

TWITTER BOOTSTRAP 4

SUCCINCTLY

BY **PETER SHAW**

Twitter Bootstrap 4 Succinctly

By
Peter Shaw

Foreword by Daniel Jebaraj



Copyright © 2018 by Syncfusion, Inc.

2501 Aerial Center Parkway

Suite 200

Morrisville, NC 27560

USA

All rights reserved.

Important licensing information. Please read.

This book is available for free download from www.syncfusion.com on completion of a registration form.

If you obtained this book from any other source, please register and download a free copy from www.syncfusion.com.

This book is licensed for reading only if obtained from www.syncfusion.com.

This book is licensed strictly for personal or educational use.

Redistribution in any form is prohibited.

The authors and copyright holders provide absolutely no warranty for any information provided.

The authors and copyright holders shall not be liable for any claim, damages, or any other liability arising from, out of, or in connection with the information in this book.

Please do not use this book if the listed terms are unacceptable.

Use shall constitute acceptance of the terms listed.

SYNCFUSION, SUCCINCTLY, DELIVER INNOVATION WITH EASE, ESSENTIAL, and .NET ESSENTIALS are the registered trademarks of Syncfusion, Inc.

Technical Reviewer: David McQuiggin

Copy Editor: Courtney Wright

Acquisitions Coordinator: Hillary Bowling, online marketing manager, Syncfusion, Inc.

Proofreader: Graham High, senior content producer, Syncfusion, Inc.

Table of Contents

The Story behind the <i>Succinctly</i> Series of Books	6
About the Author	8
Introduction.....	9
Chapter 1 The Grids, the Grids, the Beautiful Grids.....	13
Containers.....	13
Rows	15
Columns	16
The new method.....	17
The old method.....	19
The best of both.....	19
Chapter 2 Content is King	21
Headings.....	21
Lists.....	25
Other typography features	29
Text colors.....	34
Block quotes and abbreviations	37
Displaying computer code	39
Chapter 3 An Image is Worth a Thousand Words.....	43
Responsive images and thumbnails	43
Media lists	47
Applying captions	50
Chapter 4 The Turning of the Tables	52
Pretty tables	57
Components of a Solid Foundation.....	61

Chapter 5 Feeding the User with Status	63
Alerts	63
Badges	67
Tooltips and popovers	69
Chapter 6 A Button-Shaped Form of Madness	73
Button basics	73
Button groups	78
Chapter 7 Navigating around the World	84
A basic navigation set	84
Actively marking the current entry	87
Full navigation bars	90
Chapter 8 The House of Cards	94
The Twitter card component	94
Headers and footers	95
Card colors	97
Chapter 9 A Form of Data Entry	100
Form validation	103
The Remainder of the Component Soup	107
Chapter 10 Bootstrap's Batman Utility Belt	108
Display utilities	108
Borders	109
Flex utilities	111
Sizing, margins, and padding	114
There's more	114
The Final Word	115

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 Face-book to help us spread the word about the *Succinctly* series!



About the Author

As an early adopter of IT back in the late 1970s, I started out with a rather small home computer known as a ZX81. Back then it was state-of-the-art: it had a full-screen 80 x 60 character display with 1 KB of memory, and a full operating system and programming language installed on the chips inside of it.

It wasn't long before this machine was replaced with an even more powerful 16 KB Tandy TRS-80, and ultimately, a BBC Model B. I thought I'd hit the top of the computing ladder the day my 32 KB Model B arrived, boasting 16 color graphics, 4-channel sound, and a massive 32 KB of memory. Little did I realize back then where it would lead me to.

After leaving school I spent a couple of years doing various low-grade computer courses, and it was at this time I obtained what I still consider my first real business-class computer, an Acorn Archimedes A5000. I studied at Newcastle University and ultimately ended up working with Windows and MS-DOS machines.

Over the years I've worked in many different IT environments, from being a fully trained GSM/3G communications engineer, working for one of the UK's largest telecoms companies, to doing small WordPress websites. I've covered a lot of ground in my 30-plus-year career so far. Today I run my own IT consultancy called Digital Solutions Computer Software UK, covering just about any IT need you might have.

I'm most often reachable via my Twitter account [@shawty_ds](#), and I help run LiDNUG ([Lidnug.org](#)), one of the most popular .NET user groups on the LinkedIn platform. If you want to read more about what I do, you can find my blog at [shawtyds.wordpress.com](#).

Please remember to thank Syncfusion for making this book a reality, along with many others in the series (there are well over 100 now, seven of which I've authored). Syncfusion does this as a service to the development community to help spread knowledge and make it easy for folks to learn and pick up new skills. Folks like me do this because we want to share our knowledge for the next generation to make use of. There are, unfortunately, some out there who try to hijack this for their own financial gain, so please, if you just bought this book from somewhere like Amazon, send it back, get a refund, report it as a pirate copy to Amazon and to Syncfusion, and then download a completely free copy from Syncfusion's website.

Finally, I just want to extend a big thank you to my friends and virtual internet family, and in particular, my long-suffering wife and technology widow, Samantha. I couldn't do half of what I do without you there to support me. You put up with my mood swings when a bit of code doesn't work, or I'm frustrated at what to write in my current book or blog post, and you somehow even manage to do all this even on those occasions when I'm working away from home and all we have is Skype between us. This book and many other things wouldn't have happened if you weren't along for the ride with me.

Introduction

About five years ago, I was asked to write a book for the Syncfusion *Succinctly* series about what was at the time a very little-known CSS framework called Bootstrap. Back then, it was officially only on version 2, and like many developers, I was dabbling with it to see what it could do.

What I found was something that would help me so much when putting together web-based user interfaces for applications I was writing, and something that I just couldn't put down.

Like many developers, I'm not particularly blessed in the graphics and design department. I can put together a functional, just barely usable UI, and let's just leave the rest to your imagination.

Bootstrap 2 changed all of that—pretty much overnight I found that, while I'm still not a design god, my UIs suddenly started to look and feel much more professional, even if they did all look the same. Moreover, my clients started to notice, and quickly prototyping things and letting them try stuff out actually became a simple process with almost instant feedback. Finally, because many designers were also familiar with Bootstrap, the work I was doing meant it was also now easier for them to take my layouts and add a touch of spice to them.

Not long after the first book was created, Bootstrap 3 was released, and Syncfusion asked me to write a second book to cover the newcomer. [Twitter Bootstrap 3 Succinctly](#) was published in December the same year.

In the past five years, Bootstrap has risen to become pretty much the number one front-end layout framework for at least 90 percent of new web projects created. It's available in every package source you can imagine, from NuGet to Bower and everything in between. It's the default in every template produced by every version of Visual Studio since VS 2010, and it continues to be the default even in the latest .NET Core SPA templates for the next-generation .NET platform.

This book was written to cover Bootstrap's next evolutionary step, from version 3 to version 4.

A lot has changed in Bootstrap 4, but many of the old favorites are still present. In the five years since version 3 debuted, the widespread adoption of HTML5 has made a number of new features and functionality available to web developers. Today, as you'll see later in this book, we are positively spoiled with the ease with which complicated user interfaces can be designed and laid out, even by developers who, like me, are design challenged.

If you're new to the world of Bootstrap, I would encourage you to read the previous book on [Bootstrap 3](#) before reading this one. Doing so will help you understand some of the older stuff still useable in Bootstrap 4, but that I won't go into much in this book.

Please note that as we move through the book, I'll start to refer to Bootstrap as simply BS, or BS3 or BS4 when referring to specific versions.

Bootstrap 4 has taken a long time to reach its initial beta stage, and even as I write this, it still has not made it to release candidate stage. We have, however, been promised that as of the date of me writing this, nothing is going to change in the existing toolkit as it stands. However, as with anything in the IT and software development industry, there can never be any guarantee of that.

Unlike in the previous book, I'm not going to start with a "What's changed" section; instead, I'm just going to jump straight in with some simple and practical examples that you can try. As we move on, I'll build these examples into more complex designs, while introducing aspects of the framework to help you.

Most of the examples will work if you have the HTML and CSS file on your machine, and open the HTML file in your chosen browser. I would, however, recommend that you use the latest version of Chrome (60.0.3112 as of the date I write this).

I would also suggest that you use an environment where you can serve the code to your browser, such as IIS7/8 if you're on Windows, or Apache/Nginx if you're using a Mac or Linux, or simply just IIS Express if you're using Visual Studio. Whichever you choose, it's beyond the scope of this book for me to take you through setting this up. If you're serious about web development, then it's a time investment that's worth making.

Lastly, we'll be using the same template HTML for every exercise in this book, so rather than repeat it in each chapter, I'll show you what it looks like here.

Code Listing 1: Basic Bootstrap 4 template

```
<!DOCTYPE html>
<html lang="en">
<head>
  <!-- Required meta tags -->
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width, initial-scale=1, shrink-
to-fit=no">

  <!-- Bootstrap CSS -->
  <link rel="stylesheet"
href="https://stackpath.bootstrapcdn.com/bootstrap/4.1.0/css/bootstrap.min.cs
s" integrity="sha384-
9gVQ4dYFwwWSjIDZnLEWnxCjeSWFphJiwGPXr1jddIh0egiu1Fw05qRGvFX0dJJZ4"
crossorigin="anonymous">
  <link rel="stylesheet"
href="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0-
beta.2/css/bootstrap.min.css"
crossorigin="anonymous">
</head>
<body>
  <!-- Page content goes here -->
  <!-- Optional JavaScript -->
  <!-- jQuery first, then Popper.js, then Bootstrap JS -->
  <script src="https://code.jquery.com/jquery-3.2.1.slim.min.js">
```

```

crossorigin="anonymous"></script>
<script
src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.14.0/umd/popper.min.js"
integrity="sha384-
cs/chFZiN24E4KMATLdqvsezGxaGsi4hLG0zlxWp5UzB1LY//20VyM2taTB4QvJ"
crossorigin="anonymous"></script>
<script
src="https://stackpath.bootstrapcdn.com/bootstrap/4.1.0/js/bootstrap.min.js"
integrity="sha384-
uefMccjFJAIV6A+rW+L4AHf99KvxDjWSu1z9VI8SKNVmz4sk7buKt/6v9KI65qnm"
crossorigin="anonymous"></script>
</body>
</html>

```

This template is the minimum recommended HTML snippet to use before adding any BS-specific content. It pulls in the required jQuery and Bootstrap scripts and resources from a cloud-based CDN, which means you will need a live connection to the internet while using it.

You can download a local copy of BS [here](#). Once you do this, you can work entirely offline. The copy used throughout this book is version 4.1 of the stable release, and all code examples have been tested and work against it.

To use a local copy, you need to download and unzip the ZIP files available in the compiled CSS and JS section of the download page. In the ZIP file, you will find a **js** and a **css** folder; if you copy your HTML template to the same folder where you placed these two unzipped folders, you can alter the template in Code Listing 1 as follows.

Code Listing 2: Listing 1 altered to use a local copy of Bootstrap

```

<!DOCTYPE html>
<html lang="en">
<head>
  <!-- Required meta tags -->
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width, initial-scale=1, shrink-
to-fit=no">

  <!-- Bootstrap CSS -->
  <link rel="stylesheet" href="css/bootstrap.min.css">
</head>
<body>
  <!-- Page content goes here -->

  <!-- Optional JavaScript -->
  <!-- jQuery first, then Popper.js, then Bootstrap JS -->
  <script src="https://code.jquery.com/jquery-3.3.1.slim.min.js"
integrity="sha384-
q8i/X+965Dz00rT7abK41JStQIAqVgRVzpbzo5smXKp4YfRvH+8abtTE1Pi6jizo"

```

```
crossorigin="anonymous"></script>
  <script
src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.14.0/umd/popper.min.js" integrity="sha384-
cs/chFZiN24E4KMATLdqvsezGxaGsi4hLG0zIXwp5UZB1LY//20VyM2taTB4QvJ"
crossorigin="anonymous"></script>  <script
src="/js/bootstrap.min.js"></script>
</body>
</html>
```

Please be aware that if you do a local installation like this, you may very well need to look in the CSS and JS folders to ensure that the versions you are using match.

You'll notice that in each `<script>` and `<link>` tag in the previous code samples, there are still version numbers, and while these were correct when this book was written, they may have changed by the time you try the code yourself.

The bottom line is this: if you get 404 errors loading Bootstrap or any of the files required in the template, double-check that you have the correct versions and that the file names match any that you may have installed locally.

Ready to get started?

Great! Then let's begin...

Chapter 1 The Grids, the Grids, the Beautiful Grids

For as long as I can remember, developers using HTML technologies have struggled immensely with laying out even the most basic of grid patterns. If you're old enough to remember HTML4 and (*shudder*) even versions 2 and 3, then you'll be all too familiar with the abuse HTML tables received as many folks (me included) tried to use them to control and constrain their layouts.

With Bootstrap, all of this is a thing of the past.

In recent years, the CSS specification has brought forward two new layout modes, called Flexbox and CSS Grid, the latter of which is the newest kid on the block. BS4 takes full advantage of everything that Flexbox has to offer, and this has not only given us the original world-class pre-Flexbox grid that BS3 already had, but has seriously supercharged what BS4 has available.

BS4 is fully Flexbox to the core, and uses it for just about everything. All block-level containers and elements are now controlled using Flexbox, enabling some very interesting layouts to be created.

Before we talk more about that, let's start with some simple examples.

Containers

Before you can do anything with any of the Bootstrap grid tools, you need a container. Containers are the daddy, the alpha, the *numero uno*, and they contain everything else that your layout will use.

Containers come in two flavors: a fluid container whose width is always 100 percent of the width of its parent (usually the page), and a standard container that will always use a center column fixed width, whose size is dependent on the overall parent width.

Using the template from Code Listing 1 or 2, add the following just after `<!--Page content goes here-->`.

Code Listing 3: Our first container

```
<div class="container" style="background-color: yellow">Hello Bootstrap</div>
```

If you save your file and load it into your browser, you should see something like the following figure.



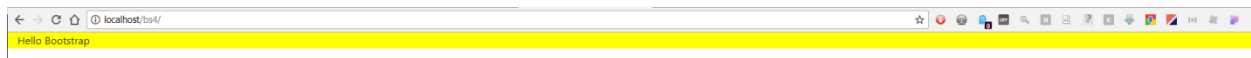
Figure 1: Our first container, as seen in Chrome

What you've just created is a fixed-width container, and if you change the width of your browser window, you should see that it remains centered, and where possible, keeps an equally spaced margin at each side.

Everything you just added is regular HTML; the magic part is the class that was added to it.

Most of Bootstrap's magic is applied simply by choosing the correct class to add to your code. In this case, we added **container**, and we also added some extra style so that you could see the background of the `<div>` tag it created.

Load your HTML file back into your editor and change the class so that it reads **container-fluid** rather than **container**, and then press **F5** to refresh your browser. The result should look like Figure 2.



*Figure 2: Our first container with **container-fluid** applied*

You'll notice that the container now occupies 100 percent of the width of the page. If you were creating a layout that required you to use the full page width, then you've just created the base container to add the rest of your content into.

You might be wondering what the purpose of the fixed width container is—surely most of the layouts you're going to want to do will be full width, right?

Well, yes and no. If you're building a blog or magazine-style site, then you'll definitely want to use a fixed width, and in fact, anything that has a high density of words to read should use a fixed-width container. This is because if the width is too great, then people have to move their heads side to side as they read.

With a balanced central column, usually all that has to move is a person's eyes.

If you're creating an application on the other hand, then there is an expectation that the interface will use the full browser window. Not only that, but there is an expectation that the user interface will actually adapt itself to varying sizes of page width depending on the device being used.

When you create your base container, you're creating more than just a placeholder—you're creating a flexible box that will resize itself, and even in some cases lay itself out differently depending on the screen size being used. This technique is called responsive design, and BS4 (like its predecessor) fully supports it with a mobile-first strategy.



Note: Mobile-first strategy means that you concentrate on making your design work on many different mobile platforms first, and then you concentrate on what it will look like on the desktop.

In short, BS4 sets all of this up for you, and adds a staggering amount of support to get you up and running quickly by just adding a simple class to your HTML tag.

Rows

Containers are all well and good, but most layouts also consist of rows and columns to divide the space used into different zones, depending on the designer's needs.

Applications, for example, generally have a header, body, and footer. Usually only the content within the central body scrolls; the rest normally stays in a fixed position.

Reading layouts usually have a header and a content area, and then a footer, but the entire page usually scrolls, with the header scrolling up and off the screen, and the footer scrolling in as the bottom of the page is reached. Reading layouts usually also have boxouts in the margins or advertising space in the header.

The first step to making all this happen is to divide your layout into rows.

Using rows is rather like defining rows in a table; you need to work out how you want to use the vertical space, and how many slots you want, before you can start to divide those slots horizontally.

Open your HTML file (or create a new one using the template) and make sure your body code looks like the following listing.

Code Listing 4: Our first container with a row

```
<div class="container" style="background-color: yellow">  
  <div class="row" style="background-color: green">Hello Bootstrap Row</div>  
</div>
```

All being well, you should see something similar to Figure 3.



Figure 3: Our container with a row in it

You might be forgiven for thinking, “That looks just the same as the first container example, except now it’s green. What’s the point of adding it?” I can understand why you would come to that conclusion.

A row is designed to consume 100 percent of its parent space, but unlike a container, it doesn’t set anything up to support responsive design. It simply makes sure that it follows whatever the parent container width is, and this includes nesting rows.

Add two more `<div>`s of different colors to the listing in Code Listing 4, so that it now looks like this:

Code Listing 5: Fixed container with three rows defined

```
<div class="container" style="background-color: yellow">  
  <div class="row" style="background-color: green">Hello Bootstrap Row</div>
```

```
<div class="row" style="background-color: red">Hello Bootstrap Row</div>  
<div class="row" style="background-color: blue">Hello Bootstrap Row</div>  
</div>
```

Save and reload that into your browser, and you should have something that looks the same as Figure 4.



Figure 4: Three rows

You should see that each row sits neatly on top of another, and that each obeys the 100 percent space available within the parent container.

If you change your container to a fluid one just as you did before, then you'll see that all three rows will also follow the container and its width.

What you have now, however, are three fully self-contained rows, and just like rows in a spreadsheet or rows in a table, each of them is capable of being divided vertically.

What's important is that you're partitioning off sections of your document, and as you'll see later in the book, you could fix the height on each of these rows so that you always have a fixed height header and footer, and a different height content area.



Note: In a later chapter, I'll show you how to create some common layouts using the tools BS4 provides for you, one of which will be the “golden layout,” or the fixed header, footer, and scrolling content region.

And that's all there is for rows. You can nest them if you want to, but there's very little need for that. What's more likely to happen is that you'll produce a static template of some description (similar to the template we defined earlier), and you'll use a front-end framework, such as Aurelia, React, Redux, or Vue, to host that template. The template will in turn have its own container and row layouts on a smaller scale, and if everything is nested correctly when your app is compiled to production, the components and the template should just re-size as needed for the display.

Columns

Of course, having rows in a layout is not much use unless you can further divide those rows into columns, and BS4 definitely has you covered here with some rather surprising abilities.

First things first: to make things simpler, let's change our body code so that we are back to having just one row.

Code Listing 6: Reset back to one column

```
<!-- Page content goes here -->
<div class="container" style="background-color: yellow">
  <div class="row" style="background-color: green">Hello Bootstrap
Row</div>
</div>
```

This should take your document back to looking as it did in [Figure 3](#). If you want to keep what you've done previously, then by all means save this under a new name and start a new document.

In BS3, you always had to specify the specific number of units that a column used, and the maximum number of units allowed per row was 12 (unless you created a custom BS3 build).

In BS4 this is still true, and you can specify the row sizes just as you used to, but BS4 also has a new way of specifying columns (thanks to Flexbox) that allows you to just set a number of columns and have the columns work out how much space they should consume.

I'll start with the newer way of doing things, and then quickly cover the old way so that you can see the advantages and disadvantages of both.

The new method

As with everything Bootstrap, it's all about the CSS class names you add to your markup. With columns, you use the **col-XXXXX** classes.

Take the code from Code Listing 6 and change it so that it looks as follows.

Code Listing 7: Three equal-sized columns

```
<!-- Page content goes here -->
<div class="container" style="background-color: yellow">
  <div class="row" style="background-color: green">
    <div class="col-sm" style="background-color: pink">Col 1</div>
    <div class="col-sm" style="background-color: hotpink">Col 2</div>
    <div class="col-sm" style="background-color: deeppink">Col 3</div>
  </div>
</div>
```

Once you save and load this into your browser, you should see something like the following figure.



Figure 5: Three equal-spaced columns

The first thing you should notice is that all three columns each have the same amount of space.

You've gotten this far by just adding three extra `<div>`s to your row and telling your browser that these are small equal columns (that's what the `sm` suffix means). However, the best part is yet to come.

If you resize your browser so that it's more of a mobile width rather than a desktop width, you should see the following output.

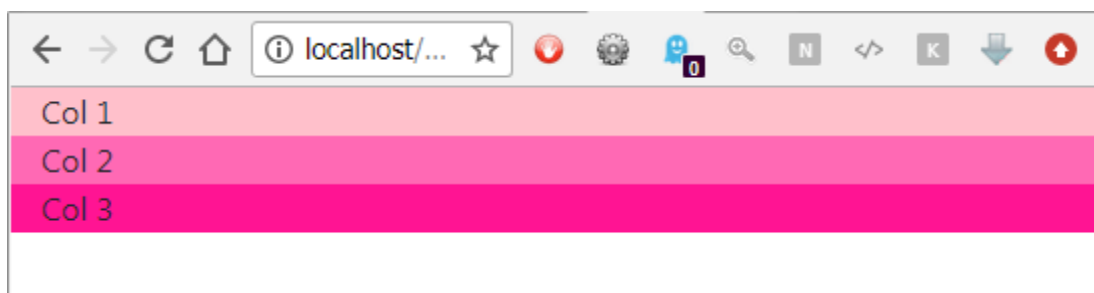


Figure 6: The same layout in Figure 5 with the width reduced

Your three columns have magically repositioned themselves so that they stack neatly on top of each other as the display gets narrower. It's the `sm` suffix that decides at what point this switch happens, and there are a few different suffixes defined in the base BS4 grid system.

Here are the suffix and sizing options, as provided in the BS4 documentation.

	Extra small <576px	Small ≥576px	Medium ≥768px	Large ≥992px	Extra large ≥1200px
Max container width	None (auto)	540px	720px	960px	1140px
Class prefix	<code>.col-</code>	<code>.col-sm-</code>	<code>.col-md-</code>	<code>.col-lg-</code>	<code>.col-xl-</code>
# of columns	12				
Gutter width	30px (15px on each side of a column)				
Nestable	Yes				
Column ordering	Yes				

Figure 7: Bootstrap column sizing options

Each of these measurements is based on the standard size of 12 units in a row. So in our example from Code Listing 7, each column will be 4 units in size.

Because we didn't specify any actual column unit sizes, however, BS4 automatically worked that out for us. If we wanted to, we could have quite easily done the following:

Code Listing 8: Three equally sized columns with the sizes specified by the class

```
<!-- Page content goes here -->
<div class="container" style="background-color: yellow">
  <div class="row" style="background-color: green">
    <div class="col-sm-4" style="background-color: pink">Col 1</div>
    <div class="col-sm-4" style="background-color: hotpink">Col 2</div>
    <div class="col-sm-4" style="background-color: deeppink">Col 3</div>
  </div>
</div>
```

The old method

All that's changed is the class name; we've simply added a **-4** after the **sm** suffix to tell BS4 that it should always use 4 units (out of the 12 available) for that column.

Again, if you resize your browser, you should see exactly the same effect as seen in Figures 5 and 6.

In previous versions of Bootstrap, you had to work this way all the time, so you always had to remember what you set your sizes to, and often had to mentally picture your layouts and the space they used. If you exceed the number of units you have available in a row, then your columns will spill onto the next line. In some cases, this may be what you want; for example, if you were creating an image gallery. In most cases, however, it's not the desired output, so you always need to ensure that you don't exceed 12 units in one row.

The best of both

The crux of the two methods is that in BS4 you can easily mix things and use both side by side.

For example, let's suppose you want your center column to always take up 6 units of space no matter what, but your two side columns should always just divide the remaining space between themselves. In BS3, you'd have to make sure that you had a set number of units for all three columns specified at all the different unit sets you wanted to support, leading to class names such as **col-xs-2**, **col-sm-2**, **col-md-1**, **col-lg-1**, and so on.

In BS4, it's as simple as doing this:

Code Listing 9: Three columns with a fixed center

```
<!-- Page content goes here -->
<div class="container" style="background-color: yellow">
  <div class="row" style="background-color: green">
    <div class="col-sm" style="background-color: pink">Col 1</div>
    <div class="col-sm-6" style="background-color: hotpink">Col 2</div>
    <div class="col-sm" style="background-color: deeppink">Col 3</div>
  </div>
```

```
</div>
```

Which when rendered, will give you:



Figure 8: The column layout produced by Code Listing 9

As you can see, the center column is now the widest of the three, and the left and right columns share what's left between them.

Depending on your needs, you may still need to specify multiple suffix breakpoints on your classes using **sm**, **lg**, and others where needed, but you won't have to make every single size specific.

It gets better than this though—I've left the best for last.

If you remove the suffix altogether, then your columns will work at all breakpoints. This means if you rewrite Code Listing 9 so that it reads as shown in Code Listing 10, you'll get exactly the same output as in Figure 8, but this time you don't have to specify multiple suffixes because it now works with all sizes.

Code Listing 10: Listing 9 revised to remove the breakpoint suffixes

```
<!-- Page content goes here -->
<div class="container" style="background-color: yellow">
  <div class="row" style="background-color: green">
    <div class="col" style="background-color: pink">Col 1</div>
    <div class="col-6" style="background-color: hotpink">Col 2</div>
    <div class="col" style="background-color: deeppink">Col 3</div>
  </div>
</div>
```

You do, however, have a price to pay: you lose some of the automatic stacking of columns. Using the non-suffix method, BS will attempt to make your row stay consistent across small displays. With an appropriate arrangement of columns and rows, it's still possible to take advantage of the clever things BS can do; you just need to get creative.

There's much more you can do with columns and rows, such as vertical and horizontal alignment, nesting, offsets, and floating containers. You can find the rest of this in the [official documentation](#).

Chapter 2 Content is King

No modern website or web application is complete without content. From grids of products for sale, to witticisms and block quotes of people spanning far and wide across our digital world, content is the stuff we fill our pages with, the thing that we use to define our images, headings, sections, and much more.

Most developer types don't really think about content when producing web layouts, and to be fair, most of them are happy to just put in some kind of placeholder and let the designers deal with it afterwards.

BS4, like its predecessors, makes it easy to make your UI look great with very little work. To start with, many of the common things like headings, lists, `<pre>` tags, and forms are already defined for you. In many cases, you don't even need to specify any classes of any kind—just using the correct semantic tags in the correct order is enough.



Note: *Semantic tags are part of the HTML5 standard that set out to make tags in an HTML document self-documenting. We're all familiar with an `<h1>` being a heading tag level 1, or a `<p>` tag being a paragraph, but under HTML5 we now have tags such as `<section>`, `<article>`, and `<aside>` to mark the content inside them as particular document entities. `<form>`, `<input>`, and other similar tags are also designed with the same goal in mind, so that software or scripts working with the content can make some assumptions about what a specific piece of content is being used for.*

As you'll see in this chapter, BS4 can make even the simplest of text blocks look great, even if it's just placeholder content you're producing.

