



# Twitter Bootstrap 3

**Succinctly**

by Peter Shaw

# Twitter Bootstrap 3 Succinctly

---

By

Peter Shaw

Foreword by Daniel Jebaraj



Copyright © 2014 by Syncfusion Inc.

2501 Aerial Center Parkway

Suite 200

Morrisville, NC 27560

USA

All rights reserved.

## **I** mportant licensing information. Please read.

This book is available for free download from [www.syncfusion.com](http://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](http://www.syncfusion.com).

This book is licensed for reading only if obtained from [www.syncfusion.com](http://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:** Zoran Maksimovic, [@zoranmax](https://twitter.com/zoranmax), [www.agile-code.com](http://www.agile-code.com)

**Copy Editor:** Courtney Wright

**Acquisitions Coordinator:** Hillary Bowling, marketing coordinator, Syncfusion, Inc.

**Proofreader:** Darren West, content producer, Syncfusion, Inc.

# Table of Contents

<b>The Story behind the <i>Succinctly</i> Series of Books</b> .....	<b>6</b>
<b>About the Author</b> .....	<b>8</b>
<b>Introduction</b> .....	<b>9</b>
What's Changed.....	9
New Installation Methods.....	11
Device Support .....	12
<b>Chapter 1 Migrating from Version 2 to Version 3</b> .....	<b>14</b>
Class Changes.....	14
Migrating the Grid System .....	15
Other Migrations .....	20
<b>Chapter 2 Common Pitfalls</b> .....	<b>23</b>
Internet Explorer Backwards Compatibility Modes .....	23
Internet Explorer 10 Device Viewport .....	24
Safari Percent Rounding.....	25
Android Stock Browser .....	25
And the Rest? .....	25
<b>Chapter 3 Changed CSS Features</b> .....	<b>27</b>
Typography Changes.....	27
List Changes .....	29
Table Changes.....	29
Form Changes .....	34

Output generated by code sample 18.....	34
Button Changes .....	43
Image Changes.....	46
Helper and Visibility Changes.....	47
<b>Chapter 4 Changed Components Features .....</b>	<b>55</b>
Glyphicon Changes.....	55
Button Changes .....	56
Input Group Changes.....	63
Navigation Changes.....	65
Basic Navigation .....	66
Navbar Navigation .....	71
Label and Badge Changes .....	77
List Group Changes .....	80
Media Objects and Custom Thumbnail Changes .....	84
Panel Changes .....	89
Other changes .....	92
<b>Chapter 5 Changed JavaScript Features .....</b>	<b>96</b>
Modals.....	96
Tabs .....	99
Tooltips and Popovers .....	100
Collapsible Panels .....	102
Carousel.....	103
<b>Chapter 6 Customizing Bootstrap 3.....</b>	<b>106</b>
<b>Closing Notes .....</b>	<b>110</b>

# The Story behind the *Succinctly* Series of Books

Daniel Jebaraj, Vice President  
Syncfusion, Inc.

**S**taying 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](mailto:succinctly-series@syncfusion.com).

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

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



# About the Author

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

In time this small, 1k machine became a 16k Tandy TRS-80, followed by an Acorn Electron, and eventually, a 4mb arm powered Acorn A5000.

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

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

With over 30 years of experience in the IT industry within varied platforms and operating systems, I have a lot of knowledge to share.

You can often find me hanging around in the Lidnug.NET users group on LinkedIn that I help run, and you can easily find me in the usual places such as Stack-Overflow (and its GIS-specific board) and in twitter (@shawty\_ds), and now also on Pluralsight, where my various videos are available.

I hope you enjoy the book, and gain something from it.

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



# Introduction

Welcome to the second book in the *Succinctly* series that covers the Twitter Bootstrap (BS) UI and CSS framework.

In my first book, I laid down the initial groundwork and got you started with Bootstrap 2.

Since that book was released, however, Bootstrap 3 has become the mainstream version.

In this book we're going to continue to build on that groundwork and move forward into using BS v3. We'll look to see where things are different, and as we explore the new features, we'll see that a big chunk of what was present in v2 is still applicable to v3.

If you've not yet read the first book on Bootstrap 2, then I encourage you to do so, as I will be referring to it at various points throughout this book.

The style and layout in this book will also be slightly different from the first one, in that it will be more of an extension to the first book rather than an independent book in its own right.

Why? I believe that in order to understand the entire Bootstrap landscape, you need to examine it from the beginning. You need to be able to understand what it set out to achieve and how.

You will be able to pick up just this book and learn the basics of BS v3, but you'll get a much deeper, better understanding if you read the v2 book first.

## What's Changed

So what's changed from v2 to v3?

Quite a lot.

The major change between the two versions is that v3 is now "Mobile First." Bootstrap v2 was a responsive layout CSS kit, but its mobile and responsive features were always second place to its rich UI features. In fact, in order to make the responsive stuff work correctly, you had to include a second CSS file whose sole purpose was to enable the responsive, mobile features and nothing else.

In v3 this whole situation has been completely reversed. The entire framework is now mobile-friendly and responsive out of the box, and it now takes extra work to adapt your layouts for larger screen formats. Don't get me wrong—it's by no means a huge amount of work. Most of what you need to change is still just simply swapping classes about and structuring your HTML mark-up correctly.

The other major change is in the naming of classes and API calls. Many of the class names that were introduced in v2 are now either deprecated or have been renamed to something more suitable to their intended purpose.

There has also been a major effort to rename classes to be more consistent. For example, in v2, for items that targeted the RED error color, we had the following classes:

- Buttons - **btn-danger**
- Text - **text-error**
- Table Rows - **tr.error**
- Labels - **.important**
- Badges - **.important**
- Alerts - **.error**
- Progress Bars - **progress-danger**

Now, in v3, these have been consolidated so that naming is similar across all components as follows:

- Buttons - **btn-danger**
- Text - **text-danger**
- Table Rows - **.danger**
- Label, Badge, Alert - **.danger**
- Progress Bars - **progress-bar-danger**

As you can see, consistency is now a major player in the v3 classes, and many other similar changes have been made across the entire framework.

There have been quite a few minor changes too. For example, the box-model used by Bootstrap has now been improved considerably, with all elements now using **border-box** as the default CSS box sizing model.

The grid system has been extended and improved too, and instead of being one monolithic grid system with optional classes, it's now comprised of four tiers of grid sizes specifically aimed at phones, tablets, desktops, and large desktops.

All the JavaScript stuff has also now been name-spaced to reduce conflict with other JavaScript code; the available events are now better named to reflect their purpose and are much more clearly documented.

**Modals** and **Navbars** have been vastly improved in terms of responsiveness, and along with the class-naming changes, the sizing classes for all of the components (Inc Nav Modals) have now been aligned.

Component-wise, some of the older, less frequently used components have been deprecated and removed, the most notable of which is the **Accordion** component. But don't despair—the accordion has been replaced with a brand new **collapsible-panel** component that's much more flexible than its predecessor.

We also have a new, narrow **jumbotron**, new **panel** types, list groups, and much more.

Finally, the one change that EVERYONE will notice is the look and feel: **TWB V3 is flat**. It has a single colored, new interface, but with rounded corners flat.

The hover classes no longer have nice graduated effects in them, and the progress bars and buttons no longer look semi-3D as they did in v2. Instead, what the maintainers of Twitter Bootstrap have decided to do is to make it easier to customize the look and feel of the elements that are in the CSS.

The maintainers have even made available a “Bootstrap” theme that makes v3 look like the original v2 design to get you started. In the last chapter of this book, I'll show you how simple it now is to override the various classes, and show you how to overhaul the flat theme to take on your own look and feel.

For now though, if you want the v2 experience, you'll need to make sure that you also include the appropriate CSS file as required. If you want to stick with the v3 default, you don't need anything extra.

## New Installation Methods

When BS2 was first released, the only way of getting it was via a download from the project website. This was covered extensively in the first book with an in-depth discussion of exactly which files were in the zip file and why.

Because so many people were using Bootstrap, it didn't take long before it was made available for free on a CDN by the folks who run MaxCDN. This tradition has continued into the v3 code base, and you can now simply get your chosen standard v3 installation by using the following HTML script tags:

```
<link rel="stylesheet"
href="//netdna.bootstrapcdn.com/bootstrap/3.1.1/css/bootstrap.min.css">

<link rel="stylesheet" href="//netdna.bootstrapcdn.com/bootstrap/3.1.1/css/bootstrap-
theme.min.css">

<script src="//netdna.bootstrapcdn.com/bootstrap/3.1.1/js/bootstrap.min.js"></script>
```

If you use the CDN versions, please remember that you will also need to make sure you include a recent copy of JQuery; none of the BS3 JS functionality will work unless you do.

As well as the CDN link, there is now also a direct download on the [getbootstrap.com](http://getbootstrap.com) site, not only for the standard JavaScript & CSS bundle as described in the previous book, but also a direct link to a zip file of the original “Less” sources and a conversion to “Sass” for those folks who would like to be able to include a fully variable-based, customizable version of the kit in your own build system using tools like Grunt to automate things.

In addition to the new sources, you can now also use Bower to install everything you need using the following from your project folders command line:

```
bower install bootstrap
```

Bower is an HTML scaffolding and package system designed to automate much of the application boiler plate. You don't need to understand it to understand BS v3, but a quick Google search for "Bower" will turn up many references to deepen your understanding.

On a similar note, BS v3 can also be installed using the NuGet package manager if you're working in Visual Studio. Like Bower, this will automate much of the process for installing and setting up the required files. Be careful though—there are many Bootstrap packages available in NuGet, some of which will cause you more work than they save you.

Finally, the TWB customizer is still available at <http://getbootstrap.com/customize/>. It's been greatly restructured and rebuilt to provide more options and an easier, more intuitive look at what you're changing and why.

In fact, it's now so easy to change things using the customizer that your designer or design team can do 90 percent of the work needed to set the colors and branding of your download before the files required are even downloaded. This enables your design department to start making color and layout choices immediately, while you work on the page layouts and additional code for the required interactivity. Then, all that's required is for you to simply replace your default files with the files provided to you by your designers, and an instant theme change should occur.

## Device Support

Officially, BS v3 only supports the latest round of HTML 5-compliant browsers and Internet Explorer 10 and above.

In reality, the framework also supports IE8 and IE9; however, there are some features of CSS3 and HTML5 that don't quite work as expected. In particular, if you want the responsive elements to work correctly, you'll need to use **respond.js**, which you can download from the following GitHub repository: <https://github.com/scottjehl/Respond>.

Once you add **respond** to your project, it's basically just a general expectation to what was and was not added to the various browsers, at various times.

The parts that will be most noticed as missing are as follows:

For IE8:

**border-radius** is NOT supported

**box-shadow** is NOT supported

**transform** is NOT supported

**transition** is NOT supported

**placeholder** text is NOT supported

This essentially means that placeholder text in input elements won't show, and anything that uses rounded corners or drop shadows, or has any kind of transition or transformation on the element, won't display correctly.

For IE9:

**border-radius** IS supported

**box-shadow** IS supported

**transform** IS supported (but only with **-ms** prefix)

**transition** is NOT supported

**placeholder** text is NOT supported

Things are slightly improved where IE9 is concerned—rounded corners and drop shadows are now fine, and transformations will also work, as long as they also have a **-ms** prefix version.

