

Yeah, I think I see pretty much a 90 percent show of hands.

Chances are, though, with that opening question I hit a nerve. In fact, I'm so confident that I did that I've now got you all wanting to know why.

As a developer, our job basically is to write software; this is something that requires deep, logical thought and an exceptionally analytical mind.

It's a rewarding and challenging job, one that we all love to do. But it has one fatal flaw:

Non-developers.

Those of you who work in an enterprise will know exactly and straight away what I'm about to say. I'm talking about the managers, the users, the project leads, the admin clerks, and the marketers—the list goes on and on. All of them have one thing in common: they don't understand what it is you do, why, or how.

They ask for an update on the major, super-important project you're doing for them (and we all know that every project is major, super-important). So, you explain to them that the database is working, the database access layer is coming along, and the business rules are in good shape. And the whole time, you try to keep everything in layman's terms, but then you hear, *"Yeah, Yeah, Yeah…Okay, but what does it look like?"*

So, you fire up your project with its half-written UI done in a standard black on white (or grey) layout, only to be greeted by cries of,*"What on earth is that! It looks terrible! We're paying you to write software, not re-create the drawings of a 3-year-old child!"*

We've all been there. We've all suffered the curse of those who only look at the visual side of things, such as the project manager who is blind to anything that's not "pretty" or the CEO who thinks the color mauve has the most RAM.

But what can you do about it?

You're a developer, not a graphics designer. Sure, you can make it look reasonably good, but you should be spending time writing the code. After all, if there's no code, there's nothing to add a UI to. Problem is, the non-developers don't understand this. And, to be fair, you shouldn't expect them to. Unfortunately, though, they are very results-driven, and for them to be able to see a good-looking UI is a result. And when they see results, your life is that much easier.

Bottom line? You need Twitter Bootstrap**.**

Chapter 1 What is Twitter Bootstrap?

Apart from being a godsend for the regular developer, Twitter Bootstrap is a CSS and JavaScript UI framework that was written by two of Twitter's senior developers to make sure they got a consistent look and feel across all of the projects they were creating for Twitter at the time. Now, yes, I can hear the groans already: "Not another UI framework! We already have jQuery UI and Kendo UI and a million others, why on earth do we need another?"

Twitter Bootstrap (TWB) is different. Its purpose is not to create dialogs and sliding effects (although it does do them rather well), nor is its purpose to allow you to apply themes to your creations.

TWB is designed to help you non-designer types to balance your layouts and designs.

I'm not a designer, I'm a developer myself. However, I've read more than a few papers on this subject, and two of the most important things to know when designing a UI are balancing your layout elements and the harmony of your chosen colors.

Color harmony is an easy one to understand, but balance is a little trickier. Balance means that there's consistency and flow through your UI. It means that proportions are equal even when they're not, and font sizes are always scaled by the same ratios and weights, among many other things.

TWB helps you to achieve this in a very standard way by not only providing standard grid sizes and page layout tools, but by also providing a standard set of base CSS rules for different headings, text sizes, sidebars, wells, lead text, and much, much more.

The magic part, however, is that it's designed by developers for developers; in most cases all you need to add is a simple class or some HTML 5 data attributes.

In this book, I'll take you on your first tour of Twitter Bootstrap and hopefully make your life a little more bearable when dealing with the visual-only types.

Chapter 2 Adding Bootstrap to Your Project

So, now that I have your attention, how do we add this magic to our projects?

Well, first off, it's only CSS and JavaScript so you only need to attach them in the usual way by adding links and script tags into your HTML code.

However, to get the best bang for your buck, there is a defined way that the TWB folks recommend you wire things up. More on that in a moment.

First things first. Let's head over to the TWB site and download what we need.

Note: At the time this book was written, Bootstrap Version 2 was the current release and Version 3 was still in testing. Since then, however, Version 3 is now out of testing and Version 2 is no longer as widely used. This book is written using the Version 2 CSS classes, so if you are using Version 2 then all should be fine. If, however, you are using Version 3, you must be aware that there have been some major changes to the names of some of the CSS rules. Fortunately, the creators of the framework have made a conversion chart that shows exactly what has changed. This conversion chart can be found at http://getbootstrap.com/getting-started/#migration. The original Version 2 documentation is still available but it's in a subfolder of the site (as shown below).

The official Twitter Bootstrap site can be found at http://www.getbootstrap.com.

However, as mentioned earlier, this will take you to the new Version 3 portal. To find the Version 2 documentation, you need to go to http://www.getbootstrap.com/2.3.2.

If you have found the right place, then you should see something like the following:

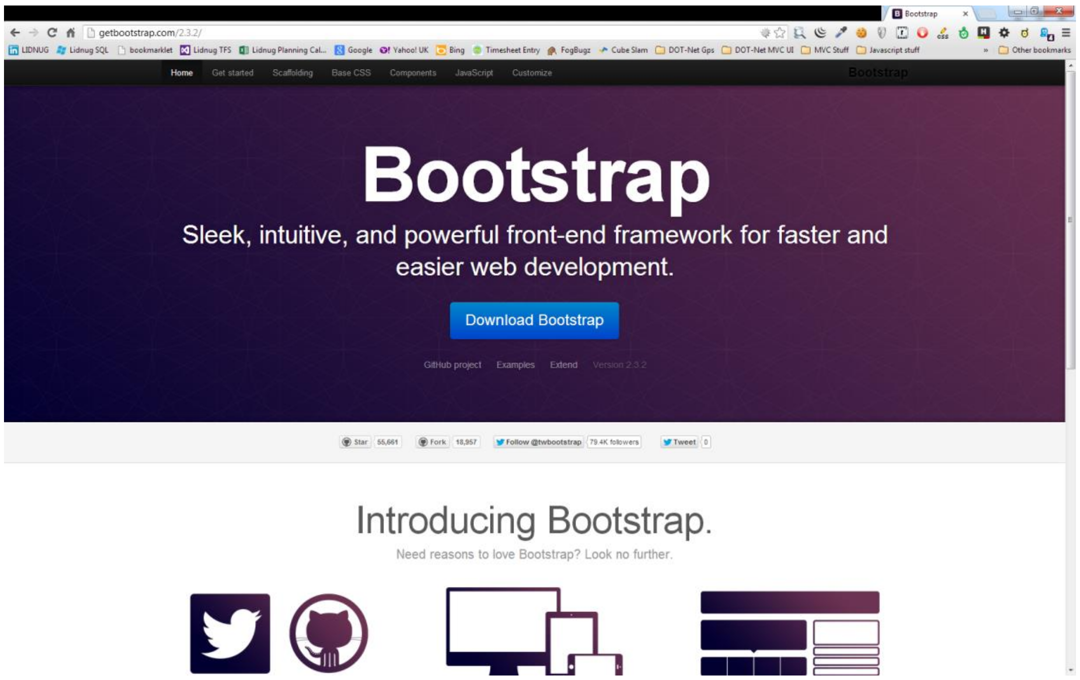

Bootstrap
Sleek, intuitive, and powerful front-end framework for faster and easier web development.
Download Bootstrap
Introducing Bootstrap.
Need reasons to love Bootstrap? Look no further.

Figure 1: Twitter Bootstrap 2.3.2 home page

From here, if you click the big blue Download Bootstrap button, it will download the main, full, uncustomized zip file that contains everything you will need to use TWB.

Download Choices

Just like many packages, TWB can be customized in many different ways.

One of its strengths is that the entire framework is built using "Less", the CSS preprocessor. With "Less," you can define variables and constants in your CSS code which can then be substituted at build/deploy time to make system-wide changes to the look and feel of your project.

This means you could, for instance, define something as **`Orange = #FF6000`**.

Then, you could refer to it in other areas of your code in the following manner:

```
background-color: Orange;

border: 1px dashed Orange;
```

Or something similar.

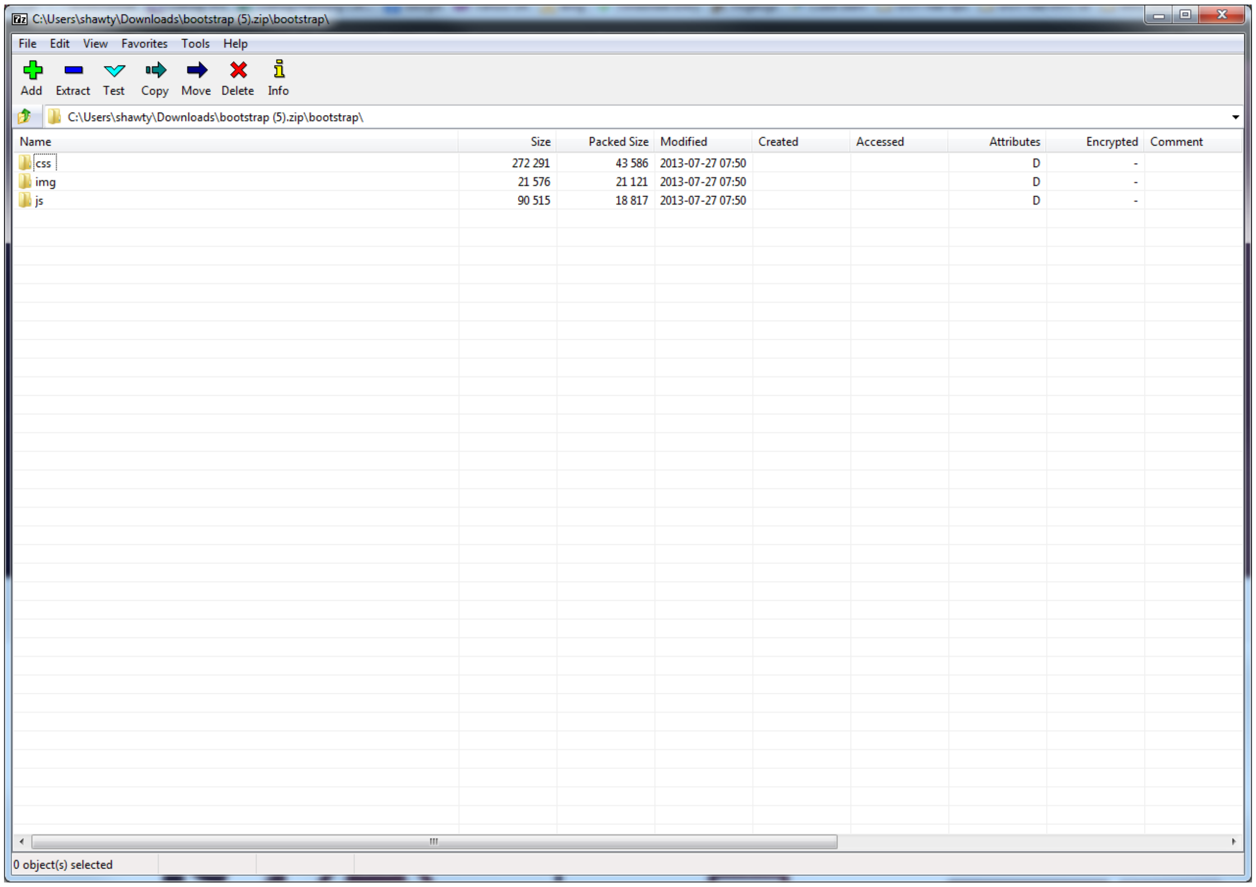
The customization page on the TWB site allows you to tweak all of the values it uses for things such as sizes, colors, fonts, and much more, as well as choose exactly which modules do or do not get added to the final download.

For now, however, we are just going to use the download that you can get from the blue button. We'll come back to what can and cannot be customized later on.

If you've already clicked the download button, then in your downloads folder you should now have a file called Bootstrap.zip.

If you click on this file, it should open in whatever application on your system handles zip archive files. I'm running on Windows 7 and using 7-Zip.

Once your zip file manager opens, you should see a number of folders similar to the following image:



C:\Users\shawty\Downloads\bootstrap (5).zip\bootstrap\

Name	Size	Packed Size	Modified	Created	Accessed	Attributes	Encrypted	Comment
css	272 291	43 586	2013-07-27 07:50			D	-	
img	21 576	21 121	2013-07-27 07:50			D	-	
js	90 515	18 817	2013-07-27 07:50			D	-	

0 object(s) selected

Figure 2: Zip client showing Bootstrap download

The CSS folder contains the core TWB style sheets. Note that these are already preprocessed and have had any "Less" commands in them turned into real CSS statements.

The JS folder contains the core TWB JavaScript files. The files in here will be different if you've done a custom download and opted not to include some of the features TWB provides.

The IMG folder contains graphics files that TWB requires in order to display the small bitmap icons it has as part of its package.

TWB has definitions in its CSS files at approximately lines 2285 and 2309; these definitions show that the IMG folder needs to be in the same folder as the CSS. If you need to change the location, you will also need to change the rules in the CSS file.

Personally, I generally remove any path info, then place the files in the same folder as my CSS files. However, for now, we'll just make the assumption that all three folders will be in the root of your website.

The Individual Files

CSS

In the CSS folder, you'll find four different files. These should be as follows:

CSS File Name	Description
Bootstrap-responsive.css	This is the main, uncompressed, responsive TWB style sheet.
Bootstrap-responsive.min.cs	This is the minified version of the responsive TWB style sheet.
Bootstrap.css	This is the main, uncompressed TWB style sheet.
Bootstrap.min.cs	This is the minified version of the standard TWB style sheet.

The “minified” versions are as expected: they are the reduced size versions of the style sheets so as to reduce download time when your site is accessed from a remote browser. The non-minified versions are full-fat, complete with comments, white space, and everything else.

The CSS comes in two varieties. The first is just a standard, non-responsive version. This style sheet defines the grids, layouts, fonts, and everything else that TWB documents. However, it does not provide rules that allow for the layout to fluidly resize when your website is viewed on a mobile device (whether it’s a tablet, smartphone or other reduced-size display).

The "responsive" version, on the other hand, includes the extra bits needed to add this to your layout.

Typically, you'd include the main style sheet while you design your layout. Then, once it's finished, you'd include the responsive rules just after your main CSS link. This will override those rules in the main sheet that it needs to, enabling the responsive features for your site.

In the new Version 3 build, this is all now built in, as Version 3 has a mobile-first policy. It’s expected that you will work the opposite way around.

We won't be dealing with the responsive stuff until later in the book so, for now, the only one you really need to pay any attention to is the main style sheet.

JS

In the JS folder, you should (if you're using the main uncustomized download) find the following files:

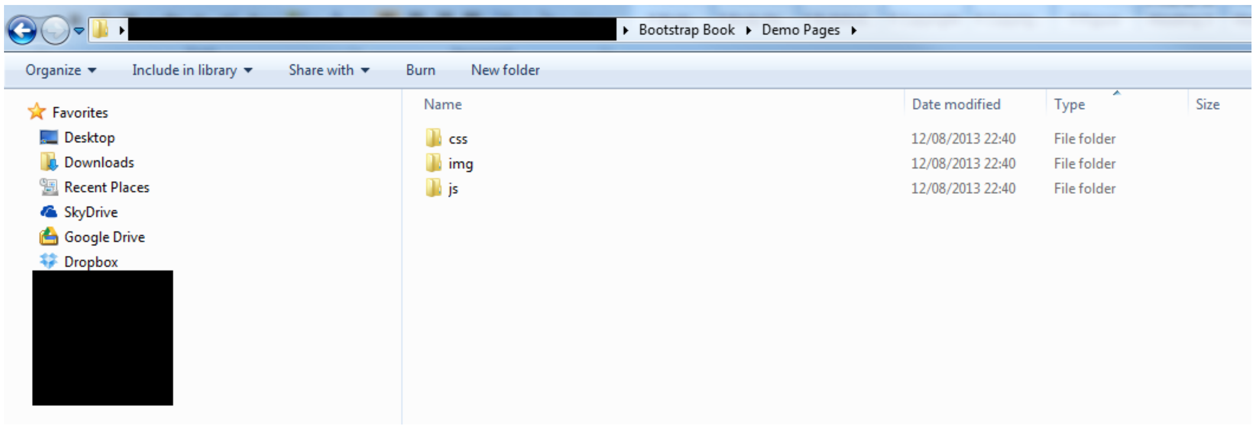
JS File Name	Description
Bootstrap.js	This is the main bootstrap JavaScript file.
Bootstrap.min.js	This is the minified version of the main JavaScript file.

Just as with the CSS, the minified file is a reduced-size version designed for the deployed version of your website. The main JS file is the only one needed if you have not decided to customize. If you're using a customized download, then you may find separate modules in here such as Modal.js or ScrollSpy.js, but I generally don't bother customizing the JS stuff as it's much easier to maintain in one file.

IMG

As previously mentioned, you should find two PNG files in here. These are the glyph icons, one set is black, the other set is white. They are used to render the icons that are provided with the framework.

Before we move on, copy the three folders from your zip file, as is, to the same folder that you'll be creating your base HTML files and templates in. In my case, that looks something like this:



Name	Date modified	Type	Size
css	12/08/2013 22:40	File folder	
img	12/08/2013 22:40	File folder	
js	12/08/2013 22:40	File folder	

Figure 3: Folder view showing layout of Bootstrap folders

The Recommended Way to Add the Files to Your Project

So, now we finally get to the good part.

As I mentioned before, the folks at TWB recommend you use the following HTML 5 code as a base for all your TWB-based projects:

```
<!DOCTYPE html>
<html>

  <head>
    <meta http-equiv="X-UA-Compatible" content="IE=9; IE=8; IE=7; IE=EDGE" />
    <meta charset="utf-8" />
    <title>My Site</title>
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <link href="css/bootstrap.css" rel="stylesheet" type="text/css" />
  </head>

  <body>

    <!-- document code goes here -->

    <script src="js/jquery-2.0.2.js" type="text/javascript"></script>
    <script src="js/bootstrap.js" type="text/javascript"></script>

  </body>
</html>
```

Code Sample 1: Twitter Bootstrap basic code

As you can see in the previous code example, you'll also need to download and add jQuery to your project. If you're not using any of the JS features in TWB, you won't need this but it's highly recommended that you do. I won't put the details in here; however, there's an excellent jQuery book in the Syncfusion e-book library and plenty of other information online on using it.

Once you have the files downloaded, create a file in your project folder and name it helloworld.html, then copy the code in previous code sample into this file using a text editor of your choosing.

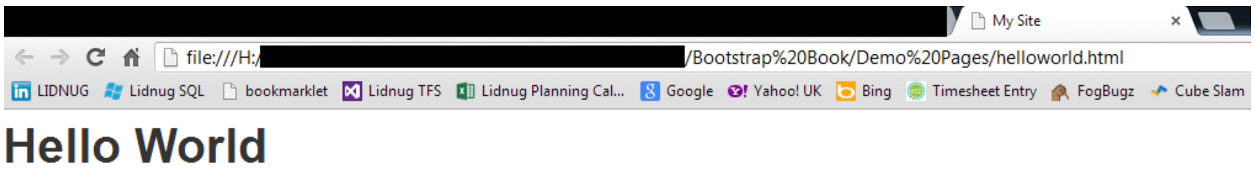
Change the following line:

```
<!-- document code goes here -->
```

To the following:

```
<h1>Hello World</h1>
```

Then load the helloworld.html file into your browser. If everything has worked as expected, you should be greeted by something that looks like the following:



file:///H:/[redacted]/Bootstrap%20Book/Demo%20Pages/helloworld.html

Hello World

Figure 4: Helloworld.html displayed in the Chrome browser

Congratulations! You've just created your first Twitter Bootstrap-enabled web page.

There's much more to come between here and the end of this book, and during the course of the various chapters we'll create several files in our project folder.

All pages we create, however, will be derived from the template shown above. From this point on, whenever I create a file, I will assume that you will create the new file and then copy and paste the base template code above into it.

Because of this, I'll no longer present the code in my samples, and will show only the HTML code for the part of TWB that I'm demonstrating.

To make things easier, you may want to do what I have done, and that is to copy helloworld.html to a file such as template.html. Then, when you need to create a new file, all you then need to do is copy your template, rename it, and then load it into your text editor.

Because everything we'll be doing here is normal basic text, any simple text editor will do. In my case I'm using Ultra Edit and the Visual Studio editor, but this is purely for things like the IntelliSense and color/syntax highlighting.

The examples presented in this book can be completed on any platform, using any plain text editor, to view your output. However, I strongly recommend that you use the Chrome browser.

Twitter Bootstrap requires an HTML 5-capable browser, as it uses a lot of the new HTML 5 and CSS3 features. This is not to say that any of the code presented here will not work in any other browser, but it will look best in Chrome. For the most part, everything looks okay in recent versions of Firefox and Internet Explorer, too.

Legacy browsers, however, are a totally different ballgame. There is some graceful degradation in some sections of the framework, and using things like the IE5 shim and toolkits such as Modernizr will help you get a similar look and feel in older browsers. But the TWB developers do not guarantee that everything will work exactly as anticipated in anything less than the current crop of up-to-date browsers.

Chapter 3 Twitter Bootstrap Scaffolding

As mentioned previously, TWB provides a grid allowing you to line up and set up columns in your document very easily.

Anyone who's ever tried to do columns in an HTML page, especially equal height and equal spaced, will know just what a chore it is.

TWB makes this easy by dividing the horizontal space on the page into 12 equal columns; these columns can be further divided down into multiples of those 12 columns, but your total can never add up to more than 12.

You don't, however, have to be stuck with 12 columns. If you do a customized download as mentioned in the last chapter, one of the options you can change using “Less” is the number of columns in your CSS.

To give you an idea of how this works, here's a screenshot of the nine-column grid as shown on the TWB website:

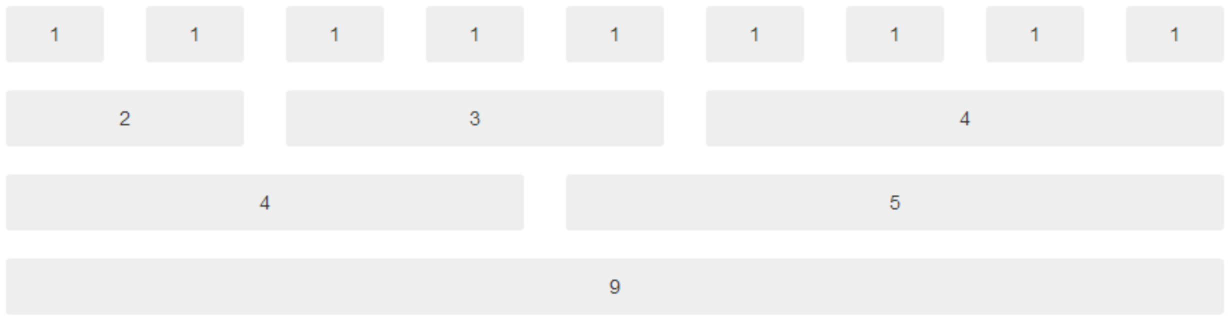


Figure 5: Nine-column grid from the Twitter Bootstrap documentation

To use this grid system, it's as simple as using the various spanX classes defined in the TWB main CSS file.

If you have the 12-column standard grid, then you'll have classes named span1 through span12. If you've customized your copy, then you'll have spanX up to the maximum number of columns you specified.

The one thing to remember, however, no matter how many columns you have, your display will always adapt to have a 940-pixel parent container, and a 724 or 1170-pixel grid width depending on your viewport size.

Any viewport width that is 767 pixels or less will always become a fluid layout, and columns will start to stack vertically and flow down the page instead of across, even without the responsive features enabled.

Making Hello World Look Better

Open up the helloworld.html file you created in the last chapter, and find the line that looks like this:

```
<h1>Hello World</h1>
```

Change this line so that it now looks like the following:

```
<div class="container">
  <div class="row">
    <div class="span12">
      <h1>Hello World</h1>
    </div>
  </div>
</div>
```

You don't have to make your indenting identical, but it will help you to read things clearly.

If you save this and hit F5 on your browser, you should see your Hello World text jump the center of the browser but with a slightly left offset.

What you've just done is created a standard TWB parent container (a page if you like), then in that container a row (much like a row in a table), and finally within that row, you've created a default, left-aligned single column spanning the whole 12 columns of the default grid.

If you now change your code to the following:

```
<div class="container">
  <div class="row">
    <div class="span6">
      <h1>Hello</h1>
    </div>
    <div class="span6">
      <h1>World</h1>
    </div>
  </div>
</div>
```

You should now see that your Hello World text has split into two parts, with a large gap between the words. The change you made has now put both words into their own separate six-column grid spanning the entire 12 column row, or in simple terms, you've just created two equal-width columns that will stretch to "balance" and fit the size and flow of your document.

Of course, you don't have to create them equal; you can specify span widths of anything you like, just as long as the total adds up to the total number of columns you have available. For example, try the following:

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

```
  <div class="row">
    <div class="span3">
      <h1>Hello</h1>
    </div>
    <div class="span9">
      <h1>World</h1>
    </div>
  </div>
</div>
```

You should see that the word "World" now jumps closer to the left but everything still looks to be in a nicely scaled proportion.