Headings

Nothing special here—headings have always been (and always will be) specified using the standard `<hx>` HTML tag. The x is typically a value from 1 to 6, giving you six different levels of heading.

A good example of the use of headings is in this book. If you look at the header “Headings” at the beginning of this section, and then scroll up to the start of this chapter and look at the chapter title, you can see heading levels 1 and 2 from my Microsoft Word template in play.

Add the following body code to your BS4 template file.

Code Listing 11: Content headings

```
<!-- Page content goes here -->
<div class="container">
  <div class="row">
    <div class="col">
```

```

    <h1>Heading Level 1</h1>
    <h2>Heading Level 2</h2>
    <h3>Heading Level 3</h3>
    <h4>Heading Level 4</h4>
    <h5>Heading Level 5</h5>
    <h6>Heading Level 6</h6>
  </div>
</div>
</div>

```

If all goes OK, then your browser should display this:

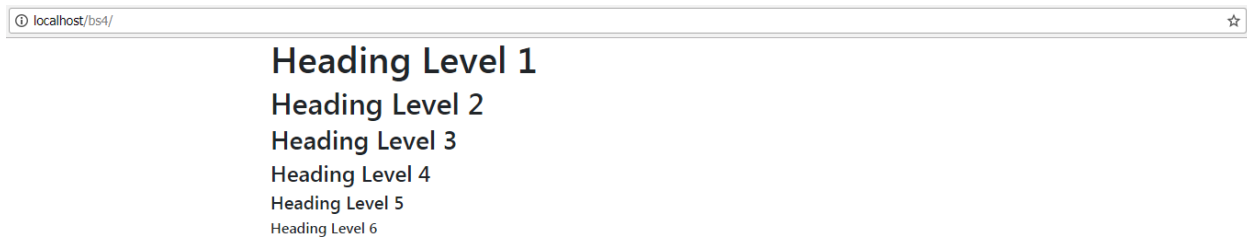


Figure 9: The heading levels produced by Code Listing 11

If you've used previous versions of Bootstrap, then you might notice that the fonts in the previous example look like the default browser sans-serif fonts, rather than the custom font that BS3 and BS2 used.

This is intentional, as the BS authors decided that it was much better to return to a native font stack. There are two main reasons for this. First, a native font stack will always render faster than a custom one, and second, it's not up to BS to dictate what font your design should use—it's the decision of the designer, and always should be.

Because there are so many different methods to manage font loading in modern web development, maintaining a font stack that works exactly the way everyone using BS4 might want to use it would be a monumental task. It's simpler to let those working on the project decide what to use instead, and have BS4 concentrate on layout and sizing.

Headings aren't just boring and straightforward, however. BS4 includes a heading style for larger, more prominent headings, too.

Change the code in Code Listing 11 so that it looks like this:

Code Listing 12: Headings using the display size class

```

<!-- Page content goes here -->
<div class="container">
  <div class="row">
    <div class="col">
      <h1 class="display-1">Heading Level 1.1</h1>
      <h1 class="display-2">Heading Level 1.2</h1>
    </div>
  </div>
</div>

```

```

    <h1 class="display-3">Heading Level 1.3</h1>
    <h1 class="display-4">Heading Level 1.4</h1>
  </div>
</div>
</div>

```

You'll now see your headings become much larger, but also thinner and easier to read.

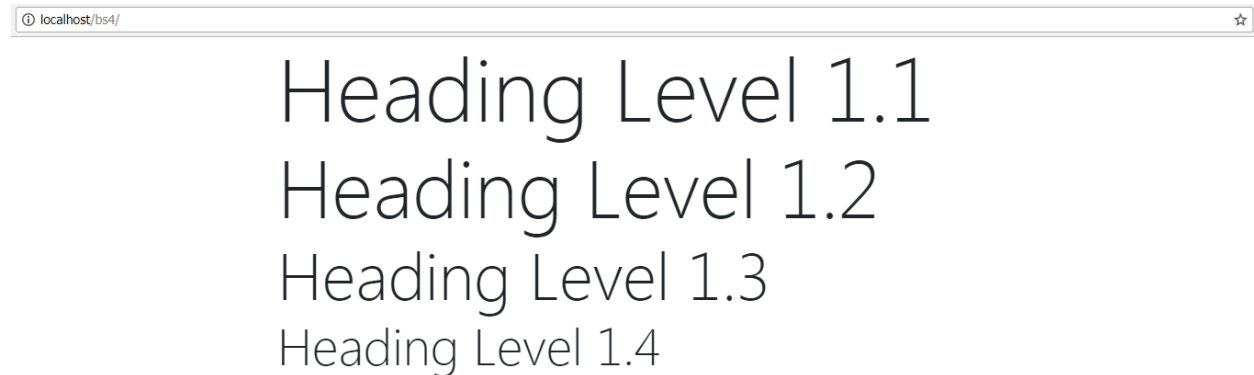


Figure 10: The output from Code Listing 12 showing the `display` class being added to a header tag

You can apply the display size classes to any of the `<h>` tags you like. In fact, you can apply them to any block-level element, but the content using them will always take on the same size. So, an `<h1>` and an `<h2>` tag using the `display-1` class will be indistinguishable from each other when rendered in the browser.

If you need to keep your HTML5 valid (something you should always try to do), then there are places where you might want to use a heading tag but are unable to because it would mean having invalid HTML content.

One example of this might be when you want to mark off a section using a `` tag, but that section might need to use its own `<h>`-style tags in its content.

Under HTML5, it's invalid to place a block element inside of an inline element, and a `` tag is an inline element.

Fear not: BS4 has your back covered here, as the following example shows.

Code Listing 13: An `<h1>` tag created in both invalid and valid formats showing the `hx` BS4 class

```

<!-- Page content goes here -->
<div class="container">
  <div class="row">
    <div class="col">
      <span>
        <h1>This is an invalid heading</h1>
        <span>This is an inner paragraph. The paragraph is in an inner

```

```

span, so it's valid.</span>
    </span>
  </div>
</div>
<div class="row">
  <div class="col">
    <span>
      <span class="h1">This is a valid heading</span><br/>
      <span>This is an inner paragraph. The paragraph is in an inner
span, so it's valid.</span>
    </span>
  </div>
</div>
</div>

```

If you render that in the browser, you'll see the following output.

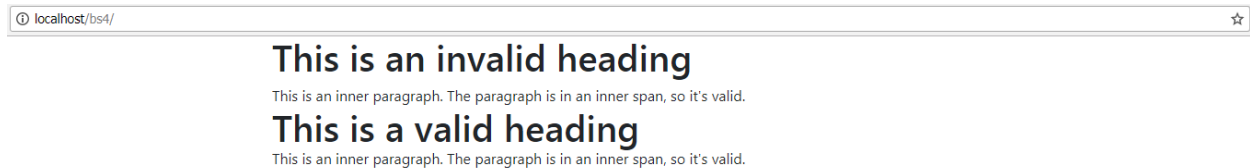


Figure 11: The output from Code Listing 13

To the inexperienced eye, nothing looks out of place, and most non-technical users wouldn't think twice about it. However, if you look at the code in Visual Studio, you'll see a green squiggly.



Figure 12: Visual Studio showing that the HTML code is invalid.



Note: The green squiggly in Figure 12 is specific to Visual Studio 2017 (the editor I'm using while writing this book). If you are using a different editor, the highlighting and error notifications may be different, or in some cases, may not even be shown at all. As this is just a warning, it's nothing major. If you are keen to ensure your HTML code is 100 percent valid, there are many validation tools and online services you may wish to explore.

There are other instances where this comes in handy too, such as in headers in tables, or inside field groups next to form inputs. BS4 provides `<h1>` to `<h6>` classes, and each of them is sized to match the size that would be given, should the actual `<hx>` tag be used instead.

Lists

Just like heading tags, lists are also handled in place. When you use the correct tags in the correct order, BS4 will simply handle it for you, as the following example shows.

Code Listing 14: Standard list items

```
<!-- Page content goes here -->
<div class="container">
  <div class="row">
    <div class="col">
      <ul>
        <li>List Item 1</li>
        <li>List Item 2</li>
        <li>
          List Item 3<ul>
            <li>List Item 3a.</li>
            <li>List Item 3b.</li>
            <li>List Item 3c.</li>
          </ul>
        </li>
        <li>List Item 4</li>
        <li>List Item 5</li>
      </ul>
    </div>
  </div>
</div>
```

This should produce the following output.

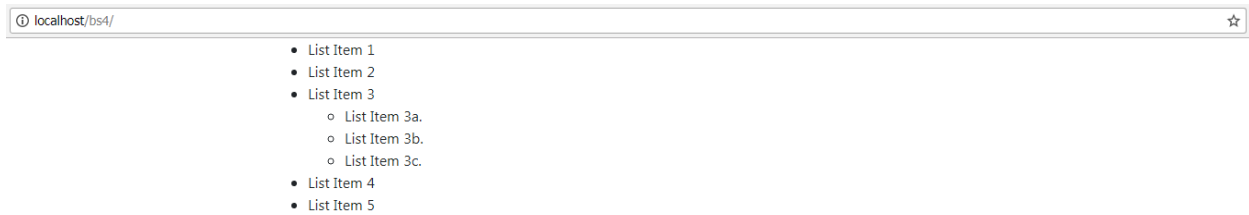


Figure 13: The output produced by listing 14

As you can see, the nesting is properly handled, and the font is sized, indented, and balanced correctly.

Change the `` tags to `` tags in Code Listing 14, and you'll see the same is also true of an ordered list.



Figure 14: The output produced by Code Listing 14, with the `` tags changed to `` tags

If you wish, you can remove the list styling and just have BS4 handle the spacing and margins for you. Change Code Listing 14 to add the extra `list-unstyled` class, so that it looks like the code in Code Listing 15.

Code Listing 15: Standard list items with the `un-styled` class

```
<!-- Page content goes here -->
<div class="container">
  <div class="row">
    <div class="col">
      <ul class="list-unstyled">
        <li>List Item 1</li>
        <li>List Item 2</li>
        <li>
          List Item 3<ul>
            <li>List Item 3a.</li>
            <li>List Item 3b.</li>
            <li>List Item 3c.</li>
          </ul>
        </li>
        <li>List Item 4</li>
        <li>List Item 5</li>
      </ul>
    </div>
  </div>
</div>
```

```
</div>
```

Note that all we changed was the class on the first `` tag, nothing else. The inner `` tag is a new set of list items, so it automatically has the styles turned back on, leading to a list that should look like the image in Figure 15.

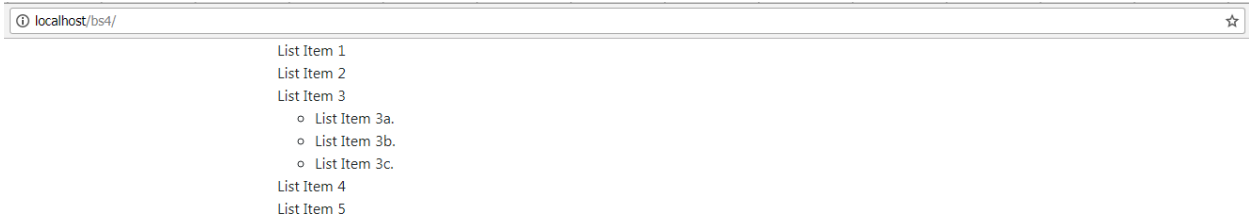


Figure 15: The output produced by a normal list tag with the `un-styled` class added

Using the class `list-unstyled` in this way allows you to not worry about the inner parts of your lists maintaining their styling while still allowing you to treat the parent list differently.

One more class you have at your disposal when it comes to regular list tags is the `list-inline` class. As its name suggests, this class puts all your list items in a single line, which you could use if you needed a horizontal navigation bar or drop-down list, for example.

Using the `inline` class is a little more involved, as not only do you need to apply the `inline` class to the ``, but you also need to apply a class to each of the `` elements, as Code Listing 16 shows.

Code Listing 16: Standard list using the inline classes

```
<!-- Page content goes here -->
<div class="container">
  <div class="row">
    <div class="col">
      <ul class="list-inline">
        <li class="list-inline-item">List Item 1</li>
        <li class="list-inline-item">List Item 2</li>
        <li class="list-inline-item">List Item 4</li>
        <li class="list-inline-item">List Item 5</li>
      </ul>
    </div>
  </div>
</div>
```

Pop the body code in Code Listing 16 into your template and render it in the browser. You should now see that each of your elements is inline horizontally, rather than stacked vertically as they were previously.



Figure 16: List items displayed using the inline styles

The last thing I'd like to show you before we move on is the default definition list layout. Definition lists are quite useful because they can convey useful information, such as lists of names and descriptions, or abbreviations along with their meanings. As with most of the other semantic tags, this gives meaning to the content, allowing automated processes to make better sense of it.

Add the following code to your template file.

Code Listing 17: Standard definition list

```
<!-- Page content goes here -->
<div class="container">
  <div class="row">
    <div class="col">
      <dl>
        <dt>HTML</dt>
        <dd>Hypertext Markup Language, a variation of XML that defines the
structure of a webpage to the client displaying it.</dd>
        <dt>CSS</dt>
        <dd>Cascading Style Sheets, not exactly a language, but a list of
instructions telling the HTML client how to style the layout defined in the
HTML code.</dd>
        <dt>JavaScript</dt>
        <dd>The scripting language used inside of HTML documents to add
interactivity and programmed features.</dd>
        <dt>Style Sheet</dt>
        <dd>A plain text file that is usually separate from the HTML
document and holds CSS definitions to apply to the document.</dd>
      </dl>
    </div>
  </div>
</div>
```

Next, render it in your browser.

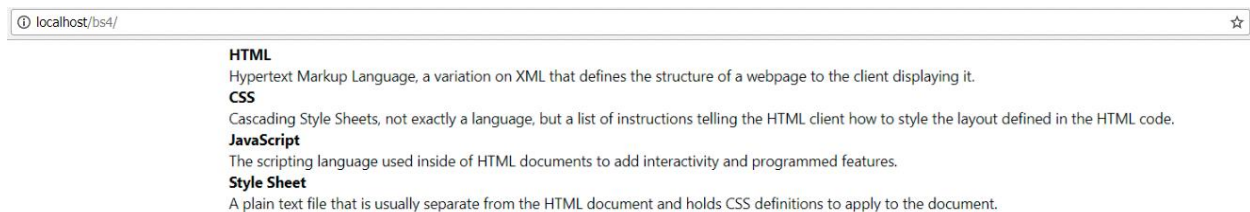


Figure 17: Standard definition list produced by Code Listing 17

Again, there are no extra classes required, just standard markup, but BS4 formats and lays out margins and such automatically for you.

In the actual BS4 docs, there's a longer example that shows the use of grids, columns, and rows to produce a side-by-side list, and some nested examples. The only classes that needed to be added were the classes to define the column layout.

Other typography features

As well as heading and list tags, paragraphs and other standard text elements are also treated in more or less the same way. By adding a `<p>` tag to your content and filling it in, BS4 will automatically set it up with the default layout to work with the rest of the styles.

Like the heading tags, paragraphs will not be given a custom font, but you can still make some easy changes by using them in combination with other tags.

The main tags to be used in combination with the paragraph are:

- `<mark>`: Used to highlight text as though it was marked with a highlighter.
- `<s>`: Used to strike out text.
- ``: Used to put a line of deletion through text.
- `<small>`: Used to decrease the size of the text slightly, and mark it as less important.
- ``: Makes text bold.
- ``: Used to emphasize a piece of text, but without making it more important than the rest of the paragraph.
- `<u>`: Underlines the text.
- `<ins>`: Adds an underline to the text inside of it, with the intention of marking it as an addition to the original text.

As with all the previous examples, change the body code in your template to the following:

Code Listing 18: Standard paragraph typography styles

```
<!-- Page content goes here -->
<div class="container">
  <div class="row">
    <div class="col">

      <p><mark>Highlighted</mark> text is easy to achieve by using the
<mark>mark tag</mark>. Again, semantics are important.</p>
      <p><s>Using an "S" tag inside a paragraph will strike the text
through, indicating that the text is no longer correct or accurate.</s></p>
      <p><del>Using the "DEL" tag has a similar effect to "S", but the
meaning is different, in that the text is to be or has been
deleted.</del></p>
      <p><small>The "SMALL" tag is mostly used for the small print, and
will be liked by legal folks :-)</small></p>
      <p><strong>And "STRONG" is used to make text bolder so it stands
out.</strong></p>
```

```

    <p><em>Whereas the "EM" tag is designed to emphasize text by making
it appear italic.</em></p>
    <p><u>The "U" tag will underline your text.</u></p>
    <p>And finally, "INS" is designed to make it look like text <ins>has
been inserted</ins> into a paragraph.</p>

    </div>
  </div>
</div>

```

And render it in your browser:

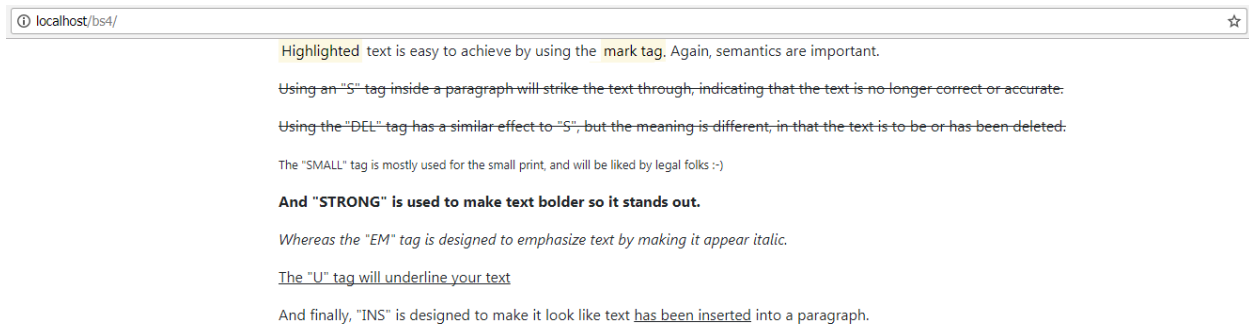


Figure 18: Standard typography styles

As with most of what we've seen so far, you can see straightaway that it's all semantic.

The `<u>` and `<ins>` tags both give exactly the same style, but to anything looking at the page content, it would be able to tell that one is an insertion, while another is simply decorative. The `<s>` and `` tags work the same way, allowing you to use custom CSS to visually style things for differentiation, but without running the risk of hurting the semantics on your page.



Note: Google will give a page that follows semantic rules a better ranking in its search index than a page that does not, especially if you use things like correctly marked-up postal addresses and site names. In fact, much of the information that Google displays to the right of a search page comes from it being able to look inside your page code and understand which parts refer to names, addresses, maps, dates, times, etc. Using semantic tags is not just about helping Bootstrap, it's about the world stage and your visibility, too.

If you add the `lead` class to a paragraph tag, BS4 will render that paragraph in a larger font to make it look like an opening or lead paragraph. Add the code in Code Listing 19 to your page to see this in action.

Code Listing 19: The paragraph *Lead* class

```

<!-- Page content goes here -->
<div class="container">

```

```

<div class="row">
  <div class="col">

    <p class="lead">I am the lead paragraph in this document. My size is
increased to make me look more prominent compared to the rest of the text
below me.</p>
    <p>I'm just a standard paragraph tag. I'm not more important than the
one above me, but I'm also not the start of the text on this page or in this
document.</p>

  </div>
</div>
</div>

```

Code Listing 19 should give you the output shown in Figure 19, where you can clearly see which of the two paragraphs is intended to be the opening one.

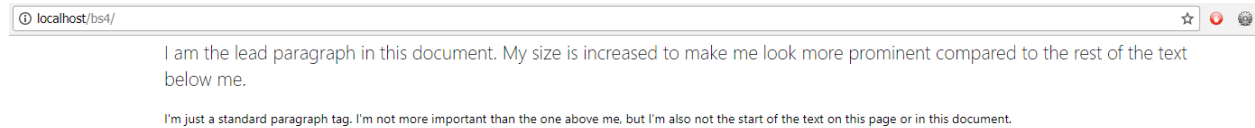


Figure 19: Paragraphs in the lead and non-lead style as produced by Code Listing 19

Once we start to move away from marking things up using semantic tags, and into using just the styling classes that BS4 provides us, we can start to take advantage of things like justified and aligned text, and contextual colors that can be automatically swapped with custom BS4 color palettes and custom BS4 CSS builds.

Text alignment comes in four different flavors: **left**, **center**, **right**, and **justified**. **left** sticks the text to the left of the parent container, **right** positions it to the right of the parent container, and **center** positions it in the center with equal space on either side. **justified** alignment tries to make each line in the paragraph equal to the width of the parent container.

Notice that I describe this in terms of the parent container. The text utilities in previous versions of Bootstrap were not particularly suited to being used at anything other than row level or page level, and would often look slightly out of place when used. With BS4 using Flexbox behind the scenes for its layout, the text utilities now use the new CSS rules and styles available to align text within the rows, columns, and regular `<div>` tags that you would now use in your page layouts.

This makes the layout of items much more flexible, but more importantly, it also makes it simpler to make reusable chunks of content and nest them to quite deep levels.

The code in Code Listing 20 gives you an example of the alignment utilities in action.

Code Listing 20: BS4 alignment classes

```

<!-- Page content goes here -->

```

```

<div class="container">
  <div class="row">
    <div class="col">

      <p class="text-justify">...trimmed...</p>
      <p class="text-left">Left Aligned Text</p>
      <p class="text-center">Center Aligned Text</p>
      <p class="text-right">Right Aligned Text</p>
      <p class="text-justify">...trimmed...</p>

    </div>
  </div>
</div>

```

I've trimmed down the amount of text in the justified paragraphs to make it fit in the book, but feel free to add lots of new text in the place of `...trimmed...`. The more text you add, the better the effect will be. I used the [Blind Text Generator](#) to generate the text for my example.

Once you have some text in and the code from Code Listing 20 added to your BS4 template, you should see something similar to Figure 20.

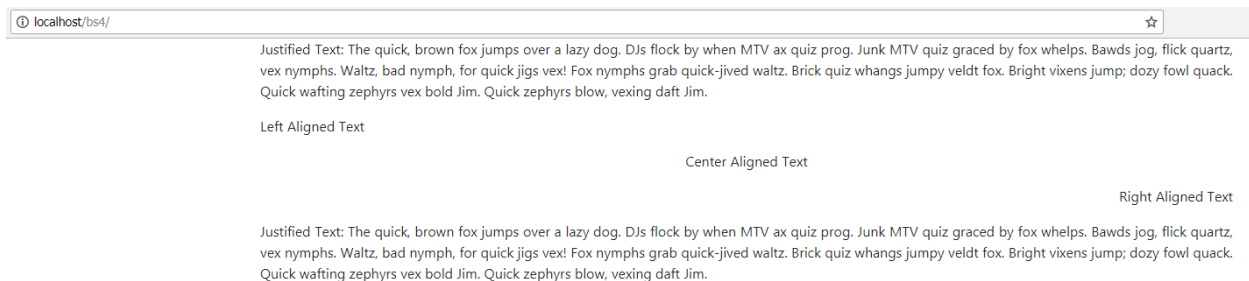


Figure 20: Output produced by the alignment classes code listing

The text utilities also have classes to control wrapping and overflow. By adding the `overflow` class, you allow the text to break outside the boundaries of the parent container, and if that container is the page, potentially disappear off the edge of the page. To be fair to the BS4 development team, I'm sure they had some specific use cases in mind for this, but as of yet, I've not actually found a use for it.

The truncation class, on the other hand, has many uses.

The basic idea of the truncation class is to take a string that's far too big for its container, shorten it to fit within the width, and then append three dots to the end to show that it's been shortened. Code Listing 21 shows both classes being used.

Code Listing 21: Wrapping and truncating classes

```

<!-- Page content goes here -->
<div class="container">

```



```

<div class="row">
  <div class="col" style="background-color: lightblue">

    <p class="text-nowrap">...trimmed...</p>
    <p class="text-truncate">...trimmed...</p>

  </div>
</div>
</div>

```

Again, as with the justification example, you'll need to replace the **trimmed** text with something more substantial. Once you do, you should see something similar to Figure 21.

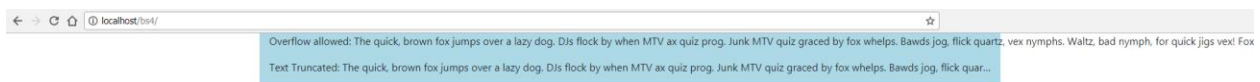


Figure 21: The output from Code Listing 21

As you can see in Figure 21, the first string with no wrapping flows straight outside the row, the container, and eventually off the right edge of the page. The truncated paragraph reaches the edge of the container and is then terminated.

If the container becomes wider or narrower, the truncated text will be altered to fit as much as it can inside the container.

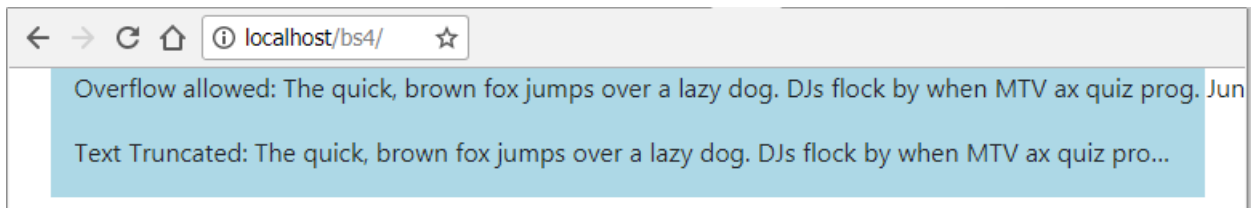


Figure 22: The output from Code Listing 21 in a narrower window

The last of the actual transformation and styling utilities include classes to force lowercase, uppercase, and capitalized text, as well as setting bold, normal, lightweight, and italic text.

The **bold**, **normal**, **light**, and **italic** styles should only be used in cases where semantic layout is **not important**. If you need to make the client or browser scripting aware of the fact text is bold or italic, then you must use the semantic tag approach shown earlier in this chapter. The classes here are for decoration only, and will mean nothing to most screen readers and page scripts.

Code Listing 22 shows the remaining classes in use.

Code Listing 22: The remaining text utility classes

```

<!-- Page content goes here -->
<div class="container">

```

```

<div class="row">
  <div class="col">

    <p class="text-lowercase">All of the text IN THIS PARAGRAPH should be
forced to be in LOWERCASE ONLY, no MaTtEr WHat the Case Of The TEXT is.</p>
    <p class="text-uppercase">All of the text IN THIS PARAGRAPH should be
forced to be in UPPERCASE ONLY, no MaTtEr WHat the Case Of The TEXT is.</p>
    <p class="text-capitalize">All of the text IN THIS PARAGRAPH should
be forced to INITIAL CAPITALS while leaving the case of every OTHER lEtTeR
set to what it was.</p>
    <p class="font-weight-bold">All of the text in this paragraph should
be BOLD.</p>
    <p class="font-weight-normal">All of the text in this paragraph
should be NORMAL.</p>
    <p class="font-weight-light">All of the text in this paragraph should
be LIGHTWEIGHT.</p>
    <p class="font-italic">All of the text in this paragraph should be
ITALIC.</p>

  </div>
</div>
</div>

```

In my version of Chrome, Code Listing 22 renders as shown in Figure 23.