Transitions and placeholders on the input elements, however, are still sadly missing.

The official support matrix for the current version of BS3 in current browsers looks like this:

**Table 1: Official support matrix**

	Chrome	Firefox	IE	Opera	Safari
Android	Yes	No		No	
iOS	Yes			No	Yes
Mac OS X	Yes	Yes		Yes	Yes
Windows	Yes	Yes	Yes	Yes	No

If you are targeting IE9 and IE8, and are using **respond.js** to support those efforts, then please be aware of the following points:

You will need to refer to the **respond.js** docs if you're hosting CSS, etc. on a different domain (for example a CDN) to mitigate cross-domain problems.

Browser security will cause you problems with **file://** and **@import** based resource references.

Specific to **@import** is that **respond.js** cannot read the rules properly, which is important to Drupal users, as Drupal uses **@import** quite heavily.

Older IE compatibility modes will stop Bootstrap from working completely, not just with **respond.js**, so be careful if you're testing for backwards compatibility using a modern IE in emulation mode—the results will most likely NOT be what you expect.

# Chapter 1 Migrating from Version 2 to Version 3

So what's involved in migrating from Bootstrap 2 to Bootstrap 3? In truth, not a great deal.

Despite the many changes, there's still not a huge amount for you to actually change, and the changes you do need to make are generally just class renames where applicable.

One of the things you might want to do, especially if you've been using BS only for general web app development and not mobile or any kind of responsive design, is to disable the responsive features in BS3.

This is easy enough to do, but not at all recommended.

You can achieve this as follows:

- Do not add the `meta` tag containing the device width and other initial sizing info to the head of your document.
- Do override the `width` on your elements that are marked up with a class of `container`, and make sure you use `style= 'width: xxx !important'` when you do so.
- Do make sure that any width overrides are processed AFTER the main Bootstrap CSS rules have been loaded.
- Do remove ALL collapsing and expanding behaviors and rules from ALL `navbar` constructs within your document.
- Do change all grid layout classes to use ONLY `col-xs-*` classes and none of the other four levels.

If you're targeting IE8 and IE9, you will still need to make sure you use `respond.js`, even if you do disable responsiveness as outlined.

## Class Changes

As I mentioned earlier, there have been many class name changes between the two versions, and many classes have been deprecated and withdrawn.

One thing that will (and already has if you look at Stack Overflow) come as a surprise to many is the withdrawal of the fluid width classes.

In version 2, if you wanted a full-width elastic container, then you had to do something like the following:

Code Sample 1: Version 2 Flexible Container

```
<div class="container-fluid" id="myParentContainer">
  <div class="row-fluid" id="mycontentrow">
    <h1>A headline</h1>
    <p>Some paragraph text</p>
  </div>
</div>
```

In version 3 the **container** and **row-fluid** classes no longer exist.

So how do you get a fluid container? Simple: you don't.

Rather than wrap your contents in a **container** and then a **row**, you simply don't wrap them in anything.

You can still use the grid system to provide enclosing containers for your content, so that things line up nicely with Bootstrap's grid, but you no longer need to put a container around those collections of **<div>** elements before you use them.

In fact, if you use **container** and **row** (the non-fluid versions still exist) then you'll end up with all your content being in the 1024-pixel, central column automatically, and be able to use the entire page width if you do not.

## Migrating the Grid System

Then next biggest class change is the grid system itself.

In version 2 you typically created grids in the following manner:

Code Sample 2: Version 2 Grid Classes

```
<div class="container">
  <div class="span2">Content here</div>
  <div class="span10">Content here</div>
</div>
```

This code would give you two containers that neatly filled the 12 grid squares horizontally that all layouts had (typically a side bar).

In version 3, the "medium level" grid is now the equivalent of the v2 **span** classes, so to rewrite the previous code for V3 you simply do the following:

Code Sample 3: Version 3 Grid Classes Equivalent to 'Span'

```
<div class="container">
  <div class="col-md-2">Content here</div>
```

```
<div class="col-md-10">Content here</div>
</div>
```

However, whereas version 2 had only one level of grid size, version 3 now has four levels. Each level is tailored for the expected main target device that you anticipate your end product will be running on.

These grid units are now named as follows:

Extra small devices: **col-xs-\***

Small devices: **col-sm-\***

Medium devices: **col-md-\***

Large devices: **col-lg-\***

Media queries are used internally for BS3 to decide just which of the aforementioned grid classes to use, and the different sizes are defined as follows:

Extra small: display width less than 768 pixels

Small: display width greater than or equal to 768 pixels, or less than 992 pixels

Medium: display width greater than or equal to 992 pixels, or less than 1,200 pixels

Large: display width greater than or equal to 1,200 pixels

You can code up multiple versions of your grid for BS3 to decide which type to use when targeting multiple displays. For example if you did the following:

*Code Sample 4: Multiple Grid Size Declarations*

```
<div class="container">
  <div class="col-xs-2">Content here</div>
  <div class="col-xs-10">Content here</div>
  <div class="col-sm-2">Content here</div>
  <div class="col-sm-10">Content here</div>
  <div class="col-md-2">Content here</div>
  <div class="col-md-10">Content here</div>
  <div class="col-lg-2">Content here</div>
  <div class="col-lg-10">Content here</div>
</div>
```



BS3 will hide and unhide the containers as required, depending on the width of the device display and the operation of the media queries.

As with previous versions of the grid system, there are 12 columns horizontally across all the different sizes, so whichever grid size is displayed, you will always still get 12 grids on every device.

The column width itself does change, however, so you may need to plan the content in those grids to take advantage of the differing sizes. The sizes for each of them are as follows:

**col-xs-\*** = Auto sizing, no fixed dimensions

**col-sm-\*** = 60 pixels

**col-md-\*** = 78 pixels

**col-lg-\*** = 95 pixels

The gutter margin in all cases will remain at 15 pixels on each side of the grid container, giving an overall gutter of 30 pixels. This size will be consistent no matter which grid size level you're using.

Nesting and offsets work as they did previously, but as with the grids themselves, by way of a slight renaming of the actual classes used.

To apply an offset, simply use **col-md-offset-\***, remembering to replace the **md** with **xs**, **sm**, or **lg** as needed, depending on your grid size.

Nesting is done simply by nesting containers under control of the **col-xx-\*** classes inside each other, where they will resize and behave as they did in previous BS versions.

The following examples show the correct way to achieve both of these techniques:

*Code Sample 5: Nested Grids in Version 3*

```
<div class="col-md-9">
  Level 1: .col-md-9
  <br/>
  <div class="col-md-6">
    Level 2: .col-md-6
  </div>
  <div class="col-md-6">
    Level 2: .col-md-6
  </div>
</div>
```

This example will give you a grid that looks like the following:

Level 1: col-md-9

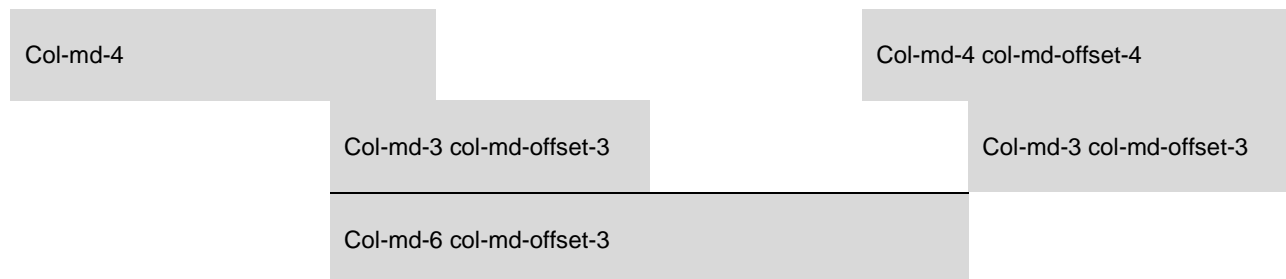
Level 2: col-md-6

Level 2: col-md-6

Code Sample 6: Offset Grids in Version 3

```
<div class="col-md-4">
  .col-md-4
</div>
<div class="col-md-4 col-md-offset-4">
  .col-md-4 .col-md-offset-4
</div>
<div class="col-md-3 col-md-offset-3">
  .col-md-3 .col-md-offset-3
</div>
<div class="col-md-3 col-md-offset-3">
  .col-md-3 .col-md-offset-3
</div>
<div class="col-md-6 col-md-offset-3">
  .col-md-6 .col-md-offset-3
</div>
```

This example will give you a layout as follows:



BS3 also brings something new to the table when it comes to offsetting and nesting, and that's something called column ordering.

If you want your columns to be presented in a different order to how you define them in your HTML document, then you can use the new **col-xx-pull-\*** and **col-xx-push-\*** classes to push or pull your grid layouts into the order you want them. For example:

```
<div class="col-md-8">8 columns of content</div>
```

```
<div class="col-md-4">4 columns of content</div>
```

If you render those in your document, as expected, you'll get the following:

8 columns of content

4 columns of content

If, however, you modify the above code to add push and pull modifiers as follows:

```
<div class="col-md-8 col-md-push-4">8 columns of content</div>  
<div class="col-md-4 col-md-pull-8">4 columns of content</div>
```

When you render your document, you should see your layout change as follows:

4 columns of content

8 columns of content

Finally, if you're using the *Less* CSS source versions of Bootstrap, you have complete control over the grid sizes by changing the following variables:

**@grid-columns:** controls the number of grids horizontally (default 12)

**@grid-gutter-width:** the total margin around each grid (default 30 pixels)

**@grid-float-breakpoint:** the minimum size below which we have “extra small” devices (default 768 pixels)

So now that we have the new grid system under control, is there anything else you need to know?

The more astute of you may be thinking, “But that's crazy—with all those multiple sets of `<div>` elements and offsets with `col-xx-xx` classes, all for different size displays, I might as well just create four different sites, with four different resolutions in mind!” To be honest, I wouldn't blame you, except for one thing: each of these new layout size levels are designed to work on the same markup, at the same time, and occupy the same space.

Let's take the code in the previous code sample 4, and rewrite it to do this the recommended way:

*Code Sample 7: The Recommended Way of Providing Multi-Resolution Layouts*

```
<div class="container">  
  <div class="col-xs-2 col-sm-2 col-md-2 col-lg-2">Content here</div>  
  <div class="col-xs-10 col-sm-10 col-md-10 col-lg-10">Content here</div>  
</div>
```

Ok, so you might end up with the class list from hell on your elements, but one set of markup will adapt to all display sizes, and resize itself where needed.

This also works with the various offset, order, and nesting classes.

## Other Migrations

In addition to those we've already discussed, the following class names also need to be changed if you're migrating from a V2 layout to a V3 layout (Note: the following table has been taken directly from the Bootstrap 3 docs and was correct at the time of writing. As Bootstrap matures, however, this may not remain so).