Let's add some style rules so we can see what's happening. Add the following code after the "Link" tag that includes the TWB CSS file but before the closing "Head" tag in your webpage:

```
<style>
  .span3
  {
    background-color: red;
  }

  .span9
  {
    background-color: green;
  }

</style>
```

If everything has worked as expected, you should see the following when you refresh your browser after saving the modified file:

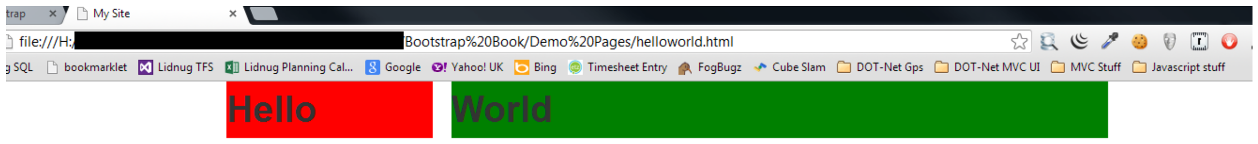


Figure 6: Helloworld.html modified to show a background color on the grids

You'll notice straight away exactly where the grids you specified using the span tags are defined. Try changing the spans in the two divs to different values and see what the effect is. Remember, though, if you want to see the background colors, you will also need to change the names on the style rules to match the ones used in your grids.

If you start changing the numbers in your spanX classes now, you'll easily see how everything is divided up. Remember, though, that you'll also need to change the style rules to match.

spanX-enabled columns can also be nested so that columns are balanced inside other columns. When you do this, however, you no longer have your 12-column grid. Your grid size instead becomes the width of the parent grid minus 1.

Revisiting our three and nine-column example we did just a moment ago, change your code so that it's now changed from the following:

```
<div class="container">
  <div class="row">
    <div class="span3">
      <h1>Hello</h1>
    </div>
    <div class="span9">
      <h1>World</h1>
    </div>
  </div>
</div>
```

To the following:

```
<div class="container">
  <div class="row">
    <div class="span6">
      <h1>Twitter Bootstrap</h1>
      <div class="span3">
        <h2>Hello</h2>
      </div>
      <div class="span3">
        <h2>World</h2>
      </div>
    </div>
  </div>
</div>
```

And change the style rules in the head section of your page to the following:

```
<style>
  .span6
  {
    background-color: red;
  }

  .span3
  {
    background-color: green;
  }
</style>
```

Save your document and hit refresh. If everything's worked as expected, you should see the following:

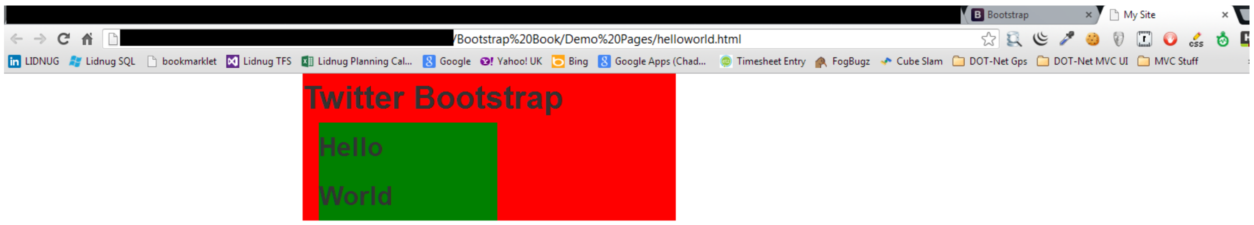

Bootstrap
My Site
[redacted]/Bootstrap%20Book/Demo%20Pages/helloworld.html
LIDNUG Lidnug SQL bookmarklet Lidnug TFS Lidnug Planning Cal... Google Yahoo! UK Bing Google Apps (Chad... Timesheet Entry FogBugz Cube Slam DOT-Net Gps DOT-Net MVC UI MVC Stuff
Twitter Bootstrap
Hello
World

Figure 7: Hello World example revised to nest rows

Why have the columns stacked on top of each other?

When you nest your spans, you need to also allow some room for the padding and other parts of the CSS to expand. If there's not enough room in your nested span, then TWB will attempt a responsive render and stack the columns as best it can.

In the segment of code you just wrote, if you change the following:

```
<div class="span6">
```

To the following:

```
<div class="span7">
```

Remember to change the style name too, then save and refresh your page. You should see things spread out a bit better and it should look something like the following:

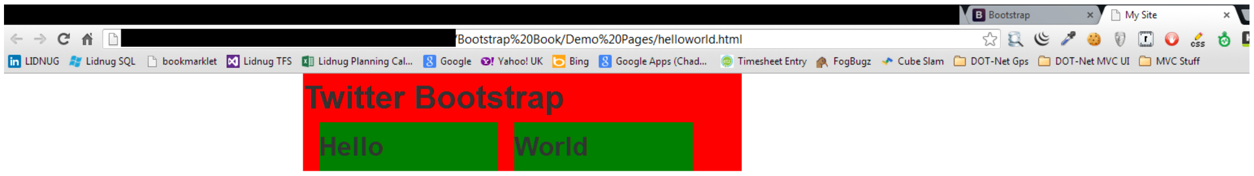

Bootstrap
My Site
[redacted]Bootstrap%20Book/Demo%20Pages/helloworld.html
LIDNUG Lidnug SQL bookmarklet Lidnug TFS Lidnug Planning Cal... Google Yahoo! UK Bing Google Apps (Chad... Timesheet Entry FogBugz Cube Slam DOT-Net Gps DOT-Net MVC UI MVC Stuff
Twitter Bootstrap
Hello
World

Figure 8: Hello World example revised to nest rows without stacking

There are some CSS rules in the style sheets that you use to make changes to this, including fluid spans (which we'll see in just a moment). But, for the most part, it's usually easier and faster to just plan your layout to always take an extra column into account when needed.

As well as a standard grid and nesting, we can also offset columns to create space where there is no content or to make space for something else you may wish to place using your own CSS rules.

Just like using the spanX classes, columns are offset by adding **`offsetX`** to the class, usually alongside the spanX controlling the width. Let's make one more change to our Hello World example.

Change the body code to:

```
<div class="container">
  <div class="row">
    <div class="span3">
      <h2>Hello</h2>
    </div>
    <div class="span3 offset2">
      <h2>World</h2>
    </div>
  </div>
</div>
```

And your styles in the header to:

```
<style>
  .span3
  {
    background-color: green;
  }
</style>
```

When you save and refresh, you should see the following:

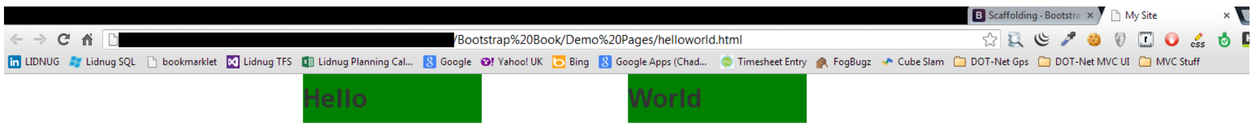


Figure 9: Hello World example showing column offsets

As you can see, there is now a two-column gap between the first and second three-column spans.

Offsets, just like normal spans, can also be nested. TWB will always keep a balanced look and feel no matter what rules you use.

Fluid Grids

The next trick that TWB has up its sleeve in the scaffolding system comes in the form of fluid grids. Fluid grids are just like the fixed grids we've already seen but, rather than use fixed pixel widths for its column layouts, it bases all sizing information on percentages.

This means that a fluid grid will use all the space it has available to display, usually to the full width of your browser window.

Let's create a new HTML document based on the template I mentioned earlier. Save this in your project folder as fluidhelloworld.html. Once that's done, add the following body code and styles as you have done previously:

```
<style>
  .span6
  {
    background-color: green;
  }
</style>

<div class="container-fluid">
  <div class="row-fluid">
    <div class="span6">
      <h2>Hello</h2>
    </div>
    <div class="span6">
      <h2>World</h2>
    </div>
  </div>
</div>
```

You'll notice that this code sample is not that much different from the previous one. In fact, if you look closely, the only two changes are to change **`container`** to **`container-fluid`** and **`row`** to **`row-fluid`**.

If you save and refresh your page, you should see the following:

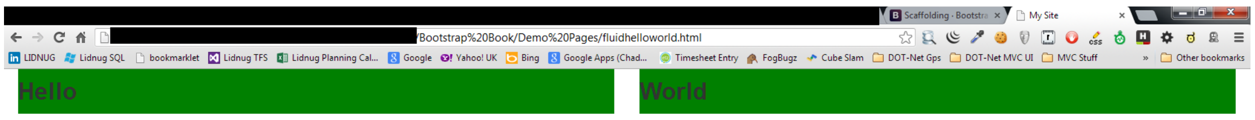

Scaffolding · Bootstra
My Site
[redacted]Bootstrap%20Book/Demo%20Pages/fluidhelloworld.html
Hello
World

Figure 10: Hello World example using fluid scaffolding layout

Immediately, you can see that your two **`span6`** columns have expanded to the entire width of the browser window. Using **`container-fluid`** makes your root container use the entire width available but still be divided up into 12 columns.

Everything that you have tried before with nesting and offsets all work in exactly the same way as with a fixed grid. You can even combine them to get the best of both worlds.

For example, you could have a container that contained **`row-fluids`** making the inner content elastic. The parent framework would be neatly centered in the page and handled responsively where required.

A common use for the container-fluid layout is to divide your page into a regular content and sidebar type arrangement using the following body code:

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

```
    <div class="span2">
      <!--Sidebar content-->
    </div>
    <div class="span10">
      <!--Body content-->
    </div>
  </div>
</div>
```

Responsive Design

And now, finally, we come to the scaffolding section, the one you've all been waiting for.

Twitter Bootstrap is not only a good general purpose CSS framework, it's also a Responsive Design framework. To enable the responsive features, however, you need to include the responsive CSS style sheet just after you include the main one in your page header, like so:

```
<!DOCTYPE html>
<html>

  <head>
    <meta http-equiv="X-UA-Compatible" content="IE=9; IE=8; IE=7; IE=EDGE" />
    <meta charset="utf-8" />
    <title>My Site</title>
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <link href="css/bootstrap.css" rel="stylesheet" type="text/css" />
    <link href="css/bootstrap-responsive.css" rel="stylesheet"
type="text/css" />
  </head>

  <body>

    <!-- document code goes here -->

    <script src="js/jquery-2.0.2.js" type="text/javascript"></script>
    <script src="js/bootstrap.js" type="text/javascript"></script>

  </body>
</html>
```

Code Sample 2: Twitter Bootstrap basic code enabled for responsive features

Note: Twitter Bootstrap 3 will be mobile-first in design. This means that you will no longer need to make sure that you add the responsive features CSS, as the responsive features will be available by default. However, everything has new class names in Version 3 so a lot of reading will be needed to know what's what.

When you enable the responsive features, it's very important that you also include the proper meta tag in the head of your page. This is the tag you can see in Code Sample 2 with the name "viewport." The responsive features in TWB will not work correctly unless this metatag is set to an appropriate value.

So, what exactly does it mean to enable responsive features?

In recent years, as JavaScript and CSS have gotten more powerful, it's become possible to actually have facilities built directly into the webpage rendering engine that can assist with page formatting for different sized screens. The best part about it is that these facilities no longer need to be implemented in many separate different files with large amounts of JavaScript and jQuery code needed to tie everything together.

Instead, all the developer needs to do is create his or her page as usual, for one single display type, and then add in some extra CSS rules to control what happens when a smaller display is encountered.

These facilities are enabled through the use of something called media queries.

TWB handles all of this for you transparently. You, as the developer, don't need to know which queries do what and how, or how some rules override other rules at specific times depending on what's being displayed.

Just as with the basic grid system, most of the responsive features in TWB are enabled simply by specifying certain class names on the appropriate elements in your document.

Please remember, however, that even though many devices and browsers are very powerful these days, you still don't want to be enabling features that you don't need. If your app is never going to be seen on anything other than a desktop PC with a widescreen, then there's no point in enabling the responsive parts of TWB. Plus, in some cases, doing so may even slow things down or cause some unexpected output.

In order for Responsive Design to work correctly, certain display width boundaries have to be used. For TWB these are defined as follows:

Display Type	Layout Width	Column Width	Gutter Width
Large Display	1200px and up	70px	30px
Default	980px and up	60px	20px

Display Type	**Layout Width**	**Column Width**	**Gutter Width**
Portrait Tablets	768px and up	42px	20px
Phones to Tablets	767px and below	Fluid columns only available	
Phones	480px and below	Fluid columns only available	

There are three categories of class names for the responsive features. These are phone, tablet, and desktop.

In each of these three categories, each has a "hidden" and "visible" class that makes sure a given element is either only visible in a given category or is always hidden in a given category.

Rather than type out a lengthy description here, the best way to show you how the classes work is to just direct you to look at "scaffolding" page in the TWB docs at http://getbootstrap.com/2.3.2/scaffolding.html.

If you scroll down to the bottom of the document, you'll see a well-laid-out table, with green squares showing the various combinations of class and device size that's easy to understand.

However, in simple terms:

- **`class="visible-phone"`** will only ever be visible on a phone-sized display whereas **`class="hidden-phone"`** will always be hidden on a phone-sized display.
- **`visible-tablet/hidden-tablet`** and **`visible-desktop/hidden-desktop`** work in exactly the same way.

By adding these classes to your spans and containers for your gridded layouts, you can make sure that sidebars are hidden on phones, but shown on desktops and all manner of other combinations. How you combine them is simply a matter of deciding what needs to appear where.

To finish this chapter off with a simple example, paste the following code into a template (as shown in Code Sample 1):

```
<div class="container-fluid">
  <div class="row-fluid">
    <div class="span2 visible-tablet visible-desktop">
      <!--Sidebar content-->
    </div>
    <div class="span10 hidden-phone">
      <!--Body content-->
    </div>
    <div class="span12 visible-phone">
```

```
      <!--Body content-->
    </div>
  </div>
</div>
```

Code Sample 3: Responsive elements

In the previous sample, on anything from 768px upwards, you'll get a 10-column fluid content area with a two-column fluid side bar. Because they are fluid, they'll expand to fit the full area available for the layout to render into.

Below 768px, the display will change to a 12-column fluid grid, the full width of the display with no side bar visible.

Or, in Twitter Bootstrap terms, on desktops and tablets, you'll have your 10/2 sidebar content division the whole width of the display; on phones you'll have only a 12/0 content area visible full width.

If you display this in your desktop browser and resize the width of the viewport, you should see quite quickly the effect it has on different display widths.

Chapter 4 Twitter Bootstrap Base CSS Classes

Like the scaffolding features, many of TWB's tricks are applied via simply adding classes onto the elements you wish to apply them to. A lot of the features available, however, are also applied directly to the element types themselves and, in many cases, groups of elements, too.

All of the general HTML elements that you know and use already have styling built in to the element level of the tags. As you've already seen in the last chapter, heading 1 <H1> has already been used.

All of the heading tags, from H1 through to H6, have some default styling for typography applied to them, as has the <p> tag for standard body copy along with <strong> for bold text, <em> for italic and <small> for the fine print. As per the HTML 5 spec, you can still use <b> and <i> for bold and italics and they still are styled the same way.

All of the above come under the Standard Typography classes in the base CSS, along with a few others.

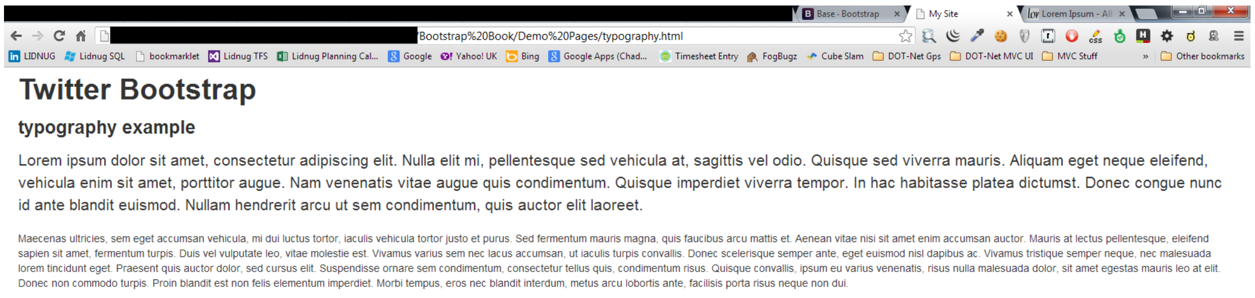
Lead Body Copy

If you use the standard body text **`<p>`** but apply the class **`lead`** to it, you'll get a paragraph styled to look like lead body copy or an emphasized opening paragraph. Put the following body text into a new file named typography.html:

```
<div class="container-fluid">
  <div class="row-fluid">
    <h1>Twitter Bootstrap</h1>
    <h3>typography example</h3>
      <p class="lead">
      Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nulla elit mi,
      pellentesque sed vehicula at.
      </p>
      <p>
      Maecenas ultricies, sem eget accumsan vehicula, mi dui luctus tortor,
iaculis     vehicula tortor justo et purus. Sed fermentum mauris magna, quis
faucibus arcu     mattis et. Aenean vitae nisi sit amet enim accumsan auctor.
Mauris at lectus  pellentesque, eleifend sapien sit amet.
      </p>
  </div>
</div>
```

Code Sample 4: Basic typography

If you've copied the code correctly into a recommended template as we have been doing, then you should end up with something that looks like the following:



Twitter Bootstrap

typography example

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nulla elit mi, pellentesque sed vehicula at, sagittis vel odio. Quisque sed viverra mauris. Aliquam eget neque eleifend, vehicula enim sit amet, porttitor augue. Nam venenatis vitae augue quis condimentum. Quisque imperdiet viverra tempor. In hac habitasse platea dictumst. Donec congue nunc id ante blandit euismod. Nullam hendrerit arcu ut sem condimentum, quis auctor elit laoreet.

Maecenas ultricies, sem eget accumsan vehicula, mi dui luctus tortor, iaculis vehicula tortor justo et purus. Sed fermentum mauris magna, quis faucibus arcu mattis et. Aenean vitae nisi sit amet enim accumsan auctor. Mauris at lectus pellentesque, eleifend sapien sit amet, fermentum turpis. Duis vel vulputate leo, vitae molestie est. Vivamus varius sem nec lacus accumsan, ut iaculis turpis convallis. Donec scelerisque semper ante, eget euismod nisl dapibus ac. Vivamus tristique semper neque, nec malesuada lorem tincidunt eget. Praesent quis auctor dolor, sed cursus elit. Suspendisse ornare sem condimentum, consectetur tellus quis, condimentum risus. Quisque convallis, ipsum eu varius venenatis, risus nulla malesuada dolor, sit amet egestas mauris leo at elit. Donec non commodo turpis. Proin blandit est non felis elementum imperdiet. Morbi tempus, eros nec blandit interdum, metus arcu lobortis ante, facilisis porta risus neque non dui.

Figure 11: Twitter Bootstrap basic typography example

Alignment and Emphasis

The base CSS has a bunch of classes that make aligning your text amazingly easy, as well as the ability to emphasize using standard color classes (all alterable via a custom download as mentioned previously). These color classes are standard on many of the styles and components that TWB provides throughout the framework.

Text alignment is really easy to use; you simply just add **`class = "text-left", class = "text-center" or class = "text-right"`** to the text you wish to align. This class works on any text element as well (and many non-text elements), so you can use it to align headings, lead body text, and all sorts of other things.

The emphasis text is just as easy to use, with the following class names:

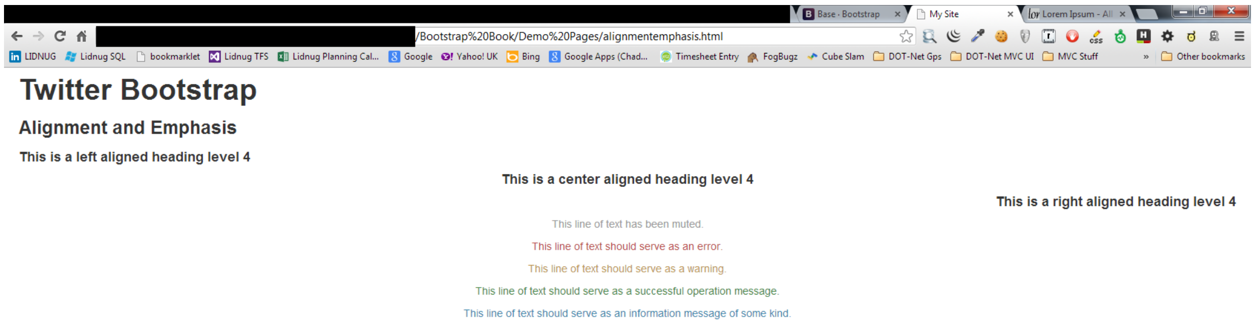
- muted: Designed for text that looks to be dumbed down or disabled.
- text-warning: Shows the text in an off-yellow color designed for warnings.
- text-error: Shows the text in a red color designed for error messages.
- text-info: Shows the text in a powder blue color, designed for general information messages.
- text-success: Shows the text in a green color, designed for successful operation and everything is OK messages.

Create a new template called alignmentemphasis.html and enter the following body text into it:

```
<div class="container-fluid">
  <div class="row-fluid">
    <h1>Twitter Bootstrap</h1>
    <h3>Alignment and Emphasis</h3>
    <h4 class="text-left">This is a left aligned heading level 4</h4>
    <h4 class="text-center">This is a center aligned heading level 4</h4>
    <h4 class="text-right">This is a right aligned heading level 4</h4>
    <p class="text-center muted">This line of text has been muted.</p>
    <p class="text-center text-error">This line of text should serve as an
error.</p>
    <p class="text-center text-warning">This line of text should serve as a
warning.</p>
    <p class="text-center text-success">This line of text should serve as a
successful operation message.</p>
    <p class="text-center text-info">This line of text should serve as an
information message of some kind.</p>
  </div>
</div>
```

Code Sample 5: Text alignment and emphasis

As with all the previous samples, once you save and load the page into your browser, you should be greeted with the following:



[redacted] Base · Bootstrap My Site Lorem Ipsum - All

[redacted]/Bootstrap%20Book/Demo%20Pages/alignmentemphasis.html

LIDNUG Lidnug SQL bookmarklet Lidnug TFS Lidnug Planning Cal... Google Yahoo! UK Bing Google Apps (Chad... Timesheet Entry FogBugz Cube Slam DOT-Net Gps DOT-Net MVC UI MVC Stuff » Other bookmarks

Twitter Bootstrap

Alignment and Emphasis

This is a left aligned heading level 4

This is a center aligned heading level 4

This is a right aligned heading level 4

This line of text has been muted.

This line of text should serve as an error.

This line of text should serve as a warning.

This line of text should serve as a successful operation message.

This line of text should serve as an information message of some kind.

Figure 12: Alignment and emphasis example

Abbreviations, Addresses, and Blockquotes

Moving on to more standard typography classes, the <abbr>, <address>, and <blockquote> tags all have styling appropriate to their use which, again, falls in line with the standard balanced look and feel that we've seen everywhere else in TWB up to now.

If you've been looking at the new HTML 5 semantic markup tags, then it's no surprise that each tag has specific meaning. If you haven't been looking, then it should be reasonably obvious the intended use and scenario for each element.

Abbreviations are generally intended to be used inline in the following manner:

```
<p><abbr title="Twitter Bootstrap">TWB</abbr> is totally awesome</p>
```

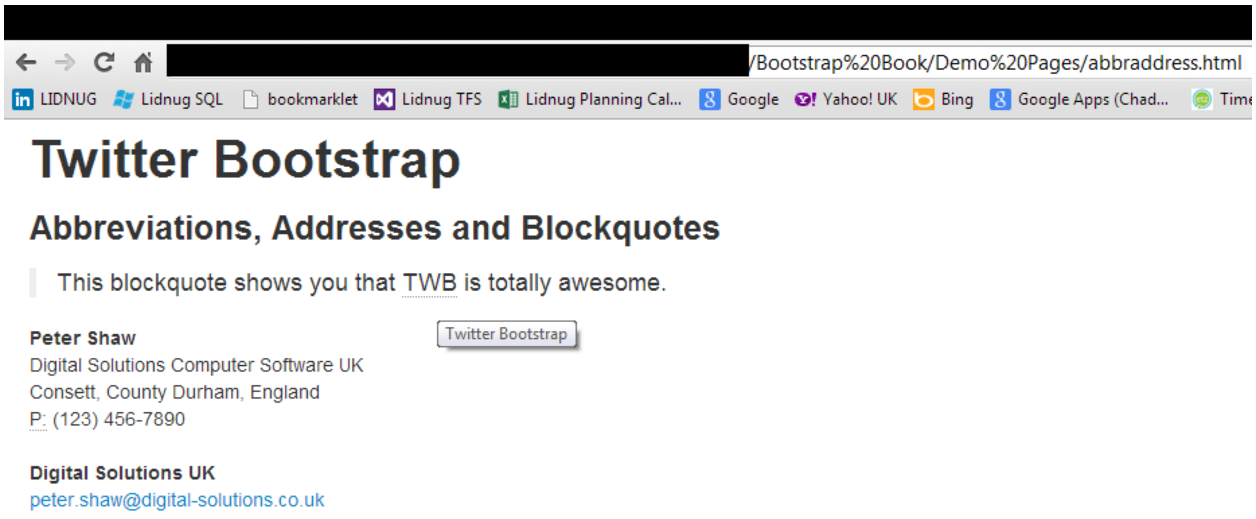
Addresses and block quotes, on the other hand, are designed to be self-describing entities used independently of other elements, and expected to be marked up as such.