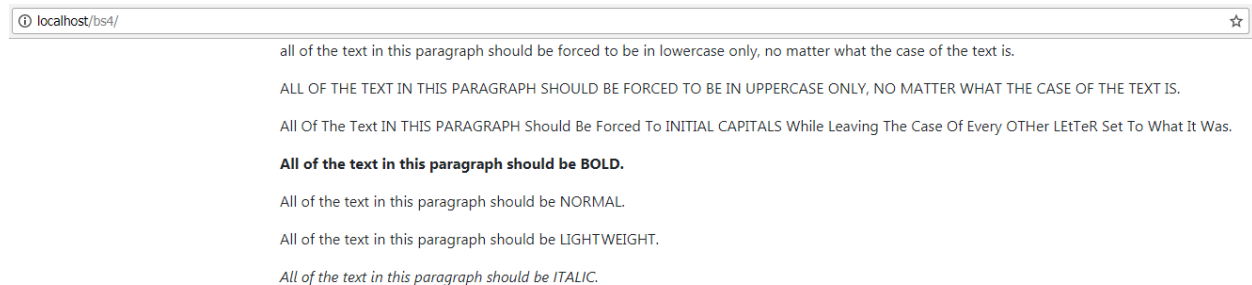


Figure 23: Output produced by Code Listing 22

Text colors

If you've done any work with Bootstrap, even the earlier versions, you might be aware that one of the best features is its color classes.

While the naming has changed significantly in BS4 compared to previous versions, the color classes still allow you to maintain color palette consistency throughout your layouts and designs. This is done by creating classes that are named by function, rather than by a color name. These classes are customizable by using Sass as your CSS build tool, or by creating a custom BS4 build using the tools available on the Bootstrap website.

The custom class will then appear with the exact same color each time it is used, meaning you don't have to remember hex values to use the colors your design needs.

You can also swap color palettes very easily using various techniques, knowing that any changes will instantly change site-wide without any extra work.

You can see the 10 color classes available by adding the code in Code Listing 23 into your template and loading it into your browser.

Code Listing 23: BS4 color classes

```
<!-- Page content goes here -->
<div class="container">
  <div class="row">
    <div class="col">

      <p class="text-primary">.text-primary</p>
      <p class="text-secondary">.text-secondary</p>
      <p class="text-success">.text-success</p>
      <p class="text-danger">.text-danger</p>
      <p class="text-warning">.text-warning</p>
      <p class="text-info">.text-info</p>
      <p class="text-light bg-dark">.text-light</p>
      <p class="text-dark">.text-dark</p>
      <p class="text-muted">.text-muted</p>
      <p class="text-white bg-dark">.text-white</p>

    </div>
  </div>
</div>
```

Once this is opened in your browser, you should see the following output.

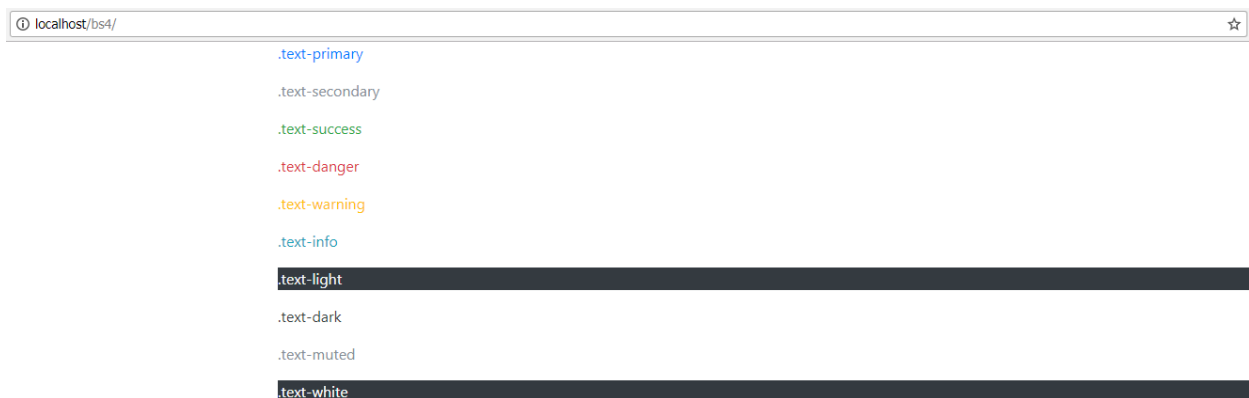


Figure 24: The BS4 default color classes

The example in Code Listing 23 and the accompanying figure is the exact same example that's available in the BS4 documentation.

The light and white colors have been rendered on a dark background, since they wouldn't otherwise be visible on the page.

As you can see, each of the classes has a contextual name, which gives you an idea of its intended use. **danger**, for example, will always be red in the default color scheme.

You can easily remap the default colors to new shades, but in order to do this, you must be working with the Sass sources directly. There are 10 extra color swatches, and nine extra levels of gray from strength 100 to 900 in steps of 100, allowing you to produce a very nice grayscale effect.

In the default download of BS4, however, none of the extra colors and grays are enabled by default. You can find out how to add the extra colors in the [documentation](#).

If you're including the CSS from a CDN or direct download as we are here, then you're restricted only to the contextual colors.

The colors are not just for regular text either; there are a number of contextual classes to set background colors, too. Code Listing 24 shows an example of these.

Code Listing 24: BS4 background color classes

```
<!-- Page content goes here -->
<div class="container">
  <div class="row">
    <div class="col">

      <div class="bg-primary text-white">.bg-primary</div>
      <div class="bg-secondary text-white">.bg-secondary</div>
      <div class="bg-success text-white">.bg-success</div>
      <div class="bg-danger text-white">.bg-danger</div>
      <div class="bg-warning text-dark">.bg-warning</div>
      <div class="bg-info text-white">.bg-info</div>
      <div class="bg-light text-dark">.bg-light</div>
      <div class="bg-dark text-white">.bg-dark</div>
      <div class="bg-white text-dark">.bg-white</div>

    </div>
  </div>
</div>
```

When rendered in Chrome, you should get a series of `<div>` tags, each with a different background color, as shown in Figure 25.



Figure 25: BS4 background colors as produced in Code Listing 24

As was shown in the foreground colors example, you can mix the background classes and the foreground classes to get any combination you need, and as with the text colors, you can change the palette (if you use Sass directly).



Tip: If you're using a recent version of Visual Studio, Sass style sheets can be handled and built on the fly at project build time. If you're using an older version, there are some really good plug-ins in the Visual Studio Marketplace that will allow you to add Sass to your project with the same ease you add regular CSS.

Block quotes and abbreviations

Adding abbreviations and block quotes to a BS4-enabled document is so simple that no explanations are required. Both of them are tag oriented: abbreviations use the `<abbr>` semantic tag, and block quotes use `<blockquote>`. Code Listing 25 shows the body code to put into your template to demonstrate the classes.

Code Listing 25: Abbreviation and block quote example

```
<!-- Page content goes here -->
<div class="container">
  <div class="row">
    <div class="col">

      <p><abbr title="Bootstrap Version 4">BS4</abbr> has lots of features
to help you create stunning page and application layouts. Use industry
standard <abbr title="Hypertext Markup Language Version 5">HTML5</abbr>
markup along with cutting-edge <abbr title="Cascading Style Sheets Version
3">CSS3</abbr> features to make web apps that almost jump out of the
screen.</p>

      <hr/>

      <blockquote class="blockquote">
        <p class="mb-0">It's the same for users too, you just keep doing
what it looks like you're supposed to be doing, no matter how crazy it
seems.</p>
      </blockquote>
```

```

    </div>
  </div>
</div>

```

I've added a `<hr>` tag between the two parts so you can see where the block quote is.

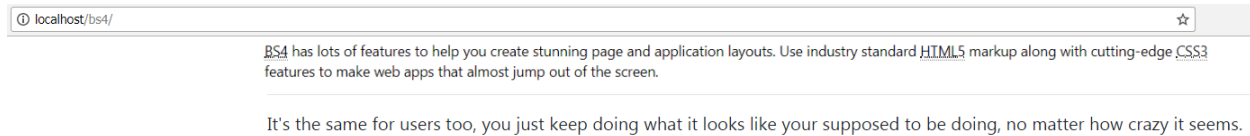


Figure 26: Output from the abbreviation and block quote example

As you can see from Figure 26, the abbreviated words are underlined with a dotted line, and if you hover over them, a tooltip with the full title text will appear.

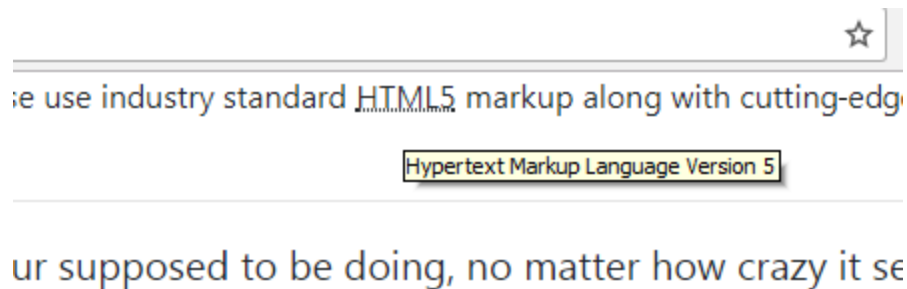


Figure 27: Code Listing 25 with one of the abbreviation tooltips displayed

Unfortunately, on a Windows screenshot, you can't see the pointer, which changes to a question mark and pointer symbol inviting you to hover or click on the item.

Adding a source to a quote is as simple as adding an extra line. Change the code in Code Listing 25 to match Code Listing 26. The output should look the same as Figure 28.

Code Listing 26: Listing 25 changed to add a block quote source

```

<!-- Page content goes here -->
<div class="container">
  <div class="row">
    <div class="col">

      <p><abbr title="Bootstrap Version 4">BS4</abbr> has lots of features
to help you create stunning page and application layouts. Use industry
standard <abbr title="Hypertext Markup Language Version 5">HTML5</abbr>
markup along with cutting-edge <abbr title="Cascading Style Sheets Version
3">CSS3</abbr> features to make web apps that almost jump out of the
screen.</p>

```

```

<hr/>

<blockquote class="blockquote">
  <p class="mb-0">It's the same for users too, you just keep doing
  what it looks like you're supposed to be doing, no matter how crazy it
  seems.</p>
  <footer class="blockquote-footer">Kevin Flynn in <cite
  title="Source Title">Tron</cite> (1982)</footer>
</blockquote>

</div>
</div>
</div>

```

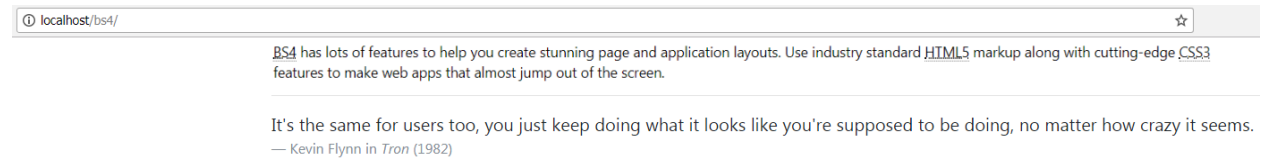


Figure 28: The example from Code Listing 25 with a block quote source added

Displaying computer code

If you've read any kind of technical blog, you know that many of them have specific markup for displaying snippets of computer code, or inline styles to highlight things like key strokes, math formulas, and other similar things.

BS4 has five different classes available to help you mark up this kind of content in your pages. You can easily display inline code and blocks of code, variables, user input instructions, and expected output from a text mode computer application.

There is, however, no syntax highlighting that will color-code text depending on its language or layout—you'll need to add that yourself. There are plenty of JS-based toolkits available that are BS4-friendly and will work with a multitude of different languages.

The easiest markup to use is for an inline code snippet. This simply highlights the given code snippet in line with the regular flow of text, be it a paragraph, span, or `<div>`.

Change your template's body to the following:

Code Listing 27: Inline code example

```

<!-- Page content goes here -->
<div class="container">
  <div class="row">

```

```

<div class="col">

    <p>This is a normal paragraph, but within it I've marked up a
<code>&lt;p&gt;</code> tag, and a <code>&lt;div&gt;</code> tag.</p>

    </div>
</div>
</div>

```

As you can see, it's again a case of using the correct semantic tags. As in previous examples, once BS4 sees these tags it marks up and, in this case, applies a slight color and font change to the text so that it stands out from the surrounding text. Once you render it in the browser, you should see something like the following output.

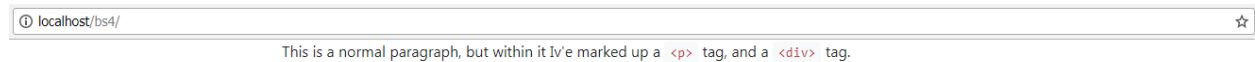


Figure 29: Inline code produced by Code Listing 27

The `<code>` tag can also be used to format multiple code lines, but unlike the inline version, your text color and font won't change. Your text will, however, preserve things like white space, line breaks, and indentation.

To use multiple-line code blocks, wrap your one and only `<code>` tag inside a `<pre>` tag, as Code Listing 28 shows.

Code Listing 28: Using the `<code>` tag inside a multiline block

```

<!-- Page content goes here -->
<div class="container">
  <div class="row">
    <div class="col">

      <pre><code>
        &lt;p&gt;Sample text here...&lt;/p&gt;
        &lt;p&gt;And another line of sample text here...&lt;/p&gt;
      </code></pre>

    </div>
  </div>
</div>

```

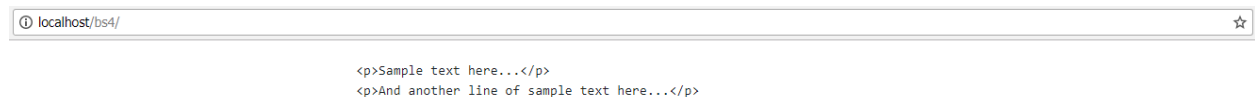


Figure 30: Output from code block in Code Listing 28

Note that the code chosen for display inside the tag has its `<` and `>` symbols in the tags converted to `<` and `>`.

This is a requirement of the HTML spec, not Bootstrap, and the same applies anywhere in an HTML5 page layout. If you need to display angle brackets, then you must do so using the correct ampersand-based formats.

There isn't space in this book to go into a full description of encoding things like angle brackets. If you wish to follow up on this subject and general HTML5 topics, two really good sources are [HTML5 Doctor](#) and [Mozilla Developer Network](#) (MDN).

Moving on, we can also use the `<var>` tag for displaying variables. Its usage is identical to using a code block inline, and it's designed to be used inline, as opposed to being used in a code block.

The `<var>` tag will italicize the text inline, while keeping the surrounding base format. Code Listing 29 shows this in action.

Code Listing 29: Variable markup example

```
<!-- Page content goes here -->
<div class="container">
  <div class="row">
    <div class="col">

      <p>This is a paragraph with some var tags inline: <var>y</var> =
<var>m</var><var>x</var> + <var>b</var></p>
      <h1>This is a header with some var tags inline: <var>y</var> =
<var>m</var><var>x</var> + <var>b</var></h1>

    </div>
  </div>
</div>
```

Rendered in the browser using the template, you should see the following output.

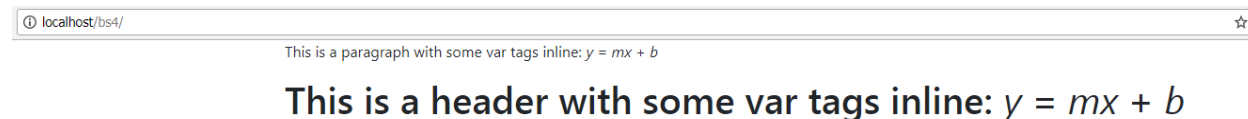


Figure 31: Variable markup output from Code Listing 29

As you can see, no matter what the surrounding markup is, the variable font will be narrower and italicized, but it will retain the size and font of the markup they're used in. This includes other BS4 classes such as `lead` on paragraphs, and the `color` and alignment utility classes.

The last group of code-related content formatting highlights keyboard input and expected output. Like code blocks, this comes in the form of two HTML5 tags, and no classes.

If you want to show a user that they're expected to press some keys or type some input, you can use the `<kbd>` tag. For the expected output, the `<samp>` tag is used.

Code Listing 30 shows how to use these two tags. Change your template code so that you have the following code.

Code Listing 30: Using keyboard and sample output tags

```
<!-- Page content goes here -->
<div class="container">
  <div class="row">
    <div class="col">

      <p>In order to use the terminal you need to enter <kbd>cmd</kbd> in
your run application field.</p>
      <p>After you have entered cmd, you need to click OK or press
<kbd>Enter</kbd> to activate the console.</p>
      <p>If everything works OK, you should be greeted with something
similar to:</p>
      <p class="font-weight-bold">
        <samp>Microsoft Windows [Version 6.1.7601]</samp><br/>
        <samp>Copyright (c) 2009</samp><br /><br/>
        <samp>C:\Users\user></samp><br />
      </p>
      <p>If you type <kbd>dir</kbd> and press <kbd>Enter</kbd> you should
now get a listing of your files.</p>

    </div>
  </div>
</div>
```

I've added an extra **font-weight-bold** class to the paragraph containing the output just to make it easier to see which section is which. Once you load the page into your browser, you should see the following output.

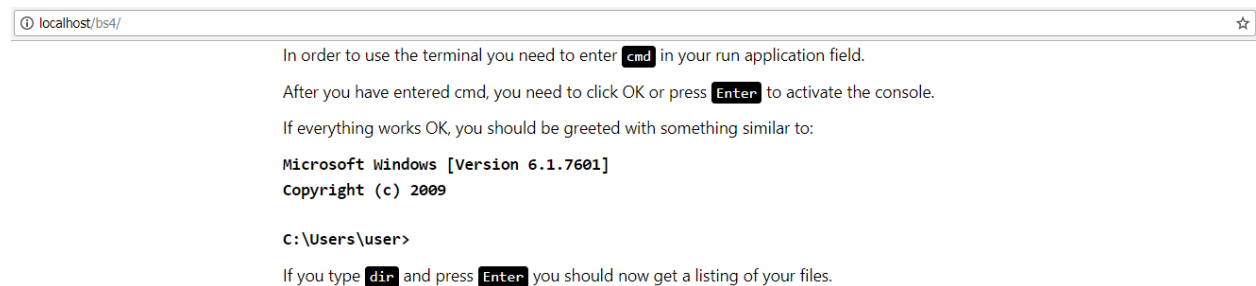


Figure 32: The output produced by Code Listing 30

Chapter 3 An Image is Worth a Thousand Words

No webpage would be complete without the ability to add images and pictures to it. BS4 has an wide array of tools and classes available to help you position your images just right.

Because the `` tag is a block-level object, we can already say that lining `` tags up onto regular grids and positioning them alongside written content is extremely easy given the row, column, and grid structures we've already seen.

Bootstrap goes even further than that, though. It includes premade block-level structures composed of `<div>`s, whose sole purpose is to allow you to create card-like structures, or lists of media where thumbnails and content are perfectly aligned.

As we'll see in this chapter, a Bootstrap-controlled picture is not just worth a thousand words—it's an entire lifetime of artistic opportunities with the ability to communicate just about any message you need to. Gone are the days of folks looking at your designs and being able to say, "Oh, I see you used Bootstrap."



Note: Many of the image examples the are used in the Bootstrap documentation use `holder.js`, which can be added to your projects by following the instructions on the [holder GitHub page](#). For the examples in this book, however, I'm going to keep things simple and use the excellent [online placeholder service](#). Feel free to substitute any images or other services you prefer to use by swapping the URLs that look like <http://via.placeholder.com/350x150> for the ones you prefer to use.

Responsive images and thumbnails

We'll start by looking at the responsive image classes provided by BS4. BS4 is designed to be mobile first, so everything it does, it does with the assumption that you're designing for a mobile experience. With images, this means that every image you place under control of Bootstrap is automatically set up so that it will move, adapt, and scale with the display it's being viewed on.

This also means that in many cases, sizing your images is not dependent on the image tag itself, but is the responsibility of the parent container. This doesn't mean that the image classes are not important—you still have to use them to achieve the best output—but it does mean that you no longer have to have specific image sizes for different parts of your layout. In many cases you can use the same imagery multiple times in multiple places, even at different sizes and scales.

Image quality and size does still need to be considered, however, so it's not a complete get-out-of-jail-free card.

The simplest usage of a basic responsive image class is a standard `` tag with the `img-fluid` class applied to it, as Code Listing 31 shows.

Code Listing 31: Using the responsive image class

```
<!-- Page content goes here -->
<div class="container">
  <div class="row">
    <div class="col">

    </div>
  </div>
</div>
```

Before I show you the rendered output, there's one thing I'd like to point out. Because the image service I use requires me to specify a size for the image placeholder, I cannot create small images and have them size upwards to consume the parent space. This is not an issue if you're generating your own images, but it does make it a bit of a challenge to demonstrate the feature to the reader.

What I've done for the example in Code Listing 31 is made the image wider than the parent container (in this case, just a standard BS row enclosed in a centered container).

BS4 will attempt to reduce the size of the image to fit, which you can see in Figure 33. I've used the excellent [Web Developer tools](#) by Chris Pederick in ruler mode to show the actual width of the image versus the width I asked for.

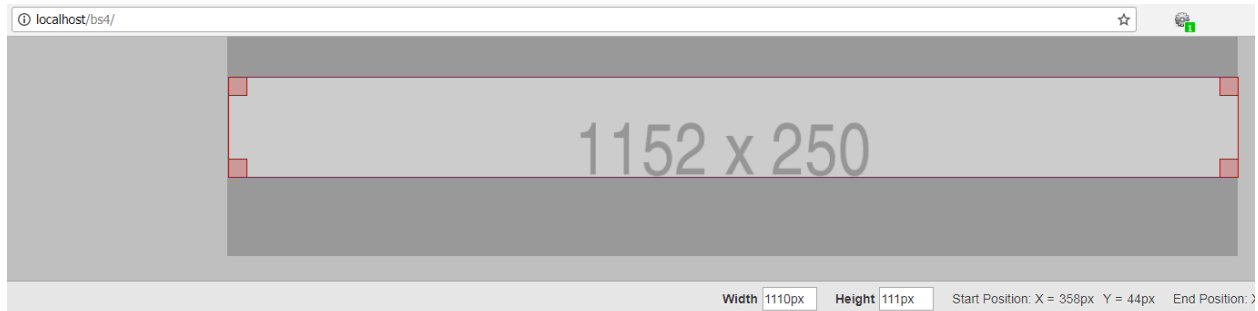


Figure 33: The output from Code Listing 31 with the Web Developer ruler enabled to display the image's actual width

As you can see from the code and the output, I requested an image 1,152 pixels wide, but received a 1,110-pixel-wide image instead.

In many cases, if you don't specify any sizing, then the image will size to the exact space available. If you're just using one measurement, then BS4 will size things so that the ratio is maintained. In the example for Code Listing 31, I could have also shown that the height had been reduced from 250 pixels down to 242 to maintain the ratio of the requested size.

When adjusting and testing sizes, it's definitely better to try different sizes rather than just going straight for what you think a good size will be. Fixing sizes in BS4-controlled layouts can often lead you to grief and mis-sized images that don't look like they fit correctly.

Making smaller image thumbnails for lists and such is also very easy to do; you just use the **img-thumbnail** class in place of the **img-fluid** class.

Code Listing 32: Using the image thumbnail class

```
<!-- Page content goes here -->
<div class="container">
  <div class="row">
    <div class="col">

    </div>
  </div>
</div>
```

The main difference between the **img-thumbnail** and the **img-fluid** classes is that with the thumbnail, a thin border and spacing is applied around the image, giving it an attractive framed look.

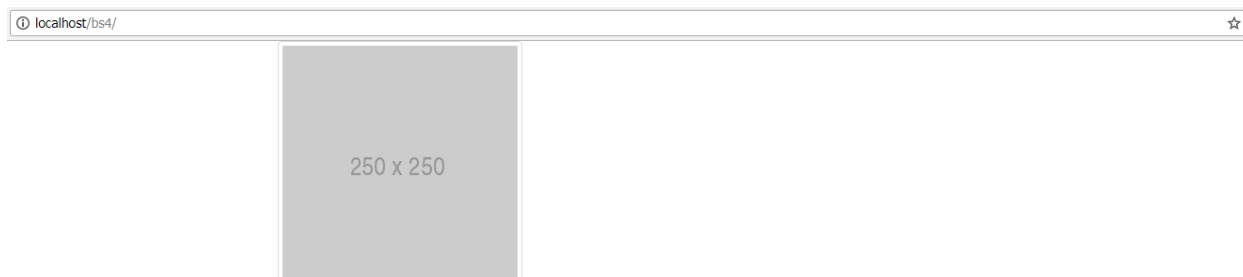


Figure 33: Our image with the thumbnail class applied

The border can also be tweaked using the border utilities, which we'll cover in a later chapter. You can set highlights and colors, and use a number of premade, rounded corner settings to design the frame just the way you want it.

Images can easily be aligned left or right by adding the **float-left** or **float-right** classes to them as follows.

Code Listing 33: Aligning images

```
<!-- Page content goes here -->
<div class="container">
  <div class="row">
    <div class="col">

    </div>
  </div>
</div>
```

You might be wondering why BS4 chooses to use **floats** for this task when we've already established that it uses Flexbox throughout, and there are more than enough features in Flexbox to allow any type of alignment we would wish to use.

The answer is that the alignment classes are designed to align within a parent container. So, if you had a content area that held images and other content too, and you wanted that content to flow around the image, the best tool for the job is **float**. (This is what **float** was originally designed for.) Code Listing 34 shows how you might do this. The image in Figure 34 shows you what to expect from Code Listing 33. When we reach the chapter on the rest of the utility classes that we've not yet covered, we'll see the extra Flexbox classes that could potentially help us align things in this scenario.

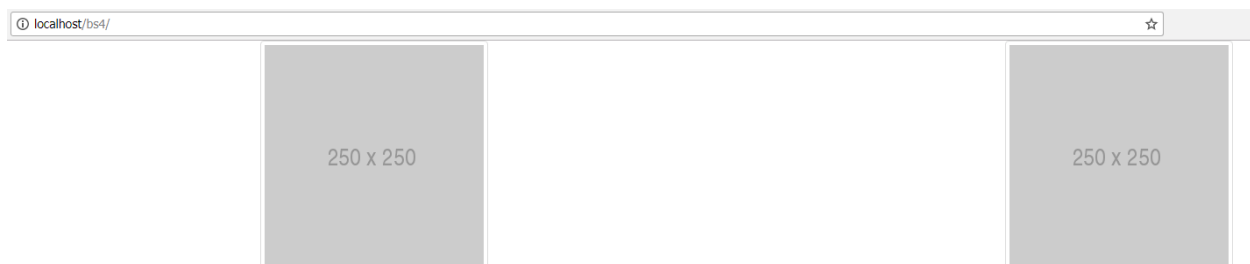


Figure 34: Images aligned using the **float** classes

One last note on the **float** classes: the BS4 toolkit has settings applied to it, so that if you do use the **floats** where they are intended, they actually apply things like clear fixes where needed. This is so that you don't have to worry about the **floats** throwing off any of your alignment and causing problems.