Table 2: V3 class name changes

Bootstrap version 2 class name	Bootstrap version 3 class name
.row-fluid	.row
.span*	.col-md-*
.offset*	.col-md-offset-*
.brand	.navbar-brand
.nav-collapse	.navbar-collapse
.nav-toggle	.navbar-toggle
.btn-navbar	.navbar-btn
.hero-unit	.jumbotron
.icon-*	.glyphicon .glyphicon-*
.btn	.btn .btn-default
.btn-mini	.btn-xs
.btn-small	.btn-sm
.btn-large	.btn-lg
.alert-error	.alert-danger
.visible-phone	.visible-xs
.visible-tablet	.visible-sm
.visible-desktop	<b>Split into</b> .visible-md .visible-lg
.hidden-phone	.hidden-xs
.hidden-tablet	.hidden-sm
.hidden-desktop	<b>Split into</b> .hidden-md .hidden-lg
.input-block-level	.form-control
.control-group	.form-group

Bootstrap version 2 class name	Bootstrap version 3 class name
.control-group.warning .control-group.error .control-group.success	.form-group.has-*
.checkbox.inline .radio.inline	.checkbox-inline .radio-inline
.input-prepend .input-append	.input-group
.add-on	.input-group-addon
.img-polaroid	.img-thumbnail
ul.unstyled	.list-unstyled
ul.inline	.list-inline
.muted	.text-muted
.label	.label .label-default
.label-important	.label-danger
.text-error	.text-danger
.table .error	.table .danger
.bar	.progress-bar
.bar-*	.progress-bar-*
.accordion	.panel-group
.accordion-group	.panel .panel-default
.accordion-heading	.panel-heading
.accordion-body	.panel-collapse
.accordion-inner	.panel-body

As previously mentioned, most of the changes have been made to bring conformity to the naming scheme used by the various classes. But many of them have also been renamed because their overall purpose has changed.

We'll go into in more detail in upcoming chapters in this book, but for now, if you're doing a conversion, then Table 2 will tell you everything you need in order to retarget a v2 layout to v3.

You might want to consider using a custom job in something like a **Grunt.js**<sup>1</sup> task, so that when you run your build system, these changes are performed automatically. This will allow your developers to remain productive using v2 while gradually making the move to v3.

So what exactly has been added to Bootstrap that's new, and what exactly has been removed?

We'll start with what's been removed, and we'll cover what's been added in more detail in Chapter 3, "Changed CSS features." It's more important that you know what's been removed in this chapter, since this is the chapter you're likely to be referring to when migrating your layouts.

---

<sup>1</sup> <http://gruntjs.com>

First we'll start with what's been removed where forms are concerned, and unfortunately, that's quite a lot. We no longer have a specific type for a search form **form-search**, and the shaded bar typically found at the foot of a form **form-actions** has also been deprecated in v3.

Also gone is the class typically used to display validation errors, **control-group-info**, and its help counterpart, **help-inline**. None of these four classes have any recommended replacement in the v3 code base, meaning that to construct equivalents of them, you will need to use other elements and classes where applicable.

Continuing with forms, the main **controls** class used to wrap entire control sets is gone, along with **controls-row**. Instead, you are advised to use **row** or the new **form-group** class. Forms have also lost most of the sizing classes; the fixed-size classes such as **input-mini**, **input-small**, **input-medium**, **input-large**, **input-xlarge**, and **input-xxlarge** have now all gone away, along with the block level control class **input-block-level**. Instead, you are now advised to control your form element sizes using **form-control** in combination with the new sizes and layouts available in the grid system.

From an individual control point of view, the **inverse** classes have been removed from buttons and other similar controls, and we've also lost the **dropdown-submenu** class in favor of just using split drop-down buttons to create the same functionality.

For tabs, the **tabs-left**, **tabs-right**, and **tabs-below** classes no longer exist, which means we now only have the ability to put tabs at the top of the content, left-aligned.

Staying with tabs, the class to work with content in a pill-based tab setup has also been removed, meaning that **pill-pane** & **pill-content** should now use the general **tab-content** & **tab-pane** classes.

Finally, the various **navbar** classes are not without casualties: **navbar-inner**, **navbar-divider-vertical**, **nav-list**, and **nav-header** are no longer part of the framework.

In most cases, there are no direct equivalents in v3 for these classes, although there are some similarities in other classes that may prove useful. For example, **nav-list** and **nav-header** can be recreated using **List** groups.

There's a quick reference chart to all of these in the migration guide on the Bootstrap 3 website, which can be found at: <http://getbootstrap.com/migration>

# Chapter 2 Common Pitfalls

While researching for this book, I've come across a number of potential problems, all of which are going to cause you some grief—either with the transition from v2 to v3, or even just when jumping straight in to v3.

A few of these are applicable to v2 as well, but in general I'm including them here because they bit me and left me scratching my head in a lot of cases.

## Internet Explorer Backwards Compatibility Modes

This one hurt, really hurt. For about a week and a half, I was going around in circles, trying to figure out why my nicely crafted layout was not displaying as expected in the latest version of IE11.

It turned out that I had a malformed `meta` device tag.

As you may recall from the BS2 book, there's a basic template that's recommended as the starting point for all sites based on the bootstrap framework. It looks something like this:

*Code Sample 8: Bootstrap HTML Basic Template*

```
<!DOCTYPE html>
<html>

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

  <body>

    <!-- document code goes here -->

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

  </body>
</html>
```

If you notice in the `meta` tag just inside the `head` element, you'll see that we have compatibility keys in there to allow the modern IE rendering engine to know what is and is not supported when trying to render things in a backward compatible way.

In BS3, none of this works correctly. Instead, the recommended method is to remove all your content type keys, leaving ONLY `IE=EDGE`, and no others.

The best way to do this is to create a layout that works perfectly in an HTML5-standard way, include the various versions, then either force the IE debugger to adopt a specific version, or do something to the document source that will force IE to attempt to render the content in a way that it would under IE9 or earlier.

Over and over again, I looked at the `meta` tag, not realizing that it was the cause of my problems. So if you get any weird rendering errors in IE11, have a look in the debugger and see just what mode IE believes it should be displaying your page in.

## Internet Explorer 10 Device Viewport

Yes folks, another IE-related problem. IE10 can't tell the difference between `device width` and `viewport width`; the result of this is that IE10 gets its CSS media queries wrong a lot of the time (not just in Bootstrap, but other frameworks too). The fix is simple enough: add a dummy CSS rule to your site-wide CSS styles that looks like this:

```
@-ms-viewport { width: device-width; }
```

That generally fixes things, except in one case. With Windows Phone versions earlier than Update 3, the device will not interpret things correctly and put the page into desktop view. To fix this, the following CSS rules and JavaScript code are needed:

*Code Sample 9: Windows Phone 8 IE Fix*

```
@-webkit-viewport { width: device-width; }
@-moz-viewport { width: device-width; }
@-ms-viewport { width: device-width; }
@-o-viewport { width: device-width; }
@viewport { width: device-width; }

if (navigator.userAgent.match(/IEMobile\/10\.0/))
{
  var msViewportStyle =
    document.createElement('style') msViewportStyle
    .appendChild(
      document.createTextNode( '@-ms-viewport{width:auto!important}' ) )
    document.querySelector('head').appendChild(msViewportStyle)
}
```



I can't take credit for this fix however—it's clearly detailed in the BS3 documentation online. There is also more information on the subject in the Windows 8 developer guidelines.

## Safari Percent Rounding

In some versions of Safari, the rendering engine struggles with the number of decimal places in percentage values.

These percentages are used often in the `col-*-1` grid classes, and as a result, you'll see errors in the rendering of 12-column layouts when this is encountered.

There is a bug open in the BS3 bug report system, but there's little they can do to resolve it. The BS3 docs do suggest trying to add a **pull-right** to your last column, but the best course of resolution seems to be manual tweaking of your percentage-based values until a balance is found.

## Android Stock Browser

At this time, Android versions 4.1 and above ship with the “Browser” app as the default web browser. BS3 (and many others) fail to render correctly in the Browser app due to the large number of problems in the browser's code base, and more so in the CSS engine where there are a large number of known problems.

There is a JavaScript-based solution to patch your layouts in the BS3 docs, but the best resolution is for the user to use the Chrome app instead, which is by far a better and more stable browser for Android in general.

## And the Rest?

There are quite a few more things to be aware of, and again, most of these are documented in the BS3 docs and cover things like pinch-based zooming, virtual keyboards, and how different types of view ports react where media queries are concerned.

In fact, a quick scan of the most common issues on [www.stackoverflow.com](http://www.stackoverflow.com) tells us that a great many of the problems revolve around scrolling, resizing, zooming, and general touch screen-based issues that seem to stem from either things being too small, or not being sensitive enough for average finger sizes.

Many of these size issues can be resolved by downloading the BS3 source distribution and either altering the **Less** variables and mixins available, or by customizing things using the customization tools available; it's now no longer a good idea to just download and use BS3 unless you're only targeting desktop apps.

If you're targeting multiple platforms and expecting full responsiveness, then you really need to be doing a lot of customization in the hooks provided by the framework authors.

# Chapter 3 Changed CSS Features

So far you've already seen a number of the new CSS features that are available in version 3 of Bootstrap. In this chapter we'll go into a little more detail about just what is considered new and what's not.

In many cases these new classes are just renames of the old ones, but we treat them as new here so that you can easily make the distinction.

## Typography Changes

The various classes that make up BS3's typography section haven't changed as much as some of the other elements. Tags **H1** through **H6** are still treated the same way as they were in v2, with the addition that you can now use `<small>` in line with any header element without it first having to be wrapped in a `div` using the `page-header` class. This means no extra markup now, unless you want your block heading to be underlined with a different paragraph margin.

*Code Sample 10: BS V2 Page Header*

```
<div class="page-header">
  <h1>This is my super web page <small>It's the best there is</small></h1>
</div>
```

In v2 you had no choice but to wrap your **H1** in a `page-header` div, as this was the only way the small tags output would be neatly lined up. This has now been rectified and also applied to all levels of header, rather than just the first three.

*Code Sample 11: BS V3 Page Header*

```
<h1>This is my super web page <small>it's the best there is</small></h1>
```

Continuing on, the standard body copy class has no changes, and remains at a default size of 14 pixels with a line height of about 1.4.

Body copy is applied to all text inside a paragraph tag automatically, so no classes are needed unless you want to use some of the special features (as we'll see soon).

Lead body copy (paragraphs with the class name of `lead`) also have no changes to their names or styling, and as with v2, are designed to give your opening paragraph a bit more emphasis than the other regular body copy. Combining these three CSS rules, you might have something like the following:

```
<div class="container">
  <div class="page-header">
    <h1>This is my super webpage <small>It's the best there is</small></h1>
  </div>

  <p class="lead">Welcome to my super-duper webpage, there's no other webpage like it
in the whole world, my page is the best thing on the Internet that you should
visit</p>
  <p>On this fantastic page I have text and some more text, and there's even some
text for you to read, as well as a nice looking page title</p>
</div>
```

When rendered in your browser, it should look like the following:

# This is my super webpage It's the best there is

---

Welcome to my super-duper webpage, there's no other webpage like it in the whole world, my page is the best thing on the Internet that you should visit

On this fantastic page I have text and some more text, and there's even some text for you to read, as well as a nice looking page title

Figure 1: The output from code sample 12

The `<small>` tag can now also be used on its own too, as its styling is now correctly handled and applied in the context in which it's used, so its styling will follow its position in the document no matter what the parent tag.

Similarly, bold text is still created using the `<strong>` tag and italics using the `<em>` tag; as with many of the typography classes, this is no different to the v2 framework.

The alignment classes also maintain their same class names of `text-left`, `text-center`, `text-right`, and `text-justify`, and still perform the same functions as described in the BS2 book that precedes this one.