Add the following code sample to a new file, then save and load the file into your browser:

```
<div class="container-fluid">
  <div class="row-fluid">
    <h1>Twitter Bootstrap</h1>
    <h3>Abbreviations, Addresses and Blockquotes</h3>

    <blockquote>
      <p>This blockquote shows you that <abbr title="Twitter
Bootstrap">TWB</abbr> is totally awesome.</p>
    </blockquote>

    <address>
      <strong>Peter Shaw</strong><br>
      Digital Solutions Computer Software UK<br>
      Consett, County Durham, England<br>
      <abbr title="Telephone number">P:</abbr> (123) 456-7890
    </address>

    <address>
      <strong>Digital Solutions UK</strong><br>
      <a href="mailto:#">peter.shaw@digital-solutions.co.uk</a>
    </address>

  </div>
</div>
```

Code Sample 6: Abbreviations, addresses, and blockquotes

The result should look as shown in the following image:


[redacted]

[redacted]/Bootstrap%20Book/Demo%20Pages/abbraddress.html

Twitter Bootstrap

Abbreviations, Addresses and Blockquotes

This blockquote shows you that TWB is totally awesome.

Twitter Bootstrap

Peter Shaw
Digital Solutions Computer Software UK
Consett, County Durham, England
P: (123) 456-7890

Digital Solutions UK
peter.shaw@digital-solutions.co.uk

Figure 13: Abbreviations, addresses and blockquotes sample

Lists

No framework should be without classes to assist the creation and manipulation of the humble list item. Twitter Bootstrap, being no exception, has six different style sets that can be used.

The first two are simply just default styling for the humble <ul> and <ol> list containers.

As with the normal parent element types, UL is an unordered list whereas OL will present an ordered list with each line being prefixed with a number.

You can also un-style the root level of your UL or OL, too. This gives us our third type of list which allows us to list items in a neat block format but to not list bullets and numbers until you get more than one level deep. This is used by adding the "unstyled" class to any normal list collection.

The fourth style of list that TWB provides is an inline list. This is used a lot with other components such as the navigation and breadcrumb components. In this style, the items in the list are laid out horizontally rather than vertically. Again, as with the un-styled type, you simply need to add a class called "inline" to the list container.

HTML 5 adds a new list type called a description list; like the standard UL and OL tags, this new tag is also given some default styling which allows definition lists to be created with simple markup that, by default, remain balanced with the rest of the document.

The sixth and final type is a horizontal description list. Adding the dl-horizontal class to the <dl> element causes the title’s <dt> elements to be lined up horizontally next to the <dd> elements, giving a nicely formatted vertical list with horizontally lined up labels.

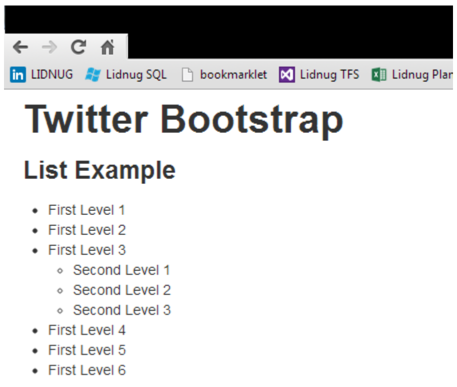
First, let's do a normal, unordered list.

Add the following body text into a document template and save it as listexample.html:

```
<div class="container-fluid">
  <div class="row-fluid">
    <h1>Twitter Bootstrap</h1>
    <h3>List Example</h3>
    <ul>
      <li>First Level 1</li>
      <li>First Level 2</li>
      <li>First Level 3</li>
      <ul>
        <li>Second Level 1</li>
        <li>Second Level 2</li>
        <li>Second Level 3</li>
      </ul>
      <li>First Level 4</li>
      <li>First Level 5</li>
      <li>First Level 6</li>
    </ul>
  </div>
</div>
```

Code Sample 7: Standard unordered list

If you save and render that in your browser, you should see a standard list as follows:


[redacted]

Twitter Bootstrap

List Example

- First Level 1
- First Level 2
- First Level 3
  - Second Level 1
  - Second Level 2
  - Second Level 3
- First Level 4
- First Level 5
- First Level 6

Figure 14: Standard unordered list example

If you now change your <ul> and </ul> tags in the previous code sample, then save and refresh your browser window; this will change to a standard ordered list with the bullet points replaced by numbered list items.

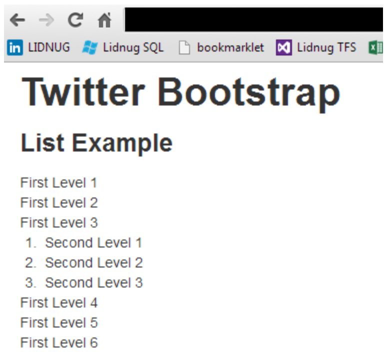
If you now change your root <ol> tag from:

```
<ol>
```

To the following:

```
<ol class="unstyled">
```

You'll see that the formatting will then be removed on immediate children items only but leave the rest intact:



Twitter Bootstrap

List Example

First Level 1
First Level 2
First Level 3
1. Second Level 1
2. Second Level 2
3. Second Level 3
First Level 4
First Level 5
First Level 6

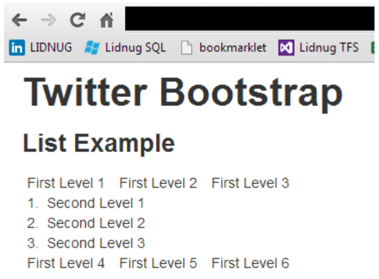
Figure 15: Ordered list with "unstyled" class applied

To demonstrate the fourth type of list available, change the unstyled class attribute to inline. Your code should now look as follows:

```
<div class="container-fluid">
  <div class="row-fluid">
    <h1>Twitter Bootstrap</h1>
    <h3>List Example</h3>
    <ol class="inline">
      <li>First Level 1</li>
      <li>First Level 2</li>
      <li>First Level 3</li>
      <ol>
        <li>Second Level 1</li>
        <li>Second Level 2</li>
        <li>Second Level 3</li>
      </ol>
      <li>First Level 4</li>
      <li>First Level 5</li>
      <li>First Level 6</li>
    </ol>
  </div>
</div>
```

Code Sample 8: Standard ordered list

When you save and render your page in the browser, you should get the following:



[redacted]

Twitter Bootstrap

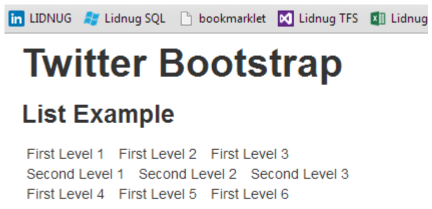
List Example

```
First Level 1   First Level 2   First Level 3
1. Second Level 1
2. Second Level 2
3. Second Level 3
First Level 4   First Level 5   First Level 6
```

Figure 16: Inline list example

As you can see, like the unstyled attribute, inline style affects only the immediate children. But, if you have anything else breaking up your document flow, it will create multiple-columned, equal-width rows from each of the items in the list.

If you also apply an inline class to the inner <ol> tag, you'll get an arrangement suitable for a multi-level, horizontal superfish menu:



Twitter Bootstrap

List Example

```
First Level 1   First Level 2   First Level 3
Second Level 1   Second Level 2   Second Level 3
First Level 4   First Level 5   First Level 6
```

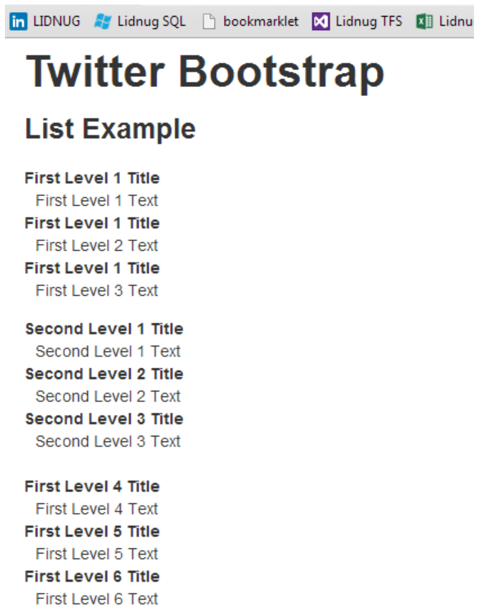
Figure 17: Nested inline list example

For the fifth and sixth types, we need to change our ordered list from Code Sample 8 into an HTML 5 definition list tag as follows:

```
<div class="container-fluid">
  <div class="row-fluid">
    <h1>Twitter Bootstrap</h1>
    <h3>List Example</h3>
    <dl>
      <dt>First Level 1 Title</dt>
      <dd>First Level 1 Text</dd>
      <dt>First Level 1 Title</dt>
      <dd>First Level 2 Text</dd>
      <dt>First Level 1 Title</dt>
      <dd>First Level 3 Text </dd>
      <dl>
        <dt>Second Level 1 Title</dt>
        <dd>Second Level 1 Text </dd>
        <dt>Second Level 2 Title</dt>
        <dd>Second Level 2 Text </dd>
        <dt>Second Level 3 Title</dt>
        <dd>Second Level 3 Text </dd>
      </dl>
      <dt>First Level 4 Title</dt>
      <dd>First Level 4 Text </dd>
      <dt>First Level 5 Title</dt>
      <dd>First Level 5 Text </dd>
      <dt>First Level 6 Title</dt>
      <dd>First Level 6 Text </dd>
    </dl>
  </div>
</div>
```

Code Sample 9: Standard definition list

If you save and load that into your browser, it should look something like this:



Twitter Bootstrap

List Example

First Level 1 Title
First Level 1 Text
First Level 1 Title
First Level 2 Text
First Level 1 Title
First Level 3 Text

Second Level 1 Title
Second Level 1 Text
Second Level 2 Title
Second Level 2 Text
Second Level 3 Title
Second Level 3 Text

First Level 4 Title
First Level 4 Text
First Level 5 Title
First Level 5 Text
First Level 6 Title
First Level 6 Text

Figure 18: Standard definition list example

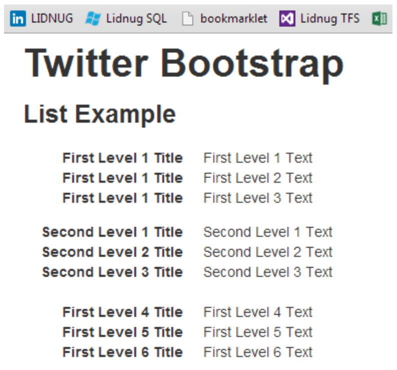
And, finally, change each of these tags:

```
<dl>
```

To the following:

```
<dl class="dl-horizontal">
```

Once you refresh your browser, you'll see the sixth type of list available:



Twitter Bootstrap

List Example

First Level 1 Title	First Level 1 Text
First Level 1 Title	First Level 2 Text
First Level 1 Title	First Level 3 Text
Second Level 1 Title	Second Level 1 Text
Second Level 2 Title	Second Level 2 Text
Second Level 3 Title	Second Level 3 Text
First Level 4 Title	First Level 4 Text
First Level 5 Title	First Level 5 Text
First Level 6 Title	First Level 6 Text

Figure 19: Definition list sample with horizontal formatting applied

Tables

I only have one thing to say about tables in Twitter Bootstrap: they are awesome.

Like anyone who has ever had to style tables will tell you, it's not a happy task. You have rules for row striping, border collapsing, and individual cell spacing. Then there are odd cells out, such as vertical label columns, or far right ends or rows where, for one cell, you have to have a separate rule just to remove one small bit of padding because it just looks out of place with the whole border in there, too.

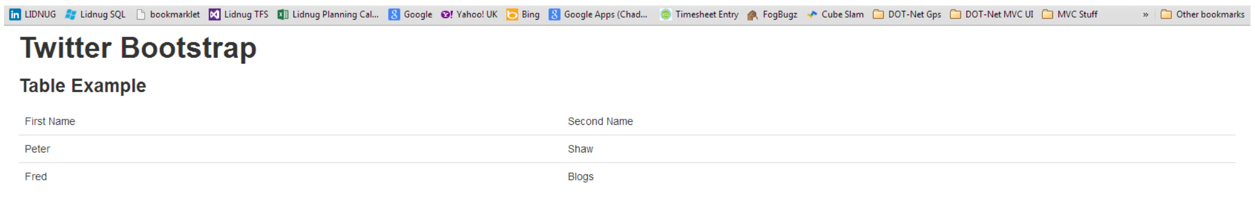
In TWB, all you need to do is make your table as normal, then apply the table class to it, and that's it.

Start a new template file, and add the following body content:

```
<div class="container-fluid">
  <div class="row-fluid">
    <h1>Twitter Bootstrap</h1>
    <h3>Table Example</h3>
    <table class="table">
      <thead>
        <tr>
          <td>First Name</td>
          <td>Second Name</td>
        </tr>
      </thead>
      <tbody>
        <tr>
          <td>Peter</td>
          <td>Shaw</td>
        </tr>
        <tr>
          <td>Fred</td>
          <td>Blogs</td>
        </tr>
      </tbody>
    </table>
  </div>
</div>
```

Code Sample 10: Standard table

As before, if you save and load this into your browser, you should see the following:



Twitter Bootstrap

Table Example

First Name	Second Name
Peter	Shaw
Fred	Blogs

Figure 20: Twitter Bootstrap basic table

It may not be much to look at, but if you notice, straight away you can see that this is a responsive table.

If you resize your browser, it will resize and balance the two columns appropriately. If you add a third column, you'll see that everything will resize nicely and the table will remain balanced.

We're still using a fluid grid here, but you can wrap this in spans with offsets and all manner of things, and everything will just resize nicely and keep the correct proportions.

You'll notice also that I've marked up the sample code using thead and tbody; this is sensible not just for TWB but for HTML 5 in general.

There are a large number of plug-ins (which you'll see later) that add extra features to TWB and need tables to be structured like this to work. For now, though, it helps us style the header row much more easily.

Just as you did when we were looking at the scaffolding earlier on in this book, add the following style rules to the table example you just created, in the page's "head" section:

```
<style>
  .table thead tr td
  {
    font-weight: bold;
    font-size: 1.2em;
  }
</style>
```

Code Sample 11: Table header styles

I'm not going to do a screenshot for this one but, once you add, save, and refresh the page in your browser you should see the first row go bold and slightly larger.

However, because I've made the rule target Twitter’s existing "table" class, you'll see that if you remove the class="table" from your table markup and then refresh the page, the bold will disappear at the same time as the TWB formatting, and re-appear once you add the class back in.

If you add a second table to your document and add the TWB table class, that will automatically have bold entries in the header line if you mark the table up using thead and tbody. If you were also using tfoot for the table footer, you can also extend the rule the same way.

I'm not going to dwell too much on it here, as it's a general CSS feature (cascading, which is the C in CSS) but Twitter Bootstrap uses it quite extensively in a lot of places such as here in the tables where all the heavy drudgery work is done for you, leaving you only needing to extend the styling on the bits with which you need to be concerned.

TWB, however, makes it much easier than that. If you change the <td> tags on the thead section to <th>, TWB will automatically make them bold for you and keep the font sizes in proportion. You can even use <th> in other places and the font will be made bold and highlighted.

You can also use a <caption>...</caption> tag between the starting <table> and <thead> tags to add a caption to your table, which TWB will then pay special attention to also.

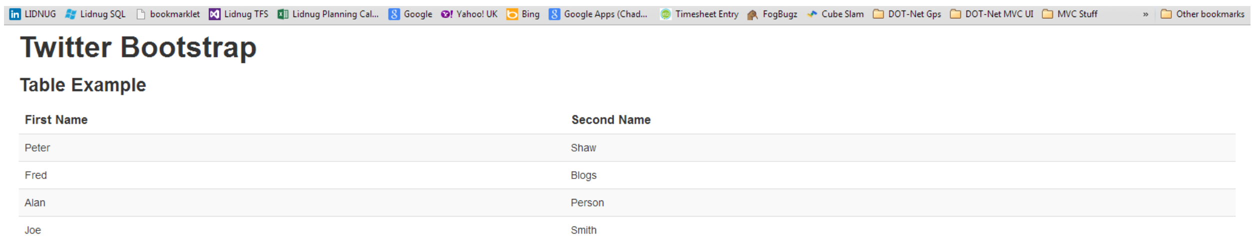
The table class also has some class extensions of its own. Say you want to make your odd/even rows striped. Try changing your table definition from:

```
<table class="table">
```

To the following:

```
<table class="table table-striped">
```

If you've done it correctly, you should see something similar to the following image (I've added a few more rows just so you can see the stripes):



Twitter Bootstrap

Table Example

First Name	Second Name
Peter	Shaw
Fred	Blogs
Alan	Person
Joe	Smith

Figure 21: Twitter Bootstrap striped table example

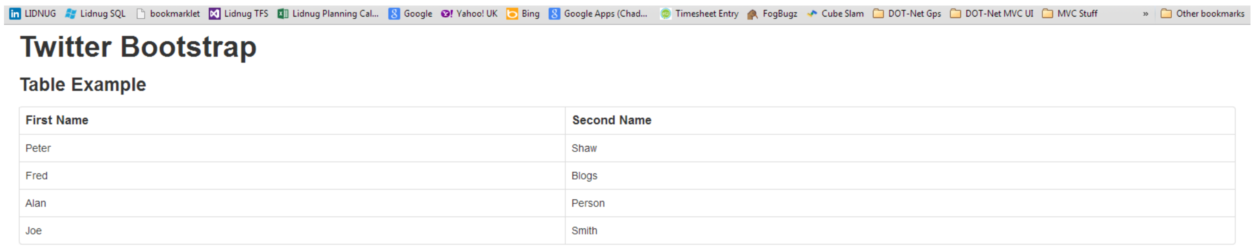
Now, let's try adding a full border. Change your table from:

```
<table class="table table-striped">
```

To the following:

```
<table class="table table-bordered">
```

You should end up with:



Twitter Bootstrap

Table Example

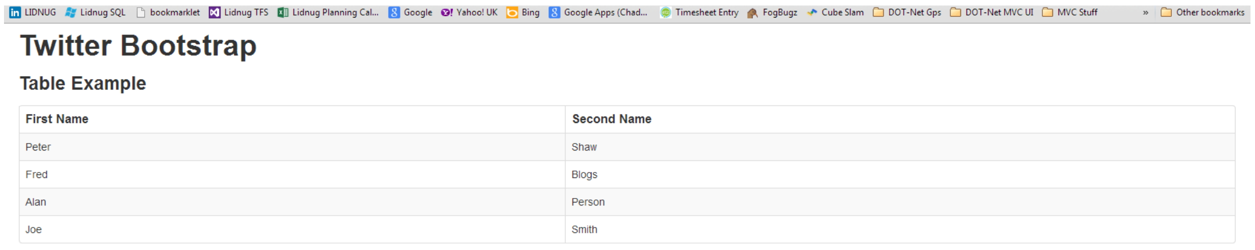
First Name	Second Name
Peter	Shaw
Fred	Blogs
Alan	Person
Joe	Smith

Figure 22: Twitter Bootstrap bordered table example

And, if you want to, there's nothing at all stopping you from adding them both:

```
<table class="table table-bordered table-striped">
```

This is displayed in the following image:



Twitter Bootstrap

Table Example

First Name	Second Name
Peter	Shaw
Fred	Blogs
Alan	Person
Joe	Smith

Figure 23: Twitter Bootstrap table with striped and bordered optional classes applied

Two other optional classes also exist. These are table-hover, which highlights the row the mouse cursor is currently hovering over, and table-condensed, which squashes the table up a bit more by cutting the padding margins around the cells in half.

As stated previously, you can use these classes on their own with just a plain table or you can combine them to make some very attractive tables. The previously mentioned table plug-in that exists for TWB also adds a handful of new styles of its own.

Before we leave tables, there's one other set of classes you'll find immensely useful and those are the optional row classes.

The optional row classes, unlike the ones we've seen so far, are applied to the <tr> tags within the table and not the table itself. They are designed to give special highlighting to that whole row.

Like the text color highlighting classes we saw previously, the intended use for these row classes is to communicate status information on the rows contents. They have similar names, too, as follows:

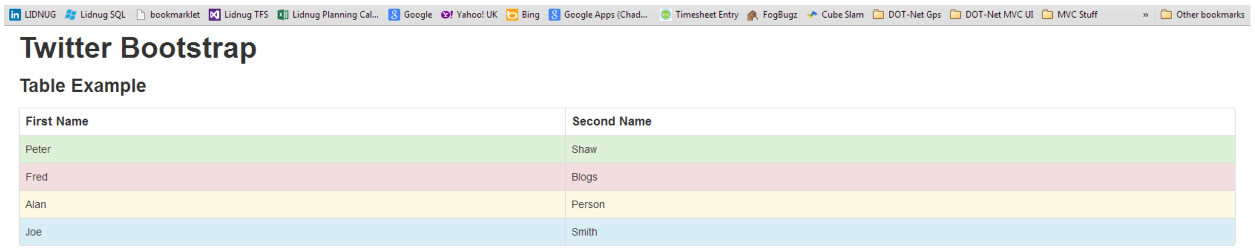
- success: Displays in the color defined for successful operations (usually green).
- error: Displays in the color defined for error operations (usually red).
- warning: Displays in the color defined for warning operations (usually yellow).
- info: Displays in the color defined for informational operations (usually blue).

Change your previous table example (from Code Sample 10) so that it looks as follows:

```html
<div class="container-fluid">
  <div class="row-fluid">
    <h1>Twitter Bootstrap</h1>
    <h3>Table Example</h3>
    <table class="table">
      <thead>
        <tr>
          <td>First Name</td>
          <td>Second Name</td>
        </tr>
      </thead>
      <tbody>
        <tr class="success">
          <td>Peter</td>
          <td>Shaw</td>
        </tr>
        <tr class="error">
          <td>Fred</td>
          <td>Blogs</td>
        </tr>
        <tr class="warning">
          <td>Alan</td>
          <td>Person</td>
        </tr>
        <tr class="info">
          <td>Joe</td>
          <td>Smith</td>
        </tr>
      </tbody>
    </table>
  </div>
</div>
```

Code Sample 12: Optional row classes

If you save and refresh that in your browser, you should see something like this:



Twitter Bootstrap

Table Example

First Name	Second Name
Peter	Shaw
Fred	Blogs
Alan	Person
Joe	Smith

Just like any other stock colors, these colors can be modified by doing a custom download as described earlier, in the chapter on installing Twitter Bootstrap in your project.

Chapter 5 Forms

The features added for forms and webpage interaction in TWB are, put simply, nothing short of stunning. There is just so much stuff to cover that it really does deserve an entire chapter unto itself.

Before Twitter Bootstrap came along, I could easily spend a large chunk of my time on the UI of an application, doing nothing more than trying to lay out forms and make everything look nice and lined up.

Any web developer who's ever had to do a form on a webpage will know exactly what I mean, and they'll know just how painful an experience it can be to get it right.

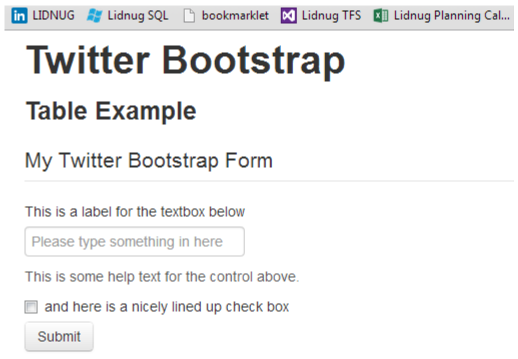
Best of all, forms marked up under TWB don't need any master class added to the form as you did with tables. All you simply have to do is mark your forms up using recommended semantic HTML 5 layout techniques. TWB will simply just do what it needs to and give you a default, left-aligned, vertically-stacked responsive form.