Code Listing 34: An example of how you might use the alignment classes with content in a parent container to allow text to wrap an image

```
<!-- Page content goes here -->
<div class="container">
  <div class="row">

    <div class="col-sm-6">
      
      <p>--- Snipped ---</p>
      <p>--- Snipped ---</p>
    </div>
    <div class="col-sm-6">
```

```

        
        <p>--- Snipped ---</p>
        <p>--- Snipped ---</p>
    </div>

</div>
</div>

```

Code Listing 34, like the others with large chunks of text, has had the text replaced with --- Snipped ---, allowing you to substitute your own long text for the full effect.

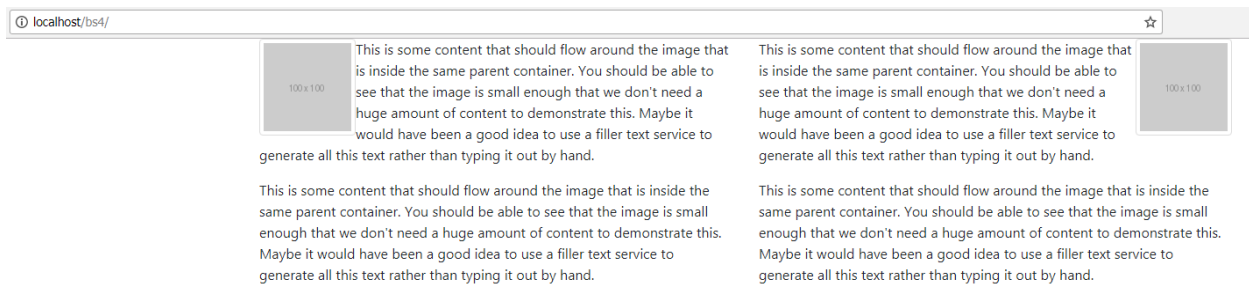


Figure 35: The output from Code Listing 34, showing alignment with text and images

If you want to center an image without the Flexbox utilities, the quickest way is to wrap the entire `` tag inside a `<div>` tag, and then apply the `text-center` alignment class to it, as seen earlier in the text alignment section.

If you're using the `picture` element, then you'll need to apply the BS4 classes to the inner `` tags, and not to the outer `<picture>` tags. If you try to apply the classes to the `<picture>` tags, the alignments won't work as expected. Unfortunately, this means that if you have multiple `` sources inside your `picture` element, then you will have to mark up each one separately.

Media lists

Media lists are great. There, I said it!

A lot of layouts today require you to create some kind of conversation or item and thumbnail-style list of items. BS4 (and BS3) has the perfect tool for this job, called a media list. The concept is simple: you have vertically stacked rows, each with a small, perfectly aligned thumbnail image, a header, and a small text body.

Unfortunately, it does require a bit of nesting to get the right setup, but it's an ideal way to list all sorts of things, from product items to emails in a web email inbox, and beyond.

To create a single media list item, put the following code in your Bootstrap template.

Code Listing 35: A single simple media list object

```
<!-- Page content goes here -->
<div class="container">
  <div class="row">
    <div class="col">

      <div class="media">
        
        <div class="media-body">
          <h4>My Media Item</h4>
          <p>This is a paragraph of text associated with my media item,
telling you what it's all about.</p>
        </div>
      </div>

    </div>
  </div>
</div>
```

Render your template in the browser.

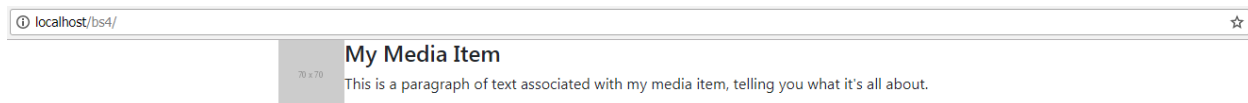


Figure 36: A media list item as produced by Code Listing 35

You'll notice that the text is pushed right up to the thumbnail. You can improve this spacing by using one of the BS4 spacing classes on the inner `` tag. In this case, we'll use `mr-3` (I'll cover this more in the utilities chapter).

Code Listing 36: A single simple media list object with spacing added

```
<!-- Page content goes here -->
<div class="container">
  <div class="row">
    <div class="col">

      <div class="media">
        
        <div class="media-body">
          <h4>My Media Item</h4>
          <p>This is a paragraph of text associated with my media item,
telling you what it's all about.</p>
        </div>
      </div>

    </div>
  </div>
</div>
```



```
</div>
</div>
```

Re-rendering your page now, you'll see the text and image have better spacing.

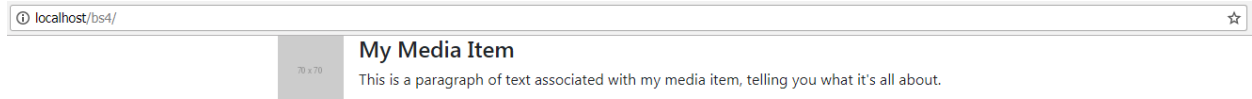


Figure 37: Better spacing applied to the image

You can also place the image after the `<div>` containing the `media-body` class. This will automatically align the image to the right of the container.

Code Listing 37: Media body with the image following it

```
<!-- Page content goes here -->
<div class="container">
  <div class="row">
    <div class="col">

      <div class="media">
        <div class="media-body">
          <h4>My Media Item</h4>
          <p>This is a paragraph of text associated with my media item,
telling you what it's all about.</p>
        </div>
        
      </div>

    </div>
  </div>
</div>
```

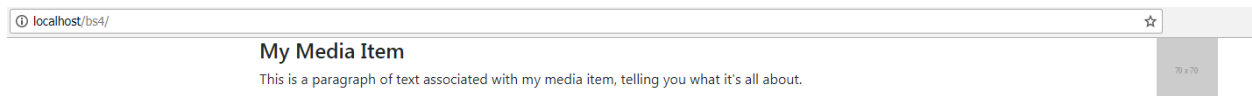


Figure 38: Image is now right-aligned, just by moving the thumbnail position

There are a number of other things you can do too, such as having two images, one on each side of the text, or aligning the bottom of the image to the bottom of the text in the container. All of these extras are done using the Flexbox utilities and other spacing stuff that we'll cover later.

Applying captions

By using the HTML5 `<figure>` and `<figcaption>` tags, you can add perfectly aligned captions to your images, allowing you to annotate things correctly.

Because we are venturing back into semantic meaning again here, as well as adding `alt` attributes to our images, you should also try to use the figure-related tags and classes where possible so that screen readers and other assistive technologies can better describe your images.

Please remember that the idea is to describe your image, not to tell the screen reader what can be seen on the screen. While this may seem easy, it's quite harder than many people realize.

It's easy to say "a picture of a football," but it's a lot harder to say "a typical football, with black and white regular hexagon-shaped patches stitched together with strong binding." In most cases, users of your application who are using a screen reader will thank you for the longer descriptions.

Code Listing 38: Using `<figure>` and `<figcaption>` to add captions to your images

```
<!-- Page content goes here -->
<div class="container">
  <div class="row">
    <div class="col">

      <figure>
        
        <figcaption class="figure-caption">Figure 1: A typical football,
with black and white regular hexagon-shaped patches stitched together with
strong binding.</figcaption>
      </figure>

    </div>
  </div>
</div>
```

The football image I used in this example was linked online, so rather than upset anyone, you'll need to add your own image link in the code. Here's how mine looked when rendered in the browser:



Figure 1: A typical football, with black and white regular hexagon shaped patches stitched together with strong binding.

Figure 39: Our caption example for a football

Chapter 4 The Turning of the Tables

Not much has changed between BS3 and BS4 in terms of what was originally available—everything you were used to using in BS3 still works the same way in BS4. There are, however, some new classes that allow you to create some rather different table appearances.

The most basic example to use is still the good old `<table>` tag along with `<th>`, `<td>`, `<tr>`, `<thead>`, and `<tbody>` where needed, and then apply a `table` class to the main table element.

Code Listing 39: Basic BS4 table

```
<!-- Page content goes here -->
<div class="container">
  <div class="row">
    <div class="col">

      <table class="table">
        <thead>
          <tr>
            <th scope="col">#</th>
            <th scope="col">Name</th>
            <th scope="col">Price</th>
            <th scope="col">Position</th>
          </tr>
        </thead>
        <tbody>
          <tr>
            <th scope="row">1</th>
            <td>Bootstrap 3</td>
            <td>Free</td>
            <td>1</td>
          </tr>
          <tr>
            <th scope="row">2</th>
            <td>Bootstrap 4</td>
            <td>Free</td>
            <td>2</td>
          </tr>
          <tr>
            <th scope="row">3</th>
            <td>Foundation</td>
            <td>Free/Paid</td>
            <td>3</td>
          </tr>
          <tr>
            <th scope="row">3</th>
            <td>UI Kit</td>
```

```

        <td>Free</td>
        <td>4</td>
    </tr>
</tbody>
</table>

</div>
</div>
</div>

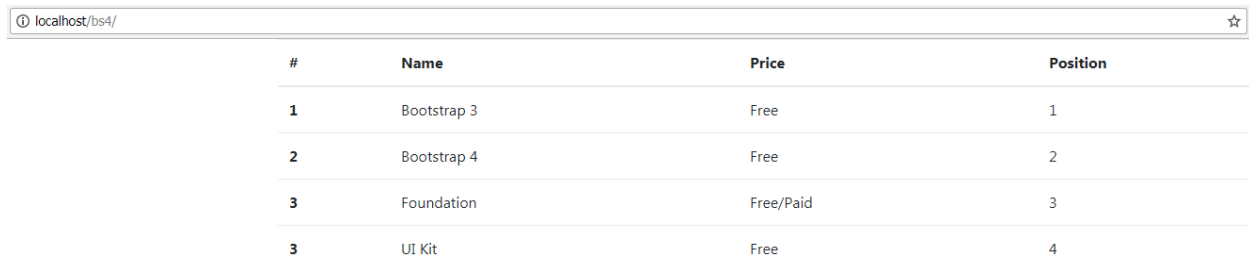
```

The important thing to note about Code Listing 39 is the use of `<thead>` and `<tbody>`.

You **must** structure your tables in this manner; otherwise, Bootstrap won't be able to style them correctly for you. This is a good thing, because the HTML5 specification stipulates that tables should be constructed this way so that screen readers and assistive technologies can better describe the parts of a table.

If you've gotten this far, you're well aware that BS4 is not only designed for mobile first, but also leans heavily on the semantics and structure that is designed to make things easier for assistive technologies. I haven't covered it much in this book, but BS4 also has a lot of support for things like ARIA roles, and if you're keen on marking up your layouts to be as friendly as possible with screen readers, then you should also be adding ARIA attributes to your HTML. I'm not going to use ARIA in this book, but be aware (especially if you're reading the Bootstrap docs) that the BS4 team does push ARIA very heavily, and it is something you should absolutely take into consideration when marking up your code.

If your table worked correctly, then you should see the following in your browser.



#	Name	Price	Position
1	Bootstrap 3	Free	1
2	Bootstrap 4	Free	2
3	Foundation	Free/Paid	3
3	UI Kit	Free	4

Figure 40: A basic BS4 table produced by Code Listing 39

One of the new style options added to tables in BS4 is the ability to give the entire table an inverted or dark style, and still use all the other stuff, such as borders and contextual colors.

To make your table entirely black, it's as simple as adding an extra class name to the `<table>` tag, as Code Listing 40 shows.

Code Listing 40: Creating an inverted table

```

<!-- Page content goes here -->
<div class="container">

```

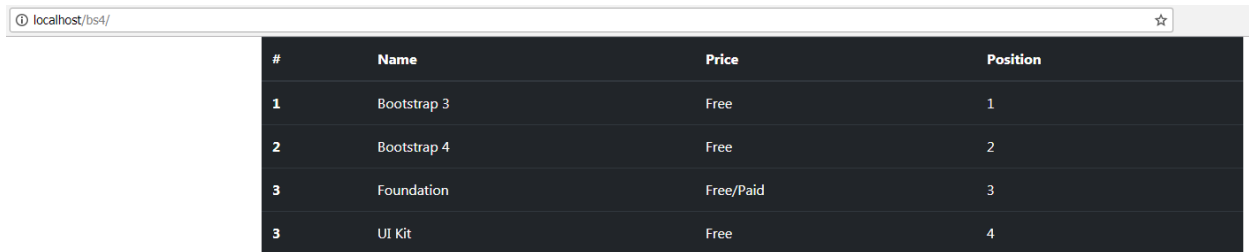
```

<div class="row">
  <div class="col">

    <table class="table table-dark">
      <thead>
        <tr>
          <th scope="col">#</th>
          <th scope="col">Name</th>
          <th scope="col">Price</th>
          <th scope="col">Position</th>
        </tr>
      </thead>
      <tbody>
        <tr>
          <th scope="row">1</th>
          <td>Bootstrap 3</td>
          <td>Free</td>
          <td>1</td>
        </tr>
        <tr>
          <th scope="row">2</th>
          <td>Bootstrap 4</td>
          <td>Free</td>
          <td>2</td>
        </tr>
        <tr>
          <th scope="row">3</th>
          <td>Foundation</td>
          <td>Free/Paid</td>
          <td>3</td>
        </tr>
        <tr>
          <th scope="row">3</th>
          <td>UI Kit</td>
          <td>Free</td>
          <td>4</td>
        </tr>
      </tbody>
    </table>
  </div>
</div>

```

Rendered in the browser, Code Listing 40 should look as follows.



A screenshot of a web browser window with the address bar showing 'localhost/bs4/'. The browser displays a table with a dark background. The table has four columns: '#', 'Name', 'Price', and 'Position'. The data rows are as follows:

#	Name	Price	Position
1	Bootstrap 3	Free	1
2	Bootstrap 4	Free	2
3	Foundation	Free/Paid	3
3	UI Kit	Free	4

Figure 41: Inverted table produced by Code Listing 40

If you just want to invert the table header, then you can add the **dark** class just to the **<thead>** tag, as shown in Code Listing 41.

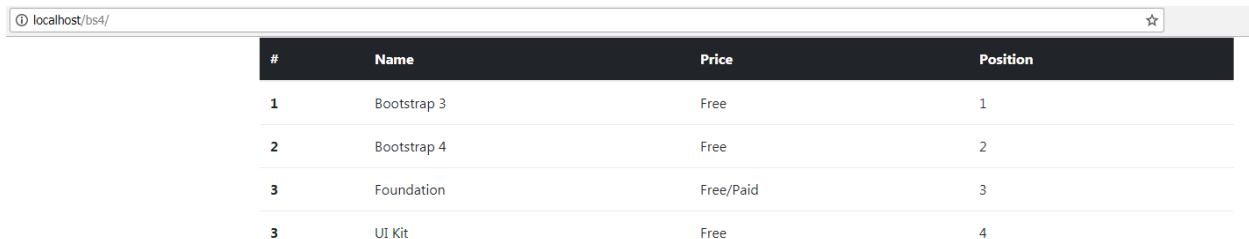
Code Listing 41: Inverting the header only

```
<!-- Page content goes here -->
<div class="container">
  <div class="row">
    <div class="col">

      <table class="table">
        <thead class="thead-dark">
          <tr>
            <th scope="col">#</th>
            <th scope="col">Name</th>
            <th scope="col">Price</th>
            <th scope="col">Position</th>
          </tr>
        </thead>
        <tbody>
        </tbody>
      </table>

    </div>
  </div>
</div>
```

In Code Listing 41 I've removed the main table body, as it's exactly the same as in Code Listing 40. The bit to pay attention to is the addition of the extra class in the **<thead>** tag, which produces the table seen in Figure 42.



A screenshot of a web browser window with the address bar showing 'localhost/bs4/'. The browser displays a table with a dark header and a light body. The table has four columns: '#', 'Name', 'Price', and 'Position'. The data rows are as follows:

#	Name	Price	Position
1	Bootstrap 3	Free	1
2	Bootstrap 4	Free	2
3	Foundation	Free/Paid	3
3	UI Kit	Free	4

Figure 42: Table with an inverted header

You can also use **thead-light** and **table-light** to apply a light gray color, giving you several different combinations of colors to choose from, which you can freely mix and match to your heart's content.

You might have noticed that the header text is slightly bolder than the rest. You might assume this is because the text is in a `<th>` tag, and that is partly the reason. If you look closely, however, you'll notice that the text in the first column is also in bold.

If you look at the code in Code Listings 40 and 41, you'll notice that each `<td>` tag in the first column has a **scope** attribute attached to it.

BS4 notices this and makes the font slightly bolder to show that the row or column is a header in its appropriate direction. You can find more information on this technique in this [W3C Working Group note](#).



Note: The *scope* attribute is not in common use, and is deprecated, so it may not be supported at all in future browsers. Still, if you use it, BS4 will see it and will change the appearance accordingly.

You can make the rows in your table striped by adding the **table-striped** class to the `<table>` tag, and you can add borders by adding **table-bordered**, as shown in Code Listing 42.

Code Listing 42: A striped and bordered table

```
<!-- Page content goes here -->
<div class="container">
  <div class="row">
    <div class="col">

      <table class="table table-bordered table-striped">
        <thead>
          <tr>
            <th scope="col">#</th>
            <th scope="col">Name</th>
            <th scope="col">Price</th>
            <th scope="col">Position</th>
          </tr>
        </thead>
        <tbody>
          <tr>
            <th scope="row">1</th>
            <td>Bootstrap 3</td>
            <td>Free</td>
            <td>1</td>
          </tr>
          <tr>
            <th scope="row">2</th>
            <td>Bootstrap 4</td>
```



```

        <td>Free</td>
        <td>2</td>
    </tr>
    <tr>
        <th scope="row">3</th>
        <td>Foundation</td>
        <td>Free/Paid</td>
        <td>3</td>
    </tr>
    <tr>
        <th scope="row">3</th>
        <td>UI Kit</td>
        <td>Free</td>
        <td>4</td>
    </tr>
</tbody>
</table>

</div>
</div>
</div>

```

Rendered in Chrome, this should give you the following output.

localhost/bs4/ ☆				
#	Name	Price	Position	
1	Bootstrap 3	Free	1	
2	Bootstrap 4	Free	2	
3	Foundation	Free/Paid	3	
3	UI Kit	Free	4	

Figure 43: BS4 table with the striped and bordered classes applied to it

You can also add the various dark and light prefixes as shown in Figure 43, and the styling will adapt to the chosen color scheme.

Pretty tables

Many tables have things like hover colors and row highlighting. BS4 provides all of these too, and it's all easily achievable using simple, intuitive class names.

Add **table-hover** to the table class list.

Code Listing 43: The table-hover class

```
<!-- Page content goes here -->
<div class="container">
  <div class="row">
    <div class="col">

      <table class="table table-hover">
        <thead>
          <tr>
            <th scope="col">#</th>
            <th scope="col">Name</th>
            <th scope="col">Price</th>
            <th scope="col">Position</th>
          </tr>
        </thead>
        <tbody>
          <tr>
            <th scope="row">1</th>
            <td>Bootstrap 3</td>
            <td>Free</td>
            <td>1</td>
          </tr>
        </tbody>
      </table>

    </div>
  </div>
</div>
```

I'm not going to show the output for Code Listing 43, as it's not easy to capture a moving hover bar in a screenshot, but if you put the code in Code Listing 43 into your template file, open the file in your browser, and then move your pointer over the table, you'll see your highlight bar.

You can color individual rows or cells using a set of contextual colors that have the same identifiers as those used for the text coloring utilities.

If you want to apply color to the entire row, apply it to the `<tr>` tag, and if you want to color only the cell, apply it to the `<td>` tag.

Code Listing 44: Table colors

```
<!-- Page content goes here -->
<div class="container">
  <div class="row">
    <div class="col">

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

```

    <tr>
      <th scope="col">#</th>
      <th scope="col">Name</th>
      <th scope="col">Price</th>
      <th scope="col">Position</th>
    </tr>
  </thead>
  <tbody>
    <tr class="table-active">
      <th scope="row">1</th>
      <td>Bootstrap 3</td>
      <td>Free</td>
      <td>1</td>
    </tr>
    <tr class="table-primary">
      <th scope="row">1</th>
      <td>Bootstrap 3</td>
      <td>Free</td>
      <td>1</td>
    </tr>
    <tr class="table-secondary">
      <th scope="row">1</th>
      <td>Bootstrap 3</td>
      <td>Free</td>
      <td>1</td>
    </tr>
    <tr class="table-success">
      <th scope="row">1</th>
      <td>Bootstrap 3</td>
      <td>Free</td>
      <td>1</td>
    </tr>
    <tr class="table-danger">
      <th scope="row">1</th>
      <td>Bootstrap 3</td>
      <td>Free</td>
      <td>1</td>
    </tr>
    <tr class="table-warning">
      <th scope="row">1</th>
      <td>Bootstrap 3</td>
      <td>Free</td>
      <td>1</td>
    </tr>
    <tr class="table-info">
      <th scope="row">1</th>
      <td>Bootstrap 3</td>
      <td>Free</td>
      <td>1</td>
    </tr>
  </tbody>
</table>

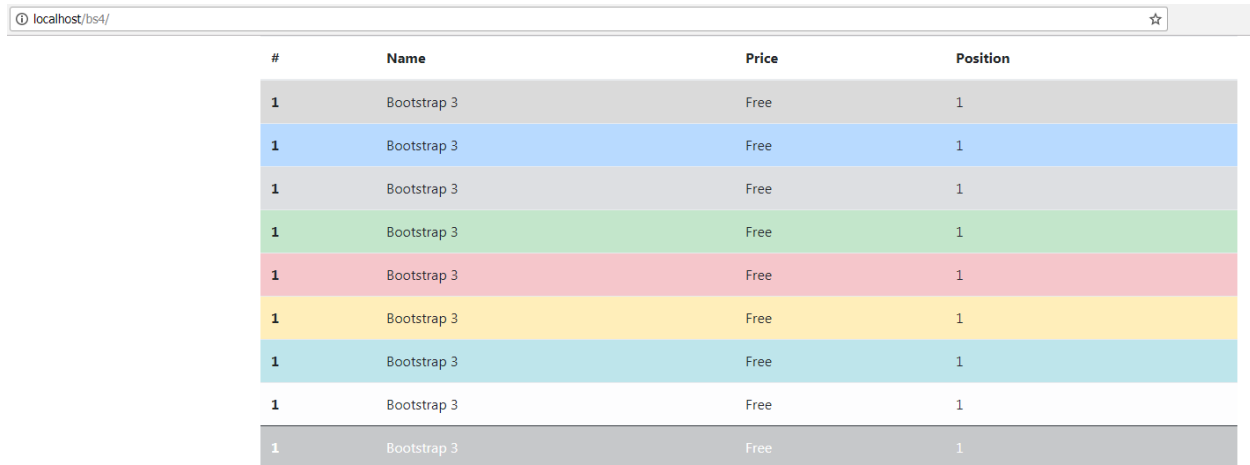
```

```

    </tr>
    <tr class="table-light">
      <th scope="row">1</th>
      <td>Bootstrap 3</td>
      <td>Free</td>
      <td>1</td>
    </tr>
    <tr class="table-dark">
      <th scope="row">1</th>
      <td>Bootstrap 3</td>
      <td>Free</td>
      <td>1</td>
    </tr>
  </tbody>
</table>

</div>
</div>
</div>

```



#	Name	Price	Position
1	Bootstrap 3	Free	1
1	Bootstrap 3	Free	1
1	Bootstrap 3	Free	1
1	Bootstrap 3	Free	1
1	Bootstrap 3	Free	1
1	Bootstrap 3	Free	1
1	Bootstrap 3	Free	1
1	Bootstrap 3	Free	1

Figure 44: Bootstrap 4 table colors

There's a whole lot more you can do with tables, such as making them scroll horizontally when the screen is too narrow, and using text classes to apply brighter colors when using **table-dark** and other modifiers.

You can find the rest of these and more in the [official docs](#).

Components of a Solid Foundation

One thing that BS4 has no shortage of is components. According to the dictionary, a component is:

A part or element of a larger whole, especially a part of a machine or vehicle. Something that is part of a larger assembly.

So a component is essentially a part of something larger—a small cog in a large machine, so to speak. Well in BS4, that couldn't be more true.

If you think of your web application as the machine, and your pages and UI as large parts of the machine that allow users to interact with it, then components are smaller parts of those pages that enable the “UI machine” to function correctly.

So why did I go to the lengths I did to define this?

In the past year or so, there's been a country-sized shift toward the use of “web components.”

This actually started some time ago with Google releasing PolymerJS and a number of other toolkits that added “modular” features to their respective frameworks. At the time, a big fuss was made about actually supporting a component-based modular service directly in the browser.

As is usual in the world of web development, however, this took considerable time and effort before it started to catch on, and before the idea started to make sense to a lot of developers.

Just recently, this idea has not only gained a huge amount of support and traction, but the browsers themselves are now starting to deliver on the promised native implementation in the JavaScript, HTML, and CSS APIs to actually make it a reality.

So where does Bootstrap fit into all of this?

Even since version 2, Bootstrap has provided something that it calls “Bootstrap components.”

The intention then and now is to provide premade “cogs” for you to add into your pages and combine to provide a larger machine with which to drive your site.

A lot of the functionality for this is provided by the Bootstrap.js file and is the main reason why BS relies so heavily on having jQuery included in your project.



Note: If you don't intend to use any JavaScript functionality in Bootstrap, and all you want is the styles, you usually don't have to include the JavaScript files. Likewise, many of the components available are purely styling, and don't need the JavaScript, either. Always check in the [BS docs](#), as you may find that you don't actually need the JS file.

The Bootstrap components list is extensive and covers pretty much anything you might need. Because most of what it does is dependent on how you lay out your HTML5 tags and what BS4 classes you assign to them, a vast amount of it can be used easily inside many of the application frameworks in use today.

Angular, Vue, React, and Aurelia all embrace the philosophy of component-based architectures, and every one of them can work flawlessly with BS4 with little to no effort on the developer's part.

Personally, I favor Aurelia with .NET Core because it allows me to build clean, easy-to-maintain UIs backed onto a very fast, lightweight C# and MVC-based, cross-platform REST interface using the Microsoft MVC frameworks. My [blog](#) has a number of articles covering different subjects related to software development in this environment.

I don't have space to cover every component in the BS4 toolset, as it would take me two books just to cover the basics (and then probably a third one to finish off the stragglers), so the next chapters are going to cover the most-used and newer components introduced in BS4.

Warning: There's going to be a **lot** of HTML code in the following chapters, and I'm not going to be reproducing the full template file for every example as I have done so far. Only the inner body parts necessary to produce the example will be shown.

Chapter 5 Feeding the User with Status

Providing feedback for the user is an important part of any web application. The first set of components we'll be looking at will be alerts, badges, popovers, and tooltips.

Alerts

Alerts are simple. They're a single `<div>` tag with a role attached (for ARIA reasons) and a couple class names added to them. Alerts are a block element by nature and will expand to fill the space available to them, so if you need to control their size, then you either need to customize your CSS and parent element to constrain them, or you need to use the BS4 grid system to ensure they appear where you want them to.

Alerts are also contextual and use the same color naming as other color-related features you've seen so far in this book. The following HTML listing demonstrates all the different alert styles available.

Code Listing 45: BS4 alert styles with inline links

```
<!-- Page content goes here -->
<div class="container">
  <div class="row">
    <div class="col">

      <div class="alert alert-primary" role="alert">
        I'm your PRIMARY alert, I'm the one you should use for general
        notifications.
        <a href="#" class="alert-link"> This is a LINK</a>
      </div>
      <div class="alert alert-secondary" role="alert">
        I'm your SECONDARY alert, I'm the one you should use for general
        notifications that may be slightly less important.
        <a href="#" class="alert-link"> This is a LINK</a>
      </div>
      <div class="alert alert-success" role="alert">
        I'm your SUCCESS alert, I'm the one you should use when something
        went well.
        <a href="#" class="alert-link"> This is a LINK</a>
      </div>
      <div class="alert alert-danger" role="alert">
        I'm your DANGER alert, I'm the one you should use when something
        didn't go well.
        <a href="#" class="alert-link"> This is a LINK</a>
      </div>
      <div class="alert alert-warning" role="alert">
```

I'm your WARNING alert, I'm the one you should use when something might possibly not go well.