Likewise, abbreviations are still created using the `<abbr>` tag with the `title` attribute acting as the full description of the abbreviation. There is one new class, `initialism`, that can be added to an abbreviation tag and gives the rendered output a slightly smaller look and feel than the surrounding text.

Addresses (using the `<address>` tag) and block-quotes (using the `<blockquote>` tag) also have no changes in the CSS or base rules between v2 and v3 of the framework.

The final few tags that remain in the typographic category include `<code>`, used to create an inline code sample. Again, this has not changed in any way, with its intended use still being for code samples that sit in line with regular body text.

For code samples (or anything that is plain text) that must remain formatted as per indentation and carriage returns, you should still use the `<pre>` tag; again the styling here has not changed from v2, and layout using this tag should still behave as expected.

The final typographic element is the addition of a new element called `<kbd>`.

The purpose of this new tag is to display text in a way that indicates the user should enter the information into the computer in some way, generally by typing it.

For example:

*Code Sample 13: '<kbd>' Tag Example*

```
<p>Open up a command prompt by typing <kbd>cmd</kbd> into the box and clicking on the button market 'Run', when it opens type in <kbd>myprogram</kbd> and press enter, at which point the app should run</p>
```

Which when rendered in an HTML document, should look something like the following:

Open up a command prompt by typing `cmd` into the box and clicking on the button market 'Run', when it opens type in `myprogram` and press enter, at which point the app should run

*Figure 2: Output produced by code sample 13*

## List Changes

In general, the normal list elements made up of `<ul>`, `<ol>`, and `<dl>` elements have not changed; the layout is still as it was in BS2, with no extra classes being needed. Note also that the `list-unstyled`, `list-inline` along with the `dl-horizontal` classes for definition lists also remain unchanged in BS3, and have the same behavior as in BS2.

There are some changes in the list elements however, but since these occur with the specialist classes used to create menus and navigation lists, we'll be covering those when we address the changes to the navigation elements in the next chapter.

## Table Changes

Tables are still styled just as they were in BS2 by creating a standard `<table>` arrangement, then adding a `table` class to the markup. As in BS2, tables should always be constructed using the full range of `<table>`, `<thead>`, and `<tbody>` HTML elements as the following example shows:

*Code Sample 14: Marking Up a Table to Be Used by Bootstrap*

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

```

<tr>
  <th>Col A</th>
  <th>Col B</th>
  <th>Col C</th>
</tr>
</thead>
<tbody>
  <tr>
    <td>Val A</td>
    <td>Val B</td>
    <td>Val C</td>
  </tr>
  <tr>
    <td>Val A</td>
    <td>Val B</td>
    <td>Val C</td>
  </tr>
</tbody>
</table>

```

This markup should give you the following:

Col A	Col B	Col C
Val A	Val B	Val C
Val A	Val B	Val C

Figure 3: Output produced by code sample 14

As with BS2, the classes to add the optional styles to a table, **table-striped**, **table-bordered**, **table-hover**, and **table-condensed** work exactly the same as in BS2; these extra classes are added as secondary classes to the main table class on the table element itself.

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

This code, for example, will give you a table that has an outer border and alternating colors on each table row.

One thing to note however, is that table striping now uses the **:nth-child** pseudo selector, which means that it will no longer work in IE8 or earlier without a poly-fill or other fix to help.

The remainder of the table classes for coloring your table rows have changed slightly. Firstly there is a new class called **active**. Before BS3, the **active** class was not available on all elements, but mainly just on navigation and button elements. From BS3 onwards it can now be applied to a **<tr>** element to show that row as a highlighted row, which by default is a light grey.

The remaining contextual classes, as with BS2, are designed to highlight the table rows to show different conditions, and are the same, except for one small change.

The class name representing a dangerous or negative action has been renamed to **danger**. In BS2 the red class was named **error**. Other than that, the classes are applied the same way to the `<tr>` element as the following code shows:

*Code Sample 15: Optional Row Colorings*

```
<table class="table">
  <thead>
    <tr>
      <th>Class</th>
      <th>Col B</th>
      <th>Col C</th>
    </tr>
  </thead>
  <tbody>
    <tr class="active">
      <td>Active</td>
      <td>Val B</td>
      <td>Val C</td>
    </tr>
    <tr class="success">
      <td>Success</td>
      <td>Val B</td>
      <td>Val C</td>
    </tr>
    <tr class="info">
      <td>Info</td>
      <td>Val B</td>
      <td>Val C</td>
    </tr>
    <tr class="warning">
      <td>Warning</td>
      <td>Val B</td>
      <td>Val C</td>
    </tr>
    <tr class="danger">
      <td>Danger</td>
      <td>Val B</td>
      <td>Val C</td>
    </tr>
  </tbody>
</table>
```

When rendered, it should look something like this:

Class	Col B	Col C
Active	Val B	Val C
Success	Val B	Val C
Info	Val B	Val C
Warning	Val B	Val C
Danger	Val B	Val C

Figure 4: Output produced by code sample 15

There are two new additions to the classes used to support tables. First, there is a new responsive class that takes in to account the size of your grid system and provides either vertical scroll bars or a realigned table to fit differing size displays. Secondly, there's the ability to use the aforementioned colored row classes on individual cells, rather than just entire rows, as was the case in BS2.

To use the coloring classes on a cell level, you just need to add the classes to individual `<td>` or `<th>` elements as follows:

Code Sample 16: Applying the Option Row Classes to Individual Cells

```
<table class="table">
  <thead>
    <tr>
      <th class="active">Class</th>
      <th class="success">Col B</th>
      <th class="info">Col C</th>
    </tr>
  </thead>
  <tbody>
    <tr>
      <td class="warning">Val A</td>
      <td class="danger">Val B</td>
      <td>Val C</td>
    </tr>
  </tbody>
</table>
```

This code should result in the following:

Class	Col B	Col C
Val A	Val B	Val C

Figure 5: Output generated by code sample 16

The final table-related addition is a class called **table-responsive**, which you can use by applying it to a `<div>` element that wraps the entire `<table>` as follows:



Code Sample 17: New BS3 Responsive Table Example

```
<div class="table-responsive">
  <table class="table table-bordered">
    <thead>
      <tr>
        <th>Col A</th>
        <th>Col B</th>
        <th>Col C</th>
      </tr>
    </thead>
    <tbody>
      <tr>
        <td>Val A</td>
        <td>Val B</td>
        <td>Val C</td>
      </tr>
    </tbody>
  </table>
</div>
```

When this new class is used on a display greater than 768 pixels (that is, any display using a sizing class other than `*-xs-*`), then the table display will behave like a normal bootstrap-responsive table. However, if the table is displayed on a device that targets an `*-xs-*` class and is less than 768 pixels, the container will be altered so that a vertical scroll is available, allowing the entire table to be moved left and right without affecting the rest of the page.



Figure 6: Output from code sample 17 on a device greater than 768 pixels in width

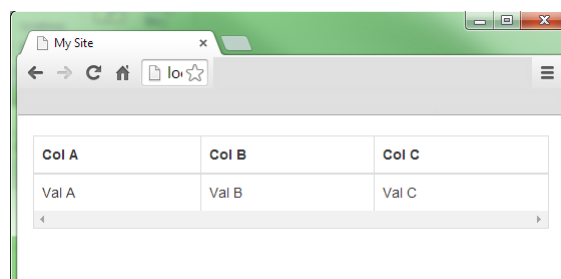


Figure 7: Output from code sample 17 on a device less than 768 pixels in width

## Form Changes

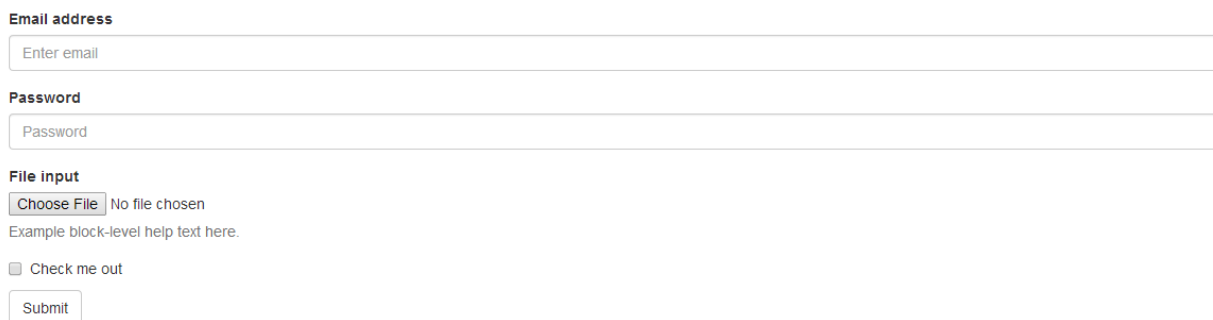
As with BS2, all normal form elements by default have a minimum level of markup that gives them a base style. This means that simply just marking up a normal form tag and associated controls will give your form the default Bootstrap look and feel.

Take the following example:

*Code Sample 18: Basic BS3 Form Example with no Classes*

```
<form>
  <div>
    <label for="exampleInputEmail1">Email address</label>
    <input type="email" id="exampleInputEmail1" placeholder="Enter email">
  </div>
  <div>
    <label for="exampleInputPassword1">Password</label>
    <input type="password" id="exampleInputPassword1" placeholder="Password">
  </div>
  <div>
    <label for="exampleInputFile">File input</label>
    <input type="file" id="exampleInputFile">
    <p>Example block-level help text here.</p>
  </div>
  <div>
    <label>
      <input type="checkbox"> Check me out
    </label>
  </div>
  <button type="submit">Submit</button>
</form>
```

If we render this example in a browser, we'll see that we get a reasonably good output without adding any extra classes, as you can see in the following image:



The image shows a rendered HTML form with the following elements:

- Email address:** A text input field with the placeholder text "Enter email".
- Password:** A password input field with the placeholder text "Password".
- File input:** A file input field with the text "Choose File" and "No file chosen". Below it is a paragraph of text: "Example block-level help text here."
- Checkbox:** A checkbox with the label "Check me out".
- Submit button:** A button with the text "Submit".

Output generated by code sample 18



**Note:** Since the previous example was written, there has been a minor update to the BS3 code. If you try the example as it is written here, the output will likely not look as expected. The change that has been made in BS3 seems to now mean that just marking up a form without any BS3 classes will not have the effect of giving the form a consistent look and feel. I've left the example in this book, as it agrees with the information that is still present on the documentation site, and as such still appears to be the official advice by the framework authors.

As I mentioned in the migration section, the classes and components around HTML forms have been some of the biggest casualties when it comes to class name changes, but this is for a good reason.

Before BS3, many of the classes used for forms were very narrow in scope—there were individual classes for many individual purposes, rather than a single class that covered many bases. For example, there were separate classes to handle the alignments of check boxes and radio buttons, and there were separate classes to handle input boxes and text areas with respect to their row alignment.

In BS3 many of these classes have been deleted and are now all rolled up under a smaller number of classes and elements.