Start a new template HTML file just as we have done all the way throughout this book, and add the following body content to it:

```
<div class="container-fluid">
  <div class="row-fluid">
    <h1>Twitter Bootstrap</h1>
    <h3>Forms Example</h3>
    <form>
      <fieldset>
        <legend>My Twitter Bootstrap Form</legend>
        <label>This is a label for the textbox below</label>
        <input type="text" placeholder="Please type something in here">
        <span class="help-block">This is some help text for the control
above.</span>
        <label class="checkbox">
          <input type="checkbox"> and here is a nicely lined up check box
        </label>
        <button type="submit" class="btn">Submit</button>
      </fieldset>
    </form>
  </div>
</div>
```

Code Sample 13: Basic form code

You should have a form in your webpage that looks something like the following once you save and load your document:



Twitter Bootstrap

Table Example

My Twitter Bootstrap Form

This is a label for the textbox below

Please type something in here

This is some help text for the control above.

and here is a nicely lined up check box

Submit

Figure 24: Basic Twitter Bootstrap form

As you can see, it takes very little to get something that's visually pleasing to the eye, and as with everything else, you can easily override and customize any individual bits you wish to.

Like the table classes, however, there are a whole load of optional classes you can add to your forms.

Some are designed for special cases such as the search classes, which are designed to make a very modern-looking, rounded search box, right through to the more generic classes for the individual input types.

Search Forms

We've all seen them, and the visually pleased crowd loves them. That's right, those nice-looking search boxes with the sharply rounded corners and nicely lined up controls that we see on the menu bar of most websites.

Well, using Twitter Bootstrap you can create one of these things in just four lines of HTML 5 markup. Don't believe me? Try changing the forms example you created just a moment ago to the following:

```
<div class="container-fluid">
  <div class="row-fluid">
    <h1>Twitter Bootstrap</h1>
    <h3>Forms Example</h3>
    <form class="form-search">
      <input type="text" class="input-medium search-query">
      <button type="submit" class="btn">Search</button>
    </form>
  </div>
</div>
```

Code Sample 14: Basic form code

If all goes well, you should see the following when you refresh your page:

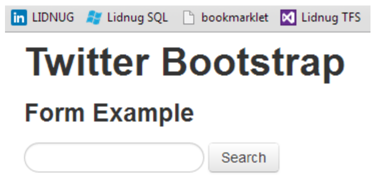


Figure 25: Search form example

Something else you get for free when you start using the form classes in Twitter Bootstrap is highlighting on the fields themselves. I'll show you more on this soon but, for now, if you click in the search box, you should see it highlight with a blue aura as the following image shows:

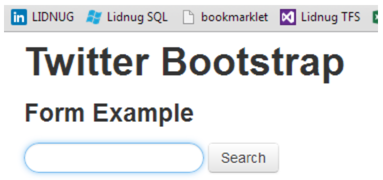


Figure 26: Search form example with highlighted field

Search forms are only one specialized example of how TWB can help you; another is the inline forms.

Inline Forms

Inline forms are typically used in places like navigation bars and page headers for elements such as miniature login forms. To make an inline form in Twitter Bootstrap, it should come as no surprise that all you need to do is add yet another class name (I did mention that this was all about CSS and JavaScript).

Change the code sample you made in Code Sample 13 earlier so that it looks like the following code:

```
<div class="container-fluid">
  <div class="row-fluid">
    <h1>Twitter Bootstrap</h1>
    <h3>Forms Example</h3>
    <form class="form-inline">
      <input type="text" class="input-small" placeholder="Email">
      <input type="password" class="input-small" placeholder="Password">
      <label class="checkbox">
        <input type="checkbox"> Remember me
      </label>
      <button type="submit" class="btn">Sign in</button>
    </form>
  </div>
</div>
```

Code Sample 13: Inline form code

Save and refresh, and your search form should now have changed to look like this:

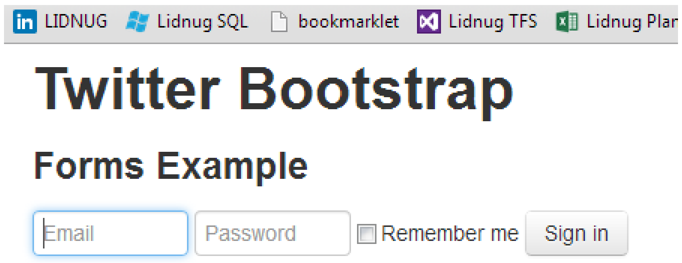


Figure 27: Inline form example

As you'll see later when we get to the subject of things like navigation bars, it would be really easy to take this form and add it into your site-wide navigation.

Before we move on to what's available for the individual form controls, let's look at one more type of form: the horizontal form.

Horizontal Forms

To use the horizontal form feature, you add the form-horizontal class to your form. But, unlike the other forms you've seen so far, you have to do a little more work. Different collections of controls need to be wrapped in control groups and your form layout has to be structured in a specific way.

As you dig deeper into TWB, you'll find that there are a few cases where it's about more than just adding the correct classes to your elements. In some cases (particularly when we get to the section on components), it's about the order in which you create your HTML elements and the actual tags you use.

We won't worry about this too much at the moment but it is something that you need to keep at the back of your mind, as you will need to remember it later on.

Before I present you with the next code sample, I'm going to take a moment to introduce you to control groups.

Control Groups

When you create a form normally, you often don't really give it much thought. You just add labels and input fields, and then you add the various buttons and fix up the form handlers to accept your data inputs.

If you look logically at your form from a high level however, through a user's eyes, you will soon start to see patterns.

The labels above or next to the input fields, the collections of radio buttons, and the rows of buttons all have a separate reason for existing on the page.

If you were to draw boxes around each of these lines, you'd see that you in most cases have a one-column table.

Now, add into the mix things such as validation messages, help text, and other artifacts designed to help the end user navigate the form. Then, consider things such as ARIA compatibility hints for things like screen readers and navigation aids to help those who have impairments. All of a sudden you have a whole jumble of things that require some structure to make related things look related.

The creators of Twitter Bootstrap solved this problem with something called a control group.

A control group is nothing more than a standard <div> tag that surrounds a given row of elements in your form (similar to how a row class works in the base scaffolding). But the other classes in TWB understand and respect the meaning that this parent element has on the controls it contains. We'll see this in better detail very soon when I cover validation states.

Continuing on with the Horizontal Form

So, getting back to our form examples, change the body code in your form so that it looks like the following:

```
<div class="container-fluid">
  <div class="row-fluid">
    <h1>Twitter Bootstrap</h1>
    <h3>Form Example</h3>
    <form class="form-horizontal">
      <div class="control-group">
        <label class="control-label" for="inputEmail">Email</label>
        <div class="controls">
          <input type="text" id="inputEmail" placeholder="Email">
        </div>
      </div>
      <div class="control-group">
        <label class="control-label" for="inputPassword">Password</label>
        <div class="controls">
          <input type="password" id="inputPassword" placeholder="Password">
        </div>
      </div>
      <div class="control-group">
        <div class="controls">
          <label class="checkbox">
            <input type="checkbox"> Remember me
          </label>
          <button type="submit" class="btn">Sign in</button>
        </div>
      </div>
    </form>
  </div>
</div>
```

Code Sample 15: Horizontal form

As with previous examples, if you save and refresh your page, you should see something that looks like this image:

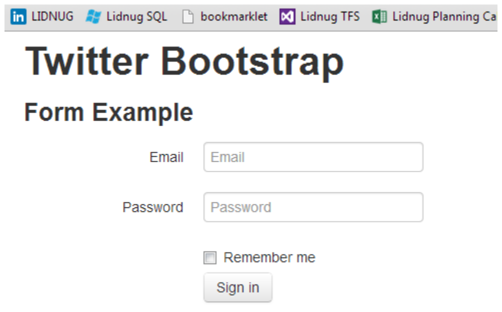


Figure 28: Twitter Bootstrap horizontal form example

As you can see, the use of structure and control groups has made sure that the text label for the check box is neatly lined up, the labels for the input fields are nicely lined up, and the buttons and other furniture all share a common left margin.

With just a few extra lines of code and a sprinkling of classes, your form has a balanced look to it that is pleasing to the eye.

Validation States

If you recall just a moment ago when I was discussing control groups, I mentioned something called validation states. Validation states are optional classes that you can add to your control groups that then affect every element contained in that group in some visual way, allowing you to convey extra information to the end user about the state of the inputs in the form.

Just like the optional classes you've seen previously to style table rows and text items, the validation state classes follow a similar naming convention and default color set that is easily customizable by using the custom download and/or "Less" version of the main TWB style sheet in your project. The names used by the different validation styles are as follows:

- warning: Shows the validation state in a yellow warning color.
- error: Shows the validation state in a red color to indicate an error state.
- info: Shows the form state in a powder blue color to indicate an informational state.
- success: Shows the form in a green color to indicate all is okay.

Let's try these states and see what they look like.

In Code Sample 14, change the first control group in your form to include the warning validation state on the first control group in the form. The code should look something like:

```
<div class="control-group warning">
  <label class="control-label" for="inputEmail">Email</label>
  <div class="controls">
    <input type="text" id="inputEmail" placeholder="Email">
  </div>
</div>
```

If you then save and refresh your form, you should be greeted with the following:

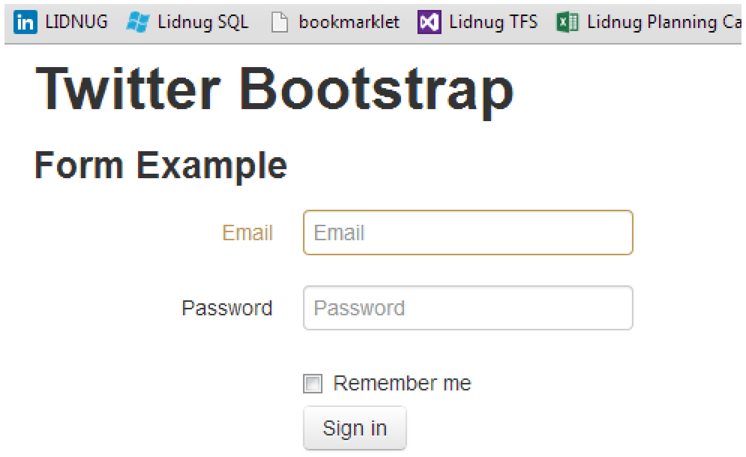


Figure 29: Horizontal form showing warning state applied to first control

Try changing the warning class to one of the other three and you should see the different colors affecting the entire group. I've added three new controls to my form just to show them all alongside each other.

I've also added the following span:

```
<span class="help-inline">Example help text</span>
```

It's marked up with a help-inline class to each of the <div> tags containing the actual input controls so you can see that the validation state also affects them, too.

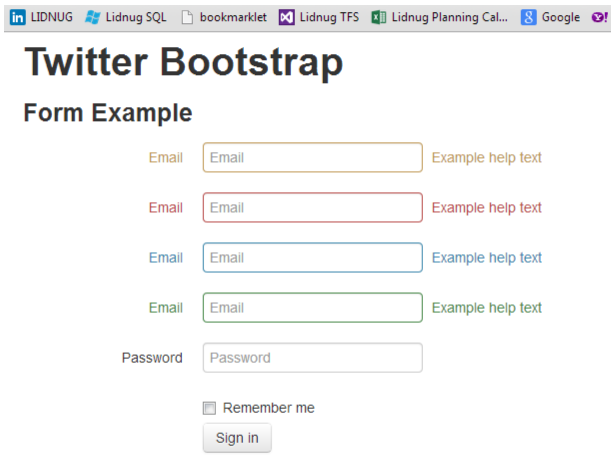


Figure 30: Validation states example

Individual Controls Support

Most of the standard input types in the HTML standard are supported, styled, and handled automatically by Twitter Bootstrap. If you use the specific input types added by the HTML 5 specification where browser support permits and styles can be changed, TWB will also attempt to handle the rendered controls appropriately.

Unfortunately, browser support for the new HTML 5 control types is still a little patchy at present. Chrome is considered to be the browser with the best support for the different types at present; however, even that is still a little short of its target.

Don't be put off from using them, though. One of the good things about HTML 5 is that, if the browser does not support the control type you're trying to render, then it will (or it should) render as a plain old text box instead.

The following code example, while not a Twitter Bootstrap-specific bit of code, shows all the new input types, marked up in appropriate control groups:

```
<div class="container-fluid">
  <div class="row-fluid">
    <h1>Twitter Bootstrap</h1>
    <h3>Form Inputs Example</h3>
    <form>
    <div class="control-group">
      <label class="control-label">Text Input</label>
      <div class="controls">
        <input type="text" />
      </div>
    </div>
    <div class="control-group">
```

```
        <label class="control-label">Search Input</label>
        <div class="controls">
          <input type="search" />
        </div>
      </div>
      <div class="control-group">
        <label class="control-label">Email Input</label>
        <div class="controls">
          <input type="email" />
        </div>
      </div>
      <div class="control-group">
        <label class="control-label">Url Input</label>
        <div class="controls">
          <input type="url" />
        </div>
      </div>
      <div class="control-group">
        <label class="control-label">Telephone Input</label>
        <div class="controls">
          <input type="tel" />
        </div>
      </div>
      <div class="control-group">
        <label class="control-label">Number Input</label>
        <div class="controls">
          <input type="number" />
        </div>
      </div>
      <div class="control-group">
        <label class="control-label">Range Input</label>
        <div class="controls">
          <input type="range" />
        </div>
      </div>
      <div class="control-group">
        <label class="control-label">Date Input</label>
        <div class="controls">
          <input type="date" />
        </div>
      </div>
      <div class="control-group">
        <label class="control-label">Month Input</label>
        <div class="controls">
          <input type="month" />
        </div>
      </div>
      <div class="control-group">
        <label class="control-label">Week Input</label>
        <div class="controls">
```

```
          <input type="week" />
        </div>
      </div>
      <div class="control-group">
        <label class="control-label">Time Input</label>
        <div class="controls">
          <input type="time" />
        </div>
      </div>
      <div class="control-group">
        <label class="control-label">Datetime Input</label>
        <div class="controls">
          <input type="datetime" />
        </div>
      </div>
      <div class="control-group">
        <label class="control-label">Datetime local Input</label>
        <div class="controls">
          <input type="datetime-local" />
        </div>
      </div>
      <div class="control-group">
        <label class="control-label">Color Input</label>
        <div class="controls">
          <input type="color" />
        </div>
      </div>
      </form>
    </div>
  </div>
```

Code Sample 16: HTML 5 input control types

If I render this in my default copy of Chrome (Version 28.0.1500.95 of this writing), this is what I see:

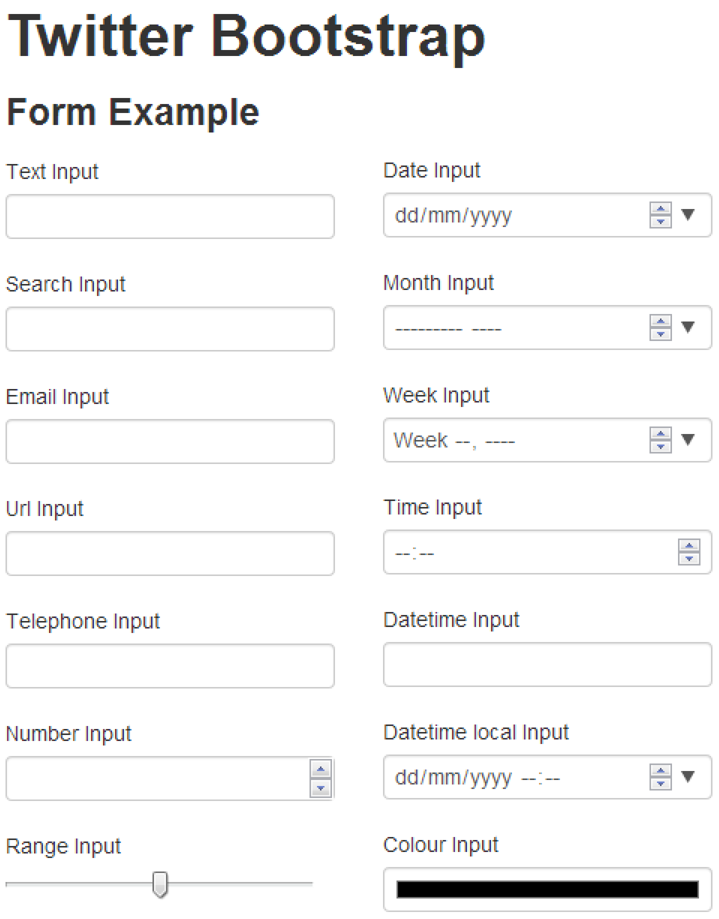


Figure 31: HTML 5 input types rendered in Twitter Bootstrap

The keen-eyed of you will notice that's two columns, not one. And, no, TWB didn't help me there (although it could have). I actually created the screenshot in that manner purely for the illustrations in this book.

If you try the code in Code Sample 15 in different browsers, you'll get different mileage depending on which browser you use. You'll also notice, especially on the mobile browsers, that some of them will adapt the onscreen keyboard or input system to reflect the input type being used.

You should also notice that, if you attempt to break the validation rules in the input types (such as entering letters in the number input, or an incorrectly formatted URL or email in the appropriate field types), the validation state colors as shown previously will take effect, even before you attempt to submit the form.

Aside from the standard input types, using the correct form for grouping classes and marking up your forms correctly, there is still one thing that always seems to be difficult to do: aligning label names with check boxes and radio buttons.

By wrapping your check boxes and radio buttons actually inside a label element as follows:

```
<label class="checkbox">
  <input type="checkbox" value="">
  This is my option
</label>
```

And adding the class of "checkbox" or "radio" to the label as required, Twitter Bootstrap will line the label up with the control so you never again have to worry about whether or not the layout of your options panels looks correct.

If you further add the inline optional class, you'll get your check boxes and radio buttons to all appear inline on the same row horizontally:

```
<label class="checkbox inline">
  <input type="checkbox" value="">
  A
</label>
<label class="checkbox inline">
  <input type="checkbox" value="">
  B
</label>
<label class="checkbox inline">
  <input type="checkbox" value="">
  C
</label>
```

This will produce the following output:

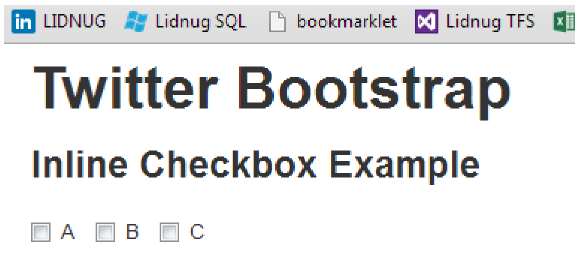


Figure 32: Inline check boxes

You can add the "inline" optional class to radio buttons in the same manner.

The rest of the input types—such as Select and Options and Select with an attribute of multiple—will also be styled to match other Twitter components and, as such, can also be used with the different span classes and validation styles, just like everything else. Unfortunately, the humble file upload control is not yet subject to the same functionality.

There are some add-on components that allow you to use a file select control and have its visual appearance look like the rest of Bootstrap. But these are largely just text boxes with extended control sets; the actual file control is hidden behind the scenes using a technique I wrote about a few years ago on my blog. The reason for this is, because of security fears, file controls don't have the same freedom of styling and JavaScript operation that other controls have. And this is to be expected, as you wouldn't want a script being able to steal files from your PC, now would you?

Extended Form Controls

There's another good reason to correctly structure your HTML 5 code when using Twitter Bootstrap, and that's so you can take advantage of the extended controls that it makes available.

An extended control is a control where you have a text label that appears to be part of the control, such as an @ sign for a Twitter name input or two decimal points after a number input, as shown in the following image:

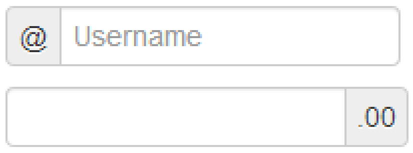


Figure 33: Extended input types

Creating these is incredibly easy; you can reproduce the above two examples using the following HTML:

```
<div class="input-prepend">
  <span class="add-on">@</span>
  <input class="span2" id="prependedInput" type="text"
placeholder="Username">
</div>
<div class="input-append">
  <input class="span2" id="appendedInput" type="text">
  <span class="add-on">.00</span>
</div>
```

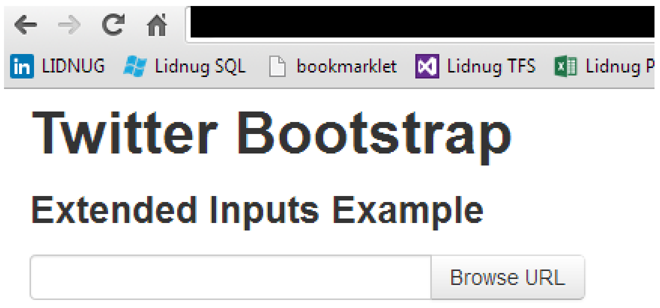
As you can see, it's as simple as wrapping the control and its pre- or post-appended part in a standard "<div>" with the class type of "input-prepend" or "input-append" depending on which end of the input control you want to add your extension to.

You can also add a class containing both, and then put a span at either side of the input control, allowing you to add an extension on both ends at the same time.

Text is not the only thing you can extend a control with either. Many of Twitter Bootstrap's components can be used in the same way. For example, the following code:

```
<div class="input-append">
  <input class="span2" id="appendedInputButton" type="url">
  <button class="btn" type="button">Browse URL</button>
</div>
```

This will create an HTML 5 URL input control with an attached browse button that looks like the following:

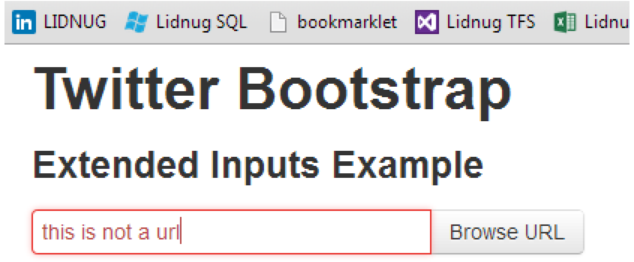

[redacted]

Twitter Bootstrap

Extended Inputs Example

Figure 34: Twitter Bootstrap extended input control with button

The validation states continue to work as expected:



Twitter Bootstrap

Extended Inputs Example

Figure 35: Twitter Bootstrap extended input control with button, showing an error in validation

You can add multiple buttons, segmented buttons with dropdown menus, styled search inputs that look like the premade search form you saw earlier, and all sorts of other additions to make your input fields really stand out.

Other classes that can be added to individual inputs are things such as "uneditable-input" and "disabled." Use both of these together on a span that holds a value from a form to display that data in a form group and style it just like any other form input. The result is not actually an input control but rather an element that has the same visual appearance but does not take part in form submission. The idea is that it can be used to display a record number in a database app but allow that record number to remain unedited and be submitted as a hidden parameter when the form is posted back to the web server.

We also have help text classes as you saw previously in the validation states example. Help text can take one of two classes:

- help-inline: Attempts to keep the text in line with its control.
- help-block: Breaks the help text out into a paragraph, starting on a new line but still attempts to keep the text left-aligned with the form controls.

Control Sizing

The final thing I'd like to cover before we finish this chapter on forms is how to use the control sizing features.

It will come as no surprise that the standard spanX classes in the base scaffolding can be used and apply to inputs just as they apply to any other class or element in the framework. Twitter Bootstrap, however, has some additional tricks up its sleeve for input fields.