```
<a href="#" class="alert-link"> This is a LINK</a>
```

```
</div>
```

```
<div class="alert alert-info" role="alert">
```

I'm your INFO alert, I'm the one you should use when something wants to tell you something, but expects it to be ignored.

```
<a href="#" class="alert-link"> This is a LINK</a>
```

```
</div>
```

```
<div class="alert alert-light" role="alert">
```

I'm your LIGHT alert. I'm a general purpose alert.

```
<a href="#" class="alert-link"> This is a LINK</a>
```

```
</div>
```

```
<div class="alert alert-dark" role="alert">
```

I'm your DARK alert. I'm a general purpose alert.

```
<a href="#" class="alert-link"> This is a LINK</a>
```

```
</div>
```

```
</div>
```

```
</div>
```

```
</div>
```

When you render Code Listing 45 in the browser, you'll notice that I've also added some anchor links in there. The BS4 framework provides an inline style specifically to be added to an `<a>` tag so that you can place links inside an alert box and give them the same look and feel as the surrounding text. You don't have to use these, and if you omit them, you'll see the usual primary color (blue by default) for the link text instead, but using the premade styles means that everything will match and look well-balanced.

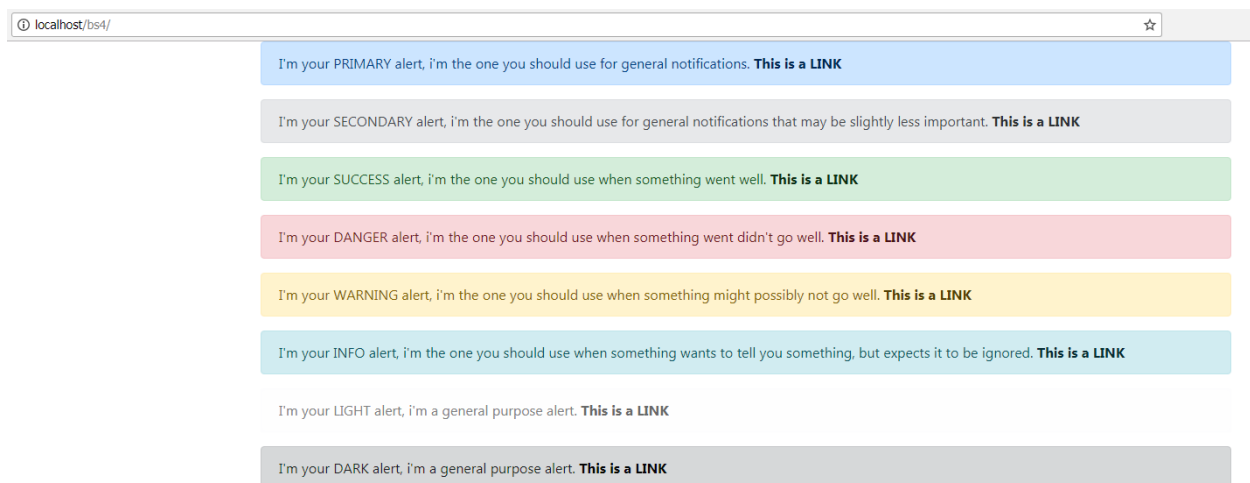


Figure 45: Alerts produced by Code Listing 45

You may also notice in Code Listing 45 that I haven't formatted the inner content in the alerts using HTML. If you don't use HTML, the alert will still be formatted to look OK, and some default padding will be added around it, suitable for one continuous line that will wrap as needed. It doesn't have to be that way though—you can (and maybe should, depending on your needs) use headers, paragraphs, and other typography features to make your alerts stand out a bit more.

Code Listing 46: A rather fancy BS4 alert box

```
<!-- Page content goes here -->
<div class="container">
  <div class="row">
    <div class="col">

      <div class="alert alert-primary" role="alert">
        <h4>That Totally Rocks!</h4>
        <p>The thing that just occurred in your application was totally
awesome. It was made all the more awesome with a little help from Bootstrap,
but it was awesome nevertheless.</p>
        <hr />
        <p>When doing awesome stuff, you can make awesome alerts that
contain regular stuff using all the other Bootstrap features too.</p>
        <div class="row">
          <div class="col">
            <p>This is a <a href="#" class="alert-link">Link</a> to some
further information.</p>
          </div>
          <div class="col">
            <button class="btn btn-primary float-right">Click this button
to do more</button>
          </div>
        </div>
      </div>
    </div>
  </div>
</div>
```

Code Listing 46 shows one way you might add some extra markup to your alert to make it look a bit more fully featured. Don't worry about the few classes in there that you haven't seen yet; you'll see them in detail soon. Figure 46 shows what our snazzy alert looks like.

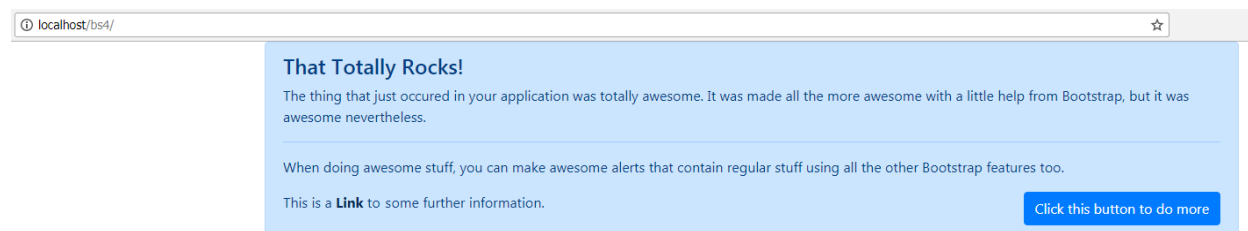


Figure 46: A snazzy alert box

Up to this point, everything you've done using the alert box you can do without needing to use any of the BS4 JavaScript library. The alert library, however, includes a close button and some extra JavaScript functionality to make it look even better.

Change the code from Code Listing 46 so that it looks like the following:

Code Listing 47: A snazzy alert box with a close icon and closing animation

```
<!-- Page content goes here -->
<div class="container">
  <div class="row">
    <div class="col">

      <div class="alert alert-primary alert-dismissible fade show"
role="alert">
        <button type="button" class="close" data-dismiss="alert" aria-
label="Close">
          <span aria-hidden="true">&times;</span>
        </button>
        <h4>That Totally Rocks!</h4>
        <p>The thing that just occurred in your application was totally
awesome. It was made all the more awesome with a little help from Bootstrap,
but it was awesome nevertheless.</p>
        <hr />
        <p>When doing awesome stuff, you can make awesome alerts that
contain regular stuff using all the other Bootstrap features too.</p>
        <div class="row">
          <div class="col">
            <p>This is a <a href="#" class="alert-link">Link</a> to some
further information.</p>
          </div>
          <div class="col">
            <button class="btn btn-primary float-right">Click this button
to do more</button>
          </div>
        </div>
      </div>
    </div>
  </div>
</div>
```

If you render this in your browser, you should see that the alert box now has a small Close icon in its top-right corner. Furthermore, if you click the icon, you'll see that the alert closes with a rather nice and simple fade effect.

The extra functionality was provided by simply adding the **alert-dismissible**, **fade**, and **show** classes to the main alert `<div>` container, and by adding an extra `<button>` tag using the **data-dismiss="alert"** attribute to link the two. **show** makes sure the alert appears when the page is rendered, and **fade** provides the fade animation (both are optional). The **dismissible** class, however, is required.



Note: Data-what? One of the many new things added in HTML5 is the `data-xxxx` attribute. When you start to use Bootstrap components, particularly the JS features, you'll start to use these more and more. Essentially a data attribute is a way for the person creating the HTML markup to add user-defined data to an HTML tag. This data can be used later in some JavaScript code, possibly written by a completely different person, to influence the behavior of the code. For example, a designer could add the name of the primary color used here, and the JavaScript programmer could then use that to program the JavaScript to use that color elsewhere. In this case, the Bootstrap JS library sees the `dismiss` data item, sees that the value is equal to `alert`, and then uses that information to look for a parent element that is set up as an alert. If it finds one, it hooks up the functionality to allow the designer to add a `close` feature to the alert box, without ever having to program any JavaScript whatsoever.

As we move through the different components, you'll start to see more of these HTML blocks being set up to use data attributes; it's the primary method that BS4 uses to enable non-programmers to add programmed features to their designs.

Like other components that use the JavaScript library, the alert library also has some calls that you can use to make the alert visible or to hide it from other parts of your application. I'm not going to cover those in this book, as the explanations and code examples in the official docs do a much better job than I can, and since this book is about using BS4 mainly for styling, I don't want it to get too complicated.

You can find the official docs at the [Bootstrap website](http://getbootstrap.com).

Badges

Badges are designed to provide contextual information to a primary notification object. For example, the tab for the inbox section of a web-based email service might show the number of new or unread messages in the folder. A good real-world example is the Twitter web app, in which you have a small number next to the notifications item in the top navigation bar, showing you how many new notifications you have.

Badges are typically used inline on a `` tag and will size themselves to match the surrounding element sizes. There are a couple of different variations, such as **Dark** and **Light**, and like many of the classes provided by BS4, they have contextual color variants too.

There's no JavaScript functionality available for badges (meaning they will work without including Bootstrap.js), but they will disappear if they are applied to an element that has no content in it. This is achieved by using the **:empty** CSS selector and setting it to **display:none** when applied. **:empty** only works when the inner text is completely empty, so things like spaces and carriage returns will not be considered empty.

Code Listing 48 shows all the various features available for badges.

Code Listing 48: Showing off the BS4 badge component

```
<!-- Page content goes here -->
<div class="container">
  <div class="row">
    <div class="col">
      <p>Standard primary and secondary badges, sized with surrounding
text.</p>
      <hr />
      <h1>Heading Level 1 <span class="badge badge-primary">I'm a primary
badge</span></h1>
      <h2>Heading Level 2 <span class="badge badge-secondary">I'm a
secondary badge</span></h2>
      <h3>Heading Level 3 <span class="badge badge-primary">I'm a primary
badge</span></h3>
      <h4>Heading Level 4 <span class="badge badge-secondary">I'm a
secondary badge</span></h4>
      <h5>Heading Level 5 <span class="badge badge-primary">I'm a primary
badge</span></h5>
      <h6>Heading Level 6 <span class="badge badge-secondary">I'm a
secondary badge</span></h6>

      <hr />
      <p>Contextual colors.</p>
      <hr />

      <div>
        <span class="badge badge-primary">Primary</span>
        <span class="badge badge-secondary">Secondary</span>
        <span class="badge badge-success">Success</span>
        <span class="badge badge-danger">Danger</span>
        <span class="badge badge-warning">Warning</span>
        <span class="badge badge-info">Info</span>
        <span class="badge badge-light">Light</span>
        <span class="badge badge-dark">Dark</span>
      </div>

      <hr />
      <p>Pill-shaped badges with contextual colors.</p>
      <hr />

      <div>
        <span class="badge badge-pill badge-primary">Primary</span>
        <span class="badge badge-pill badge-secondary">Secondary</span>
        <span class="badge badge-pill badge-success">Success</span>
        <span class="badge badge-pill badge-danger">Danger</span>
        <span class="badge badge-pill badge-warning">Warning</span>
      </div>
    </div>
  </div>
</div>
```

```

    <span class="badge badge-pill badge-info">Info</span>
    <span class="badge badge-pill badge-light">Light</span>
    <span class="badge badge-pill badge-dark">Dark</span>
  </div>

</div>
</div>
</div>

```

I was going to add some JavaScript into Code Listing 48 to toggle the content so the reader could see the effect of no content, but it's much easier if I just encourage you to experiment. Once you put this code into your template file, try removing and emptying the text inside some of the tags and then refreshing the page. You'll see that not only does the badge vanish, but also the space it would have used collapses too, closing the gap.

If you render Code Listing 48 in your browser, you should see something like the following figure.

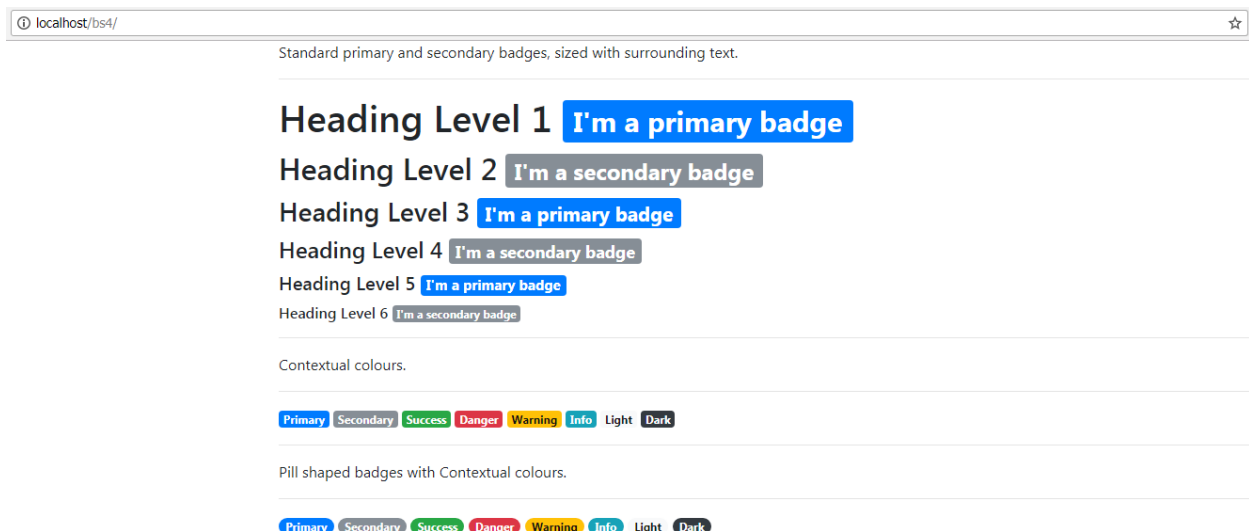


Figure 47: A demonstration of the badge component as produced by Code Listing 48

Tooltips and popovers

Tooltips are useful little bits of information, usually only one or two words that appear when you hover over a control or click a help button of some kind. To use them in Bootstrap, not only do you need the **Bootstrap.js** file, but you also need to ensure that you have **Popper.js** loaded.

Popovers are usually a little bit fuller than a tooltip, and have a title area and usually a paragraph of text. Both, however, are very similar in their operation.



Note: *Popper.js is a third-party JavaScript library used to manage popups in layers in a web application. BS4 uses this rather than writing its own implementation, as it has a proven track record and is known to be cross-platform. You can find out more at <https://popper.js.org/>.*

If you recall from the introduction, the initial BS4 template included three lines of JavaScript at the bottom of the file.

Code Listing 49: The bottom three lines in our initial template

```
<!-- Optional JavaScript -->
<!-- jQuery first, then Popper.js, then Bootstrap JS -->

<script src="https://code.jquery.com/jquery-3.2.1.slim.min.js"
integrity="sha384-
q8i/X+965Dz00rT7abK41JStQIAqVgRVzpbzo5smXKp4YfRvH+8abtTE1Pi6jizo"
crossorigin="anonymous"></script>
<script
src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.14.0/umd/popper.min.js"
integrity="sha384-
cs/chFZiN24E4KMATLdqvsezGxaGsi4hLG0zlxWp5UzB1LY//20VyM2taTB4QvJ"
crossorigin="anonymous"></script>
<script
src="https://stackpath.bootstrapcdn.com/bootstrap/4.1.0/js/bootstrap.min.js"
integrity="sha384-
uefMccjFJAIV6A+rW+L4AHf99KvxDjWSu1z9VI8SKNVmz4sk7buKt/6v9KI65qnm"
crossorigin="anonymous"></script>
```

As you can see in Code Listing 49, we include jQuery, then we include Popper.js, and finally we include the Bootstrap.js library.

If you don't intend to use tooltips and popovers, then you don't need to include Popper.js.

Another thing you need to be aware of is that tooltips and popovers must be initialized by hand once your page has finished loading. According to the BS4 documentation, this is for performance reasons' I've never tested this, however, and the few time' I've used them, performance has always been more than acceptable.

What all this means is that in your page-ready functionality, either by using jQuery's **onReady** handler, or by using the events and functions provided by whichever framework you're using, you will need to include the following small snippet of JavaScript to initialize any tooltips before you can use them.

Code Listing 50: JavaScript required to initialize tooltips and popovers

```
$(function () {  
  $('[data-toggle="tooltip"]').tooltip()  
  $('[data-toggle="popover"]').popover()  
})
```

Code Listing 50 shows the jQuery way of doing things with an anonymous function. If you're using a framework such as Angular or Aurelia, then the only lines you need are the middle ones that end with **tooltip** or **popover**. Essentially, all you need to do is call the appropriate function on each HTML5 element that has a **tooltip** or **popover** attribute. Please also note that if you are using a framework, you don't have to do the initialization using the jQuery approach shown in Code Listing 50. As long as you're able to get a list of elements that include the **data-toggle** attribute, and then call the either the **tooltip** or **popover** function on them, that, too, will work.

Because of the sheer number of variations there are on how to do this, I'm not going to attempt to give you a sample—I'll leave it up to the Bootstrap 4 documentation to show you how to apply tooltips to your elements.

You can find the relevant BS4 documentation by following these links:

- [Tooltips documentation](#)
- [Popovers documentation](#)

Here's a screenshot of the tooltips and popovers, as seen on the BS4 documentation site.

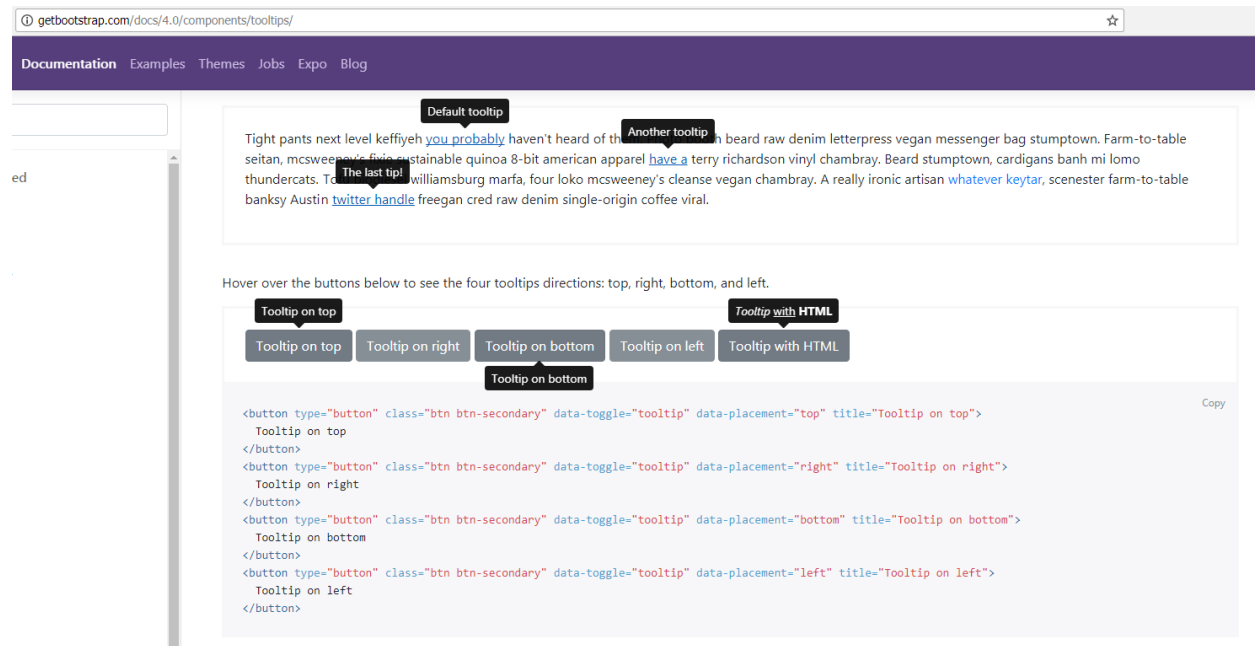


Figure 48: Tooltips demo on the Bootstrap documentation site

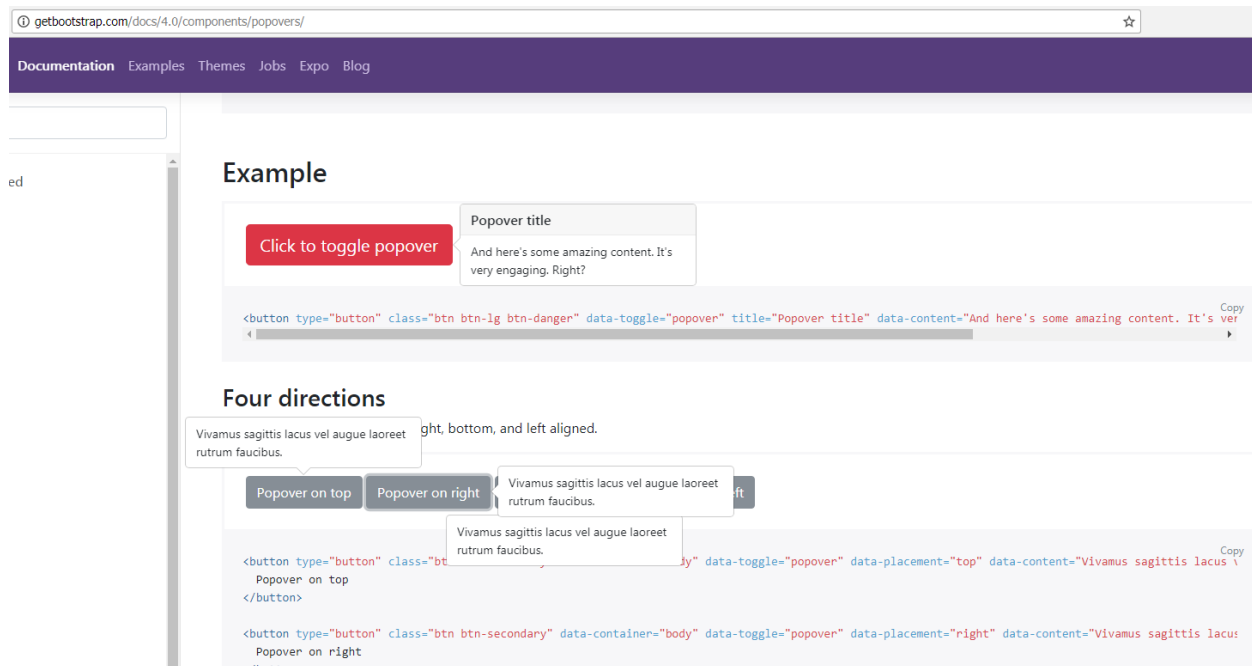


Figure 49: Popovers demo on the Bootstrap documentation site

The main difference between popovers and tooltips as far as their interactivity is concerned is that tooltips will be removed from the element automatically, usually when the mouse pointer is moved away, or a touch-up event is fired. Popovers will remain in place until explicitly told to move.

Whichever you choose, displaying them on your elements is pretty much the same in both cases.

Set **data-toggle** equal to **tooltip** for tooltips, and set it to **popover** for popovers.

To set the content, set the **title** attribute for tooltips, and set **data-content** for a popover.

Both tooltips and popovers use **data-placement** equal to **top**, **bottom**, **left**, or **right** to determine the position they appear in. Remember that you have to initialize the JavaScript by hand in a manner appropriate for the framework or page design you are using.

There are a few other lesser-used options too; if you're interested in them, everything you need can be found in the documentation.

Chapter 6 A Button-Shaped Form of Madness

A web application wouldn't be much good without some form of button to click on. Buttons allow you to perform actions, select choices, or in some cases, do unspeakable things—every villain throughout history had a big red one.

Whatever your reason for having buttons, BS4 has more than enough variety to keep even the most manic clickers happy, and as with all the other components available, it provides contextual colors and dark and light variations too.

Button basics

The simplest thing to do to create a button is to apply the `btn` class to a block-level element.

You can apply the class to anchor tags and inline elements too, but for best results, buttons should always be used with the HTML5 semantic `<button>` tag.



Note: Buttons or anchors? Many folks using HTML5 use standard `<a>` (anchor) elements to provide buttons and activate functionality in a web application. While this works, it can cause you many problems' I've seen links to delete functions placed as "get" links, and then watched in horror as Google sends its spider over a website and quickly empties the site's database. Anchors have a distinct use case over buttons, and should only ever be used for a button that either navigates to a new page or provides functionality that makes no changes. The W3C and other standards bodies all recommend that for anything other than navigation, `<button>` tags should be used where possible. Which you use is up to you; BS4 will style both exactly the same.

The simplest button you can produce is as follows.

Code Listing 51: The most basic button you can produce

```
<!-- Page content goes here -->
<div class="container">
  <div class="row">
    <div class="col">

      <button type="button" class="btn">My Button</button>

    </div>
  </div>
</div>
```

When rendered, you should see a simple gray button with whatever caption you provided.



Figure 50: The simplest button possible

Adding the `btn` class is not all, though. To apply contextual colors, you add a modifier class, as Code Listing 52 demonstrates.

Code Listing 52: Simple buttons with modifier classes added

```
<!-- Page content goes here -->
<div class="container">
  <div class="row">
    <div class="col">

      <button type="button" class="btn btn-primary">Primary</button>
      <button type="button" class="btn btn-secondary">Secondary</button>
      <button type="button" class="btn btn-success">Success</button>
      <button type="button" class="btn btn-danger">Danger</button>
      <button type="button" class="btn btn-warning">Warning</button>
      <button type="button" class="btn btn-info">Info</button>
      <button type="button" class="btn btn-light">Light</button>
      <button type="button" class="btn btn-dark">Dark</button>
      <button type="button" class="btn btn-link">Link</button>

    </div>
  </div>
</div>
```

Code Listing 52 shows the different contextual colors you can add to your buttons. In Chrome, you should see one of each type, as follows.



Figure 51: Simple buttons with contextual colors

In Figure 51, you can see that the last button is marked as a **Link** button. The idea here is to allow you to use regular semantic button tags, but to make them look like standard anchor links. The intended use is to place them in table rows as controls to activate record actions, or as help links in dialog boxes and alerts. Remember though: colors are just colors. Because this is set using an extra class, screen readers will NOT pick up on the fact that a button is dangerous or may lead to an unwanted action. With buttons, you **must** add ARIA roles (`role="button"`) and things like alternative text, so that a screen reader can help its user. You should also add a proper tab order to buttons and allow them to be focusable; this will allow the use of the Tab key or arrow keys on a keyboard to move around an application correctly.

In fact, the one big plus about using the `<button>` tag for this kind of interactivity is that you don't have to code it yourself. There's nothing stopping you from using a `btn` class on a `<div>` tag. After all, both elements are block-level elements, and they will look exactly the same when rendered in the document, but fundamentally, unless you add a lot of extra JavaScript code and make sure that you're religious about setting tab order correctly, you simply won't be able to focus and navigate the controls the way you will when using buttons.

If you don't want filled buttons, then you can use the `btn-outline-xxxx` modifier classes instead of the standard contextual colors you used in Code Listing 52. The `xxxx` should be replaced with the contextual name you want to use, as Code Listing 53 shows.

Code Listing 53: Outline contextual button styles