Taking our previous example and adding in the recommended markup as is shown in the BS3 documentation gives us the following:

*Code Sample 19: Basic BS3 Form from Sample 18 with Recommended Classes Added*

```
<form role="form">
  <div class="form-group">
    <label for="exampleInputEmail1">Email address</label>
    <input type="email" class="form-control" id="exampleInputEmail1"
placeholder="Enter email">
  </div>
  <div class="form-group">
    <label for="exampleInputPassword1">Password</label>
    <input type="password" class="form-control" id="exampleInputPassword1"
placeholder="Password">
  </div>
  <div class="form-group">
    <label for="exampleInputFile">File input</label>
    <input type="file" id="exampleInputFile">
    <p class="help-block">Example block-level help text here.</p>
  </div>
  <div class="checkbox">
    <label>
      <input type="checkbox"> Check me out
    </label>
  </div>
  <button type="submit" class="btn btn-default">Submit</button>
</form>
```

Externally, if you render this code, you'll see no difference to the output generated for sample 18. Internally, however, Bootstrap can now find and work with individual elements much more easily than it could previously.

In BS2 there was no requirement to group controls unless you wanted to work with the automatic validation classes. There's still no absolute requirement to use them, but doing so allows BS to resize and reposition things correctly when using its grid (among many other things). You'll also note that each control now has just a single **form-control** class assigned to it, rather than many different ones targeting different aspects.

Also note that the form tag itself now has a **role** assigned to it. This, apart from being good practice all around, is now enforced by BS3 to help with aria roles and standards guiding the use of web apps by people with disabilities.

Lastly, if you look at the `<p>` tag with a **help-block**, you'll notice that that is also now used for inline form text in all cases, whereas in BS2 we had a number of different classes, such as **form-info**.

Unfortunately there is still one area in the support for forms that's missing—the file upload control. Like BS2, this is due to the fact that security in all the current crop of browsers restricts the ability to style file input controls to match the rest of the input controls available.

As I mentioned in the previous book, however, there are still third-party additions out there that have this area covered, and because you can now manufacture your own upload system using the HTML5 classes, strictly speaking you don't need the file upload control—you can manufacture your own.

In BS2 there were a number of specific form types such as the **search-form**; under BS3 all of these have been rolled up into three main types of form. First, you have the standard form; as we've seen, this is a normal form with no extra classes added to the form tag. The two other form types are **form-inline** and **form-horizontal**.

The **form-inline** class is designed for forms in small, limited height places such as menu and navigation bars. A word of warning though: all input elements in this class and the other form types are sized 100 percent by default, so if you need the form to only take up a small amount of room (particularly in **nav bar** forms), you will need to put manual sizes on the individual controls.

In the example from code sample 19, adding **form-inline** or **form-horizontal** in turn should change your basic form layout to look like the following:



The image shows a horizontal form layout. On the left, there is a label "Email address" followed by an input field containing the placeholder text "Enter email". To its right is a label "Password" followed by an input field containing the placeholder text "Password". Further right is a "File input" section with a "Choose File" button, the text "No file chosen", and a small "x" icon. To the right of the file input is a checkbox labeled "Check me out". Finally, on the far right is a "Submit" button. Below the file input, there is a line of text: "Example block-level help text here."

Figure 8: Form produced by code sample 19 with the **form-inline** class added to the form tag.

The **form-horizontal** class is used to create regular top-down forms with input controls that have their associated labels to the left of them, rather than above them as the default form does. Be aware, however, that in order for form horizontal to work correctly, you need to add a little extra markup to the form in general, as shown in the following code sample.



**Note:** In today's brave new world of HTML 5, it's more important than ever to mark input elements up correctly with an associated label. Because disadvantaged users may be using aids to assist them, not providing the required pieces to allow these aids to work correctly will, going forward, be seen as a bad thing, and companies leaving them up could be shunned for doing so. Not to worry though—BS3 has you covered. If you decide you don't want labels in your forms, you can mark them with an optional class, *sr-only*. Adding this class will visually prevent the label or associated help text from appearing in your document, but will ensure that it is marked up in such a way as to be visible by screen readers and other similar devices or software.

Code Sample 20: The Code from Sample 19 Marked Up with Extra Classes to Support Form-Horizontal

```
<form role="form" class="form-horizontal">
  <div class="form-group">
    <label for="exampleInputEmail1" class="col-sm-2 control-label">Email
address</label>
    <div class="col-sm-10">
      <input type="email" class="form-control" id="exampleInputEmail1"
placeholder="Enter email">
    </div>
  </div>
  <div class="form-group">
    <label for="exampleInputPassword1" class="col-sm-2 control-
label">Password</label>
    <div class="col-sm-10">
      <input type="password" class="form-control" id="exampleInputPassword1"
placeholder="Password">
    </div>
  </div>
  <div class="form-group">
    <label for="exampleInputFile" class="col-sm-2 control-label">File input</label>
    <div class="col-sm-10">
      <input type="file" id="exampleInputFile">
      <p class="help-block">Example block-level help text here.</p>
    </div>
  </div>
  <div class="form-group">
    <div class="col-sm-offset-2 col-sm-10">
      <div class="checkbox">
        <label>
          <input type="checkbox"> Remember me
        </label>
      </div>
    </div>
  </div>
  <div class="form-group">
    <div class="col-sm-offset-2 col-sm-10">
```

```

    <button type="submit" class="btn btn-default">Sign in</button>
  </div>
</div>
</form>

```

If you render the code from this example in your browser, it should look like the following:

Figure 9: Output displayed from code sample 20 showing our form marked up as horizontal.

As you can see from the code in sample 20, the extra markup is not really that much more, and most of it exists just to line up the columns correctly so that everything sits nicely.

The main points to be aware of in sample 20 are:

1. All label controls now have a class of **control-label** added. This is not required for the other form types, and BS2 will simply ignore it.
2. Any input control that will likely render as a block-level element is now wrapped in a parent **<div>** in order to control its width using the grid system.
3. The extra class for **form-horizontal** is applied to the outer-most form tag.

For the rest of the classes and associated parts in the forms section of BS3, nothing else beyond the classes mentioned so far has changed. However, from an actual application point of view, input controls now **MUST** have a correct type on them to be styled. This means at a minimum you must have at least **type="text"** for BS3 to do its magic.

It's highly recommended that you do use the correct types, however. As you'll see soon when we get to validation groups, having the correct type will allow most of the validation stuff to actually work correctly without making any changes to your markup.

Check boxes and radio buttons, as in BS2, are stacked by default. If you wish to have them render vertically across the screen, you need to use the **checkbox-inline** and **radio-inline** classes as follows:

Code Sample 21: Inline Check Box and Radio Buttons in BS3

```

<div class="row">
  <label class="checkbox-inline">
    <input type="checkbox" id="inlineCheckbox1" value="option1"> 1
  </label>

```

```

<label class="checkbox-inline">
  <input type="checkbox" id="inlineCheckbox2" value="option2"> 2
</label>
<label class="checkbox-inline">
  <input type="checkbox" id="inlineCheckbox3" value="option3"> 3
</label>
</div>
<div class="row">
  <label class="radio-inline">
    <input type="radio" id="inlineRadio" value="option1"> 1
  </label>
  <label class="radio-inline">
    <input type="radio" id="inlineRadio" value="option2"> 2
  </label>
  <label class="radio-inline">
    <input type="radio" id="inlineRadio" value="option3"> 3
  </label>
</div>

```

Rendering sample 21 in your browser should look something like the following:



Figure 10: Output produced by sample 21

Select and multi-select boxes are marked up with standard styles just by using the elements as they stand; this is no change from BS2, where the markup and style of these elements is identical.

One new style that has been introduced in BS3 is the static control style. In BS2, you often had to use a disabled form control to represent static form data that could not be changed. BS3 changes this by providing a **form-control-static** class that can be applied to individual controls in place of the regular **form-control**, as the following code shows:

Code Sample 22: How to Create a Static Input Component in BS3

```

<form class="form-horizontal" role="form">
  <div class="form-group">
    <label class="col-sm-2 control-label">Email</label>
    <div class="col-sm-10">
      <p class="form-control-static">email@example.com</p>
    </div>
  </div>
  <div class="form-group">
    <label for="inputPassword" class="col-sm-2 control-label">Password</label>
    <div class="col-sm-10">
      <input type="password" class="form-control" id="inputPassword"
placeholder="Password">

```

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

When rendered in the browser, this produces a regular form layout, with the static control clearly marked as not being editable or looking like an input control, as the following image shows:



Figure 11: The form with a static field produced by code sample 22

The disabled style of form controls is still marked up and used in the same way as in BS2, simply by adding the “disabled” attribute to the input element as the following shows:

```
<input class="form-control" id="disabledInput" type="text" placeholder="Disabled input here..." disabled>
```

This will produce the classic disabled and shaded control look, something like the following:

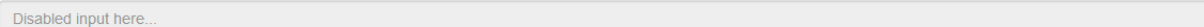


Figure 12: Classic disabled control look in both BS2 and BS3

If you add the **disabled** attribute to a field-set surrounding a form, all of the controls in that group will be disabled at the same time, using the same styling. This is something that didn't happen in BS2—you instead had to mark up each individual control.

The final thing I want to mention while on the subject of forms is the validation and focus classes. As I previously noted, adding the correct input types in HTML5 elements really helps you when it comes to using the validation classes. Why is that?

In addition to having dedicated class names and styles for you to use, the validation classes also hook the new HTML5 pseudo-elements such as **focus:**, **error:**, and others.

This means that if you mark a text box as **type="email"**, and then don't put an email address into it when submitting, the browser should automatically highlight the field red for you.

If it does not, or you have some other way of making your validation work, then you can simply use the **has-success**, **has-warning**, and **has-error** classes on your form groups as the following code shows:

Code Sample 23: Marking Up a Form in BS3 to Use the Validation Classes

```
<form role="form">
  <div class="form-group has-success">
```



```

<label class="control-label" for="inputSuccess1">Label for Form group that has-
success</label>
  <input type="text" class="form-control" id="inputSuccess1" value="Some successful
text">
  <p class="help-block">Successful block-level help text here.</p>
</div>
<div class="form-group has-warning">
  <label class="control-label" for="inputWarning1">Label for Form group that has-
warning</label>
  <input type="text" class="form-control" id="inputWarning1" value="Some warning
text">
  <p class="help-block">Warning block-level help text here.</p>
</div>
<div class="form-group has-error">
  <label class="control-label" for="inputError1">Label for Form group that has-
error</label>
  <input type="text" class="form-control" id="inputError1" value="Some error text">
  <p class="help-block">Error block-level help text here.</p>
</div>
</form>

```

If you render this code in your browser, you should see something like the following:

The screenshot shows three vertically stacked form groups. Each group consists of a label, an input field, and a help block. The first group is green, the second is orange, and the third is red. The labels are: "Label for Form group that has-success", "Label for Form group that has-warning", and "Label for Form group that has-error". The input fields contain: "Some successfull text", "Some warning text", and "Some error text". The help blocks contain: "Successful block-level help text here.", "Warning block-level help text here.", and "Error block-level help text here.".

*Figure 13: The output produced by code sample 23*

Points to note are that I've added the classes to the form groups in order to produce a static display; however, you should also try just marking up the form with the correct input types.

I've found browser support on the pseudo-classes is still a little patchy, even though there's no mention of it in the BS3 docs, so I do recommend that you also make use of the class names when manipulating your elements using JavaScript.

Also note that the label and block-level help text takes on the correct color of the group too, so you have no need of adding colors or styles to these separately to the form control group. Remember that the **form-group** sections can also use everything else we've mentioned so far to disable, shade, and resize form elements as required on a grouped basis.