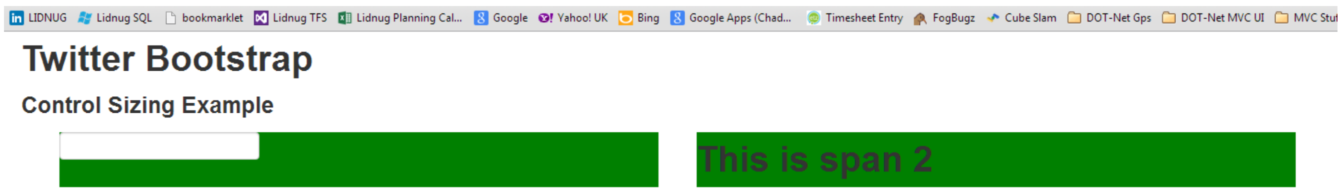
First off, we can make any input a block-level input. When we do this, the control will expand to fill one row of its entire parent container. Consider the following example of HTML code:

```
<div class="container-fluid">
  <div class="row-fluid">
    <h1>Twitter Bootstrap</h1>
    <h3>Control Sizing Example</h3>
    <span class="span5">
      <form>
        <input type="text">
      </form>
    </span>
    <span class="span5">
      <h1> This is span 2</h1>
    </span>
  </div>
</div>
```

And a style tag just to show where those span5s are (just as we did back at the beginning in the chapter on scaffolding):

```
<style>
  .span5
  {
    background-color: green;
  }
</style>
```

Then, when you save and render the page in the browser, you should see the following:



Twitter Bootstrap

Control Sizing Example

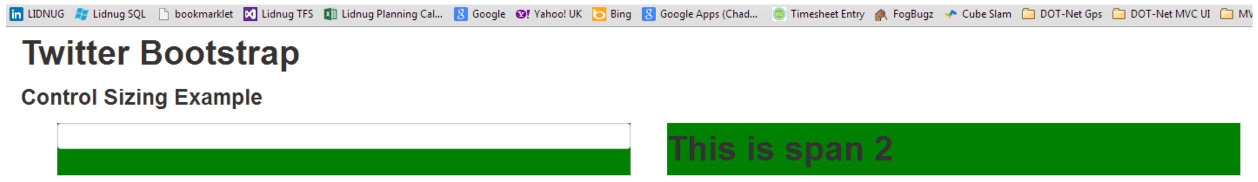
This is span 2

Figure 36: Control sizing example 1

As you can see, the input element is tucked right up in the top corner of our span element. You could fiddle with the various offsetX and spanX classes to get the sizing correct, or you can simply add the class input-block-level to your input element like so:

```
<input type="text" class="input-block-level">
```

The result should be something like this:



Twitter Bootstrap

Control Sizing Example

This is span 2

Figure 37: Control sizing example 1 with block-level optional class set

You need to take care of the vertical padding, etc., yourself but, as you can see, you don't need to worry about the input—just the container it’s in—and it will then size itself appropriately.

As previously mentioned, grid sizing can use the spanX classes directly on an input element, but instead of using the row class, TWB provides an extra class specifically for this purpose. This is the controls-row class.

This should be used wherever you would use a row class (as shown in the scaffolding chapter) but where the rows will be housing input controls primarily.

Anywhere you would reasonably expect to have a class of "controls" (see the control groups and validation state examples), you can also expect that to be a normal point in the document where you might want to use a "controls-row" class.

As an example:

```
<div class="controls controls-row">
  <input class="span4" type="text" placeholder=".span4">
  <input class="span1" type="text" placeholder=".span1">
</div>
```

When the previous code is rendered, it should give the following:

If you don't want to use the fixed grid, you can also use a relative size grid. In some ways, this is similar to the fluid layout items we saw previously but it's designed specifically for laying out your form controls.

Start a new template document, and then add the following body code to it:

```
<div class="container-fluid">
  <div class="row-fluid">
    <h1>Twitter Bootstrap</h1>
    <h3>Relative Sizing Example</h3>
  </div>
  <div class="row-fluid">
    <span class="span5">
      <form>
      <input class="input-mini" type="text" placeholder=".input-mini"><br />
      <input class="input-small" type="text" placeholder=".input-small"><br
/>
      <input class="input-medium" type="text" placeholder=".input-medium"><br
/>
      <input class="input-large" type="text" placeholder=".input-large"><br
/>
      <input class="input-xlarge" type="text" placeholder=".input-xlarge"><br
/>
      <input class="input-xxlarge" type="text" placeholder=".input-
xxlarge"><br />
      <input class="input-block-level" type="text" placeholder=".input-block-
level (for   comparison)"><br />
    </form>
  </span>
</div>
</div>
```

Code Sample 17: Relative sizing

If you save and render this with a style rule to show color behind the span5 class (green in my case), you should see something like the following:

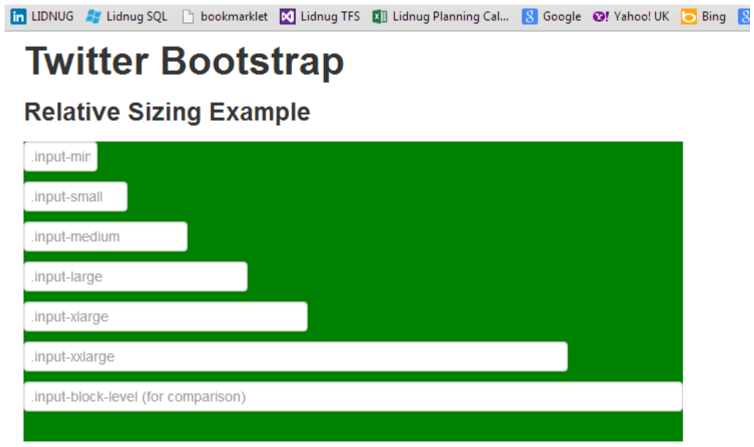


Figure 38: Twitter Bootstrap relative sizing example

Notice I've also added a block size class to one of the elements, just to compare things.

If you now start playing with the span5 on the parent span holding the controls, you should see that each of them always maintains a relative size to the container holding the elements, once again making sure that you don't need masses of nested classes to control the layout and balance of your forms.

There are a couple more minor things that are present in the forms section such as the "form-actions" class which, when used with a correctly laid out form, will ensure that control buttons and other similar things remain lined up with their parent form controls.

In this chapter, we saw one of the most impressive aspects of Twitter Bootstrap; however, some of the more astute among you may be wondering where the buttons and such are. That is the subject of the next chapter.

Chapter 6 Buttons

Just like forms, Twitter Bootstrap has a huge array of classes to help you make the best use of buttons in your forms.

Admittedly, there are not as many classes as there are for forms, but the richness of the UIs you can create using them is just as spectacular.

Before we begin, though, please note that the TWB button classes are only designed to be used with two different types of tags: the <button> tag and the <a> tag. Which one you use is entirely up to you, but remember the age-old rule: if you're trying to initiate an action, then you should use a button tag and an appropriate "get", "post," or other restful method on your form to ensure the correct response.

If you're creating buttons that are intended to perform full page navigation, then you should endeavor to use the anchor tag to provide the functionality.

When you style all your elements with the button classes, all of your buttons and anchors will look the same and it can be difficult to remember which element is where.

I'm sure the last thing any of us want is to style a delete button on a table with an anchor tag, and then watch in horror as a web spider deletes every record in our database because it tried to follow the links underneath.

I could do an entire chapter on the pitfalls of idempotence but I'll leave that to the HTTP Succinctly book that's already present in the Syncfusion library.

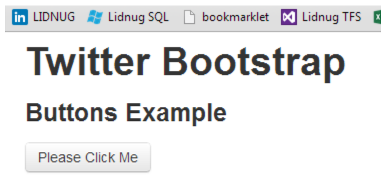
So, how do we turn a button or anchor element into a Twitter Bootstrap button? Well, just like with anything else in the framework, we add a class. In this case, the main class to be added is "btn."

Start a new template document and add the following body code to it:

```
  <div class="container-fluid">
    <div class="row-fluid">
      <h1>Twitter Bootstrap</h1>
      <h3>Buttons Example</h3>
      <button class="btn">Please Click Me</button>
    </div>
  </div>
```

Code Sample 18: Basic Twitter Bootstrap button

If all goes well, you should see the following in your browser:



Twitter Bootstrap

Buttons Example

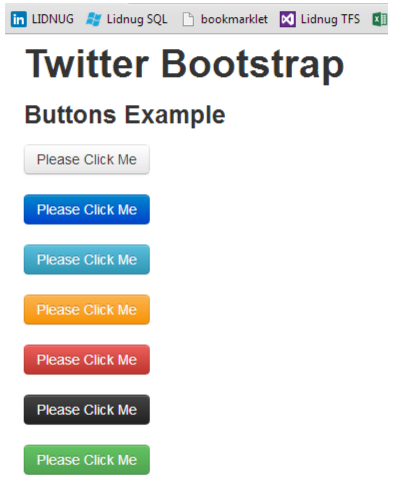
Figure 39: Basic Twitter Bootstrap button

As with everything else in TWB, you can also apply a consistent set of styles as optional style classes to add to your buttons.

Add the following extra buttons just under the button tag you placed in your new document from Code Sample 17 (Yes, I know I've spaced things out wth "<br>" tags but hey, this is just an example):

```
<br /><br /><button class="btn btn-primary">Please Click Me</button>
<br /><br /><button class="btn btn-info">Please Click Me</button>
<br /><br /><button class="btn btn-warning">Please Click Me</button>
<br /><br /><button class="btn btn-danger">Please Click Me</button>
<br /><br /><button class="btn btn-inverse">Please Click Me</button>
<br /><br /><button class="btn btn-success">Please Click Me</button>
```

If you save and refresh your page, you should hopefully see this:



Twitter Bootstrap

Buttons Example

Figure 40: Twitter Bootstrap base button styles

The best way for me to describe the button styles, or the thinking behind them at least, is just to use the information from the Twitter Bootstrap documents.

Most of them have a similar purpose to the classes you've seen previously; however, the naming is a little different.

- btn-primary: Generally strong blue, designed to highlight default actions.
- btn-info: Aqua, used as an alternative to the default styles.
- btn-success: Green, indicates a successful or positive action.
- btn-warning: Yellow, indicates caution should be taken with this action.
- btn-danger: Red, indicates a dangerous or potentially negative action.
- btn-inverse: Alternate dark gray button, not tied to a semantic action or use.

One thing to note when using these button styles (and the TWB authors also mention it in the documents) is that Internet Explorer 9 does not render buttons with these styles applied correctly.

The background gradients are not cropped on the rounded corners and text has a rather nasty grey shadow that the authors have been unable to fix. From my own point of view, I cannot say if this has been resolved in Version 3 yet or not because, as noted earlier, I still need to go through the differences.

I can confirm, however, that in both Internet Explorer 10 and Firefox version 23.0, the buttons example renders and looks correct.

There are also handy classes for setting button sizes that are relative to the size of everything else. These classes are:

- btn-large
- btn-default
- btn-small
- btn-mini

As with the relative sizes used by the form components in the last chapter, you can also apply a class of "btn-block" to your buttons; this allows your button to expand to fill its parent container with a suitable margin, in exactly the same way as the block-level optional class works for a form input field.

Replace the body code in your HTML file with the following code:

```
  <div class="container-fluid">
    <div class="row-fluid">
      <h1>Twitter Bootstrap</h1>
      <h3>Buttons Example</h3>
      <button class="btn">Please Click Me</button>
      <br /><br /><button class="btn btn-primary btn-large">Please Click Me
(Large)</button>
      <br /><br /><button class="btn btn-info btn-default">Please Click Me
(Default)</button>
      <br /><br /><button class="btn btn-warning btn-small">Please Click Me
(Small)</button>
      <br /><br /><button class="btn btn-danger btn-mini">Please Click Me
(Mini)</button>
      <br /><br /><button class="btn btn-inverse btn-small">Please Click Me
```

```
(Small)</button>
      <br /><br /><button class="btn btn-success btn-block">Please Click Me
(Block)</button>
    </div>
  </div>
```

Code Sample 19: Button sizing

If everything worked as expected, here's the output you should see:

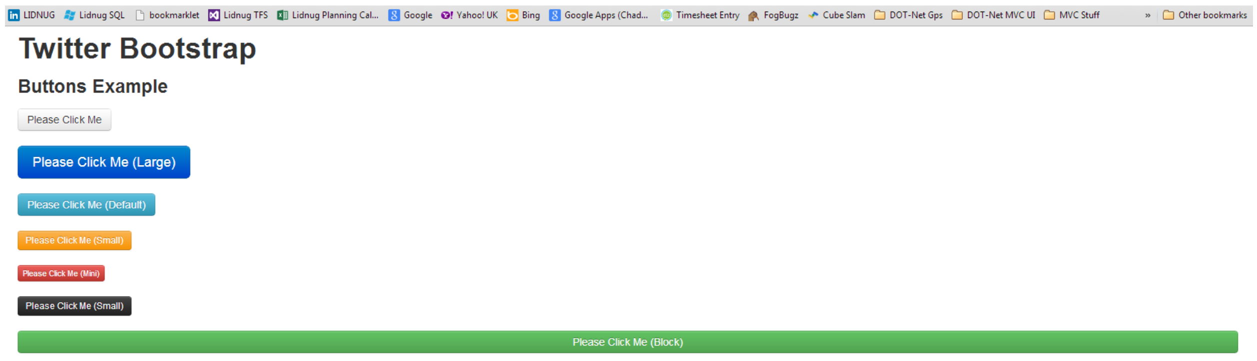


Figure 41: Button sizing example

Now, we've done nothing more than add different sizing classes to the buttons and styles that we already set up in the last example. As with everything in TWB, you can stack classes up on top of each other. I find the mini size to be particularly helpful in navigation bars for things such as logout buttons; however, if you're going to use them for this, I strongly advise also using the "inline" optional classes as seen in the chapter on forms.

Buttons can also be made to show a disabled state by adding the "disabled" class name to their class list; toggling this in client-side Javascript is a great way to give a consistent and user-focused experience in the application’s front end.

Icons

Twitter Bootstrap comes, by default, with a huge number of icons. These icons can be used in many different places such as bullet lists, menu items, and navigation items, but I mention them here as most people will want to use them with buttons.

To use an icon, it's as easy as hijacking the original <i> tag in the following manner:

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

That line of code will display the Heart Icon in front of the text "icon-heart."

Using them in a button is just as easy. If you want an icon in front of the text on your button, format your button, and input an anchor tag as follows:

```
<button class="btn"><i class="icon-heart"></i> Button Text</button>
```

This should give you something like:

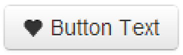


Figure 42: Default button with a heart icon

There are an absolute ton of icons provided (far too many to list here in this book). If you look at the bottom of the base CSS page in the Bootstrap docs, you'll see a cheat sheet of them all. There are also a few extra replacement icon projects available such as Font Awesome that aim to replace the built-in icons with better-looking, scalable vector font versions. They are well worth checking out.

Button Groups

If all TWB could do was make your buttons look pretty, there wouldn't be much reason for an entire chapter on them. As you may have guessed, however, they can do far more.

Many of you may be familiar with Office-style toolbars where options are grouped together into collections. For example, "left," "center," and "right" alignment buttons, as follows:

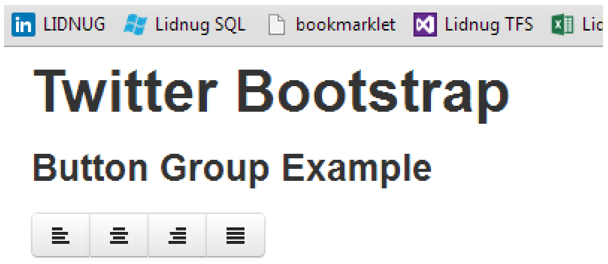


Figure 43: Twitter Bootstrap button group example

This can be very easily produced with the following body code added to a recommended HTML 5 template:

```
  <div class="container-fluid">
    <div class="row-fluid">
      <h1>Twitter Bootstrap</h1>
      <h3>Button Group Example</h3>
      <div class="btn-group">
        <button type="button" class="btn"><i class="icon-align-
left"></i></button>
```

```
        <button type="button" class="btn"><i class="icon-align-
center"></i></button>
        <button type="button" class="btn"><i class="icon-align-
right"></i></button>
        <button type="button" class="btn"><i class="icon-align-
justify"></i></button>
      </div>
    </div>
  </div>
```

Button groups are purely cosmetic and offer no extra functionality to show the buttons as one related group graphically. That is, they don't maintain state for options and radios, etc.

That's not to say that they can't be used for this, however. But you have to start using the TWB JavaScript API to make that functionality work. So, for now, I leave that until we get to the chapter on working with TWB's JavaScript features.

You can also make your button groups stack vertically. If you change the classes on the div holding your button group from:

```
<div class="btn-group">
```

To the following:

```
<div class="btn-group btn-group-vertical">
```

Then save and refresh your page, and you should see your button group change to this:

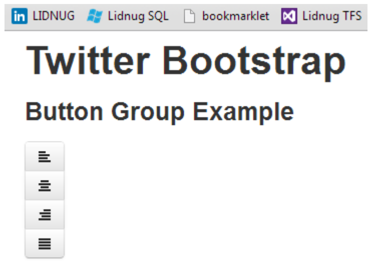


Figure 44: Vertical grouped button example

The final thing to cover before we move on is button dropdowns.

Button Dropdowns

Any button in Twitter Bootstrap can have a dropdown menu attached to it. However, for things to work correctly you must do the following:

1. Have the Twitter Bootstrap JavaScript file referenced in your project.
2. Have your buttons wrapped in a button group (even if it's only one button).
3. Use a btn-toolbar class if you have more than one button.

Once you have your HTML 5 code structured correctly to do this, there's one more new concept you'll need to learn, and that's TWB's use of HTML 5 data attributes.

Data What?

Under HTML 5, you can provide a new type of attribute to your elements known as a data attribute. All data attributes must start with "data-" but, after that, it's entirely up to you what you use them for and how.

I'm not going to go into too much detail about this. But, if you're using one of the more popular libraries in your client code such as jQuery (just as TWB does), you can use data parameters to provide information to client-side JavaScript without ever having to touch any code directly.

Twitter Bootstrap makes heavy use of data attributes (you see more of these later on in the JavaScript chapter). But for now, all you need to know to make a button that incorporates a dropdown menu is the "data-toggle" attribute.

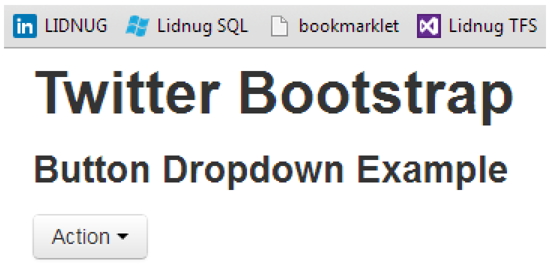
When you create your button using a standard anchor tag and then apply this data attribute to parent anchor, any <ul> element embedded within what would normally be the displayed text will be turned into a dropdown menu and be attached to your button.

Create a new HTML 5 template file as we have done with all the other examples in this book, and add the following body code to it:

```
<div class="container-fluid">
  <div class="row-fluid">
    <h1>Twitter Bootstrap</h1>
    <h3>Button Dropdown Example</h3>
    <div class="btn-group">
      <a class="btn dropdown-toggle" data-toggle="dropdown" href="#">
        Action
        <span class="caret"></span>
      </a>
      <ul class="dropdown-menu">
          <li>Menu 1</li>
          <li>Menu 2</li>
          <li>Menu 3</li>
      </ul>
    </div>
  </div>
</div>
```

Code Sample 20: Basic dropdown button

Save the file and refresh your page. You should see the following:

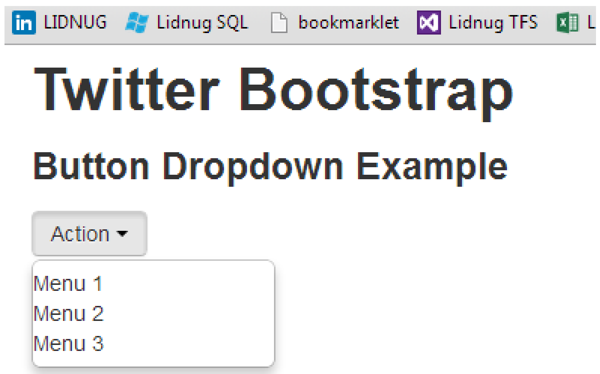


Twitter Bootstrap

Button Dropdown Example

Figure 45: Basic dropdown button

If you click on the button, you should get a menu with three options displayed under it:



Twitter Bootstrap

Button Dropdown Example

Figure 46: Dropdown button in its clicked state

As of now, I've not applied any of the dropdown menu classes to the options below as I'll be showing you more of these when we cover navigation bars in the components chapter. But there are optional classes that you can add to make the padding better, provide separator lines and, using the icon classes I introduced earlier, add icons to your menu entries.

You can also make your dropdown buttons into split buttons very easily. What's the difference?

Well, with a split button, the caret (or dropdown arrow) on the end of your button becomes the only part that triggers the menu for the button. Clicking on the main button section will actually trigger whatever action that button has in its href (or form submission if you're using a button tag).

To make Code Sample 20 into a split button, simply change it to the following code:

```
<div class="container-fluid">
  <div class="row-fluid">
    <h1>Twitter Bootstrap</h1>
    <h3>Button Dropdown Example</h3>
    <div class="btn-group">
      <button class="btn">Action</button>
```

```
        <button class="btn dropdown-toggle" data-toggle="dropdown">
          <span class="caret"></span>
        </button>
        <ul class="dropdown-menu">
            <li>Menu 1</li>
            <li>Menu 2</li>
            <li>Menu 3</li>
        </ul>
      </div>
    </div>
  </div>
```

Code Sample 21: Dropdown button with split button

Once saved and refreshed, the result from this should be a button with a dropdown menu and a split section to the button, as the following image shows:

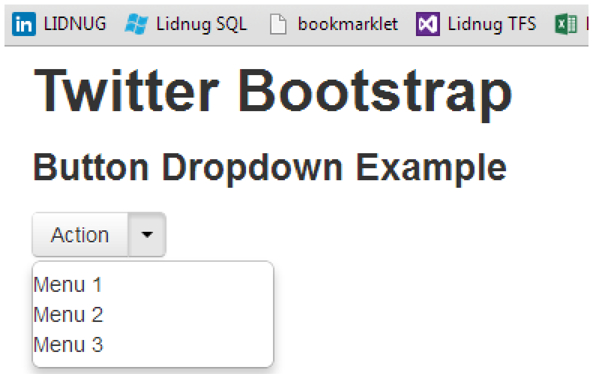


Figure 47: Dropdown button with split example

All the different button styles, classes, and sizes that you've seen so far all work perfectly well with dropdowns and button groups. Also, if you add the optional extra class "drop-up" to your parent button groups in the previous dropdown examples, you can make your dropdown menus appear above the button rather than below.

I've used these techniques with great success in toolkits such as Knockout.js where I've actually gotten different menu options to change the behavior of the split button in a Knockout view model.

They come in handy in places such as data editing grids where you can change the parent button to "edit," "delete," "disable," etc., values, and then just group your editing commands together for easy-to-navigate database management operations.

In the next chapter, we'll look at the various other Twitter Bootstrap components and start delving more into those parts of the framework that are built from correctly structured sets of elements and JavaScript functionality.

Chapter 7 Components

If you've been reading the Twitter Bootstrap docs as we've been moving through this book, you will have undoubtedly seen that the main menu has a section called components.

Personally, I think the separation that the TWB team has provided here is a little bit misleading as you can find sections on multiple things in multiple menus.

The buttons, for example, are mostly listed under the Base CSS section, but button groups are in components and radio/check box groups are in JavaScript.

My whole take on this, however, is that a component in Twitter Bootstrap is something that's just a bit more than adding a simple class to an element to style it in a certain way. Components are created from collections of elements put together in a particular way and use a particular sequence of classes in order to produce a final composite product.

That said, some of the features listed under the components section in the docs are enabled by simply adding a single class—elements such as labels, badges, and progress bars fall into this category.

Anyway, enough debating. Let's kick off this chapter with dropdown menus.

Dropdown Menus

A basic dropdown menu is produced using a standard, unordered list element with the class "dropdown-menu" applied to it as follows:

```
<ul class="dropdown-menu">
  <li><a tabindex="-1" href="#">Menu 1</a></li>
  <li><a tabindex="-1" href="#">Menu 2</a></li>
  <li><a tabindex="-1" href="#">Menu 3</a></li>
  <li class="divider"></li>
  <li><a tabindex="-1" href="#">Menu 4</a></li>
</ul>
```

However, because of the way dropdowns work, you need another element to trigger it to pop up. You saw an example of this in the last chapter with dropdown buttons.

To change the code we've just seen into a correctly formatted dropdown, we need to wrap the entire UL in a div or some other container that is marked with the "dropdown" class, and then within that div, we need to add an element that will act as the menus trigger, like so:

```
<div class="dropdown">
  <!-- Trigger element, a button, text, etc. goes here -->
  <ul class="dropdown-menu">
```

```
    <li><a tabindex="-1" href="#">Menu 1</a></li>
    <li><a tabindex="-1" href="#">Menu 2</a></li>
    <li><a tabindex="-1" href="#">Menu 3</a></li>
    <li class="divider"></li>
    <li><a tabindex="-1" href="#">Menu 4</a></li>
  </ul>
</div>
```

The commented part in that last sample shows where you would add the trigger element which, in most cases, will normally be a simple link or button element. But it could just as easily be a div styled in some way unique to your app or a standard header tag. All that matters is that the element is marked with the correct classes and data attribute to make it trigger the dropdown and show it on screen.