```
<!-- Page content goes here -->
<div class="container">
  <div class="row">
    <div class="col">

      <button type="button" class="btn btn-outline-
primary">Primary</button>
      <button type="button" class="btn btn-outline-
secondary">Secondary</button>
      <button type="button" class="btn btn-outline-
success">Success</button>
      <button type="button" class="btn btn-outline-danger">Danger</button>
      <button type="button" class="btn btn-outline-
warning">Warning</button>
      <button type="button" class="btn btn-outline-info">Info</button>
      <button type="button" class="btn btn-outline-light">Light</button>
      <button type="button" class="btn btn-outline-dark">Dark</button>
      <button type="button" class="btn btn-outline-link">Link</button>

    </div>
  </div>
</div>
```

In the browser, your buttons should change to look like Figure 52.



Figure 52: Simple buttons changed to use the outline contextual colors

There are three modifier classes to adjust the button size: **btn-lg**, **btn-sm**, and **btn-block**. **lg** and **sm** will alter the size but will maintain the button width so that it is sized to the text. Using **block**, on the other hand, will make the button expand to fill the width of its parent container. If you're controlling the layout of your buttons using the flexible grids shown at the beginning of this book, then **btn-block** is a really good modifier to use as it will ensure that your buttons adapt to whatever responsive width your various containers adapt to when responding to screen size changes.

Code Listing 54: Different button sizes

```
<!-- Page content goes here -->
<div class="container">
  <div class="row">
    <div class="col">

      <button type="button" class="btn btn-primary btn-lg">Primary</button>
      <button type="button" class="btn btn-secondary btn-
lg">Secondary</button>
      <button type="button" class="btn btn-success btn-lg">Success</button>
      <button type="button" class="btn btn-danger btn-lg">Danger</button>
      <button type="button" class="btn btn-warning btn-lg">Warning</button>
      <button type="button" class="btn btn-info btn-lg">Info</button>
      <button type="button" class="btn btn-light btn-lg">Light</button>
      <button type="button" class="btn btn-dark btn-lg">Dark</button>
      <button type="button" class="btn btn-link btn-lg">Link</button>
      <br/><br/>
      <button type="button" class="btn btn-primary btn-
sml">Primary</button>
      <button type="button" class="btn btn-secondary btn-
sml">Secondary</button>
      <button type="button" class="btn btn-success btn-
sml">Success</button>
      <button type="button" class="btn btn-danger btn-sml">Danger</button>
      <button type="button" class="btn btn-warning btn-
sml">Warning</button>
      <button type="button" class="btn btn-info btn-sml">Info</button>
      <button type="button" class="btn btn-light btn-sml">Light</button>
      <button type="button" class="btn btn-dark btn-sml">Dark</button>
      <button type="button" class="btn btn-link btn-sml">Link</button>
      <br/><br/>
    <div class="row">

      <div class="col"><button class="btn btn-outline-primary btn-
block">I'm a Block Button</button></div>
      <div class="col"><button class="btn btn-success btn-block">I'm a
Block Button</button></div>

    </div>
  </div>
</div>
```

```

    </div>
  </div>
</div>

```

If you put the code from Code Listing 54 in as the body code in your template and load it into the browser, it should produce the following output.

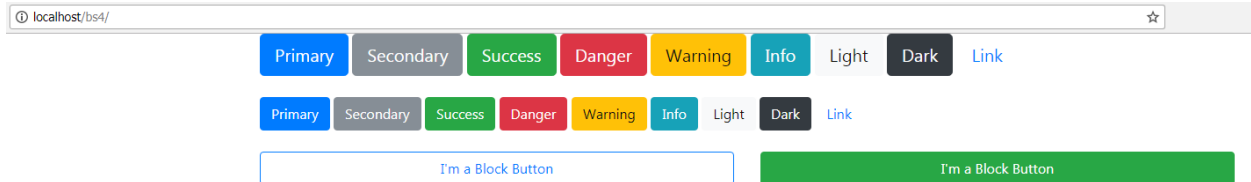


Figure 53: Different button sizes produced by Code Listing 54

While you're playing around with buttons, you should notice they have different visual styles depending on what state the button is in. When you tap one, for example, it becomes a little darker and has an inner shadow.

Usually this is all handled by BS4 internally, but there may be times when you need to make your buttons appear to be tapped from application code. You can do this by adding the **active** class to the class list for your button. Don't forget that if you do add the class, it's good practice to add the ARIA role **aria-pressed="true"** attribute so screen readers know it's marked as being pressed.

You can also make a button appear to be disabled by adding the **disabled** attribute. When you use this on a standard button tag, not only does it give you the BS4 disabled button style, but it also physically disables the button too, which means that no events fire on it.

Unfortunately, the **disabled** attribute doesn't work on **<a>** tags, so if you're using an anchor with button styles to activate things, then adding the **disabled** attribute won't have any effect. Instead, what you need to do is add the **disabled** class to the **<a>** tag, and then you need to take steps in your program code to prevent the click on the anchor from being acted upon. How you do this depends entirely on the framework and code you're using in your application, but often just involves returning **false** from your event handler.

Code Listing 55: Showing the button disabled styles

```

<!-- Page content goes here -->
<div class="container">
  <div class="row">
    <div class="col">

      <button type="button" class="btn btn-primary" onclick="alert('You
clicked button 1')">Button 1 (Enabled) Click Me</button>
      <button type="button" class="btn btn-primary" onclick="alert('You
clicked button 1')" disabled>Button 1 (Disabled) Click Me</button>
      <button type="button" class="btn btn-secondary" onclick="alert('You

```

```

clicked button 2')">Button 2 (Enabled) Click Me</button>
    <button type="button" class="btn btn-secondary" onclick="alert('You
clicked button 2')" disabled>Button 2 (Disabled) Click Me</button>

    <br /><br />

    <a href="igosomewhere" class="btn btn-success">I'm an Anchor Tag
Without a disabled class</a>
    <a href="igosomewhere" class="btn btn-success disabled"
onclick="return false;">I'm an Anchor Tag With a disabled class</a>

    </div>
</div>
</div>

```

Code Listing 55 shows an example of marking a button as disabled. I've added a simple **onclick** to the **<a>** tag to make it return **false**, so that its click is effectively disabled. Clicking on the first **<a>** tag should give you a 404 error (unless you actually have a file on your system called *igosomewhere*). If you change the **href** attribute in the two links to another page, you'll see that they behave the same as the buttons with respect to being disabled. The method I show works in plain JavaScript, but please read the notes earlier in this chapter regarding using buttons or **<a>** tags before you decide how you wish to use these in a framework.

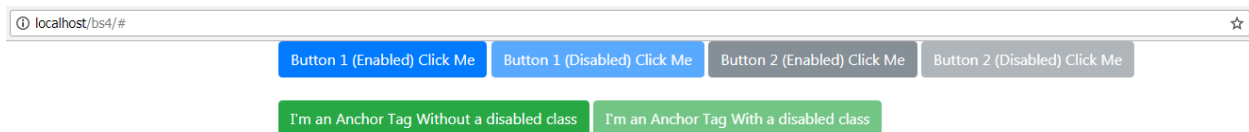


Figure 54: Disabled button styles

Button groups

In many site or page designs, you may want a collection of buttons, for example, in a toolbar or on a menu of some description. BS4 has appropriate styles and classes that not only allow you to maintain consistency when doing this, but also allow you to set up a control that behaves in much the same way that a set of radio buttons does (where only one can be selected).

Button groups are great for many aspects of an application UI because you can make components from them that are then reusable. With a framework such as Aurelia or Angular, you can put together entire button palettes, and then reuse them as a single control.

You can also group buttons with input controls, for example, by adding a search button to a search field. We'll see more about this in the chapter on forms.

Basic groups are created by wrapping a collection of **<button>** tags inside a **<div>** tag, and then applying the class **btn-group** to the **<div>**. As always, you should also put the correct ARIA attributes on the outer tag so that assistive technologies handle it correctly.

Code Listing 56: A basic button group

```
<!-- Page content goes here -->
<div class="container">
  <div class="row">
    <div class="col">

      <div class="btn-group" role="group" aria-label="A group of buttons">
        <button type="button" class="btn btn-primary">Button 1</button>
        <button type="button" class="btn btn-primary">Button 2</button>
        <button type="button" class="btn btn-primary">Button 3</button>
        <button type="button" class="btn btn-primary">Button 4</button>
        <button type="button" class="btn btn-primary">Button 5</button>
        <button type="button" class="btn btn-primary">Button 6</button>
      </div>

    </div>
  </div>
</div>
```

Code Listing 56 shows a basic horizontal group containing six buttons. Each button is independent of the other, but formatted to look like one unit, as can be seen in Figure 55.

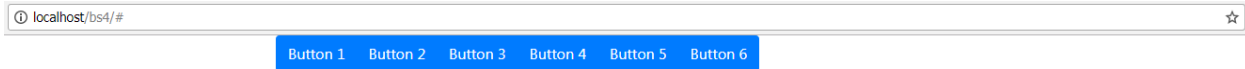


Figure 55: A basic BS4 button group

Buttons can also be grouped into a toolbar by wrapping multiple button group `<div>` tags inside an outer `<div>` tag that has a `btn-toolbar` class applied to it. This allows you to make chains of button groups while still keeping them as one functional unit. If you use some of the spacing utilities, you can also keep good, balanced spacing between the groups.

Code Listing 57 demonstrates how to do this.

Code Listing 57: A demonstration of a BS4 button toolbar

```
<!-- Page content goes here -->
<div class="container">
  <div class="row">
    <div class="col">

      <div class="btn-toolbar" role="toolbar" aria-label="An example of a
button toolbar">
        <div class="btn-group mr-2" role="group" aria-label="First Button
Group">
          <button type="button" class="btn btn-primary">Button 1</button>
          <button type="button" class="btn btn-primary">Button 2</button>
          <button type="button" class="btn btn-primary">Button 3</button>
```

```

    </div>
    <div class="btn-group mr-2" role="group" aria-label="Second Button
Group">
        <button type="button" class="btn btn-secondary">Button 1</button>
        <button type="button" class="btn btn-secondary">Button 2</button>
        <button type="button" class="btn btn-secondary">Button 3</button>
    </div>
    <div class="btn-group" role="group" aria-label="Third Button
Group">
        <button type="button" class="btn btn-success">Button 1</button>
        <button type="button" class="btn btn-warning">Button 2</button>
        <button type="button" class="btn btn-danger">Button 3</button>
    </div>
</div>
</div>
</div>

```

Again, ARIA roles are very important because a screen reader will simply announce these all as single, independent buttons, potentially confusing a screen reader, or at worst, turning assisted users away because there's "too much stuff going on."

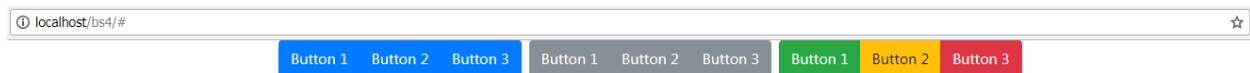


Figure 56: Button toolbar produced by Code Listing 57

Buttons can also have drop-down menus attached to them (you'll see more of this in an upcoming chapter); however, if you want to add buttons with drop-down menus into a button group, you must nest two button groups containing them.

Code Listing 58: Nested button groups to include buttons with drop-down menus

```

<!-- Page content goes here -->
<div class="container">
    <div class="row">
        <div class="col">

            <div class="btn-group" role="group" aria-label="Button group with
nested drop-down">
                <button type="button" class="btn btn-primary">Button 1</button>
                <button type="button" class="btn btn-secondary">Button 2</button>

                <div class="btn-group" role="group">
                    <button id="demoDropdown1" type="button" class="btn btn-success
dropdown-toggle" data-toggle="dropdown" aria-haspopup="true" aria-
expanded="false">

```



```

        Dropdown
    </button>
    <div class="dropdown-menu" aria-labelledby="demoDropdown1">
        <a class="dropdown-item" href="#">Menu Item 1</a>
        <a class="dropdown-item" href="#">Menu Item 2</a>
    </div>
</div>

<button type="button" class="btn btn-warning">Button 3</button>
<button type="button" class="btn btn-danger">Button 4</button>

</div>

</div>
</div>
</div>

```

You need to wrap every drop-down menu you wish to use inside its own nested **<button>** group, while all your regular button group members stay in the outermost **<button>** group tag.

The nesting can get a bit deep and tangled, so I would advise you to use drop-down menus with caution. If you're thinking about using them for a navigation system, BS4 has a better component for that purpose.

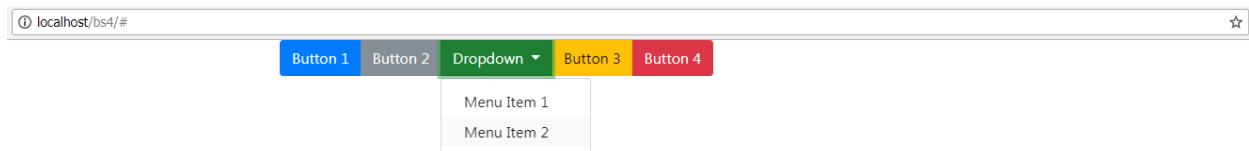


Figure 57: A button group with a nested drop-down as produced by Code Listing 58

All the usual sizing and color subclasses work the same way inside a button group, just as they do with regular buttons, so you can show disabled buttons or color buttons and set them to different sizes.

You can also change your **btn-group** class to **btn-group-vertical** so that your buttons are stacked from top to bottom, rather than running inline from left to right.

If you also add **btn-group-toggle** to your outer **<div>**, then mark up your buttons using check boxes, your button groups will work in the same visual manner that a line of check boxes does. Marking up the inner controls using radio controls gives you a button group where only one option can be selected at a time.

Code Listing 59: Check box, radio, and vertical button groups example

```

<!-- Page content goes here -->
<div class="container">
    <div class="row">

```

```

<div class="col">

  <br />
  <div class="btn-group btn-group-toggle" data-toggle="buttons">
    <label class="btn btn-primary active">
      <input type="checkbox" checked autocomplete="off">Check Box
Button
    </label>
    <label class="btn btn-primary">
      <input type="checkbox" autocomplete="off">Check Box Button
    </label>
    <label class="btn btn-primary">
      <input type="checkbox" autocomplete="off">Check Box Button
    </label>
  </div>

  <br />
  <div class="btn-group btn-group-toggle" data-toggle="buttons">
    <label class="btn btn-primary">
      <input type="radio" autocomplete="off">Radio Button
    </label>
    <label class="btn btn-primary">
      <input type="radio" autocomplete="off">Radio Button
    </label>
    <label class="btn btn-primary active">
      <input type="radio" checked autocomplete="off">Radio Button
    </label>
  </div>

  <br/>
  <div class="btn-group-vertical" role="group" aria-label="Vertical
Buttons">
    <button type="button" class="btn btn-secondary">Vertical Top
Button</button>
    <button type="button" class="btn btn-secondary">Vertical Middle
Button</button>
    <button type="button" class="btn btn-secondary">Vertical Bottom
Button</button>
  </div>

</div>
</div>
</div>

```

Code Listing 59 shows how to put together a check box, radio, and vertical button group. Note that we can mark any that are preselected in the group by adding the **active** class. In the case of the check boxes and radios, this is added on the same element that has the **checked** attribute, so that when submitting a form they may be contained in, you submit the correct values. BS4 will **not** handle the **checked** attribute for you—the only things it will change are the

styles on the button. If you wish to change the **checked** attribute on the underlying controls, then you'll need to add the required code to do this yourself, using either your framework or some standard JavaScript.

Code Listing 59 should produce a page the looks something along the lines of Figure 58.

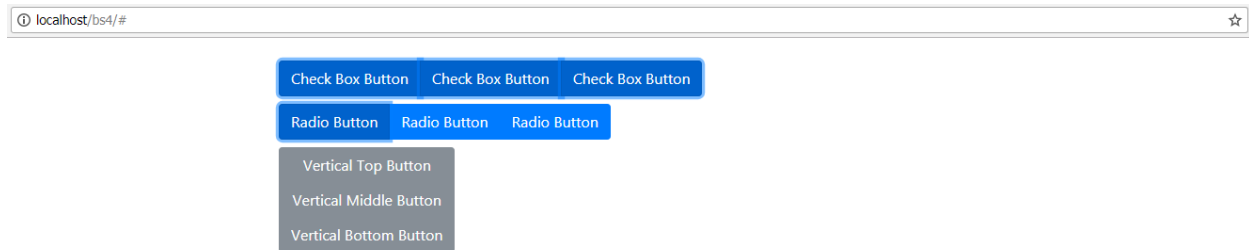


Figure 58: Check box, radio, and vertical group as produced by Code Listing 59

We're going to leave the subject of buttons here and move on to some other features. There's still a lot more that buttons can do—they have a lot of JavaScript functionality and they can be attached to data attributes to automatically open dialog boxes, menus, and many other things.

Going beyond this point means reading the [official documentation](#).

Chapter 7 Navigating around the World

I mentioned in the previous chapter that BS4 had a better way of providing sets of button links to use for navigation within an application. In this chapter, we'll see the navigation tools that BS4 provides to web application developers. Using these navigation tools is a simple process, but in this chapter, you'll start to see some large chunks of HTML code.

A basic navigation set

Many web developers will be familiar with using a `` tag that contains embedded `` tags to provide a list of items that are then styled to be used in a navigation system.

The following code listing is the simplest form of navigation we can do in BS4 using a standard `` construction.

Code Listing 60: A very basic navigation set

```
<!-- Page content goes here -->
<div class="container">
  <div class="row">
    <div class="col">

      <ul class="nav">
        <li class="nav-item">
          <a class="nav-link active" href="activelink">Active</a>
        </li>
        <li class="nav-item">
          <a class="nav-link" href="link2">Link 2</a>
        </li>
        <li class="nav-item">
          <a class="nav-link" href="link3">Link 3</a>
        </li>
        <li class="nav-item">
          <a class="nav-link disabled" href="disabledlink">Disabled</a>
        </li>
      </ul>

    </div>
  </div>
</div>
```

As you can see in Code Listing 60, the code is like much of what we've learned so far put together in a very standard way. BS4 adds its magic by way of the `nav`, `nav-item`, and `nav-link` classes.

The active link won't look any different in this chunk of code because we've done this so simply, but as you'll see soon, adding the **active** class in some cases does have a big impact.

If you add Code Listing 60 into the template we've been using and render it, your browser should look like this:



Figure 59: Basic navigation produced by the code in Code Listing 60

A point worth noting again here is the Disabled link. Just as with the warning in the chapter on buttons, the disabled link is **only** styled to look so. Clicking on it will still follow the underlying link, so you'll need to handle that in your framework if you're going to prevent it from going anywhere.

If Code Listing 60 is a little too much HTML for your liking, note that you don't have to do things that way. Because BS4 is Flexbox-first, you can also do the exact same thing using this simpler chunk of code:

Code Listing 61: A simpler way of creating the same output as Code Listing 60

```
<!-- Page content goes here -->
<div class="container">
  <div class="row">
    <div class="col">

      <nav class="nav">
        <a class="nav-link active" href="activelink">Active</a>
        <a class="nav-link" href="link2">Link 2</a>
        <a class="nav-link" href="link3">Link 3</a>
        <a class="nav-link disabled" href="disabledlink">Disabled</a>
      </nav>

    </div>
  </div>
</div>
```

If you change your template to use that code, rather than the code in Code Listing 60, and then refresh, the output should be exactly the same.

This highlights that as with most things in IT, there is always more than one way to achieve what you need to, and everything I've shown you in this book so far is only the beginning of where your journey with BS4 is heading.

When you're dealing with navigation collections like the ones in Code Listings 60 and 61, you might often need them to be centered or even right-aligned.

Anyone who's spent any time in HTML will know that this is often easier to describe than implement.

BS4, however, will make you wonder just why you ever made a fuss about it. Because of BS4's love of Flexbox, you can simply add **justify-content-center**, **justify-content-end**, or the default **justify-content-start** to your outer parent holding the collection of navigation items.

Code Listing 62 highlights this.

Code Listing 62: Aligning your navigation collections

```
<!-- Page content goes here -->
<div class="container">
  <div class="row">
    <div class="col">

      <p>Left (Start) aligned</p>
      <nav class="nav">
        <a class="nav-link active" href="activelink">Active</a>
        <a class="nav-link" href="link2">Link 2</a>
        <a class="nav-link" href="link3">Link 3</a>
        <a class="nav-link disabled" href="disabledlink">Disabled</a>
      </nav><hr/>

      <p>Center aligned</p>
      <nav class="nav justify-content-center">
        <a class="nav-link active" href="activelink">Active</a>
        <a class="nav-link" href="link2">Link 2</a>
        <a class="nav-link" href="link3">Link 3</a>
        <a class="nav-link disabled" href="disabledlink">Disabled</a>
      </nav><hr />

      <p>Right (End) aligned</p>
      <nav class="nav justify-content-end">
        <a class="nav-link active" href="activelink">Active</a>
        <a class="nav-link" href="link2">Link 2</a>
        <a class="nav-link" href="link3">Link 3</a>
        <a class="nav-link disabled" href="disabledlink">Disabled</a>
      </nav>

    </div>
  </div>
</div>
```

If you put this code into your template, then you should see the following output if your classes and layout are correct.

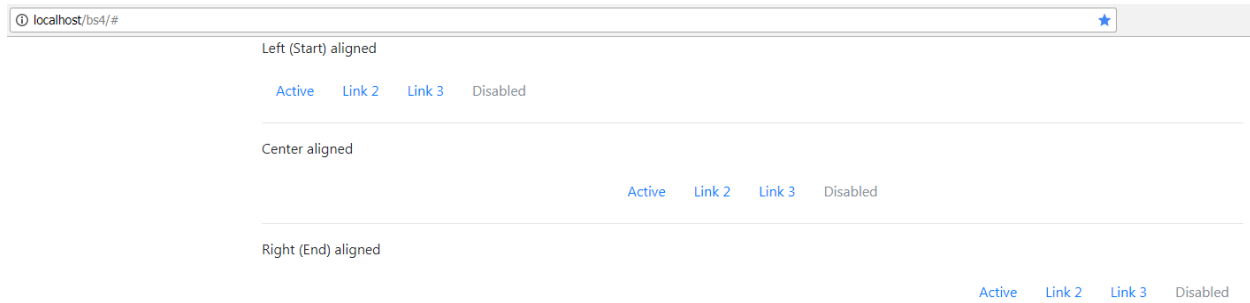


Figure 60: Aligned navigation collections

If you want your navigation collection to run vertically instead of horizontally, then you need to replace any of the justify classes with **flex-column**.

Code Listing 63: Making your navigation collection vertical

```
<!-- Page content goes here -->
<div class="container">
  <div class="row">
    <div class="col">

      <nav class="nav flex-column">
        <a class="nav-link active" href="activelink">Active</a>
        <a class="nav-link" href="link2">Link 2</a>
        <a class="nav-link" href="link3">Link 3</a>
        <a class="nav-link disabled" href="disabledlink">Disabled</a>
      </nav>

    </div>
  </div>
</div>
```

If you change the extra class on your parent container so that it looks like the snippet of code in Code Listing 63, you'll see that your **nav** items now run top to bottom. The modifier classes for left, center, right, and vertical will work on both the simple nav bar as shown in Code Listing 63, and on the **/** layout that we opened the chapter with.

Actively marking the current entry

We've been adding the **active** class to the examples we've done so far, but as of yet, you've not seen much of a change in the **nav** item marked as being the active one.

To make use of it, we need to add an outer container, such as a tab or pill set. When we do this in BS4, we'll suddenly see that we get some extra graphical goodness on our navigation collection, extra goodness that looks more like you would expect a navigation collection to look.

Code Listing 64 shows both of our layouts for navigation, and as you can see, both have a **nav-tabs** class on the parent container.

Code Listing 64: Highlighting the active choice using a tabbed navigation collection

```
<!-- Page content goes here -->
<div class="container">
  <div class="row">
    <div class="col">

      <br />
      <nav class="nav nav-tabs">
        <a class="nav-link active" href="activelink">Active</a>
        <a class="nav-link" href="link2">Link 2</a>
        <a class="nav-link" href="link3">Link 3</a>
        <a class="nav-link disabled" href="disabledlink">Disabled</a>
      </nav>

      <br /><br />

      <ul class="nav nav-tabs">
        <li class="nav-item">
          <a class="nav-link active" href="activelink">Active</a>
        </li>
        <li class="nav-item">
          <a class="nav-link" href="link2">Link 2</a>
        </li>
        <li class="nav-item">
          <a class="nav-link" href="link3">Link 3</a>
        </li>
        <li class="nav-item">
          <a class="nav-link disabled" href="disabledlink">Disabled</a>
        </li>
      </ul>

    </div>
  </div>
</div>
```

If you render Code Listing 64 in your browser, you'll see something a little odd.

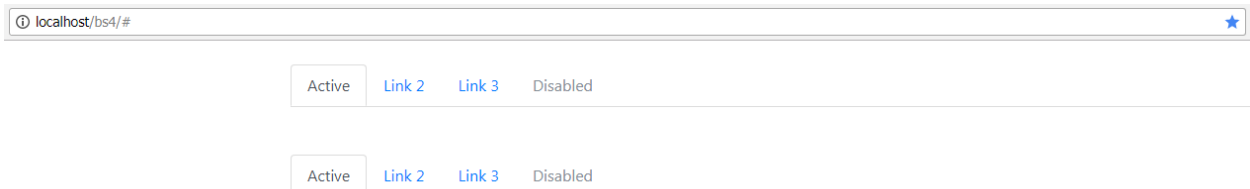


Figure 61: Nav tabs with both a flex layout and a traditional construction

When you use a simple **nav** element to create a tabbed collection, a line is drawn across the bottom of the container, whereas with the `/` way of doing things, there is a gap. I'm not sure why this is, and at the time of writing this book, I've not been able to find an answer for it. My advice here would be to use the appropriate construction, depending on if you want to see a gap or not. Whichever way you look at it, the **active** class serves to highlight the chosen option, and if you swap the classes appropriately in your framework, you'll find that you can track things very easily.

You can easily change the tabs to pills by changing the **nav-tabs** class to **nav-pills**, as Code Listing 65 shows.

Code Listing 65: Changing our tabs to pills

```
<!-- Page content goes here -->
<div class="container">
  <div class="row">
    <div class="col">

      <br />
      <nav class="nav nav-pills">
        <a class="nav-link active" href="activelink">Active</a>
        <a class="nav-link" href="link2">Link 2</a>
        <a class="nav-link" href="link3">Link 3</a>
        <a class="nav-link disabled" href="disabledlink">Disabled</a>
      </nav>