The last thing to mention for validation groups is that you can also provide optional feedback icons directly in the form controls in order to help with the state.

You do this by providing a span element IMMEDIATELY after the input element with which it should be used. This span element has the usual icon classes applied to it (which we'll see in the next chapter), along with a class of **form-control-feedback**. This MUST be put after the input control, and before any other markup in the input group, due to the way the control is repositioned to make it appear in the control. Once you add the span, you also need to add a class of **has-feedback** in the form group class list alongside the other **has-xxxxx** classes used to show the validation state.

If we expand code sample 23 to take this in to account and add feedback icons, this is what it should look like:

*Code Sample 24: Sample 23, Changed to Add Feedback Icons*

```
<form role="form">
<div class="form-group has-success has-feedback">
  <label class="control-label" for="inputSuccess1">Label for Form group that has-
success</label>
  <input type="text" class="form-control" id="inputSuccess1" value="Some successfull
text">
  <span class="glyphicon glyphicon-ok form-control-feedback"></span>
  <p class="help-block">Successful block-level help text here.</p>
</div>
<div class="form-group has-warning has-feedback">
  <label class="control-label" for="inputWarning1">Label for Form group that has-
warning</label>
  <input type="text" class="form-control" id="inputWarning1" value="Some warning
text">
  <span class="glyphicon glyphicon-warning-sign form-control-feedback"></span>
  <p class="help-block">Warning block-level help text here.</p>
</div>
<div class="form-group has-error has-feedback">
  <label class="control-label" for="inputError1">Label for Form group that has-
error</label>
  <input type="text" class="form-control" id="inputError1" value="Some error text">
  <span class="glyphicon glyphicon-remove form-control-feedback"></span>
  <p class="help-block">Error block-level help text here.</p>
</div>
</form>
```

Once we re-render with these changes, you should see the following:



Figure 14: The output generated by code sample 24

## Button Changes

The most prominent change in the classes used to style buttons in BS3 is the default style. Under BS2, simply adding the `btn` class to an input element of type button, or to an anchor tag, would give the control the default button look and feel.

From BS3 onwards, you now explicitly have to add `btn-default`; just adding `btn` on its own will now no longer have any effect.

The second main change is in the renaming of some of the base classes. Specifically, `btn-error` has been renamed to `btn-danger` so that the naming scheme matches the other similarly named class changes and brings uniformity to the Bootstrap base library.

Other than that, the base button classes remain unchanged, as the following code sample shows:

Code Sample 25: BS3 Button Classes

```
<button type="button" class="btn btn-default">Default</button>
<button type="button" class="btn btn-primary">Primary</button>
<button type="button" class="btn btn-success">Success</button>
<button type="button" class="btn btn-info">Info</button>
<button type="button" class="btn btn-warning">Warning</button>
<button type="button" class="btn btn-danger">Danger</button>
<button type="button" class="btn btn-link">Link</button>
```

When rendered in the browser, you'll see the normal flat button look the BS3 now has for controls:



Figure 15: Bootstrap buttons as produced by code sample 25

There are a few new classes for the button element, added to bring uniformity to the grid-sizing classes. These classes are **btn-lg**, **btn-sm**, and **btn-xs**; there is no **btn-md**, as the medium-sized button is the standard size used when no class is specified.

The following code sample demonstrates all the button styles at different sizes:

*Code Sample 26: All the BS3 Buttons Styles at Each of the Different Sizes*

```
<p>
  <button type="button" class="btn btn-default btn-lg">Large default button</button>
  <button type="button" class="btn btn-default">Medium default button</button>
  <button type="button" class="btn btn-default btn-sm">Small default button</button>
  <button type="button" class="btn btn-default btn-xs">Extra small default
button</button>
</p>
<p>
  <button type="button" class="btn btn-primary btn-lg">Large primary button</button>
  <button type="button" class="btn btn-primary">Medium primary button</button>
  <button type="button" class="btn btn-primary btn-sm">Small primary button</button>
  <button type="button" class="btn btn-primary btn-xs">Extra small primary
button</button>
</p>
<p>
  <button type="button" class="btn btn-success btn-lg">Large success button</button>
  <button type="button" class="btn btn-success">Medium success button</button>
  <button type="button" class="btn btn-success btn-sm">Small success button</button>
  <button type="button" class="btn btn-success btn-xs">Extra small success
button</button>
</p>
<p>
  <button type="button" class="btn btn-info btn-lg">Large info button</button>
  <button type="button" class="btn btn-info">Medium info button</button>
  <button type="button" class="btn btn-info btn-sm">Small info button</button>
  <button type="button" class="btn btn-info btn-xs">Extra small info button</button>
</p>
<p>
  <button type="button" class="btn btn-warning btn-lg">Large warning button</button>
  <button type="button" class="btn btn-warning">Medium warning button</button>
  <button type="button" class="btn btn-warning btn-sm">Small warning button</button>
  <button type="button" class="btn btn-warning btn-xs">Extra small warning
button</button>
</p>
<p>
  <button type="button" class="btn btn-danger btn-lg">Large danger button</button>
  <button type="button" class="btn btn-danger">Medium danger button</button>
  <button type="button" class="btn btn-danger btn-sm">Small danger button</button>
  <button type="button" class="btn btn-danger btn-xs">Extra small danger
button</button>
</p>
<p>
  <button type="button" class="btn btn-link btn-lg">Large link button</button>
  <button type="button" class="btn btn-link">Medium link button</button>
</p>
```

```
<button type="button" class="btn btn-link btn-sm">Small link button</button>
<button type="button" class="btn btn-link btn-xs">Extra small link button</button>
</p>
```

When rendered in the browser, this code sample should give you the following output:

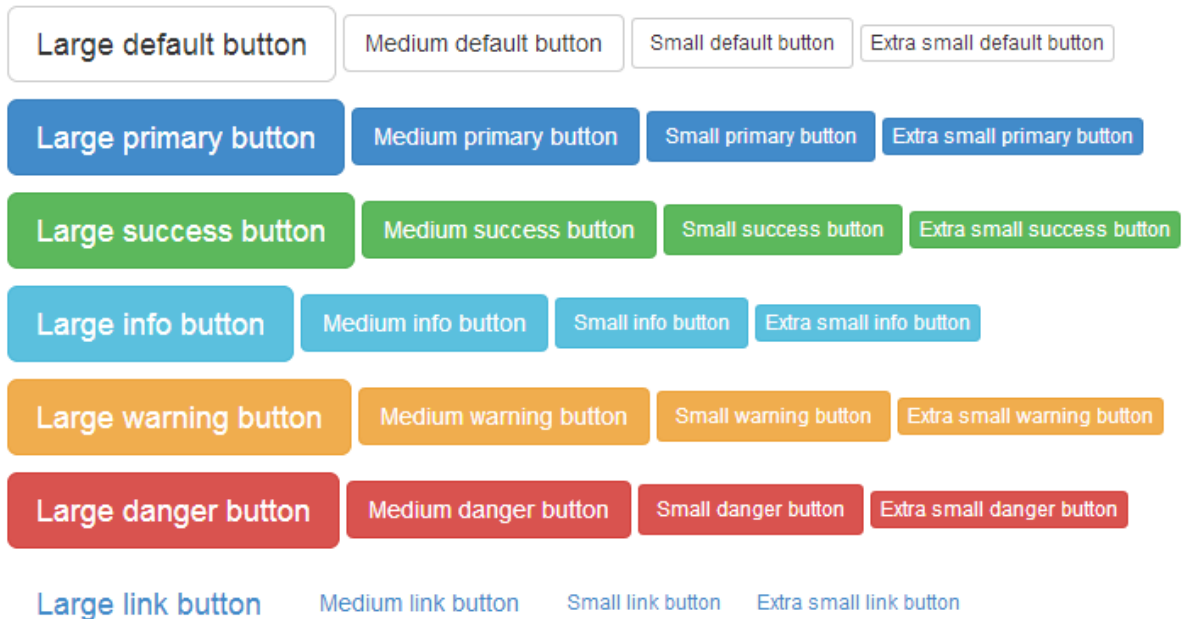


Figure 16: The output produced by code sample 26

When you add the class **btn-block** to a button or anchor element styled using any of the previous button classes, that button will stretch to fill 100 percent of the available space. This is useful when producing dialog boxes and sizing controls with BS3 grid system, as it allows you to specifically size buttons (and other elements) to maintain a good balance in your form designs.

In most cases, you won't need to set a button's active state yourself, but if you do, you can easily add the **active** class to any element marked up using the button classes. **Active** in general (on the **<button>** element at least) usually uses the **:active** pseudo-selector to change the button's style. Adding the **active** class, however, will force the button to display its active state.



**Note:** *If you're thinking of using the active state to create sticky buttons, be aware that in the following chapters, we will be discussing button states using the component and JavaScript facilities available. BS3 provides just such a sticky button using the additional features available in these facilities, so you won't need to create your own sticky buttons using active in most circumstances.*

You can also disable your buttons and mark them as inactive using the same "disabled" classes and attributes that we discussed previously with the changes in form elements.

The following code shows buttons marked up to look active and in the disabled states:

Code Sample 27: BS3 Active and Disabled Buttons

```
<p>
  <button type="button" class="btn btn-primary active">Active primary button</button>
  <button type="button" class="btn btn-default active">Active default button</button>
</p>
<p>
  <button type="button" class="btn btn-primary" disabled="disabled">Disabled primary
button</button>
  <button type="button" class="btn btn-default" disabled="disabled">Disabled default
button</button>
</p>
```

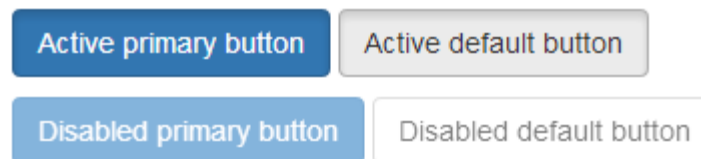


Figure 17: Output produced by code sample 27

Be aware, however, that in the case of anchor buttons, the **disabled** element/class does NOT disable the link; to make sure that a disabled state anchor link does not fire, you will need to use custom JavaScript.

For this reason, the BS3 team recommends that you use the **<button>** element where possible to mark buttons, and only use anchor tags in specific circumstances.

Let's also not forget the subject of "idempotence" and the importance of using buttons over anchors. In general, an anchor link is appropriate if the destination is a get request and making the link several times will not cause any issues by repeat activation; otherwise, use a button.

## Image Changes

CSS changes, where the humble image tag is concerned, have not been as far-reaching as in other places in BS3. Previously, images were not responsive by default, and as with many things in BS2, you had to add the optional responsive classes to get anywhere near being responsive with them.

Unfortunately, even after you added the optional responsive classes, things still were not perfect; many people cited problems with pages where Google Maps were used, among other things.

BS3 changes all of this. Now, by default, images are fully responsive with the use of an `<img>` tag. To extend and make that responsiveness even better and more fluid, you can add the `img-responsive` to any `<img>` tag to ensure that even while scaling correctly, the height and width of the image also stay in proportion to each other.

Other than that, the only other change made to the images section is the name of the `img-polaroid` class (used for generating thumbnails) to `img-thumbnail`, so that it falls in line with other similar renaming throughout the library.