Start a new template file in the same manner you have been throughout this book, and then add the following body code:

```
<div class="container-fluid">
  <div class="row-fluid">
    <h1>Twitter Bootstrap</h1>
    <h3>Dropdown Menu Example</h3>
    <div class="dropdown">
      <button class="btn dropdown-toggle" data-
toggle="dropdown">Menu</button>
      <ul class="dropdown-menu">
        <li><a tabindex="-1" href="#">Menu 1</a></li>
        <li><a tabindex="-1" href="#">Menu 2</a></li>
        <li><a tabindex="-1" href="#">Menu 3</a></li>
        <li class="divider"></li>
        <li><a tabindex="-1" href="#">Menu 4</a></li>
      </ul>
    </div>
  </div>
</div>
```

Code Sample 22: Dropdown menu

Once you save the file and render it in a browser, you'll see something similar to the first dropdown button you created in the last chapter. If, however, you now click on the button and allow the menu to display, you should see something like the following:

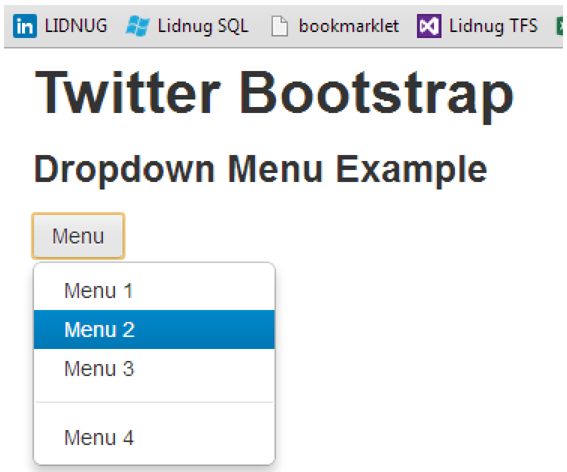


Figure 48: Twitter Bootstrap popup menu example

Immediately, you should be able to see that the menu has a shadow behind it, a vertical divider, proper padding and formatting, and highlighting of the menu items as you hover your mouse cursor over them.

We can also give menu items on a menu a disabled appearance by applying the "disabled" class alongside any other classes on the <li> element for that option.

If we now change the list item element for Menu 2 from the following:

```
<li><a tabindex="-1" href="#">Menu 2</a></li>
```

To the following:

```
<li class="disabled"><a tabindex="-1" href="#">Menu 2</a></li>
```

Then save and refresh the page, we should now see that when the menu is popped up, option number 2 is now disabled:

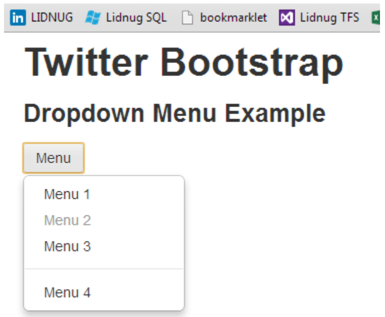


Figure 49: Dropdown menu with disabled menu item

You'll also be able to see that the highlight bar and other such effects just don't appear when you hover over the disabled option, and you're not able to click on the option to select it.

Submenus

You can also nest and cascade submenus in a TWB dropdown simply by nesting an <a> tag to act as the trigger element and then adding <ul> tags and their associated <li> elements inside of standard <li> tags in your upper-level parent container before finally applying the class "dropdown-submenu" as the following code shows:

```
<div class="container-fluid">
  <div class="row-fluid">
    <h1>Twitter Bootstrap</h1>
    <h3>Dropdown Sub Menu Example</h3>
    <div class="dropdown">
      <button class="btn dropdown-toggle" data-
toggle="dropdown">Menu</button>
      <ul class="dropdown-menu">
        <li><a tabindex="-1" href="#">Menu 1</a></li>
        <li><a tabindex="-1" href="#">Menu 2</a></li>
        <li><a tabindex="-1" href="#">Menu 3</a></li>
        <li class="divider"></li>
        <li class="dropdown-submenu">
          <a tabindex="-1" href="#">Sub Menu</a>
          <ul class="dropdown-menu">
            <li><a tabindex="-1" href="#">Sub Menu 1</a></li>
            <li><a tabindex="-1" href="#">Sub Menu 2</a></li>
            <li><a tabindex="-1" href="#">Sub Menu 3</a></li>
          </ul>
        </li>
      </ul>
    </div>
  </div>
</div>
```

Code Sample 23: Nested submenu example

If you change the body code in your dropdown page to that in Code Sample 22, and then save and refresh your page, you should be able to expand the menu and submenu as the following image shows:

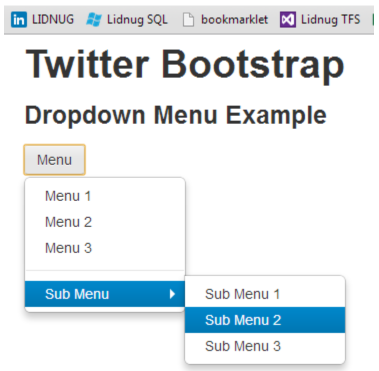


Figure 50: Dropdown menu with submenus example

Navigation Components

I would guess most of you have been waiting for this part of the book. Why do I say that? Well, when I first started to use Twitter Bootstrap, the one thing I was drawn to almost immediately was its great-looking navigation bars.

However, navigation bars are only one of the many navigation aids that TWB provides to help make the navigation in your app much more of a first-class citizen.

There are classes for tabs, pills, link lists, side menus, and more.

Most of the features provided are just as easy to use as everything else you've seen so far.

One thing to note before we dive in, nearly all of TWB's navigation aids are based on creating your link lists using standard UL/LI tags (just as with the dropdown menus in the last section). So, nesting and correct tag order is a must, as you'll get some pretty funky side effects if you miss a closing tag in some places.

Tabs

Tabs are tabs, nothing special, nothing complicated. Simple, elegant, but very functional tabs allow you to break a single screen up into multiple pages.

There are two main classes you need to know: "nav" and "nav-tabs." Another class that you'll use repeatedly is "active." This is illustrated in the following set of tabs:

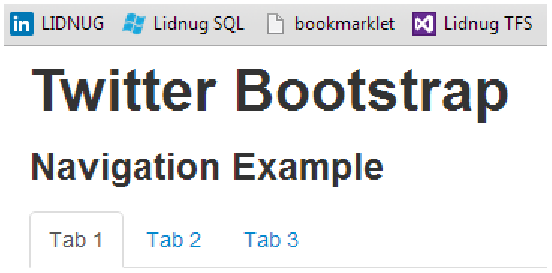


Figure 51: Basic tabs navigation example

Use the following code to create them:

```
<div class="container-fluid">
  <div class="row-fluid">
    <h1>Twitter Bootstrap</h1>
    <h3>Navigation Example</h3>
    <ul class="nav nav-tabs">
      <li class="active">
        <a href="#">Tab 1</a>
      </li>
      <li><a href="#">Tab 2</a></li>
      <li><a href="#">Tab 3</a></li>
    </ul>
  </div>
</div>
```

Code Sample 24: Basic tabs code

Using tabs doesn't stop there. Rather than just use them as a navigation aid, you can actually tie them fully together to provide a full paged content setup which, when linked correctly, will see the tab bar actually change the pages for you—without any extra JavaScript or other handling code that's usually associated with swapping tab panels.

To make a full set of tabbed pages, however, you need to obey a few rules:

1. Every LI tag in your parent tab bar must have a href that references a div ID with the content inside of it.
2. Every div that has contents for a page must have a unique ID that can be seen by the tab bar.
3. Each div with content must immediately follow the <ul> holding the tab bar, and this ul plus all associated div tags must be wrapped in a parent div with the class of "tabbable" applied to it, and
4. The collection of content divs must be wrapped in a single div tag immediately following the <ul> with the "tab-content" class applied to it.

Note, also, that each individual tab content div has the class "tab-pane" applied to it, and that by default the first tab has an "active" class to match the selected "active" class in the actual tab bar itself.

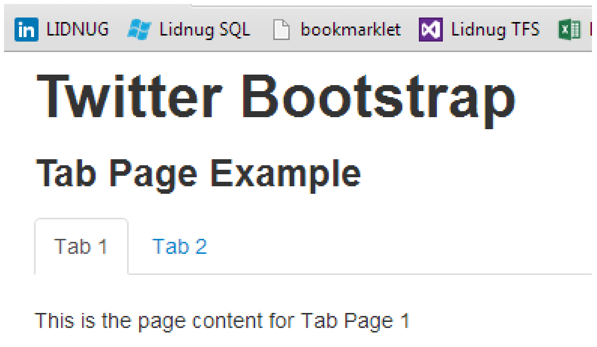
Once you have the required structure set up, the correct classes applied, and all your IDs matching, the final thing you need to make sure of is that you have data-toggle attributes on each of the anchor tags in your list items. This is so that the TWB JavaScript knows to toggle your tabs for you.

Start a new template document and add the following body code to it:

```
<div class="container-fluid">
  <div class="row-fluid">
    <h1>Twitter Bootstrap</h1>
    <h3>Tab Page Example</h3>
    <div class="tabbable">
      <ul class="nav nav-tabs">
        <li class="active"><a href="#tab1" data-toggle="tab">Tab 1</a></li>
        <li><a href="#tab2" data-toggle="tab">Tab 2</a></li>
      </ul>
      <div class="tab-content">
        <div class="tab-pane active" id="tab1">
          <p>This is the page content for Tab Page 1</p>
        </div>
        <div class="tab-pane" id="tab2">
          <p>This is the page content for Tab Page 2</p>
        </div>
      </div>
    </div>
  </div>
</div>
```

Code Sample 25: Tab pages example

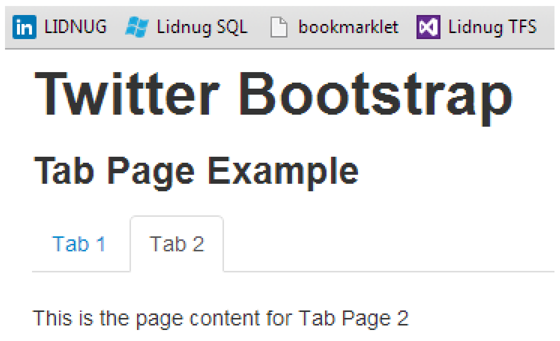
Save and load the page into your browser. You should find the following when you render it:



This is the page content for Tab Page 1

Figure 52: Tab page example

If you click on tab number two, you should find that the content changes to whatever content you included in tab number two. You should also see that the active classes update in the same way:



Twitter Bootstrap

Tab Page Example

This is the page content for Tab Page 2

Figure 53: Tab page example show the second tab

Tab pages can also be made to fade in and out of existence when switching by including the "fade" class to any div that also has "tab-pane" applied to it.

Lastly, tabs are not just limited to being shown along the top of the content. On the master parent div element (the one with the class "tabbable"), you can add any of the following to change the position of the tabs:

- tabs-below: Places the tabs below the content.
- tabs-left: Places the tabs to the left of the content.
- tabs-right: Places the tabs to the right of the content.

I'll leave experimenting with those, however, as an exercise to the reader.

Pills

Pills are very much like tabs and are created in exactly the same way with the same structure. The only difference is that pills look like small, rounded rectangles rather than page tabs, and they use a different class, "nav-pills," rather than "nav-tabs" as we did in the previous examples.

If you go back to Code Sample 25 and change the line:

```
<ul class="nav nav-tabs">
```

To the following:

```
<ul class="nav nav-pills">
```

Then save and refresh the page in your browser, you should see the example change to look like the following:

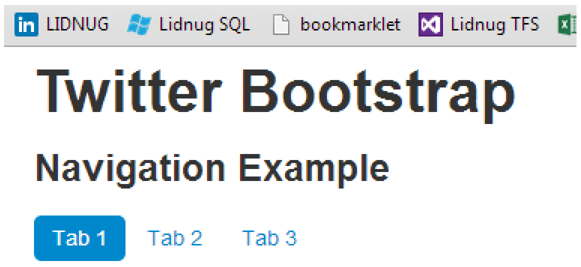


Figure 54: Pills navigation example

Before we finish up with tabs and pills, we can also add the "disabled" class just as we have elsewhere to mark a tab or pill as disabled. We can use the utility classes "pull-left" and "pull-right" to align our tabs and pills to the left or right. Finally, we can also add the "nav-stacked" class to the parent <ul> holding each list, which will convert each of our tabs or pills to a block-level element and then stack them vertically on top of each other as the following images show:

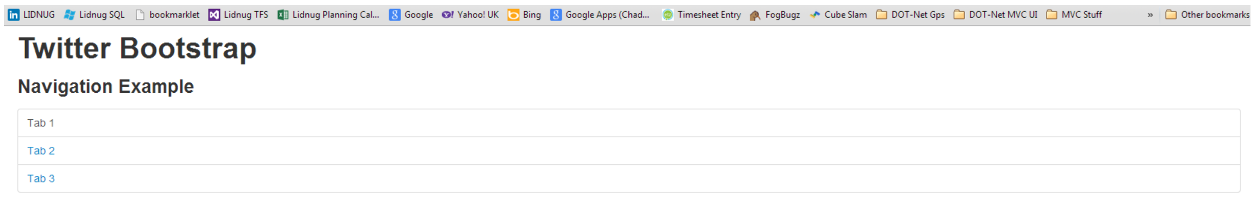


Figure 55: Stacked tabs example

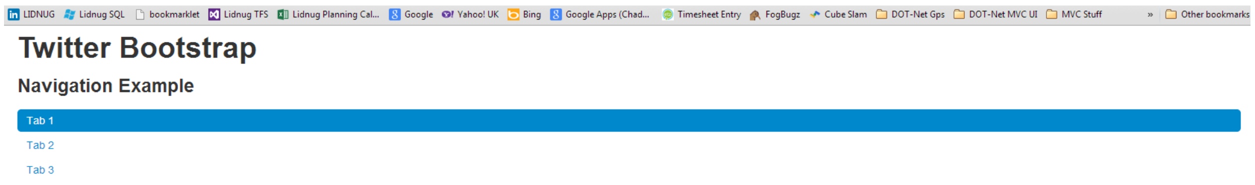


Figure 56: Stacked pills example

You'll notice that you get the appropriate hover effects for each of the entries, too.

Before we move on, please note also that, just as described in the section on dropdown menus, tabs and pills can both act as dropdown menu trigger targets when placed in the correct structure and with the correct data-toggles applied.

Navigation Lists

Navigation lists are designed for sidebars and other similar cases and couldn't be easier to construct.

All you simply have to do is create a standard, unordered list element, and then add the classes "nav" and "nav-list" to the parent <ul> tag.

Just as with the pills and nav lists, you can also add an "active" class to individual <li> items to show which of the items should be marked as the currently active entry.

You can also add the "nav-header" class on a list item. Doing so will give the item a slightly different style but, importantly, will remove it from the hover effects and selection clicks the other items will use. In some ways it's similar to adding a "disabled" class to an item which will, like other navigation aids, mark the item as un-selectable. The nav header, however, is slightly different; the idea is that you use a nav header item to separate related function groups in your sidebar menu so that you can maintain a logical structure.

You can also mark an item with the "divider" class to show the item as a horizontal divider (in the same way as you might add a divider on a dropdown menu).

Start a new template and add the following body code to it:

```
<div class="container-fluid">
  <div class="row-fluid">
    <h1>Twitter Bootstrap</h1>
    <h3>Navigation List Example</h3>
    <div class="span2">
      <ul class="nav nav-list">
        <li class="nav-header">Pages</li>
        <li class="active"><a href="#">Home</a></li>
        <li><a href="#">Profile</a></li>
        <li><a href="#">Settings</a></li>
        <li class="nav-header">Library</li>
        <li><a href="#">E-books</a></li>
        <li><a href="#">Paper books</a></li>
        <li class="divider"></li>
        <li><a href="#">Borrower List</a></li>
      </ul>
    </div>
  </div>
</div>
```

Code Sample 26: Navigation list

Save and load it into your browser, and you should get something that looks as follows:

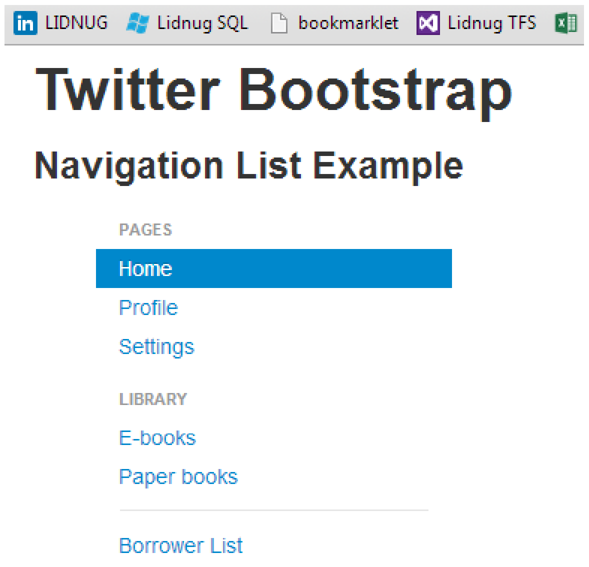


Figure 57: Navigation list example

As you can see from the code in Code Sample 26, we've wrapped the entire thing in a span2-based scaffolding element. Without that, everything would just span the width of the entire page. Nav lists are designed to be block-level vertical lists for creating static menus.

Navigation Bars

Now, we come to what's probably one of Twitter Bootstrap’s most impressive features: the humble navigation bar.

TWB's navigation bar features make the creation of menu bars across the top of sites and pages so easy, it's almost like magic.

Just as with all the navigation aids preceding it, two divs, a ul, and an anchor tag or two are all it takes to create a beautifully styled and functional navigation bar at the top of your page.

Add a few more components and optional classes, and you can create navigation bars with login and search forms embedded in them, navigation bars with dropdown menus, and a great many other combinations; the limit is your imagination.

You can also pin your bar to the top or bottom of your page, make it fluid or fixed, use it with the scaffolding classes, and much more.

Create a new template file and add the following body code:

```
<div class="container-fluid">
  <div class="row-fluid">
    <h1>Twitter Bootstrap</h1>
    <h3>Navigation Bar Example</h3>
    <div class="navbar">
```

```
      <div class="navbar-inner">
        <a class="brand" href="#">My Application</a>
        <ul class="nav">
          <li class="active"><a href="#">Menu 1</a></li>
          <li><a href="#">Menu 2</a></li>
          <li><a href="#">Menu 3</a></li>
          <li><a href="#">Menu 4</a></li>
          <li><a href="#">Menu 5</a></li>
          <li><a href="#">Menu 6</a></li>
        </ul>
      </div>
    </div>
  </div>
</div>
```

Code Sample 27: Basic navigation bar

Save and load the file in your browser, and you should see:

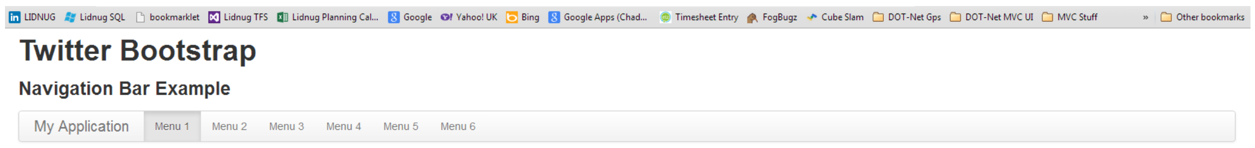


Figure 58: Twitter Bootstrap navigation bar example

In just 13 lines of HTML (six of which were the menu options themselves), you've created a great-looking navigation bar that sits nicely, lined up with everything else on the page.

You can add dividers between your menu items in exactly the same way as you do in other navigation aids. But, instead of just adding a list item with a "divider" tag, you need to add a list item with a "divider-vertical" tag (and no other content inside the list item tag itself).

If you want to add a form to your navigation bar, you can do so by using the following code sample:

```
<form class="navbar-form pull-right">
  <input type="text" class="span4" placeholder="User Name">
  <input type="text" class="span4" placeholder="Password">
  <button type="submit" class="btn">Login</button>
</form>
```

Code Sample 28: Navigation bar form

Note that I'm showing just the form item needed and not the full menu structure. If you insert this in Code Sample 26, just after the last </ul> but before the </div> tag, you should see something that looks like this:

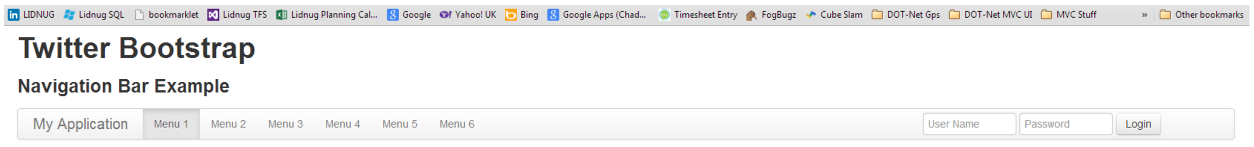


Figure 59: Navigation bar with inline form example

In a similar fashion, you can change the "navbar-form" class for a "navbar-search" or "navbar-query" class and get a search form (as shown in the chapter on forms), which will style the form inputs with the specialist styles to make them more round and search box-like.

If you want to add dropdown menus, all you need to do is mark your <ul>-based lists up in an appropriate nested fashion using the same markup you saw earlier to create a dropdown.

Add the following unordered list to your menu example after the first unordered list but before the opening form tag that creates your inline login form:

```
<ul class="nav">
  <li class="dropdown">
    <a href="#" class="dropdown-toggle" data-toggle="dropdown">
      Options
      <b class="caret"></b>
    </a>
    <ul class="dropdown-menu">
      <li><a href="#">Option menu 1</a></li>
      <li><a href="#">Option menu 2</a></li>
      <li><a href="#">Option menu 3</a></li>
    </ul>
  </li>
</ul>
```

Code Sample 29: Dropdown button for navigation bar

Save and refresh the page in your browser. If you've added the dropdown in the correct place, you should see a new menu option appear with a dropdown arrow next to it. Clicking the option should cause a menu to drop down underneath:

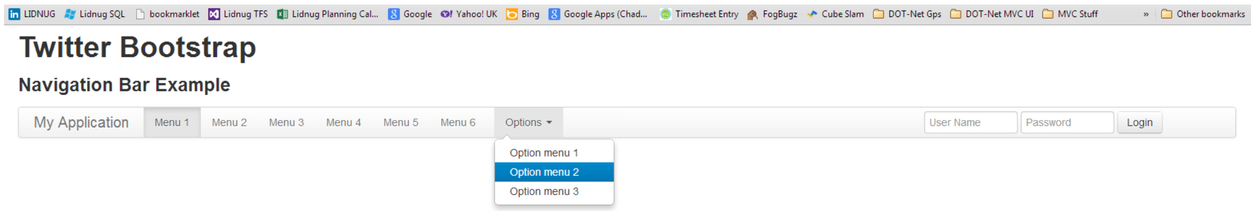


Figure 60: Navigation bar with dropdown menu

If you want to add simple text to your navigation bar, just create a paragraph tag at the appropriate place in your code as we have done with the other elements. Then, add a class of "navbar-text" to the element and watch as everything lines up neatly.

Before we finish with navigation bars, there are a few extra optional display classes you can use to affect your navigation bar in different ways.

The first of these is the "navbar-fixed-top" class which, as the name states, fixes your bar to the very top of your page.

Be aware, however, this fixing is not quite what you might expect. Fixed top navigation bars are overlaid on top of any content already present and are designed to stay in place when you scroll your page upwards, styling on top of whatever content scrolls beneath them.

This means that when your page is scrolled right to the top, you'll likely need some suitable padding or a margin on the top of your body or parent page element so that the top of your document is not lost underneath the navigation bar.

On the navbar you created previously, change the classes on the main navbar parent from:

```
<div class="navbar">
```

To the following:

```
<div class="navbar navbar-fixed-top">
```

And refresh the page.

If everything has appeared as expected, then you'll see straight away how the menu obscures the top of the page. The navigation bar should display something like the following:

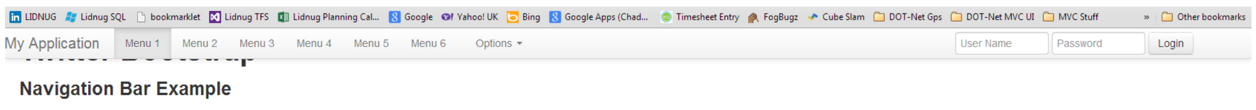


Figure 61: Navigation bar with fixed-top applied

The opposite of the fixed top optional class is "navbar-fixed-bottom" which, as expected, will do exactly the opposite and fix the menu bar to the bottom of the page.

If you use the class "navbar-static-top" rather than "navbar-fixed-top", the navigation bar will remain at the top of the page but will scroll away with the content when the page is scrolled up (rather than overlaying the content and remaining in place like the fixed one).

Creating a Responsive Navigation Bar

If you want to create a navigation bar that collapses correctly on tablet and mobile phone screens, you need to alter the structure of your HTML 5 code slightly and add a new anchor tag in to provide the collapse icon.

The first thing to do is to add a third div inside the two divs already present at the outermost layer of your menu. Give this a class of "container" or "container-fluid" (the same as the classes used in the chapter on scaffolding).