      <br /><br />

      <ul class="nav nav-pills">
        <li class="nav-item">
          <a class="nav-link active" href="activelink">Active</a>
        </li>
        <li class="nav-item">
          <a class="nav-link" href="link2">Link 2</a>
        </li>
        <li class="nav-item">
          <a class="nav-link" href="link3">Link 3</a>
        </li>
        <li class="nav-item">
          <a class="nav-link disabled" href="disabledlink">Disabled</a>
        </li>
      </ul>

    </div>
  </div>
</div>
```

When you render Code Listing 65 in your browser, you'll notice that both the `nav` element and the `/` construction render the same, so both can be used interchangeably. I suspect that the extra line in the tab version will be addressed at some point in the future as a bug. It may have been changed already in a newer version of BS4 than the one I'm using for these examples.

There's a lot more you can do with standard navigation layouts, including justifying each entry so that they all take up equal space, and reacting to JavaScript events. The rest of the documentation can be found in the [BS4 docs](#).

Full navigation bars

Navigation collections in BS4 are only the tip of the iceberg—BS4 provides a set of components designed specifically for providing rich navigation experiences at the top or the bottom of any design you create using the toolkit.

These can contain text, links, titles, drop-down menus, forms, and lots of other things.

Rather than trying to build my own sample, I'll use the opening sample directly from the BS4 documents and modify it slightly.

Code Listing 66: Standard BS4 nav bar example

```
<!-- Page content goes here -->
<div class="container">
  <div class="row">
    <div class="col">

      <nav class="navbar navbar-expand-lg navbar-light bg-light">
        <a class="navbar-brand" href="#">BS4 Succinctly</a>
        <button class="navbar-toggler" type="button" data-toggle="collapse"
data-target="#navbarSupportedContent" aria-controls="navbarSupportedContent"
aria-expanded="false" aria-label="Toggle navigation">
          <span class="navbar-toggler-icon"></span>
        </button>

        <div class="collapse navbar-collapse" id="navbarSupportedContent">
          <ul class="navbar-nav mr-auto">
            <li class="nav-item active">
              <a class="nav-link" href="#">Home <span class="sr-
only">(current)</span></a>
            </li>
            <li class="nav-item">
              <a class="nav-link" href="alink">Link</a>
            </li>
            <li class="nav-item dropdown">
              <a class="nav-link dropdown-toggle" href="#"
id="navbarDropdown" role="button" data-toggle="dropdown" aria-haspopup="true"
```

```

aria-expanded="false">
    Dropdown
  </a>
  <div class="dropdown-menu" aria-labelledby="navbarDropdown">
    <a class="dropdown-item" href="action">Action</a>
    <a class="dropdown-item" href="action2">Another action</a>
    <div class="dropdown-divider"></div>
    <a class="dropdown-item" href="somethingelse">Something
else here</a>
  </div>
</li>
<li class="nav-item">
  <a class="nav-link disabled" href="disabledlink">Disabled</a>
</li>
</ul>
<form class="form-inline my-2 my-lg-0" action="dosearch">
  <input class="form-control mr-sm-2" type="search"
placeholder="Search" aria-label="Search" name="searchterm">
  <button class="btn btn-outline-success my-2 my-sm-0"
type="submit">Search</button>
</form>
</div>
</nav>

</div>
</div>
</div>

```

The output produced by Code Listing 66 should look something like this once you put it in your template and load it into your browser.

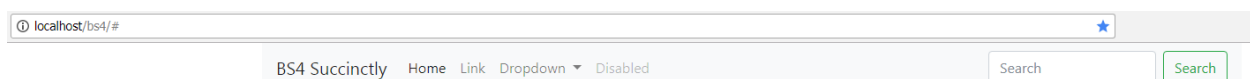


Figure 62: Standard nav bar produced by Code Listing 66

At first look, the code in Code Listing 66 is absolutely terrifying, and easily qualifies as one of the most complex examples in the book. However, once you start to break it down, it's much easier to understand.

The main container is the **nav** element at the top of the layout. This serves as the actual container that everything else is held by. It's also the main point where most of your actual navigation bar styling is performed.

The first style to note is the **navbar-expand-lg** class. With this applied, any element or container width in BS4 that is **lg** (large) or greater will expand the navigation bar to show all its links and buttons.

Try it with the example loaded and resize your browser window width.

You'll find that around the 1,000 pixel mark, the navigation bar will toggle automatically between full and folded modes.

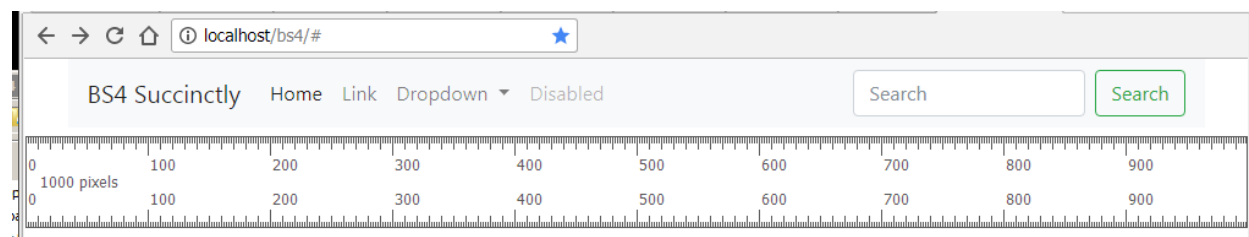


Figure 63: At about 1,000 pixels the menu will expand

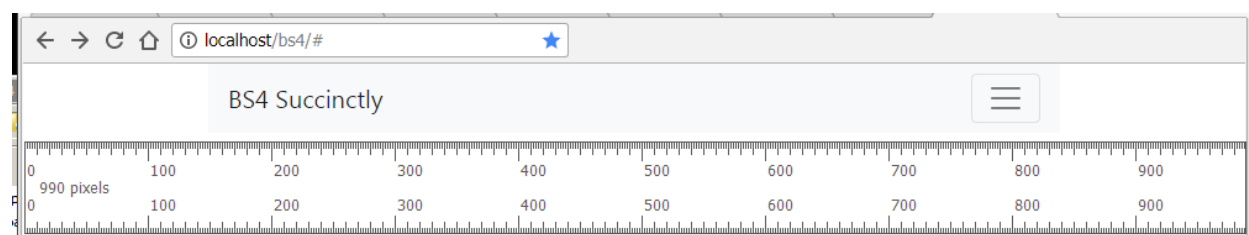


Figure 64: At just a little below 1,000 pixels, the menu bar collapses

As you can see in Figures 63 and 64, the “large” breakpoint is around the 1,000 pixel mark, and so the addition of the **navbar-expand-lg** modifier class causes the toggle at that breakpoint.

Changing the modifier for **expand** to be one of **sm**, **md**, or **xl** will allow that breakpoint to be set so the navigation bar collapses at “small,” “medium,” or “extra-large” breakpoints. The actual number of pixels each of these takes depends a lot on which device your application is running on, and what the physical screen size is. The introductory chapters in the BS4 documentation contain a detailed description of how all of this is worked out. Keep in mind that using the correct modifiers will always maintain consistency, however, and you'll generally not need to be aware of pixel sizes.

As a rule of thumb, I always set the breakpoint to the point just before I can't display all my links in the navigation bar, or when things start to look too squashed.

If my links fit on an extra-large width layout, but look like rubbish at anything below that, then the extra-large breakpoint is where I flip it. Basically, experiment and see what works best for you.



Note: For those who are wondering, the ruler you can see in Figures 63 and 64 is a small Windows desktop app I downloaded from a website called silver-software a long time ago. The website seems long gone now, but I haven't had the heart to uninstall it, as it's been a great little tool. I'm sure one of the download sites will still have it somewhere, should you wish to look for it.

The next two classes, **navbar-light** and **bg-light**, set the colors of the navigation bar. There are a lot of different combinations here, and any color that's used anywhere else can also be used here. For example, try changing the **navbar-light** to **navbar-dark**, and **bg-light** to **bg-primary**, to get a navigation bar that looks like the one in the following figure.

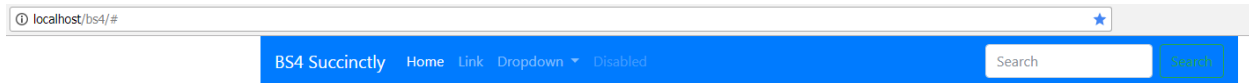


Figure 65: Navigation bar styles are easy to change

The next element is the `<div>` holding the **navbar-brand**, which forms the title of our menu bar. This can be changed for any icon that can be rendered using font-awesome or glyph icons; there's an example on the [navbar page in the BS4 docs](#) showing you how to achieve this.

After the brand text, you'll see a button marked with the class **navbar-toggler**. This is the button you see appear (with the three stripes) when the navbar collapses, as shown in Figure 64. It's this button that opens and closes the menu when you click on it. The attributes to pay close attention to here are the **data-*** attributes.

Back in the chapter on alerts, I mentioned these attributes as being the primary mechanism that BS4 uses to hook up JavaScript behaviors without you having to touch any JavaScript code. In the navbar, these attributes do exactly the same thing. In particular, if you look at the **data-target** attribute, you'll see that it contains an ID selector. If you look just below this at the first `<div>` element after the button, you'll see that the ID of that `<div>` tag matches the selector.

This marks the target element for the collapse operation, and anything that's inside that outer `<div>` will be content that is collapsed down into the menu bar when it collapses.

Inside the `<div>` that's marked to collapse, you'll see that the rest of the content is mostly made up of a `` element and a number of nested `` elements. Each element holds a **navbar** item and can be one of the following:

- Navigation link
- Drop-down link
- Form
- Standard button
- Static text

You can add other things too, but for those you need to create the containers and layouts yourself. Using the built-in styles to create any of the items I've just listed will ensure that anything inside the navbar will maintain the same vertical or central axis, so nothing will ever look as though it's not aligned and will automatically hide when a collapse operation is in effect.

I'm not going to list all the different variations that can be used; instead, you should take a look at the [BS4 docs](#), where you'll find plenty of examples of the different styles and combinations available.

Chapter 8 The House of Cards

As you may have gathered by now, a lot of the old section, well, and general header and footer components have been removed. It was these sectioning and layout components that attracted many developers, myself included, to adopt Bootstrap in the first place.

Fear not, though—the old way might be gone, but in its place comes an all-powerful, Flexbox-driven replacement with near-infinite ways to use it.

The Twitter card component

A card is a generic block-level component with specific styling geared for very specific use cases, such as a photo thumbnail with a description and a call to action below it, as Code Listing 67 shows.

Code Listing 67: A very basic Twitter card

```
<!-- Page content goes here -->
<div class="container">
  <div class="row">
    <div class="col">

      <br />
      <div class="card" style="width: 18rem;">
        
        <div class="card-body">
          <h5 class="card-title">Flying Cats</h5>
          <p class="card-text">The newest thing in pet owning circles,
.....</p>
          <a href="#" class="btn btn-primary">Learn More</a>
        </div>
      </div>
    </div>
  </div>
</div>
```

With very little markup, Code Listing 67 produces the card shown in Figure 66. Note that in the listing I've substituted the link to the cat picture I used with a link to the placeholder service mentioned in the chapter on images. Feel free to add in your own 500 × 500 image to replace the gray 500 × 500 filler image that you'll see.

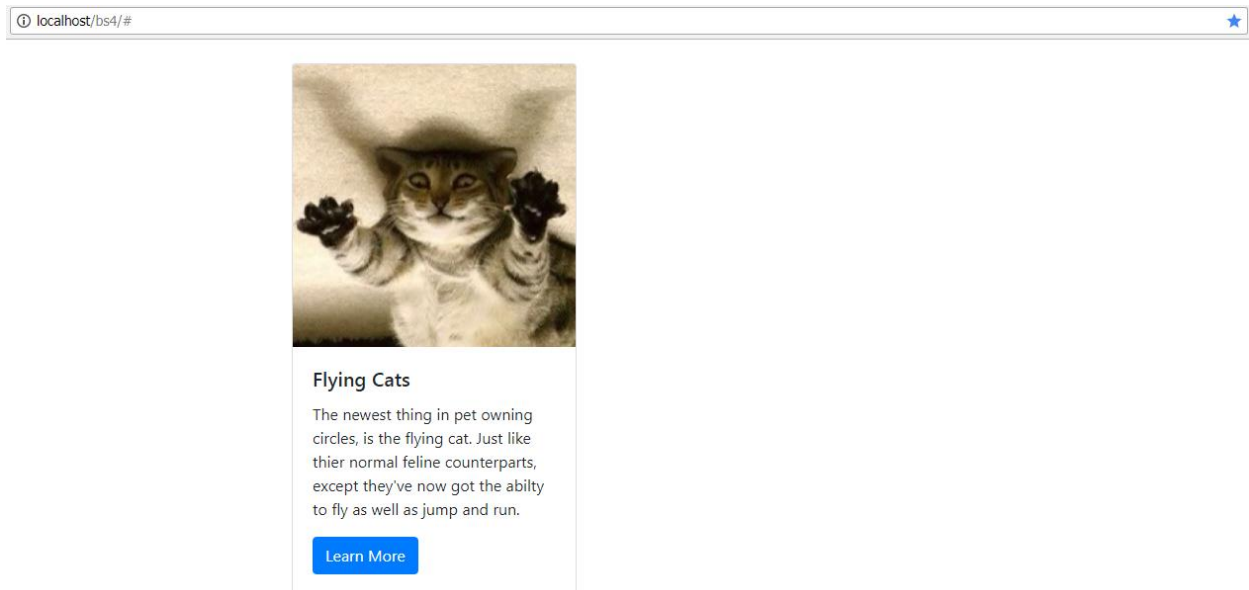


Figure 66: A basic Twitter card

As you can see from Code Listing 67, the structure is very much like everything else we've seen so far. We have an outer parent with a single class, **card** in this case, followed by a number of inner elements that each have more specific purposes, such as **card-img-top**, **card-text**, and **card-body**.

Because cards are all block-level attributes, the BS4 development team recommends using the BS4 grid layout classes to position and size the cards. This gives you absolute flexibility, and it ensures that your layouts will always allow BS4 to look after all your responsive resizing tasks for you.

We don't have space here to cover everything the card component can do, but I will show you a few notable examples.

Headers and footers

Headers and footers can be added to your cards by just adding an extra **<div>** element in the appropriate place.

Code Listing 68: Adding a header to your card

```
<!-- Page content goes here -->
<div class="container">
  <div class="row">
    <div class="col">

      <br />
      <div class="card">
        <div class="card-header">
```

```

        Information about "Peter Shaw"
    </div>
    <div class="card-body">
        <h5 class="card-title">Hello my name is SHAWTY</h5>
        <p class="card-text">This is some text telling you all about me
and what I'm doing.</p>
        <a href="#" class="btn btn-primary">Click Here to Learn More</a>
    </div>
</div>

</div>
</div>
</div>

```

The header is simply added as a nested `<div>`, immediately after the opening tag for the card itself, and then given a class of **card-header**. The rest of the markup is the same as the example in Code Listing 67. Note that the screenshot shows the card expanding to the full width of the **col** holding it. If that `<div>` was changed to something along the lines of **col-md-3**, then the card would take up exactly a quarter of the space available (remember, there are 12 cells across by default).

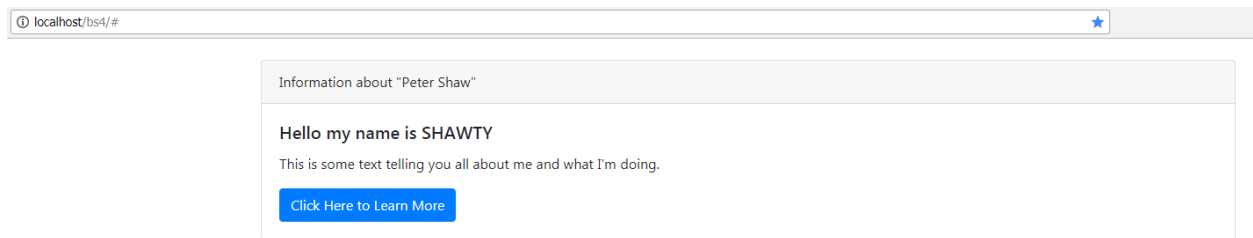


Figure 67: A Twitter card with a header

If you change the class to **card-footer** and place the `<div>` just before the closing tag of the card container, you'll get the same effect, but at the bottom of the container.

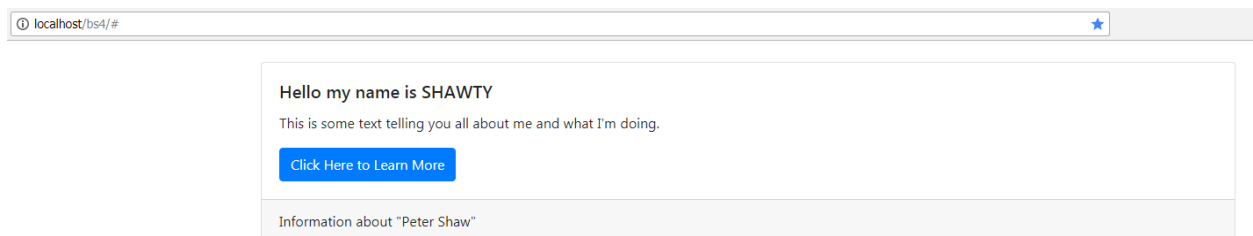


Figure 68: A Twitter card with a footer

Card colors

Just as with everything else in BS4, you can use the color classes to add contextual color to your cards. **bg-success**, **bg-light**, **bg-info**, and the rest allow you to set the background color of the card.

Code Listing 69, while a bit on the long side, should give you a good feel for the different color combinations you can use.

Code Listing 69: Some of the different color combinations a card can use

```
<div class="container">
  <br />
  <div class="row">

    <div class="col-lg-3">
      <div class="card text-white bg-primary mb-3" style="max-width:
18rem;">
        <div class="card-header">Header</div>
        <div class="card-body">
          <h5 class="card-title">Primary card title</h5>
          <p class="card-text">Some quick example text to build on the card
title and make up the bulk of the card's content.</p>
        </div>
      </div>
    </div>

    <div class="col-lg-3">
      <div class="card text-white bg-secondary mb-3" style="max-width:
18rem;">
        <div class="card-header">Header</div>
        <div class="card-body">
          <h5 class="card-title">Secondary card title</h5>
          <p class="card-text">Some quick example text to build on the card
title and make up the bulk of the card's content.</p>
        </div>
      </div>
    </div>

    <div class="col-lg-3">
      <div class="card text-white bg-success mb-3" style="max-width:
18rem;">
        <div class="card-header">Header</div>
        <div class="card-body">
          <h5 class="card-title">Success card title</h5>
          <p class="card-text">Some quick example text to build on the card
title and make up the bulk of the card's content.</p>
        </div>
      </div>
    </div>
  </div>
</div>
```

```

    </div>

    <div class="col-lg-3">
      <div class="card text-white bg-danger mb-3" style="max-width:
18rem;">
        <div class="card-header">Header</div>
        <div class="card-body">
          <h5 class="card-title">Danger card title</h5>
          <p class="card-text">Some quick example text to build on the card
title and make up the bulk of the card's content.</p>
        </div>
      </div>
    </div>

    <div class="row">

      <div class="col-lg-3">
        <div class="card text-white bg-warning mb-3" style="max-width:
18rem;">
          <div class="card-header">Header</div>
          <div class="card-body">
            <h5 class="card-title">Warning card title</h5>
            <p class="card-text">Some quick example text to build on the card
title and make up the bulk of the card's content.</p>
          </div>
        </div>
      </div>

      <div class="col-lg-3">
        <div class="card text-white bg-info mb-3" style="max-width: 18rem;">
          <div class="card-header">Header</div>
          <div class="card-body">
            <h5 class="card-title">Info card title</h5>
            <p class="card-text">Some quick example text to build on the card
title and make up the bulk of the card's content.</p>
          </div>
        </div>
      </div>

      <div class="col-lg-3">
        <div class="card bg-light mb-3" style="max-width: 18rem;">
          <div class="card-header">Header</div>
          <div class="card-body">
            <h5 class="card-title">Light card title</h5>
            <p class="card-text">Some quick example text to build on the card
title and make up the bulk of the card's content.</p>
          </div>
        </div>
      </div>
    </div>

```

```

    </div>
  </div>

  <div class="col-lg-3">
    <div class="card text-white bg-dark mb-3" style="max-width: 18rem;">
      <div class="card-header">Header</div>
      <div class="card-body">
        <h5 class="card-title">Dark card title</h5>
        <p class="card-text">Some quick example text to build on the card
title and make up the bulk of the card's content.</p>
      </div>
    </div>
  </div>
</div>
</div>

```

When rendered, you should see something similar to the following figure.

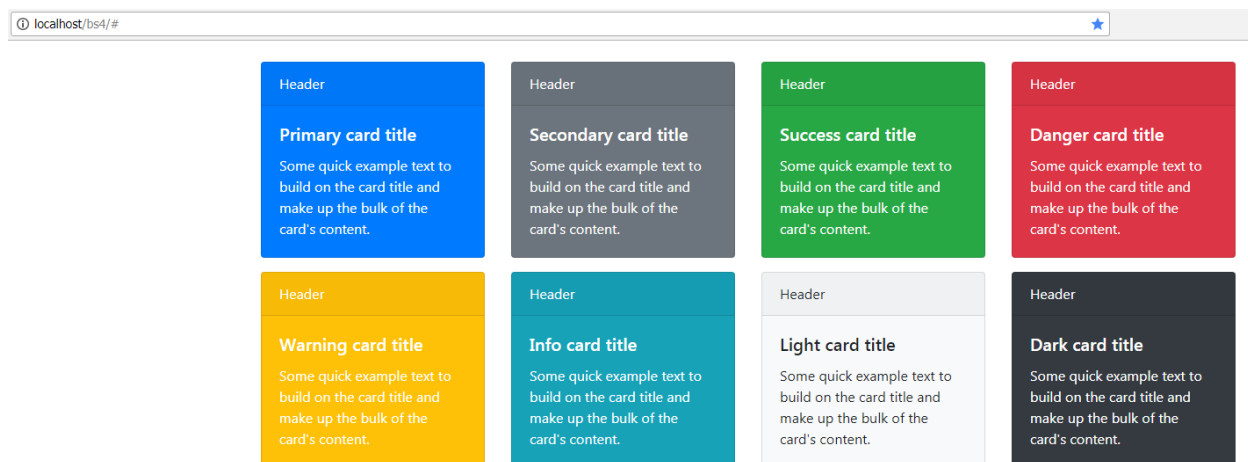


Figure 69: Different card color combinations

As with the navigation bars, the authority on using the card component can be found in the [BS4 documentation](#), and like the navbar, there's much more you can use a card for. There are special classes to include a navbar in the header of a card, so you can make small tab-driven and pill-driven content panels. There are also text alignment classes, and the ability to line up buttons and text to produce stunning dialogs. For now, however—in this book at least—it's time for us to move on.

Chapter 9 A Form of Data Entry

The one thing that hasn't changed very much in BS4 is the form components. The general premise of using the standard HTML5 `<form>` and `<input>` tags and styling them at an element level rather than adding extra classes is still very much the most basic way of using BS4 on your forms.

In reality, to get the best out of BS4's forms, you'll want to at least use the **form-group**, **form-control**, and **form-text** classes. By doing this for really simple forms, you can make sure that your inputs and their associated labels are associated with each other correctly, and more importantly, are easy to style with the validation helpers and make sense to assistive technologies using their various ARIA role attributes.

The most basic form that you can produce using these base classes in BS4 looks something like this:

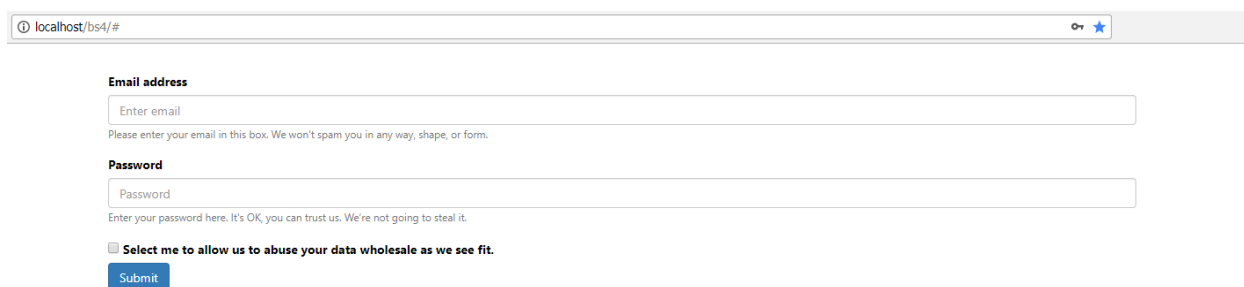


Figure 70: A very basic form

In this format, the labels are above the inputs, the inputs are wrapped in a `<div>` using the **form-group** class, along with their respective labels, and a `` or other inline element using **form-help** is placed under the `<input>` element to provide the help text.

The code to produce the form is quite simple.

Code Listing 70: The markup for the basic form shown in Figure 70

```
<!-- Page content goes here -->
<div class="container">
  <div class="row">
    <div class="col">

      <br />
      <form>
        <div class="form-group">
          <label for="email">Email address</label>
          <input type="email" class="form-control" id="email" aria-
describedby="emailHelp" placeholder="Enter email">
```

```

        <small id="emailHelp" class="form-text text-muted">Please enter
your email in this box. We won't spam you in any way, shape, or form.</small>
    </div>
    <div class="form-group">
        <label for="password">Password</label>
        <input type="password" class="form-control" id="password" aria-
describedby="passwordHelp" placeholder="Password">
        <small id="passwordHelp" class="form-text text-muted">Enter your
password here. It's OK, you can trust us. We're not going to steal
it.</small>
    </div>
    <div class="form-check">
        <input type="checkbox" class="form-check-input" id="confirm">
        <label class="form-check-label" for="confirm">Select me to allow
us to abuse your data wholesale as we see fit.</label>
    </div>
    <button type="submit" class="btn btn-primary">Submit</button>
</form>

</div>
</div>
</div>

```

There are a few things you need to take note of in Code Listing 70. First, notice that the label for the attribute is the same value as the actual ID assigned to the input control. This is not a BS4 thing, but an HTML5 thing that allows assistive tech to know which label belongs to which control.

If you don't want to use labels on your forms, then you should still create label tags in your HTML but hide them from display by using the **sr-only** class. Adding this as the final class on the other classes in the attribute will effectively hide your label from the browser rendering process, but will still allow assistive technologies to tell the user what the input field represents.

The second thing to note is the **aria-describedby** attribute on the input control. Again, like the label, the value of this is the ID of another element in the group. This time, however, it's the ID of the inline element with the class **form-text**, which, if present, will be used by screen readers to provide extra context-sensitive descriptions to the user.

The final point to note is the order of the check box and the label for the check box control. In order for the label to render correctly, the label must always follow the input. This is a limitation of HTML, not anything in BS4.

The biggest change in forms for BS4 is in the layout possibilities they provide thanks to the Flexbox-first policy that BS4 now uses.

If we strip Code Listing 70 back to just input elements in a row with **col <div>s**, we can easily achieve the following output.

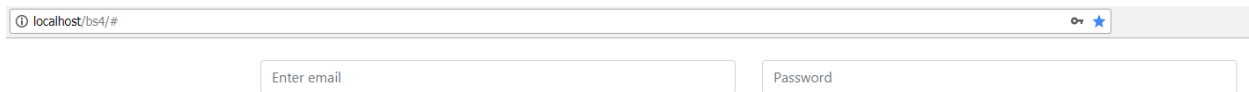


Figure 71: Inputs arranged horizontally using columns

We can use all the grid-based classes such as **col**, **col-md-***, and **row** inside and outside of **form-groups**, and with plain old input elements on their own.

The one place where this really changes, however, is in producing horizontal forms.

In previous versions of Bootstrap, there existed specific classes for aligning forms horizontally. Now, all you need to do is add an additional **row** class to your **<div>** that forms the **form-group**, and then divide the inner space using **col**-based classes on the labels and input elements as needed. For example:

Code Listing 71: A small change makes horizontal forms easy