The BS2 classes `img-rounded` and `img-circle` still work as they did previously, giving a circular and rounded-rectangle thumbnail effect.

## Helper and Visibility Changes

Finally, we come to the changes that cover those things that don't really fit into any specific categories.

Under the typography classes in BS2, you were originally introduced to a set of color classes used for setting the color of a text element to the same branding colors used in other elements in the framework.

BS3 takes this one step further and introduces the concept of having the same colors used for contextual backgrounds too.

In this case, the backgrounds are a lighter variation, as used in panels and alert boxes elsewhere. There's an added bonus: if you use these contextual color classes on anchor tags and anything else that has a hover-over set by default, the colors will automatically dim slightly to show they've been hovered over.

As with the naming in other color-based classes, the classes available here are `text-muted`, `text-primary`, `text-success`, `text-info`, `text-warning`, and `text-danger` for paragraph, span, and other inline or block-based text elements.

For background colors, the class names are `bg-primary`, `bg-success`, `bg-info`, `bg-warning`, and `bg-danger`.

The following code shows an example of using them:

*Code Sample 28: BS3 Contextual Color Classes*

```
<p class="text-muted">This paragraph is using the muted text class, <strong>typically reserved for something not really important or less prominent.</strong></p>
<p class="text-primary">This paragraph is using the primary text class, <strong>typically reserved for something important or default and visible.</strong></p>
<p class="text-success">This paragraph is using the success text class, <strong>typically reserved for an action that just succeeded or something good and congratulatory.</strong></p>
```

```

<p class="text-info">This paragraph is using the info text class, <strong>typically reserved for informal messages, such as a background job just finishing or a new file available.</strong></p>
<p class="text-warning">This paragraph is using the warning text class, <strong>typically reserved for something that might be dangerous or that needs attention but can wait a while.</strong></p>
<p class="text-danger">This paragraph is using the danger text class, <strong>typically reserved for something very important, or something that really needs attention drawing to it.</strong></p>
<br/><br/>

<p class="bg-primary">This paragraph is using normal text but with a primary background color to tell you that what you're seeing is the default status.</p>
<p class="bg-success">This paragraph is using normal text but with a success background color to tell you that what you're seeing is all good.</p>
<p class="bg-info">This paragraph is using normal text but with an info background color to tell you that what you're seeing is informative and should be read, but not always acted upon.</p>
<p class="bg-warning">This paragraph is using normal text but with a warning background color to tell you that what you're seeing could cause problems that you should be aware of.</p>
<p class="bg-danger">This paragraph is using normal text but with a danger background color to tell you that what you're seeing needs you to pay attention to it now.</p>

```

This paragraph is using the muted text class, typically reserved for something not really important or less prominent.

This paragraph is using the primary text class, typically reserved for something important or default and visible.

This paragraph is using the success text class, typically reserved for an action that just succeeded or something good and congratulatory.

This paragraph is using the info text class, typically reserved for informal messages, such as a background job just finishing or a new file available.

This paragraph is using the warning text class, typically reserved for something that might be dangerous or that needs attention but can wait a while.

This paragraph is using the danger text class, typically reserved for something very important, or something that really needs attention drawing to it.

This paragraph is using normal text but with a primary background color to tell you that what you're seeing is the default status.

This paragraph is using normal text but with a success background color to tell you that what you're seeing is all good.

This paragraph is using normal text but with an info background color to tell you that what you're seeing is informative and should be read, but not always acted upon.

This paragraph is using normal text but with a warning background color to tell you that what you're seeing could cause problems that you should be aware of.

This paragraph is using normal text but with a danger background color to tell you that what you're seeing needs you to pay attention to it now.

Figure 18: Output produced by code sample 28

One thing that was noted as being needed in BS3, and not present in BS2, is a dedicated **caret** class for drop-down indicators on buttons and other screen furniture.

You can now utilize this on your own elements by adding a class name of **caret** to your outer container—a `<span>` or `<div>` that wraps your inner content.

Another new addition in BS3, while it's not actually a class, still deserves a mention: the dialog close cross.

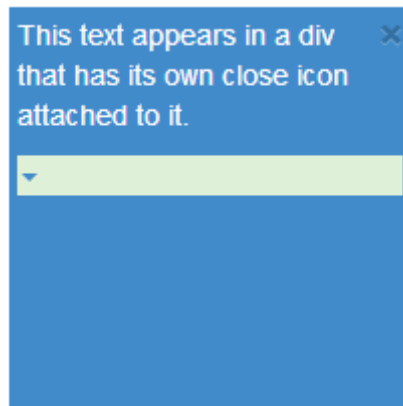


Just like the **caret**, this was present in BS2, but was not available to use separate from its intended use in modal dialogs and alert bars. However, from BS3 onwards, you can use it just fine on its own.

The following code sample shows an example of both the **caret** class and close cross in generic mark-up:

*Code Sample 29: Using the BS3 Caret and Close Cross Helpers*

```
<div class="bg-primary" style="width: 200px; height: 200px; padding: 4px;">
  <button type="button" class="close" aria-hidden="true">&times;</button>
  <p>This text appears in a div that has its own close icon attached to it.</p>
  <div class="bg-success text-primary"><div class="caret" /></div>
</div>
```



*Figure 19: Output from code sample 29*

Just as in BS2, the quick float utility classes **pull-left** and **pull-right** still exist, but they are now joined by **center-block**, which simply makes both margins automatic and centers the element in its parent, and **clearfix**, which clears out any floats that you may be using, thus restoring normal document flow.

Other new classes include the **show**, **hidden**, and **invisible** classes. **Show** pretty much speaks for itself, but what's the difference between the latter two? **Hidden** physically collapses the space used by the element, so if you have it in a full height div for example, that div will collapse down to its smallest height. However if you use **invisible**, the element retains its space (and also still occupies its place in the element flow), but it vanishes from view.

We've already seen that a new class called **sr-only** has been introduced to mark a block as being visible to screen readers and nothing else. There is now another class like this that is used for graphical headings.

If you define a page title using an image banner, a typical screen reader will be unable to tell what the text in the image says. For a long time, many authors have used a hack called image replacement to get around this. Image replacement works by wrapping the image banner in an **H1** or some other standard-type tag, putting the name in it as clear text alongside the image banner, then using CSS to move the text off screen.

What then happens is that display-wise, the image banner is seen by those with good sight, but those using a screen reader hear the reader say the actual text in the image banner.

BS3 now provides a class called **text-hide** in order to facilitate this. A simple example follows:

*Code Sample 30: Using the BS Text-Hide Class to Make Screen Reader Friendly Headers*

```
<header>
  
  <h1 class="text-hide">A graphical image based title</h1>
</header>
```

The image shows the output of the code sample in a regular PC browser. It features a large, stylized title "A Graphical Image Based Title" centered on the page. The text is rendered in a bubbly, colorful font with a red outline and a drop shadow effect. The words "A Graphical" are in green, "Image" is in yellow, and "Based Title" is in red. The background is white.

*Figure 20: Output generated by code sample 30 in a regular PC browser*

Finally, we come to the last of the last in the CSS changes section.

What good would a responsive web design framework be without utility classes to help you manage your responsive layouts?

"But hold on, we've covered that with grids," I hear you say, and yes, we have. But BS3 has one more trick up its sleeve, which in all fairness WAS present in BS2 but, didn't really work all that well.

So what's this extra magic? Let me introduce you to the responsive visibility classes.

Essentially, what these little gems do is allow you to swap and change parts of your UI depending on your grid and display size.

Let's imagine, for example, that you have a list of email inbox items, and when it's viewed on a desktop PC, each item has a preview next to it, much like in a classic email reading application. Something perhaps like the following:

Email 1 (1/1/11)	Email 2 : Received (1/1/11)
Email 2 (1/1/11)	To : A person
Email 3 (1/1/11)	Dear A Person, Blah blah blah blah blah blah
Email 4 (1/1/11)	
Email 5 (1/1/11)	
Email 6 (1/1/11)	

Figure 21: Email application layout example

Now this is great until you try to fit it on a mobile device, where you absolutely want to hide the preview pane and leave just the email list. Normally a task like this is performed using a little bit of JavaScript to change the visibility of the element by changing the element's visible settings.

BS3 has a built-in solution using CSS classes that helps you handle situations like this with great ease. Take the following bit of code:

Code Sample 31: Email Application Mockup

```
<div class="col-md-12" style="border: 1px solid black">
  <div class="col-md-3">
    <table class="table">
      <tr>
        <td>Email 1 (1/1/11)</td>
      </tr>
      <tr>
        <td class="info">Email 2 (1/1/11)</td>
      </tr>
      <tr>
        <td>Email 3 (1/1/11)</td>
      </tr>
      <tr>
        <td>Email 4 (1/1/11)</td>
      </tr>
      <tr>
        <td>Email 5 (1/1/11)</td>
      </tr>
    </table>
  </div>
</div>
```

```

    <tr>
      <td>Email 6 (1/1/11)</td>
    </tr>
  </table>
</div>
<div class="col-md-9" style="border-left: 1px solid black">
  <h1>Email 2 : <small>Received (1/1/11)</small></h1>
  <h2>To : A person</h2>
  <br/>
  <br/>
  <p>Dear A person,</p>
  <p>Blah blah blah blah blah blah</p>
</div>
</div>

```

If you render this in your browser you should get something that looks like the following:



Figure 22: Email application mock-up produced by code sample 31

If, however, you resize your browser to mobile-screen size, things start to look a bit strange:

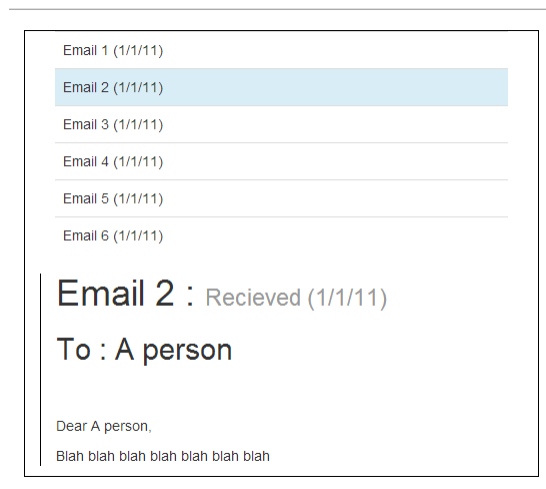


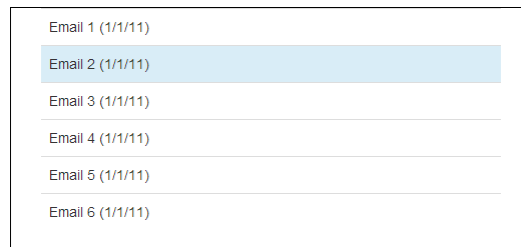
Figure 23: Email application mock-up on a resized view

This may work for some, but in general, it's a bad idea. What happens when you have 100 unread emails, and have to scroll to the bottom of the view each time to read the preview?

Take the main preview `<div>` and add a new class to it, **hidden-sm**, as follows:

```
<div class="col-md-9 hidden-sm" style="border-left: 1px solid black">
```

Then, if you refresh your browser and try resizing it, you should see that the preview `<div>` now gets hidden beyond certain widths, rather than getting stacked.