Once you've added this third div, you then need an anchor tag immediately after it to act as the collapse icon once the navigation bar is shrunk down.

After that, if you have a brand element in your bar and want that to remain shown, simply put another div immediately after it. This time, apply a class containing "nav-collapse collapse" to the element, and all your regular navigation bar content should then go inside this collapsible div.

To convert the navigation bar we've built so far into a collapsible bar, change the body code in your template to the following:

```
<div class="container-fluid">
  <div class="row-fluid">
    <h1>Twitter Bootstrap</h1>
    <h3>Navigation Bar Example</h3>
    <div class="navbar">
      <div class="navbar-inner">
            <div class="container">
              <a class="btn btn-navbar" data-toggle="collapse" data-
target=".nav-collapse">
            <span class="icon-bar"></span>
            <span class="icon-bar"></span>
            <span class="icon-bar"></span>
         </a>
         <a class="brand" href="#">My Application</a>
         <div class="nav-collapse collapse">
           <ul class="nav">
             <li class="active"><a href="#">Menu 1</a></li>
             <li><a href="#">Menu 2</a></li>
             <li><a href="#">Menu 3</a></li>
             <li><a href="#">Menu 4</a></li>
             <li><a href="#">Menu 5</a></li>
             <li><a href="#">Menu 6</a></li>
           </ul>
           <ul class="nav">
             <li class="dropdown">
               <a href="#" class="dropdown-toggle" data-toggle="dropdown">
                 Options
                 <b class="caret"></b>
               </a>
               <ul class="dropdown-menu">
                 <li><a href="#">Option menu 1</a></li>
                     <li><a href="#">Option menu 2</a></li>
                     <li><a href="#">Option menu 3</a></li>
               </ul>
             </li>
           </ul>
           <form class="navbar-form pull-right">
             <input type="text" class="span4" placeholder="User Name">
             <input type="text" class="span4" placeholder="Password">
             <button type="submit" class="btn">Login</button>
           </form>
         </div>
       </div>
     </div>
   </div>
 </div>
</div>
```

Code Sample 30: Responsive navigation bar

As you have done throughout this book, save the file and hit reload in your browser. You should see something that looks like the following:

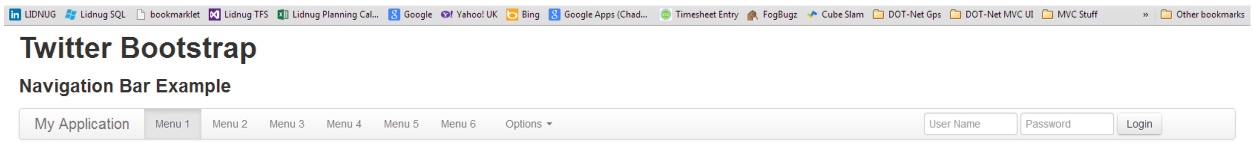


Figure 62: Responsive navigation bar example

You may notice that this doesn't appear any different from the previous navigation bar. However, if you now resize your browser window or display this on a device with a reduced size screen display, you should see something entirely different.

If I display the page in the Electric Plum iOS device simulator, for example, this is what I get in iPad landscape mode:

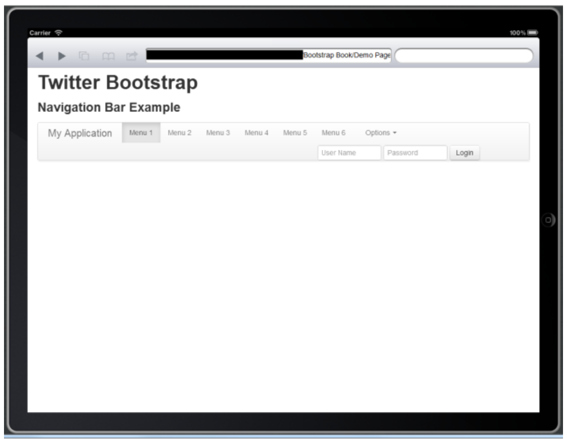


Figure 63: Responsive navigation bar displayed in an iOS device emulator for iPad

If I change the emulator to display the page in portrait iPhone mode, I get this:

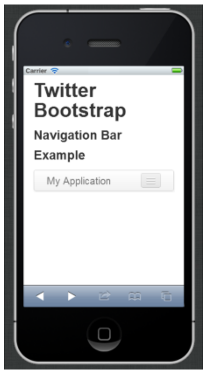


Figure 64: Responsive navigation bar displayed in an iOS device emulator for iPhone

If I then click the now displaying collapse icon, my menu will expand vertically and display my menu items, dropdowns and login form just as it did previously:

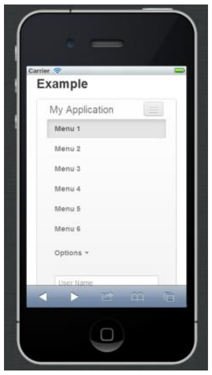


Figure 65: Responsive navigation bar displayed in an iOS device emulator for iPhone with the menu in an un-collapsed state

And just to prove it works the same way if I resize the browser window on my desktop:

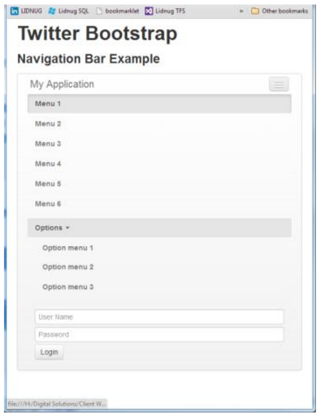


Figure 66: Responsive navigation bar shown in an un-collapsed state in a resized desktop browser

Before we leave the navigation stuff behind, I'll briefly mention the two components we've not yet covered. These are the breadcrumbs and pagination links components.

If you've ever seen a path of links at the top of a page that shows you where you are in the document, then you've seen a breadcrumb trail. To create breadcrumbs in Twitter Bootstrap, simply create an unordered list as you have done already, and then apply a class of "breadcrumb" to the <ul> element acting as a parent container.

For breadcrumbs, set the "active" class on the link that represents the page you are currently on; this will mark that link with a grey color which is then used to differentiate between links you can click on to go backwards in your page tree. A standard breadcrumb list looks like the following:

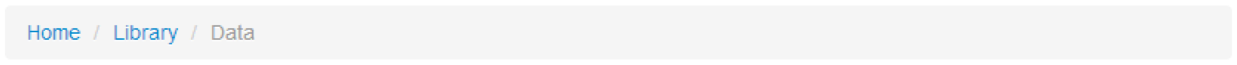


Figure 67: Twitter Bootstrap breadcrumb component

Pagination is a little bit different from the navigation aids we've seen so far. You don't apply the class to the <ul> tag but to a <div> tag that wraps the entire unordered list as the following shows:

```
<div class="pagination">
  <ul>
    <li><a href="#">Prev</a></li>
    <li><a href="#">1</a></li>
    <li><a href="#">2</a></li>
    <li><a href="#">3</a></li>
    <li><a href="#">4</a></li>
    <li><a href="#">5</a></li>
    <li><a href="#">Next</a></li>
  </ul>
</div>
```

Code Sample 31: Pagination list

This will produce the following pagination bar when rendered in the browser:

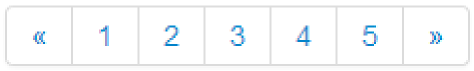


Figure 68: Basic pagination example

As with all other navigation components, there are "disabled" and "active" classes that can be applied to the individual list items as well as different sizes of pagination which can be controlled with "pagination-large," "pagination-small," or "pagination-mini" added to the class name list of the div element surrounding the pagination list.

Label and Badge Components

Twitter Bootstrap contains useful classes to produce nicely formatted labels and badges for things such as status counts and important tags on post items.

The colors on each of the classes use the same sort of naming scheme as we've already seen elsewhere, as the following code sample shows:

```
<div class="container-fluid">
  <div class="row-fluid">
    <h1>Twitter Bootstrap</h1>
    <h3>Labels and Badges Example</h3>

    <br />Labels<br />
    <span class="label">Default</span>
    <span class="label label-success">Success</span>
    <span class="label label-warning">Warning</span>
    <span class="label label-important">Important</span>
    <span class="label label-info">Info</span>
    <span class="label label-inverse">Inverse</span>
    <br /><br />Badges<br />
    <span class="badge">1</span>
    <span class="badge badge-success">2</span>
    <span class="badge badge-warning">4</span>
    <span class="badge badge-important">6</span>
    <span class="badge badge-info">8</span>
    <span class="badge badge-inverse">10</span>
  </div>
</div>
```

Code Sample 32: Labels and badges

Displayed in a browser, the previous code should look like the following:

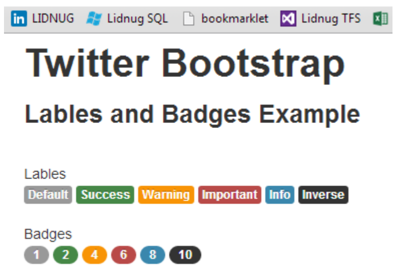


Figure 69: Labels and badges example

Hero Units and Page Headings

Under the typography section of components, you'll find two elements that are designed to make your opening page statement stand out from the crowd.

Both of them are designed with very different purposes in mind. We'll start with the page heading.

If you recall from all the examples we've done in the book so far, I've had a common header above all of them. You can see this in the last image (Figure 69) just above where the text says in an H1 tag "Twitter Bootstrap" followed in an H2 tag by "Labels and Badges example."

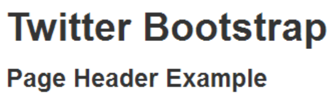
Even though I knew this page header component was available, I've deliberately not used it so far in the book, purely so I could introduce it now and give you a better idea of its intended use.

Using it couldn't be simpler either. Start a new template document and add the following body of HTML code to your page:

```
<div class="container-fluid">
  <div class="row-fluid">
    <h1>Twitter Bootstrap</h1>
    <h3>Page Header Example</h3>
    <div class="page-header">
      <h1>Twitter Bootstrap <small>Page header example</small></h1>
    </div>
  </div>
</div>
```

Code sample 33: Page header

If everything is good, you should see a page with two headers, one in the old style (that I've been using throughout this book) which should look like:


Twitter Bootstrap

Page Header Example

Figure 70: Old style header

And one below it using the new style page header, which should look like:

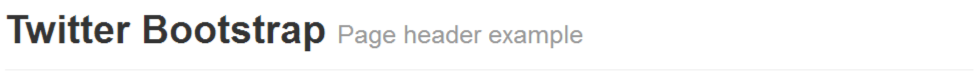

Twitter Bootstrap Page header example

Figure 71: New style header

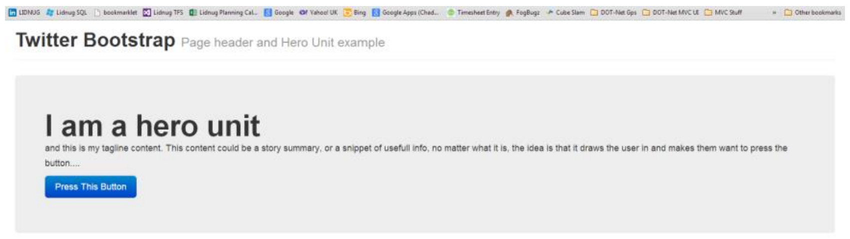
As you can see, it's a simple as wrapping an H1 tag inside a div that has the class "page-header" applied to it. The grey text to the left is created by nesting that content inside a nested small tag; the whole thing then has a nice underline added to it.

There are no optional classes for the page header but it will obey any spanX or other scaffolding classes it's placed in, so you can place and resize the content as you need to.

The next of the typographic components is the “hero unit.”

Hero units are much loved by the digital marketing enthusiast.

When you see these webpages with an extremely large call to action space at the top of the page, right before any useful content, that large area is the part that the TWB authors call a hero unit:



Twitter Bootstrap Page header and Hero Unit example

I am a hero unit

and this is my tagline content. This content could be a story summary, or a snippet of usefull info, no matter what it is, the idea is that it draws the user in and makes them want to press the button....

Press This Button

Figure 72: Hero unit example

As you can see from Figure 72, the hero unit is a block-level element so it will expand to fill any space available to it.

The code to produce the hero unit seen in the previous figure is as follows:

```
<div class="container-fluid">
  <div class="row-fluid">
    <div class="page-header">
      <h1>Twitter Bootstrap <small>Page header and Hero Unit
example</small></h1>
    </div>
    <div class="hero-unit">
      <h1>I am a hero unit</h1>
      <p>and this is my tagline content. This content could be a story
summary, or a snippet of useful info, no matter what it is, the idea is that
it draws the user in and makes them want to press the button....</p>
      <p>
        <a class="btn btn-primary btn-large">Press This Button</a>
      </p>
    </div>
  </div>
</div>
```

Code Sample 34: Hero unit code

Like the page header, the hero unit is a very simple, multiple HTML element setup and has no extra optional classes.

Thumbnails and Media Objects

Twitter Bootstrap's thumbnail component is designed for a grid-like display of multiple images, such as a gallery page.

However, the actual component and its structure are actually very flexible. And, due to its use of the scaffolding and grid system, it can create some wonderfully balanced layouts with images and text content.

Just like the navigation components, the thumbnail components are created using an unordered list with nested list items containing the thumbnails embedded inside.

Create a new template document and add the following body of HTML to it:

```
<div class="container-fluid">
  <div class="row-fluid">
    <div class="page-header">
      <h1>Twitter Bootstrap <small>Thumbnails & Media Objects
example</small></h1>
    </div>
    <ul class="thumbnails">
      <li class="span4">
        <a href="#" class="thumbnail">
          <img data-src="js/holder.js/300x200" alt="">
        </a>
      </li>
    </ul>
  </div>
</div>
```

Code Sample 35: Thumbnail image

Some of you will notice that I've used an extra bit of JavaScript in there called holder.js. This is not an official part of Twitter Bootstrap, but nevertheless is one bit of JavaScript that I believe all web developers should have by default in their project templates.

You add the following script line:

```
<script src="js/holder.js" type="text/javascript"></script>
```

Alongside all your other scripts, just above anywhere you need an image place holder, you simply swap your normal:

```
src="somefile.ext"
```

For:

```
data-src="js/holder.js/300x200"
```

Set the 300x200 to whatever size you want the thumbnail to be. You can download holder.js from its repository on GitHub or use the copy that's in the samples zip file to go with this book, if you wish. However, you can use any suitable image you have available.

Once you save and load the file, you should see something like the following:

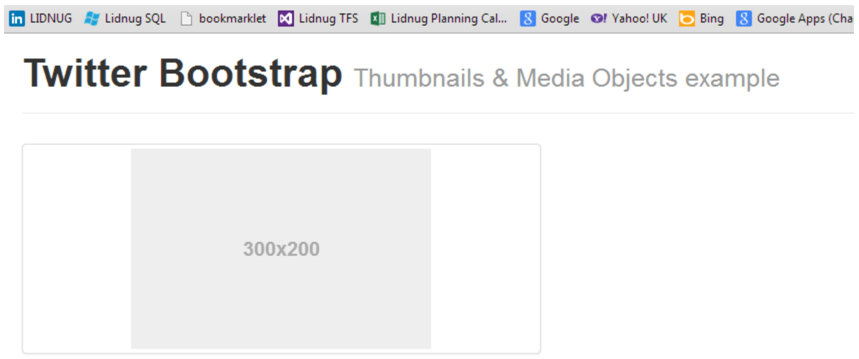


Figure 73: Basic thumbnail example

Because thumbnails respect the grid system in Twitter Bootstrap, adding more list items to the above code sizes everything proportionately and with no effort, other than adding a little extra markup following the same structure:

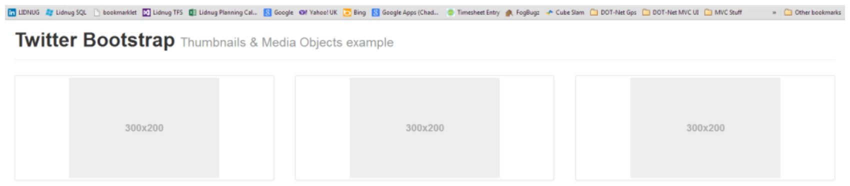


Figure 74: Multiple thumbnail example

If you want to add other content to your thumbnails, then all you need to do is change your anchor tags to div tags, and then include the content after your image or its placeholder.

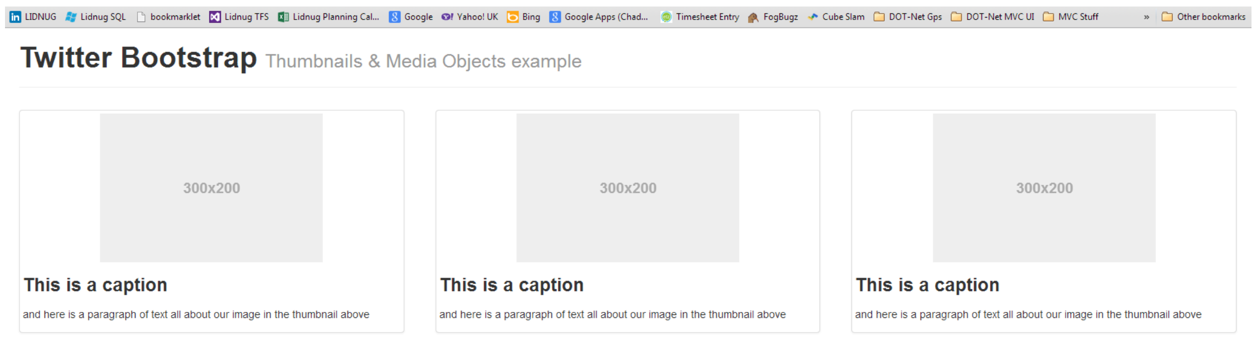
Change the body content in your thumbnails example so it looks like the following:

```
<div class="container-fluid">
  <div class="row-fluid">
    <div class="page-header">
      <h1>Twitter Bootstrap <small>Thumbnails & Media Objects
example</small></h1>
    </div>
    <ul class="thumbnails">
      <li class="span4">
```

```
        <div class="thumbnail">
          <img data-src="js/holder.js/300x200" alt="">
          <h3>This is a caption</h3>
          <p>and here is a paragraph of text all about our image in the
thumbnail above
        </div>
      </li>
      <li class="span4">
        <div class="thumbnail">
          <img data-src="js/holder.js/300x200" alt="">
          <h3>This is a caption</h3>
          <p>and here is a paragraph of text all about our image in the
thumbnail above
        </div>
      </li>
      <li class="span4">
        <div class="thumbnail">
          <img data-src="js/holder.js/300x200" alt="">
          <h3>This is a caption</h3>
          <p>and here is a paragraph of text all about our image in the
thumbnail above
        </div>
      </li>
    </ul>
  </div>
</div>
```

Code Sample 36: Multiple thumbnails with extra content

Save and refresh, and you should be presented with:



Twitter Bootstrap Thumbnails & Media Objects example

300x200

This is a caption

and here is a paragraph of text all about our image in the thumbnail above

300x200

This is a caption

and here is a paragraph of text all about our image in the thumbnail above

300x200

This is a caption

and here is a paragraph of text all about our image in the thumbnail above

Figure 75: Multiple thumbnails with extra content

The thumbnail classes also support another type of component called a media object.

Media objects are designed for list-like displays, such as those used by Twitter itself in a timeline or in a list of comments below a blog post with an avatar next to each.

Unlike the thumbnails, however, you don't create this using unordered lists and list items. You create a media object list using standard divs, nesting where needed. Add the following code to your thumbnails example, just underneath the <ul> holding the three horizontal thumbnails:

```
<div class="media">
  <a class="pull-left" href="#">
    <img class="media-object" data-src="holder.js/64x64">
  </a>
  <div class="media-body">
    <h4 class="media-heading">Media heading</h4>
    <p>a paragraph of text to go with the media heading above that's rendered
as a h4 tag, this is just a paragraph</p>
  </div>
</div>
```

Save and render your page and you should see the media object appear just below your thumbnails:

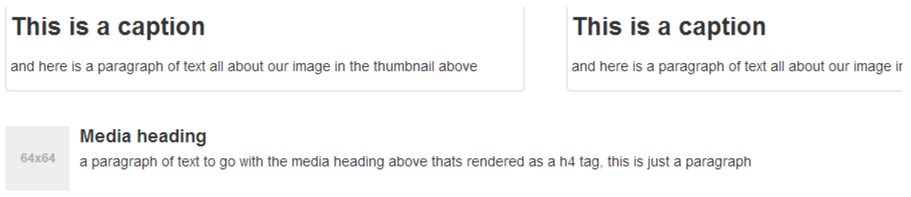

This is a caption
and here is a paragraph of text all about our image in the thumbnail above
This is a caption
and here is a paragraph of text all about our image ir
64x64
Media heading
a paragraph of text to go with the media heading above thats rendered as a h4 tag, this is just a paragraph

Figure 76: Thumbnails with media object

Add two more just below the first one, and you'll see that they all line up and stack correctly on top of one another:

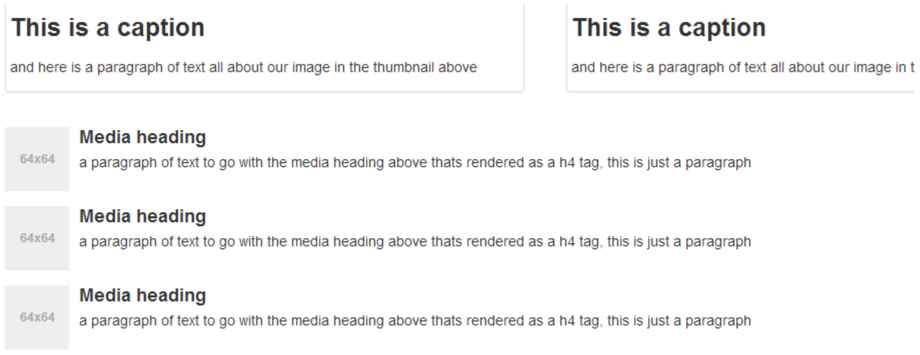

This is a caption
and here is a paragraph of text all about our image in the thumbnail above
This is a caption
and here is a paragraph of text all about our image in t
64x64
Media heading
a paragraph of text to go with the media heading above thats rendered as a h4 tag, this is just a paragraph
64x64
Media heading
a paragraph of text to go with the media heading above thats rendered as a h4 tag, this is just a paragraph
64x64
Media heading
a paragraph of text to go with the media heading above thats rendered as a h4 tag, this is just a paragraph

Figure 77: Multiple media objects

If you nest your media object divs inside the "media-body" of a parent, you'll also find they nest and indent correctly, too:

```
<div class="media">
  <a class="pull-left" href="#">
    <img class="media-object" data-src="holder.js/64x64">
  </a>
  <div class="media-body">
    <h4 class="media-heading">Media heading</h4>
    <p>Media Objects paragraph</p>
    <div class="media">
      <a class="pull-left" href="#">
        <img class="media-object" data-src="holder.js/64x64">
      </a>
      <div class="media-body">
        <h4 class="media-heading">Nested Media heading</h4>
        <p>A nested objects paragraph of text</p>
      </div>
    </div>
  </div>
</div>
```

Code Sample 37: Nested media object

When rendered, it should display the following:

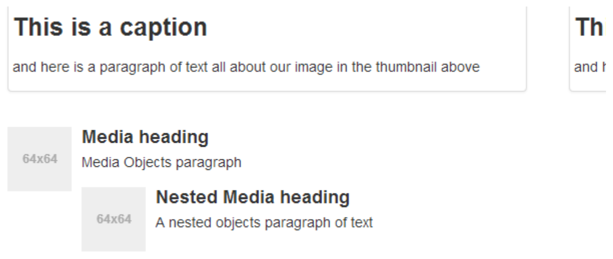

This is a caption
and here is a paragraph of text all about our image in the thumbnail above
Th
and t
64x64
Media heading
Media Objects paragraph
64x64
Nested Media heading
A nested objects paragraph of text

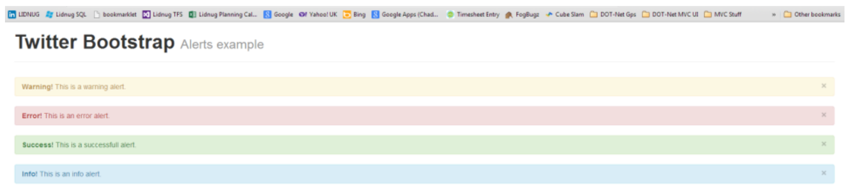
Figure 78: Nested media object

Alerts

As you've seen many times so far, Twitter Bootstrap has a number of standard colors for things such as text, buttons, and labels.

These standard colors also extend to a component specifically for creating on-screen alerts, such as the ones you see when you delete a post or enter incorrect authentication information in a login dialog.