```
<!-- Page content goes here -->
<div class="container">
  <div class="row">
    <div class="col">

      <br />
      <form>
        <div class="form-group row">
          <label for="email" class="col-sm-2 col-form-label">Email</label>
          <div class="col-sm-10">
            <input type="email" class="form-control" id="email"
placeholder="Email">
          </div>
        </div>
        <div class="form-group row">
          <label for="password" class="col-sm-2 col-form-
label">Password</label>
          <div class="col-sm-10">
            <input type="password" class="form-control" id="password"
placeholder="Password">
          </div>
        </div>
        <div class="form-group row">
          <div class="col-sm-10">
            <button type="submit" class="btn btn-primary">Sign in</button>
          </div>
        </div>
      </form>

    </div>
  </div>
</div>
```

This is in stark contrast to BS3, which involved an HTML structure that was a bit different from the general one shown in Code Listing 71, and a number of class name changes. In BS4, it's much easier to have forms switch between the horizontal layout and vertical layout, depending on the screen or device size and orientation.

Code Listing 71 should produce the following output when rendered using your BS4 template.

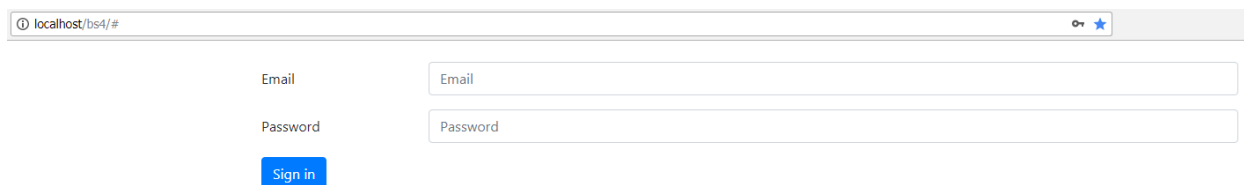


Figure 72: Horizontal forms are now much easier

If you use **col-auto** instead of **col** when breaking up your forms on the grid, you can get your controls to only use as much space as is needed to draw the control and its contents. This comes in handy if you use the **form-inline** class directly on the **<form>** tag of your form, which will force your entire form to run along one line. (This is exactly what the navbar form does internally when you add a form to a navigation bar.)

Aside from the layout possibilities, adding **disabled**, **readonly**, and the various new types as defined in the HTML5 specification will cause BS4 to react appropriately and style the form elements as needed to show them in read-only, disabled, or both states when rendered.

Form validation

The final change that might come as a bit of a shock is the validation helpers.

In BS3, you added various classes such as **has-error**, **has-warning**, and others to the actual **<form>** elements or **<div>** elements containing the form groups.

While these still exist in BS4, the recommended way is now to use the HTML5 Validation API. With the use of four new classes, everything is handled for you by BS4—you no longer have to start adding and removing lots of classes in lots of places as you did with BS3.

Take a look at the following HTML code.

Code Listing 72: BS4 form using HTML5 validation

```
<!-- Page content goes here -->
<div class="container">
  <div class="row">
    <div class="col">

      <br />
      <form class="needs-validation" novalidate>
        <div class="form-row">
```

```

        <div class="col-md-4 mb-3">
            <label for="firstName">First name</label>
            <input type="text" class="form-control" id="firstName"
placeholder="First name" required>
            <div class="valid-feedback">That's great thanks</div>
            <div class="invalid-feedback">Please enter your first name
here</div>
        </div>
        <div class="col-md-4 mb-3">
            <label for="lastName">Last name</label>
            <input type="text" class="form-control" id="lastName"
placeholder="Last name" required>
            <div class="valid-feedback">That's great thanks</div>
            <div class="invalid-feedback">Please enter your last name
here</div>
        </div>
        </div>
        <button class="btn btn-primary" type="submit">Submit form</button>
    </form>

</div>
</div>
</div>

```

Notice that it's mostly just a standard BS4 form layout using form rows, groups, and input types with labels.

Take a look at the opening `<form>` tag. Notice the **needs-validation** class that's been added to it, along with the **novalidate** attribute.

The **novalidate** attribute is added to stop the browser from attempting to apply the regular HTML5 validation to the form when it first loads. The **needs-validation** class tells BS4 **not** to style any of the input elements or groups, or to activate any validation states. What this means is that the form looks like the following figure when it's first loaded.

The screenshot shows a web browser window with the address bar displaying 'localhost/bs4/?#'. The form consists of two text input fields side-by-side. The first field is labeled 'First name' and contains the placeholder text 'First name'. The second field is labeled 'Last name' and contains the placeholder text 'Last name'. Below these fields is a blue button with the text 'Submit form'.

Figure 73: The BS4 form from Code Listing 72 when it's first loaded

There is nothing particularly surprising here, but if you now change the class on the `<form>` tag from **needs-validation** to **was-validated**, and then press **F5** to reload your form, it should now look like this:

Figure 74: The BS4 form from Code Listing 72 with the single class changed on the parent form

That one change tells BS4 that you've now validated the form, and that it now has to show the different styles to indicate that there are failures in the validation. Notice that the different colors work automatically. If you type something into one of the boxes, for example, you'll see the validation state change along with the text below the input field.

What's more, you haven't had to change any classes manually to make this happen, as you used to have to do under BS3. Instead, Bootstrap has interrogated the input controls using the HTML5 Validation API, and if you look at the two input text boxes used, you'll see they have **required** attributes.

What's happening here is you're setting the **novalidate** attribute on the form so that the HTML5 Validation API allows you to click the **Submit** button. In your **Submit** button handler, you would then check the state manually in JavaScript, and if your validation was not satisfied, you'd change the CSS class on the form to **was-validated** and let BS4 do the rest of the heavy lifting for you.

If you look at the **<div>** elements underneath the inputs, you'll see that they each have two **<div>** elements: one with a **valid-feedback** class and one with an **invalid-feedback** class on it. If the form passes validation, valid feedback elements are unhidden, and invalid feedback elements remain hidden. If validation fails, the opposite is performed, and invalid feedback elements are shown.

A suitable JavaScript function to handle this might look like the following code listing (taken straight from the [BS4 docs](#)).

Code Listing 73: A small JavaScript function for handling BS4 form validation

```
<script>
(function() {
  'use strict';
  window.addEventListener('load', function() {

    var forms = document.getElementsByClassName('needs-validation');

    var validation = Array.prototype.filter.call(forms,
function(form) {
  form.addEventListener('submit', function(event) {
    if (form.checkValidity() === false) {
      event.preventDefault();
      event.stopPropagation();
    }
  });
});
```

```
        form.classList.add('was-validated');
    }, false);
});
}, false);
})();
</script>
```

At page load, we use the jQuery **ready** handler to run a small routine that looks for and attaches a handler to every form in the page that has the **needs-validation** class. This handler intercepts the form's **submit** call so that when the form is submitted, it stops the submission from taking place. Only if **checkValidity** in the HTML5 Validation API tells it that the form has no invalid elements does the routine then reverse that block and allow the form to be submitted.

This has the benefit that, if JavaScript is disabled, the standard form submission process still works as expected, but no validation takes place.

Of course, you can add any validation mechanism you like here, and use any validation method you want. All that matters is that the form's validation state is reported as valid or invalid, and the class on the **<form>** tag is changed as required.

Again, there's more to be seen, so as with previous chapters, I encourage you to go to the [forms documentation](#) in the BS4 docs and continue from there.

The Remainder of the Component Soup

There are at least 15 other subgroups in the BS4 component set that we haven't looked at, and not very much space left in this book to cover them.

Input groups, as mentioned in the chapter on forms, have an entire section of the BS4 docs all to themselves.

In terms of feedback, you have page breadcrumbs, navigation has the paging and pagination components, and buttons have button groups and drop-downs.

There are collapsible panels, carousels, list groups, modal dialogs, and much, much more.

In a book as small as this one, I can't possibly hope to cover all the components, so if you're looking for something specific, then the place to look is the [documentation](#).

Many people faulted the documentation with being very poor for BS3 and older versions, but in BS4, the quality of the documentation has massively improved.

The BS4 team has not only documented everything they've done, but they've also improved the samples, and added a lot of guidance for using BS4 successfully in HTML in general.

There is also the [examples page](#) in the BS4 documentation.

There are many examples of using all of BS4's components and features together in ready-to-use layouts and page styles. I can honestly say that an afternoon spent playing with the code in these samples will show you more than you'll ever need about how to effectively use Bootstrap in your web-based projects.

Many of the components that I haven't covered here are still the same as used in BS3, so a quick read of [my previous book](#) is probably worth doing, as there's a lot of info in that edition that is still relevant in BS4.

Bear in mind that there are a near-infinite number of combinations of what you can do. For example, a card can contain a list group, allowing you to do some very fancy list-based designs. A dialog box can contain cards and card headers to give better dialog box styles by using just the base dialog styles and HTML structures. Navbars can be placed inside tables, and table pagination can contain drop-down button collections.

The only limit is your imagination, and a little bit of thought as to how you can combine things for your own use cases. Don't look at BS4 as a set of things used for a specific purpose; instead, see BS4 as a user interface toolkit to design the experience you need.

Chapter 10 Bootstrap's Batman Utility Belt

Finally, we come to the small fry, the scraps left at the bottom of the barrel that everyone always ignores.

In BS4's case, however, these scraps are immensely powerful and pack a heck of a punch for their weight. As you'll see in this chapter, just when you thought you'd seen it all, there are still many surprises to be had.

Display utilities

One big thing that's changed since BS3 is the display classes. In BS3, display classes were limited to a single set of hidden and show classes that were tied to breakpoints.

What this meant was that you could only "hide" or "show" for a given breakpoint. For example, for the class list of **visible-xs visible-lg**, the element would be visible only at large or extra-small breakpoints. There was no default, for example, at the other breakpoints.

If you needed to hide other elements at the same time, you typically ended up with a class list that was very long, and had lots of hides and shows, making it very difficult to read.

There were variations on display for block and inline, but these usually ended up with names such as **visible-xs-block-12** or **hidden-lg-inline-6**, or even worse combinations.

In BS4, this has been changed drastically, mostly thanks to the new Flexbox-first mentality.

The new class names are simpler, and there are more of them, covering a wider range of use cases.

As a general rule of thumb, there is now a **d-*** class type for every display mode you can think of, and each of them can have the usual breakpoint applied to them if needed so that they only take effect at a given screen size.

The best part is that you now no longer need to use the size modifiers. If you want an element to show at all screen sizes, and be a **block** element, then you simply use **d-block**, rather than **visible-xs-block visible-sm-block visible-lg-block visible-xl-block**, as you would have done with BS3.

Having one class also makes manipulation in script much easier. If you want to show or hide an element on a page, it's a simple matter of toggling **d-block** and **d-none**.

The values available for **d-{value}** are:

- **none**
- **inline**
- **inline-block**
- **block**

- **table**
- **table-cell**
- **table-row**
- **flex**
- **inline-flex**

If you're at all familiar with the standard CSS display modes, you'll realize that each of these relate to a **display: {value}** style rule. If you want to use the rule so that they only work at certain breakpoints, then **d-{breakpoint}-{value}** will achieve that for you. For example, **d-sm-block** will mean that the element this rule is attached to will display as a **block** element only when the screen width is within the small breakpoint size. The standard breakpoint values are **xs** for extra small, **sm** for small, **lg** for large, and **xl** for extra large. The BS4 docs have a very good description of the sizes [here](#).

The size modifiers are useful when you have elements you want to remove from a layout as the screen gets narrower. For example, on a small screen you might not want to use a large hero-style banner at the top of your page, so a **d-sm-none d-xs-none** will quickly remove it without having to also make sure you take into account the opposite rules to show the element when you need it (everything shows by default at **md** size).

There is one further size modifier that only applies to the display utilities, and that's the **print** modifier. If you have elements that you only want to show when the page is sent to the printer, you can use **print** as the value instead of one of the size classes. For example, if you had a menu bar that you didn't want to appear when printing a table below it, you can add the class **d-print-none**, the result being that only the table below the menu bar would be printed.

All of the display types can be modified with **print**, which means you can also create parts of your page that only show when being printed. A good use for this is to print out the link of a URL so that when printed, the destination of a link is shown next to the linked text, but is hidden when displayed in the browser.

I haven't given you any examples of the display utilities as it's quite difficult to produce meaningful screen grabs when there's nothing shown on a page. The associated page in the [Utilities section of the docs](#) has plenty of them, however.

Borders

One thing that BS3 definitely didn't have was classes to manipulate the borders of an element. BS4 not only has them, but also allows you to use them to apply the same contextual colors that can be used for alerts, text, and backgrounds. You can use them in two modes: additive and subtractive.

When you use them in additive mode, you're adding a border to the element, which means if the element you apply them to already has existing borders, you won't see any difference. A **border** class will add the border all the way around the element, while **border-{value}**, with **{value}** being **top**, **right**, **bottom**, or **left**, will add the border only to the specified side. You can combine them too, so **border-left border-top** will add the border on both the top and the left of the element they are being applied to.

Subtractive, as the name suggests, removes the specified side from an already existing element, and is achieved by adding a `-0` after any of the additive versions. For example, if you have an element and you only want the top, left, and bottom sides to have a border, you can either use **`border-top border-left border-bottom`**, which is purely additive, or you can use **`border border-right-0`**, which as you can see, is fewer characters than the additive version.

The reason there are both additive and subtractive versions is purely so you can find the best combination for the layout you're trying to produce while trying to ensure you keep your class lists as small as possible.

For those of you wanting to reproduce the old well component from BS3, you can easily do that as shown in the following listing.

Code Listing 74: Using the border utilities to produce a well with an elastic middle

```
<!-- Body content goes here -->
<div class="container">
  <br/>
  <div class="row" style="height: 100px;">

    <div class="col-1 bg-light border border-right-0">
      Left
    </div>

    <div class="col bg-light border-top border-bottom text-center">
      Middle
    </div>

    <div class="col-1 bg-light border border-left-0">
      Right
    </div>

  </div>
</div>
```

Technically, you don't need three `<div>` elements to produce a well, you just need to apply **`border`** and **`bg-light`** class rules to a single `<div>` to get the same effect. However, if you use the same approach that I used in Code Listing 74, then you can change the width of the center portion and still maintain the same side and end caps, should you need to.

Rendering Code Listing 74 in the browser should give you something similar to the following figure.

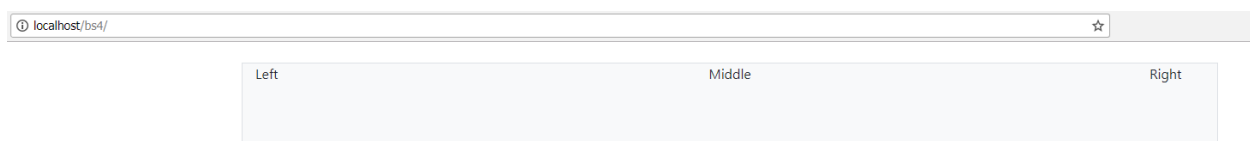


Figure 75: A well-style component produced by Code Listing 74

The border color can be changed by adding a **border-`{color}`** after the rest of the classes, with **`{color}`** being one of the common color names: **primary**, **secondary**, **success**, **danger**, **warning**, **info**, **light**, **dark**, or **white**. Change the rules on the left, middle, and right `<div>` elements in Code Listing 74 and add **border-danger** as the last class name, and you should see your output change to the following when you press **F5** to refresh.

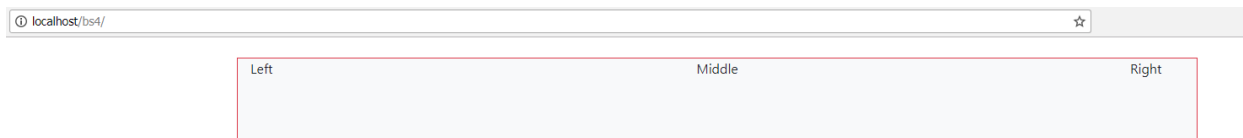


Figure 76: Our well-style component with border-danger added to the border classes

As you can probably guess, you can also change all the **bg-`{color}`** class styles, allowing you to change the background color, and if you choose to, add **text-`{color}`** classes to change the text color.

Finally, you can add a **rounded** or **rounded-`{type}`** to the element to round the corners on the element. **`{type}`** can be **top**, **right**, **bottom**, or **left** to add the rounding to only the specified side, or it can be **circle** to turn the element into a circle, or a **0** to remove the rounding.

Adding **rounding-left** to the left and **rounding-right** to the right `<div>` element in our example in Code Listing 74 will give us the following output.

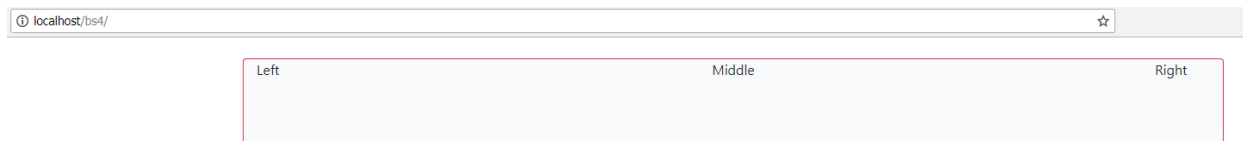


Figure 77: Our well component with rounded left and right edges

The border utilities can be applied to every component and element in the BS4 toolkit, from buttons to input fields, to cards and navbars, block, inline, or otherwise.

Flex utilities

As you've already seen, you can make any container a flex container by simply adding **d-flex** to its class list. **d-flex** is another display utility, and is the one you'll likely use the most to enable your containers to be Flexbox oriented. BS4 greatly simplifies using Flexbox, but the Flexbox standard as a whole goes way beyond that.

We don't have time to describe everything that Flexbox can do—for that I recommend the "[What the Flexbox?!](#)" series of tutorials by Wes Bos. The series is simple to understand and will fill in any gaps in knowledge you have on what I'm about to cover.

The Flexbox (flexible box) display model is a recent answer (along with the grid layout) to a long-standing problem with CSS layout: having block elements that adapt to the available screen size.

In the era of desktop PC-only browsers, this wasn't a huge problem, but as we started using ever smaller devices for our daily tasks, the sheer range of different sizes started to become a big issue in web application design. Flexbox was created to help ease this problem by allowing user interface designs to adapt themselves to different widths and orientations, without resorting to the many JavaScript and server-side rendering techniques developed over the years.

Various CSS rules to control the positioning of elements, such as sticking to the start, sticking to the end, or spacing child elements out over the width of a parent have been added to the CSS specification to manage how the browser responds to layouts. BS4 wraps many of these rule sets in CSS classes, making it simple to just add the class to an element and to achieve the required behavior.

You've already met **d-flex**, which enables Flexbox on a block-level element. **d-inline-flex** does exactly the same thing, but enables Flexbox for inline style elements such as spans. You can further modify each of these two class names to include a breakpoint size of **xs**, **sm**, **lg**, or **xl** in the same manner as you've seen with other classes.

The effect of using the sizing classes is exactly the same as elsewhere. For example, **d-sm-flex** will enable Flexbox on an element only when the screen size is in the small breakpoint range, and will disable it (default) at any other size. Like others, they can be combined, so **d-sm-flex d-lg-inline-flex** will enable block-level Flexbox for small layouts, inline Flexbox for large layouts, and default for anything else.

Flexbox containers can also specify the direction the content is intended to work in. This direction can be **row** or **column**, or **reverse** variations of the two. Adding **flex-row**, **flex-row-reverse**, **flex-column**, or **flex-column-reverse** allows you to specify the direction, and again, like **d-***, you can use the different size modifiers as required to control when it's used.

There are alignment utilities such as **align-items-start**, **align-items-middle**, and **align-items-right**, that are used to ensure that any child elements you have inside a Flexbox container are aligned to a specific position. This extends to also allowing items to be vertically positioned when using column mode.

Wrap classes allow you to move elements to the next line and continue rendering if the child element's width becomes too wide for the display. This is especially useful for designing things like thumbnail pages for photo albums and similar designs.

I'll finish this section on using the Flexbox utilities by showing you an example that will give you what some folks call the "golden layout," with a header, content, a footer, and left and right sidebars.

Code Listing 75: The golden layout, sort of

```
<!-- Body content goes here -->
<style>
  body, html {
    height: 100vh;
    width: 100vw;
  }
</style>
```



```

<div class="container-fluid d-flex flex-column p-0" style="height: 100%;
width: 100%;">
  <div class="bg-info" style="height: 100px;">
    <h1 class="text-dark text-center">Page Title</h1>
  </div>
  <div class="h-100 d-flex flex-row">
    <div class="col-2 m-2 bg-light border">Left Sidebar Area</div>
    <div class="col text-center">Main Page and Content Area</div>
    <div class="col-2 m-2 bg-light border">Right Sidebar Area</div>
  </div>
  <div class="bg-success" style="height: 50px;">
    <div class="container text-white">Page footer information</div>
  </div>
</div>

```

Unfortunately, Code Listing 75 does contain a few style attributes and tags, but these are more for the vertical sizing than anything else. While Flexbox helps us greatly, we still need to resort to using height to specifically fix an element height. More to the point, the height has to bubble all the way back up to the parent elements. None of this would work as expected without setting the HTML and body elements to the size of the browser viewport, as children of those elements would have no idea what 100 percent of the overall size is.

There are a few different ways of achieving this. What I've given you here can certainly be tidied up and done better, but as it stands you can drop it into the template we've been using for the rest of the book. If you do this and render it, you should get something similar to the following figure.

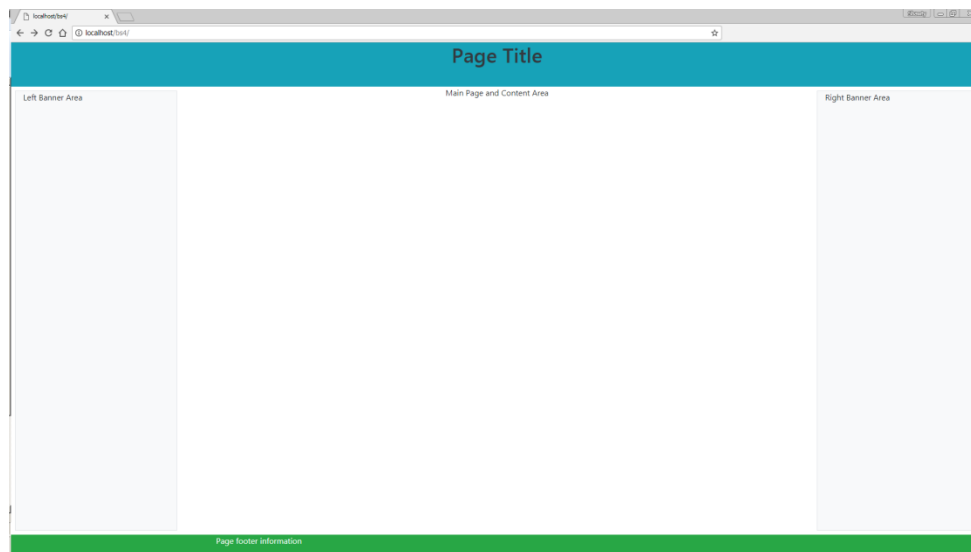


Figure 78: The layout produced by Code Listing 75

Sizing, margins, and padding

Sizing and getting the padding and margins right are an important part of making your designs look balanced. BS4 provides a lot of classes designed to make it easy for you to get this important aspect right.

Width and height have four variations for 25, 50, 75, and 100 percent. To use them, it's simply a case of specifying a **h** or a **w** for height or width, followed by a hyphen and then the percentage number. For example, **w-25** will give you 25 percent width, **h-50** a height of 50 percent, and **w-50 h-25** a width of 50 percent with a height of 25 percent.

If you want a pixel-based specific height, you are going to need to use style attributes as we did for the layout in Code Listing 75. Remember that percentage heights are relative to the parent, so 100 percent will be all of the space available provided by the element above the one you're sizing. If the parent is 200 pixels, then 25 percent will be 50 pixels, 50 percent will be 100 pixels, 75 percent will be 150 pixels, and 100 percent will be the full 200. If you don't get the parent height sized correctly, you'll get some strange sizing errors. Take this from someone who's spent more time than he would like tracking these things down—you really don't want them.

The padding and margin classes have a fairly straightforward notation once you've seen a few examples. The [BS4 docs](#) show the notation as **{property}{sides}-{size}**. **property** is the thing you want to set, and is **m** for margin and **p** for padding.

{sides} are the sides you want to apply the property to, and **{size}** is the sizing factor. If you don't specify sides, then the sizing is applied to all four sides of the element.

{sides} can be **t**, **b**, **l**, or **r** for top, bottom, left, and right, or they can be **x** for left and right and **y** for top and bottom.

{size} can be **1**, **2**, **3**, **4**, or **5**, with **1** being the default BS4 spacing value multiplied by 0.25 and **5** as the default being multiplied by 3. The multipliers for **2**, **3**, and **4** are 0.5, 1, and 1.5.

Lastly, you can specify **auto** to assign automatic sizing on the property you're setting.

So, if you want your element to have the BS4 default spacing around all four sides as a margin, you would use **m-3**. If you wanted your element to have a quarter of the default spacing on the left, but three times the default on the right, and for it to be padding, you would use **p1-1 pr-3**.

If you want an element to be centered with equal margins, then **mx-auto** will provide an automatically sized margin to the left and right of an element simultaneously.

There's more

You have utilities for close and arrow icons, and embedded content such as videos and iframes. There are float controls, clearfixes, image replacements, and more. We just don't have the space to cover all of them in this book. The [BS4 docs](#) are the definitive source of anything we haven't covered.

The Final Word

And so we've come to the end of this edition of what's now a three-book series, and it seems that with each one I've written, I've had to leave more and more out to make sure I can fit it into the page limit I have. I do hope, however, that you've gotten your feet wet and want to continue to explore Bootstrap's latest version.

Bootstrap is not the most-used CSS framework, but it enjoys the success it has because it helps us mere developers look much better at styling and design than we really are.

As Bootstrap continues to mature, its feature set becomes ever more impressive, but learning and staying up to date with it becomes more and more important. There is an absolute plethora of resources available around the internet, covering everything from premade Bootstrap templates to small snippets of code that demonstrate common tasks.

I have a list of bookmarks that overflows my vertical screen space on a 1280 × 1024 resolution when fully expanded. I'm not going to list any of them here, as they all deserve to have equal attention, and a simple Google search for "bootstrap snippets" or "bootstrap templates" will give you more than enough to entertain you for a week or more.

For now, however, it's time for me to end this book. As always, you can find me on Twitter as [@shawty_ds](#) should you want to reach out to me. This book, along with well over 100 other titles, can all be downloaded from the [SynCFusion Succinctly ebook library](#). Thanks for sticking with me on this journey, and hopefully we'll cross paths again in the future.