A TWB alert is a block-level element so it will stretch to fill whatever space is available to it in its parent element. This means that size and position can easily be controlled using the scaffolding system:



Twitter Bootstrap Alerts example

Warning! This is a warning alert. ×

Error! This is an error alert. ×

Success! This is a successfull alert. ×

Info! This is an info alert. ×

Figure 79: Alerts example

The image in Figure 79 was produced using the following HTML 5 code:

```
<div class="container-fluid">
  <div class="row-fluid">
    <div class="page-header">
      <h1>Twitter Bootstrap <small>Alerts example</small></h1>
    </div>
    <div class="alert alert-warning">
      <button type="button" class="close" data-
dismiss="alert">&times;</button>
      <strong>Warning!</strong> This is a warning alert.
    </div>
    <div class="alert alert-error">
      <button type="button" class="close" data-
dismiss="alert">&times;</button>
      <strong>Error!</strong> This is an error alert.
    </div>
    <div class="alert alert-success">
      <button type="button" class="close" data-
dismiss="alert">&times;</button>
      <strong>Success!</strong> This is a successful alert.
    </div>
    <div class="alert alert-info">
      <button type="button" class="close" data-
dismiss="alert">&times;</button>
      <strong>Info!</strong> This is an info alert.
    </div>
  </div>
</div>
```

Code Sample 38: Alerts code

You'll notice that each of them has a small, close icon on the right-hand side that's used to close the alert. This works using another of Twitter Bootstraps magic data attributes called "data-dismiss."

In the case of the above code, the line you're interested in is any of the button tags inside the alert div:

```
<button type="button" class="close" data-dismiss="alert">&times;</button>
```

I've highlighted the data dismiss attribute in bold. You'll meet this again shortly in the dialogues section, but like any other of TWB's data attributes you'll need the TWB main JavaScript file loaded to make them work correctly.

Without that data attribute, the button will function just like a regular button and won't close the alert dialog.

If you add the "alert-block" optional class to the class list of your alert div, you'll get slightly bigger padding on the bottom and top, designed to hold longer messages that may remain on screen for a longer length of time.

Progress Bars

A basic blue progress bar can be created quite simply using the following markup:

```
<div class="progress">
  <div class="bar" style="width: 60%;"></div>
</div>
```

Code Sample 39: Basic progress bar

The width on the inner "bar" class denotes the percent complete that the bar should show.

If you render this in a template page, you should see the following:

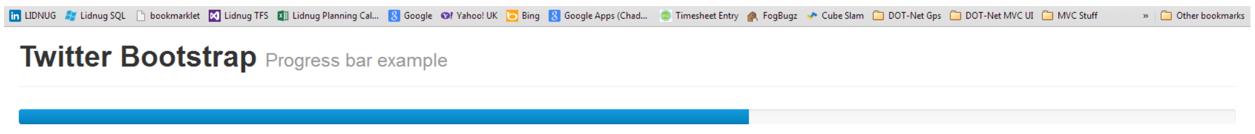


Figure 80: Progress bar example

If you add the optional class "progress-striped" to the outer div, the progress bar will appear striped:

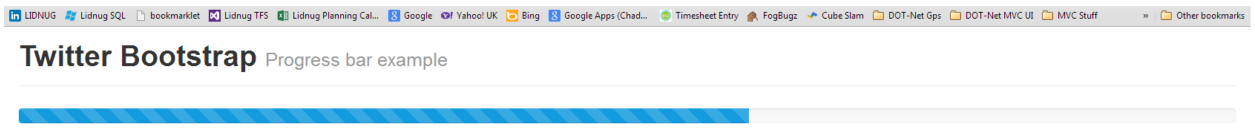


Figure 81: Striped progress bar example

You can also add the "active" optional class, which will then make the stripes appear to be animated so the bar has some movement.

You can also stack progress bars by providing multiple divs with the "bar" class applied to them inside the master div, holding the "progress" class as follows. Change the code in Code Sample 36 so that it looks like the following:

```
<div class="progress">
  <div class="bar bar-success" style="width: 35%;"></div>
  <div class="bar bar-warning" style="width: 20%;"></div>
  <div class="bar bar-danger" style="width: 10%;"></div>
</div>
```

If you load it into the browser, you should see the following:

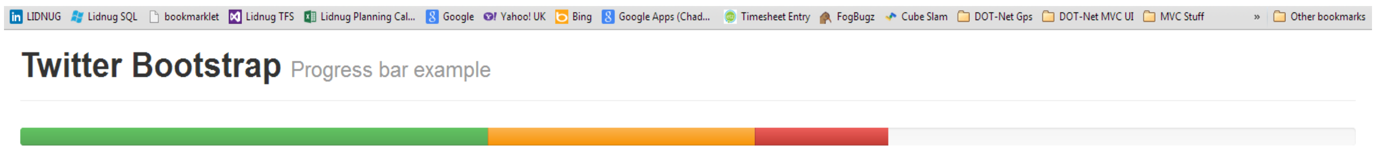


Figure 82: Stacked progress bar

There are two things to note in the above example. First, all the percent values must add up to 100 percent. Second, you provide each segment a different color by using one of the pre-made bar color classes we'll see in just a moment.

Just a word of warning on browser compatibility first, though: the TWB progress bar classes use CSS3 gradients to achieve the color effect you see, especially on the striped bars.

Because of this, Internet Explorer 7 to Internet Explorer 9 won't display things as expected, and older versions of Firefox will also have an issue with the display. Functionally, however, everything else still works as expected.

If you’re coloring individual bars, then you need to add one of the optional color classes to the parent progress div as follows:

- progress-info: A standard blue progress bar.
- progress-success: A green progress bar for successful operations.
- progress-warning: A yellow progress bar designed for warning operations.
- progress-danger: A red progress bar designed for dangerous operations.

All the Rest

So far, we've covered a huge array of styles and components in this section, and yet, there are still more in here.

If you want to make something appear in a shaded well, you can use the "well" class on a block-level element such as a sidebar to do so. You can also control the amount of padding around the contents by using "well-large" and "well-small."

There are also many utility classes, some of which you've already seen such as "pull-left" and "pull-right" to float things.

There's also the "clearfix" class that you use to clear floats when you want to reset your document flow.

You can also use the <code> tag to display text that you would like to show as computer code inline with other text paragraphs.

In the next chapter, we’re going to take a look at some of the JavaScript-driven features available in the framework.

Chapter 8 Twitter Bootstrap JavaScript

TWB's JavaScript features are all extensions built on top of jQuery. Most of them can be used simply by using the various "data-" attributes you've already seen used in this book. Unfortunately, there's just not enough room in 100 pages to give a complete top-to-bottom coverage of both the CSS framework and JavaScript framework in their entirety. So, for this chapter, I'm only going to pick the most commonly used features available.

I strongly encourage you, however, to go read the JavaScript section in the Twitter Bootstrap docs from top to bottom. I can guarantee that there will be far more there than I can fit in the pages I have remaining in this e-book.

The page for the 2.3.2 docs can be found at http://getbootstrap.com/2.3.2/javascript.html presently but be aware that this may change.

As I commented at the beginning of the book, Version 3 is on the horizon. I don't know just yet what plans the authors have for the 2.3.2 branch.

The Basics

As previously mentioned, TWB makes extensive use of data attributes to perform its no-code magic. And, while the data attribute API is great, sometimes you might just find you need to turn it off, especially if you're using other toolkits that don't play well with the framework or with jQuery.

If you need to turn the data API off, you can do so for the entire document by using a top-level block element such as "<body>" simply by doing the following:

```
$('body').off('.data-api')
```

You don't have to turn the entire data API off if you don't want to. However, if all you want to do is turn part of it off, you can do that simply by using the namespace of the section you wish to turn off:

```
$('body').off('.alert.data-api')
```

This will turn off just the data API for the alerts components.

In addition to the data API, TWB also has a programmatic API. This one is accessed and used in a fluent manner just like anything else in jQuery. For example, to show a modal you've defined, you could use the following:

```
$('#myDialog').modal('show')
```

Or you can toggle a button state as follows:

```
$('#okButton').button('toggle')
```

You'll see a few more examples very soon when I cover the actual JavaScript components.

Like any good JavaScript library, TWB also includes a no-conflict option to allow things to work in harmony with other toolkits such as Prototype and Dojo.

You can turn the conflict system on and off on a particular component quite easily, as follows:

```
var bsButton = $.fn.button.noConflict()
```

The reason for the var is so you can turn it back off again at a later point should you wish to:

```
$.fn.button = bsButton
```

The final part of the basics to know is the events structure used by Twitter Bootstrap.

TWB events are typically of a two-part form. The first event signifies something is about to happen and the second event signifies something has happened.

For example, a modal may generate a "show" event followed by a "shown" event or a button might generate a "click" event followed by a "clicked" event.

Any of the events can be canceled by returning "event.preventDefault" just as you would if you were using normal jQuery, as the following shows:

```
$('#myDialog').on('show', function(event)
{
  return event.preventDefault();
}
```

This will prevent the modal referenced with id = "myDialog" from actually showing:

The Components

As I mentioned previously, we won't be covering all of these. But, just to give you an idea of what's available, here is a full list of all the JavaScript components provided by Bootstrap:

1. Transitions
2. Modal dialogs

3. Dropdown menus
4. Scroll spy
5. Tabs
6. Tooltips
7. Popovers
8. Alert boxes
9. Buttons
10. Collapsible sections
11. Carousels
12. Type ahead
13. Fixed panels

As you'll note from previous chapters, we've already covered some of the items on this list. So, for the remainder of this book, we'll limit our descriptions to "Modal dialogs," "Tooltips," and "Popovers."

Modal Dialogs

One of the most impressive and easy-to-use features in TWB is the modal dialog component.

In fact, since I started to use them, development of UI-based alerts and input boxes has become so easy I rarely need to think about it.

Like many of the components provided by TWB, using the modal feature efficiently is all about correctly structuring your HTML 5 code.

There are five primary class names used in the construction of a modal, all of which are expected in a certain order of divs. Open a new template file and add the following body text to it:

```
<div class="container-fluid">
  <div class="row-fluid">
    <div class="page-header">
      <h1>Twitter Bootstrap <small>Modal Dialog example</small></h1>
    </div>
    <div class="modal hide">
      <div class="modal-header">
        <button type="button" class="close" data-
dismiss="modal">&times;</button>
        <h3>Modal header</h3>
      </div>
      <div class="modal-body">
        <p>I am some content that sits inside this dialog box</p>
      </div>
      <div class="modal-footer">
        <a href="#" class="btn" data-dismiss="modal">Close</a>
        <a href="#" class="btn btn-primary">Save changes</a>
      </div>
    </div>
  </div>
</div>
```

Code Sample 40: Basic modal dialog

If you save and load that into your browser, you'll notice straight away that you see nothing except the title.

Just as you might expect, modal dialogs are hidden by default. If you remove the hide class from the main outer div surrounding everything, you'll see the modal appear in your page. But, as it's not correctly active, you won't be able to do anything with it.

Your outer div must have a class of "modal" and, if you don't want it to appear inline in your document, the optional "hide" class, too.

Following that, you need to define three div sections with three classes, as follows:

- "modal-header"
- "modal-body"
- "modal-footer"

Each of the sections is exactly as the class defines it: header, body, and footer.

In the header, you'll see that we've added a data-dismiss button exactly as we did in the section on alerts. If you remember, in that section we had the word "alert" as the parameter. This time, we use "modal" so the framework knows what element we are trying to close.

Aside from that, everything else is quite standard. If we wanted to make the close button close our dialog, we would simply style it as you can see above, and then add a data-dismiss attribute just as we did for the close icon.

We can trigger the display of our dialog from any element we wish; we just have to make sure that our parent div with the class "modal hide" has a suitable ID for us to target.

Change the following line in Code Sample 40:

```
<div class="modal hide">
```

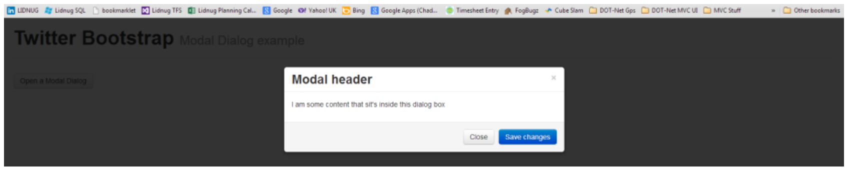
To the following:

```
<div class="modal hide" id="myDialog">
```

Then, somewhere just before that opening div tag, insert a button tag as follows:

```
<a href="#myDialog" class="btn" data-toggle="modal">Open a Modal Dialog</a>
```

If all goes well and you refresh the page and click the button, you'll see the following:



I am some content that sit's inside this dialog box

Figure 83: Basic modal dialog example

As you'll see when you click the button and, for that matter, the close button, the modal appears and disappears quite abruptly. If you add the optional class "fade" to the main parent div (alongside "modal hide"), you'll see you get a much nicer opening and closing effect.

You don't have to use an anchor tag to open a modal, either. Any element in your document that can accept a click can be used; all you have to do is add a "data-target" attribute into the mix as follows:

```
<button class="btn" data-toggle="modal" data-target="#myDialog">Open a Modal
Dialog</button>
```

The button tag could just as easily have been an image, header, paragraph, or anything else for that matter.

You can control the modal from standard JavaScript, too. You can pass a JSON object into your modal using the following:

```
$('#myDialog').modal({ … options json here … })
```

With the following options defined:

- backdrop: True, false or static. If true or false, then shaded backdrop either will or will not be displayed. If static, the shaded backdrop will be displayed but prevent clicks on the backdrop from closing the modal.
- keyboard: True or false to enable or disable keyboard interaction.
- show: True or false to show the dialog as soon as it's activated or keep it hidden.
- remote: If this is specified, then when the dialog is opened, jQuery will be used to perform an AJAX load from the URL specified in the parameter, allowing you to dynamically populate your dialogs at runtime.

Finally, you can also call the following methods on a modal using the following:

```
$('#myDialog').modal('… method name here …')
```

To make the modal react from external JavaScript, the methods available are as follows:

- toggle: If the dialog is open close it, otherwise open it.
- show: Shows the dialog, has no effect if already open.
- hide: Closes the dialog, has no effect if it's already closed.

The modal will also fire the "show," "shown," "hide," and "hidden" events which can be handled as shown at the beginning of this chapter.

Tooltips

Love them or hate them, tooltips are immensely useful in many scenarios, from attaching helpful labels to input controls, right through to those monstrous things that advertisers like to put in the middle of a paragraph that pop up when you least expect them.

In Twitter Bootstrap, using them couldn't be easier. All you need to do is use the data attributes you've seen so far in exactly the same way as the other data API examples you've tried up to now.

There is one small caveat, though, that's specific to tooltips: the fact that you have to initialize them manually.

Unlike everything else, to make use of tooltips, you'll need to manually activate any tooltips you create in a document.ready handler or something similar once your document has loaded.

The reasons for doing this are not really clear and the documentation offers no explanation either. The best way to make your tooltips work is with something like the following:

```
<p>This is some text with an embedded <a href="#" rel="tooltip" data-
original-title="I Am Tooltip!">Tooltip</a> in line with the code</p>
```

Then, you can simply just execute the following to activate them all:

```
$(document).ready(function() {
    $('[rel=tooltip]').tooltip();
});
```

From that point on, your tooltips should just simply pop up when you hover over them.

You can also use the alternative way of targeting the selectors:

```
$('#someelementid').tooltip({
  selector: "a[rel=tooltip]"
})
```

This is more useful if you have all your tooltips inside a main parent element such as a modal dialog or page article. The "#someelementid" selector would typically be the ID of a div or other block-level element.

As is common with the other components, just about everything is controlled from data attributes. The following list of attribute names should be prefixed with "data-" and added to the anchor or other inline element being used to house the tip. For example, "animation" would become "data-animation". These options can also be passed using ".tooltip" as a JSON object containing initialization parameters at initialize time:

- animation: True or false to enable or disable animation effects.
- html: True or false to allow HTML to be inserted directly into the tooltip.
- placement: "Top," "bottom," "left," or "right" to choose the pop up direction of your tooltip.
- selector: Used as described above in the initial options to define the master selector.
- title: Default tooltip title.
- trigger: "Click," "hover," "focus," or "manual" determines what action makes the tooltip appear.
- delay: Milliseconds delay before the tooltip is shown, if show is automatic. Can also be specified as delay: { show: x, hide: x } to set show and hide values independently.

Just like the modal dialogs, there are a number of fluent API calls that can be called on tooltips also.

Like others, you must use jQuery's selector engine to call them.

You can call a method using the following:

```
$('#someElement').tooltip('… method name …')
```

Valid methods are:

- 'show'
- 'hide'
- 'toggle'
- 'destroy'

Popovers

Popovers are not much different than tooltips. However, where tooltips are designed for quick help tips, popovers are more like mini dialog boxes with a title bar and content area.

They are designed to house more than a small snippet of text and are often used for image, video, and page previews, or extended explanations of snippets of text.

Because popovers are an extension of tooltips, to use them you need to make sure you have the tooltip module installed. If you're using the full "bootstrap.js" file, this isn't something you need to worry about. If, however, you've produced a custom build and included only the modules you need, you need to ensure tooltips are part of that build.

Unlike tooltips, you don't need to manually initialize them in a document.ready handler. You can just attach them to an element using plain old data attributes as follows:

```
<a href="#" class="btn" data-toggle="popover" data-placement="top" data-
content="Popover content goes here." title="Popover on top">Popover on
top</a>
```

Just as with tooltips, you can see that you can set the title, placement, and content of the popover (and allow it to be generated dynamically) using exactly the same methods as with tooltips.

Should you so wish, however, you can also build your popovers yourself directly in HTML code:

```
<div class="popover left">
  <div class="arrow"></div>
  <h3 class="popover-title">Popover left</h3>
  <div class="popover-content">
    <p>Pop over content goes here.</p>
  </div>
</div>
```

Code Sample 41: Basic popover

Personally, I prefer the dynamic approach but, if you build them using HTML, you can keep them separate from the element they are assigned to and attach or detach them as needed.

As you've seen, and in a similar fashion to tooltips, you can set the options to a popover as a JSON object and create it using the fluent API. Or can prefix each of the options and add them to the element to which it's attached by prefixing them with "data-."

Popovers understand everything listed under tooltips with the same parameters, along with two extra options as follows:

- content: Used to set the actual inner content of the popover, and
- container: Used to attach the popover only to a certain parent container.

The same four methods, "show," "hide," "toggle," and "destroy" are also all available. They are called in exactly the same way as for tooltips except you use ".popover" rather than ".tooltip".

All the Rest

As I've already mentioned, there's much more available in the JavaScript section. And, from what I've read in some of the Version 3 documentation, it appears that even more has been added in there, too.

The JavaScript features in Twitter Bootstrap could easily take up an entire book just on their own. So, what's still left to cover?

Well, you know the basics of dropdowns, buttons, tabs, and alerts as we covered them in the components section. In this section, we covered modals, tooltips, and popovers. This leaves the following still to be introduced:

- Transitions
- Scroll spy
- Collapse
- Carousel
- Typeahead
- Affix

Transitions are used to produce the fading and sliding effects you've seen in things such as Modal dialogs.

Scroll spy you've seen if you've been browsing the Bootstrap documentation while we've gone through this book; it's the module that keeps the blue highlight on the correct content’s entry on the left of the page as you scroll down.

Collapse is commonly used to create accordions, but anything that needs a collapsible panel is a target for this module.

Carousel is used to create a rotating slideshow, and can create some very impressive effects when used with a hero unit.

Typeahead is exactly as its name suggests: attach it to any type of input for instant predictable text pop-ups.

Lastly, affix does exactly what it says: it affixes an element to a given position on a page and keeps it there no matter how much the page scrolls.

In the next and final chapter, we'll look at some of the extensions that are available for Bootstrap, giving you even more choice in how you put it to use.

Chapter 9 Extending Bootstrap

After that little rocket ride, you might be left wondering just what there is to be extended in Twitter Bootstrap.

Well, the community, it appears, seems to think quite a lot actually.

One of the things I personally went looking for soon after starting to use TWB was a decent file upload component. I also had a client that wanted iOS-style, yes/no slide switches rather than check boxes.

There are thousands upon thousands of different plug-ins and extensions out there to go with Bootstrap. Pair that with Twitter's latest toolkit "Twitter Flight" and you've got the potential for one UI kit to rule them all.

After much exhaustive searching and looking to see what's available, here are some of my top picks in Twitter Bootstrap extensions.

Bootstrap Extension Sites

Bootsnipp

There are not many different ways you can describe Bootsnipp except to say that it has all of the things that relate to Twitter Bootstrap.

It's a real powerhouse of page templates and code samples, and has possibly the largest list of links to resources that I've seen so far.

The list of sites on its resources page includes all the other sites I'm going to list in this chapter, but more than that, it groups everything into useful categories. So, if all you're looking for is UI additions, you only need to use the category list or UI tag to find what you want quickly.

As if that's not enough, it also includes an online form builder, allowing you to drag and drop different controls into a form, which you can then export into your own template for use in your own projects.

There is also a button builder allowing you to style and add icons for both the Version 2 and Version 3 frameworks, meaning you no longer have to remember all of those classes to get your buttons just right.

Wrap Bootstrap

In some ways, Wrap Bootstrap is similar to Bootsnipp; the main difference is that they concentrate solely on templates, and I'm talking about full site templates here, not just the small page snippets that Bootsnipp offers.

There is also a paid for section on the site, and an opportunity for you to sell your own templates via their merchandising system. If you're looking for high-quality, pre-made templates ready for you to code against, Wrap Bootstrap is probably where you should be looking.

Extended Full Bootstrap Kits

Jasny Bootstrap

If you're going to download any extended version of Bootstrap, it has to be Jasny.

It has everything that the normal version has and then some. For example, the type ahead JavaScript module has been extended to work directly off an AJAX call. There are new input types that can take input masks, a file selector/upload component, row links , extra icons, and much more. Jasny is not just Bootstrap Extended, it is THE Bootstrap Extended.

Bootplus

Bootplus is not an awful lot different than Bootstrap itself; however, as the author describes it, that's not its purpose.

The idea behind Bootplus is to style things a little more like Google, specifically Google+.

The author's even added a few new components: "News Cards," "People Cards," "People Hover Cards," and "Navbar Notifications" that look and feel just like the components available in a G+ page.

UI Kits

Fuel UX

While Fuel UX gives the impression that it's a full Bootstrap with extensions, in reality it's more of a UI furniture extension kit.

Many of the extension kits out there fully override everything from CSS and basic styles, right through to custom additions. Fuel UX, however, does not.

Fuel UX concentrates on one thing and one thing only: making the input controls and other on-screen elements better and with richer functionality.

Three things that Fuel UX provides that are absolutely first-rate are a Data Grid, Tree View, and Wizard Bar; there are examples of these and more directly on the website.

Bootstrap Form Helpers

Like Fuel UX, Form Helpers adds new input types. Unlike the others, however, it doesn't in any way attempt to improve on what's already there.

Instead, what it attempts to do is add additional controls on top of those already provided, in an attempt to increase the number of input types available.

One thing it does add that seems sadly missing elsewhere is Time and Date pickers; it also adds some input elements that are unique to this kit such as Font Lists and Currency Pickers.

Useful Singular Components

Bootstrap Switch

Want an iOS-style sliding switch for your UIs? Then look no further.

Bootstrap Switch has it all. It's smoothly animated; you can change colors, text, label sizes, and everything else you can with regular TWB components.

It's also styled to look like the rest of Bootstrap and has a programmatic API to match.

Bootstrap Markdown

Markdown has taken the web development market by storm recently, and for good reason.

It allows you to add a rich in-page editor to your applications without enabling the masses to type any HTML they like into your page.

The Markdown extension gives you that nice, fluent front-end editing experience as made popular by sites such as StackOverflow, while allowing you to convert the data to and from regular HTML with nothing more than a few JavaScript calls.

All the Rest

As much as I'd love to list every single last one of them here, there are just too many extensions to even find them all.

While I was researching this book, I came across yet more sites that I never even knew existed the first time round, and the list continues to grow.

There are whole groups on YouTube producing entire series of tutorials on this amazing bit of kit, and there are sites like Tutorialzine. It pumps out Top 20 lists by the day.

Couple that with the fact that there are now at least five online drag-and-drop Bootstrap UI builders that I know of that allow you to create your layouts directly in the browser, there's no signs showing that any of this is slowing down.

Bootstrap, quite simply, puts the power of design and UI layout back in the developer's hands, leaving things in a nice, clean, easy-to-maintain state that can then be passed to a front-end graphic designer whose job it is to make things shine.

In the meantime, you can get on with the all-important task of writing your code, knowing that the visual side of what you do has now been taken care of.

Thanks for reading, and remember to continue building awesome stuff.