*Figure 24: Email application mock-up, resized, but with hiding classes added*

Just as with the grid system, there are four different sizes, and there are classes to hide and make visible. The visible classes will make the element in question visible **ONLY** at the specified screen size, and the hidden classes make the element in question hidden **ONLY** at the specified screen size. The class names are as follows:

**visible-xs** - Make visible on extra-small screens

**visible-sm** - Make visible on small screens

**visible-md** - Make visible on medium screens

**visible-lg** - Make visible on extra large screens

**hidden-xs** - Hide on extra-small screens

**hidden-sm** - Hide on small screens

**hidden-md** - Hide on medium screens

**hidden-lg** - Hide on large screens

The widths of the devices and the associated trigger points are the same as those used in the grid system in general, with **-xs** covering palm-sized tablet phones and smaller, **-sm** covering average-to-large tablets, **-md** covering most desktop computers, and **-lg** covering wide-screen desktops.

There are two final classes used in this category to assist you with handling display v's print-based layouts.

You can use **visible-print** and **hidden-print** in exactly the same manner as the size-based classes above, but this time making an element visible and invisible when a page is sent to the printer.

# Chapter 4 Changed Components Features

## Glyphicon Changes

The biggest news with the components provided by BS3 is the Glyphicons icon set.

When V3.0 beta was first released, the Glyphicons font that forms the core of the icon set in BS3 was removed. The community was up in arms about this, especially when the icon set was split into its own repository.

The split was short-lived, however, and from the production release onwards, the Glyphicons font was reinstated into the main branch, but with one new bonus.

No longer is the icon set made up of a set of bitmapped **pngs** in white and black—it's now a full-scale, vector-based font set, with **ttf**, **svg**, **wot** and other formats. Because it's a vector now, it can easily be colored using standard CSS-based color techniques, and can now be scaled to any size required.

In the current build of BS3, there are 200 or more icons to choose from, covering lots of different use cases. To use an icon, you simply add the class **glyphicon** and a **glyphicon-xxxxx** to select the actual icon you want to use. Then, you add this to a span class that's nested inside of the element you wish to display the icon in.

One rule you must obey with glyphicons is that you cannot mix the icon content with any other element. You must make sure that the icon class is applied only to its own element, and with enough padding around it to ensure sufficient space between it and any other content next to it.

The following code gives you a simple overview on how to use the glyphicon classes:

*Code Sample 32: BS3 Glyphicons Sample*

```
<h1>This is header text
  <small>with some small text and an icon <span class="glyphicon glyphicon-home"></span></small>
</h1>
<p>This is a normal paragraph, which again like the header has an icon inline with it
  <span class="glyphicon glyphicon-headphones"></span>
</p>
<div style="background-color: beige; width: 200px; height: 200px; padding: 4px;">
  <p>This is an absolute size div with a scaled up icon in it</p>
  <span class="glyphicon glyphicon-road" style="font-size: 150px;"></span>
</div>
<p class="text-success">This is a normal paragraph, with a different color applied to it
  <span class="glyphicon glyphicon-cog"></span>
</p>
```

This is header text with some small text and an icon 🏠

This is a normal paragraph, which again like the header has an icon inline with it 🗨

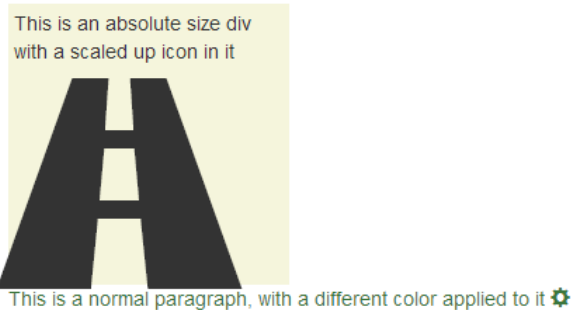


Figure 25: Output produced by code sample 32

There are far too many available icons to list them all here, but if you go to <http://getbootstrap.com/components/#glyphicons-how-to-use>, you'll find a grid complete with all the class names required to get the icon you need.

The icons work correctly in any container you add them to, and they respect the sizing classes too. So, for example, if you add one inside a `<button>` element that has a `btn-lg` class attached to it, the icon will scale as required to match the button size.

## Button Changes

Changes to buttons in BS3 are very minimal from a component point of view. There are a few new classes and recommendations, and some changes to their look and feel as mentioned, but otherwise, everything you did in BS2 should still work as expected.

From a component point of view, the `btn-group` and `btn-toolbar` classes are still the same, but it is now recommended that where possible, you try to ensure that you group buttons using `btn-group`. Also, when marking up toolbars, you should try to make sure you have a role of `toolbar` applied to them to assist screen readers and other equipment.

There is one thing you need to be aware of: due to changes in how BS3 handles button groups, and if you are using tooltips or popovers on your buttons, you will now have to add the option `container:body` to the tooltip or popover when you create it. The container doesn't have to be `body`, however; it can be any container in which the popover/tooltip/button arrangement is nested. The container needs to be there though; this, unfortunately, means that you now have no option but to create popovers and tooltips using JavaScript.

In line with the rest of the new sizing classes comes the new `btn-group-lg`, `btn-group-sm`, and `btn-group-xs`. As with the other size classes, there is NO `btn-group-md`, since medium is the default size with no extra styling added.



The following code shows a basic example of button grouping, toolbars, and different sizes.

Code Sample 33: Simple Button Group, Toolbar, and Group Sizing Example

```
<div class="btn-toolbar" role="toolbar">
  <div class="btn-group btn-group-lg">
    <button type="button" class="btn btn-default">Left</button>
    <button type="button" class="btn btn-default">Middle</button>
    <button type="button" class="btn btn-default">Right</button>
  </div>
  <div class="btn-group">
    <button type="button" class="btn btn-primary">Left</button>
    <button type="button" class="btn btn-primary">Middle</button>
    <button type="button" class="btn btn-primary">Right</button>
  </div>
  <div class="btn-group btn-group-sm">
    <button type="button" class="btn btn-success">Left</button>
    <button type="button" class="btn btn-success">Middle</button>
    <button type="button" class="btn btn-success">Right</button>
  </div>
  <div class="btn-group btn-group-xs">
    <button type="button" class="btn btn-warning">Left</button>
    <button type="button" class="btn btn-warning">Middle</button>
    <button type="button" class="btn btn-warning">Right</button>
  </div>
</div>
```



Figure 26: Output generated by code sample 33

Button nesting is still performed in the same manner as it was in BS2, thus allowing you to add drop-down button menus inside groups of normal buttons, as the following code shows (but remember, this is JavaScript functionality, so you will need to make sure JQuery is included, as well as the BS3 JS file):

Code Sample 34: Nested Buttons

```
<div class="btn-group">
  <button type="button" class="btn btn-default">Left</button>
  <button type="button" class="btn btn-default">Middle</button>

  <div class="btn-group">
    <button type="button" class="btn btn-primary dropdown-toggle" data-
toggle="dropdown">
      Right
      <span class="caret"></span>
    </button>
    <ul class="dropdown-menu">
```

```

    <li><a href="#">Menu Link 1</a></li>
    <li><a href="#">Menu Link 2</a></li>
  </ul>
</div>
</div>

```

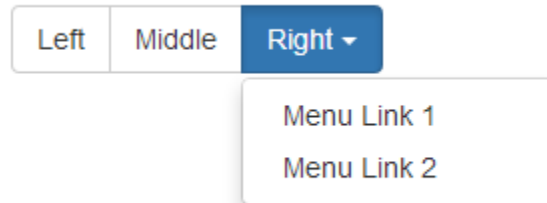


Figure 27: Output generated by code sample 34

Notice in the last code sample that I altered the button color of the drop-down button; all you need to do in order to achieve this is add individual button colors on the classes for individual buttons. This works across all button groups and toolbars—not just when nesting them—meaning that you can vary the colors of the buttons in your group depending on your usage.

Two new classes used for button layouts are the **btn-group-vertical** and the **btn-group-justified** classes.

In BS2, it was very difficult to stack your buttons vertically, or to make them take up the full width of the available space while maintaining a balanced size. These two new classes in BS3 solve both of these tricky situations. Be aware, though, that to make the justified groups work correctly, you need to mark your button groups with a little more markup than you might expect.

The next code sample shows how to use vertical groups:

Code Sample 35: Vertical Button Groups

```

<div class="btn-group-vertical btn-group-lg">
  <button type="button" class="btn btn-default">Top</button>
  <button type="button" class="btn btn-default">Middle</button>
  <button type="button" class="btn btn-default">Bottom</button>
</div>
<div class="btn-group-vertical">
  <button type="button" class="btn btn-primary">Top</button>
  <button type="button" class="btn btn-primary">Middle</button>
  <button type="button" class="btn btn-primary">Bottom</button>
</div>
<div class="btn-group-vertical btn-group-sm">
  <button type="button" class="btn btn-success">Top</button>
  <button type="button" class="btn btn-success">Middle</button>
  <button type="button" class="btn btn-success">Bottom</button>
</div>
<div class="btn-group-vertical btn-group-xs">

```

```

<button type="button" class="btn btn-warning">Top</button>
<button type="button" class="btn btn-warning">Middle</button>
<button type="button" class="btn btn-warning">Bottom</button>
</div>

```

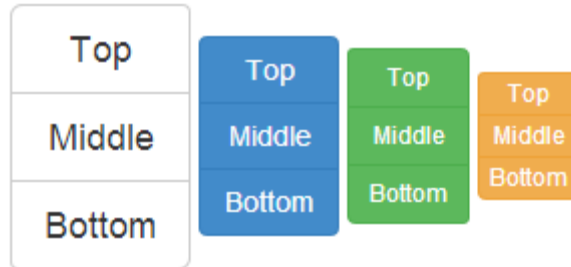


Figure 28: Output produced by code sample 35

As I just pointed out, marking up the justification classes is a little more involved, but is necessary to smooth out some of the few browser inconsistencies that, unfortunately, still exist. If you use the pattern shown in the following code to mark up your justified groups, then you should find that none of these inconsistencies present a problem for you:

Code Sample 36: BS3 Justified Button Sample

```

<div class="btn-group btn-group-justified" style="padding-top: 4px;">
  <div class="btn-group">
    <button type="button" class="btn btn-default">Left</button>
  </div>
  <div class="btn-group">
    <button type="button" class="btn btn-default">Middle</button>
  </div>
  <div class="btn-group">
    <button type="button" class="btn btn-default">Right</button>
  </div>
</div>
<div class="btn-group btn-group-justified" style="padding-top: 4px;">
  <div class="btn-group">
    <button type="button" class="btn btn-primary">Left</button>
  </div>
  <div class="btn-group">
    <button type="button" class="btn btn-primary">Middle 1</button>
  </div>
  <div class="btn-group">
    <button type="button" class="btn btn-primary">Middle 2</button>
  </div>
  <div class="btn-group">
    <button type="button" class="btn btn-primary">Right</button>
  </div>
</div>
<div class="btn-group btn-group-justified" style="padding-top: 4px;">
  <div class="btn-group">
    <button type="button" class="btn btn-success">Left</button>
  </div>

```

```

</div>
<div class="btn-group">
  <button type="button" class="btn btn-success">Middle 1</button>
</div>
<div class="btn-group">
  <button type="button" class="btn btn-success">Middle 2</button>
</div>
<div class="btn-group">
  <button type="button" class="btn btn-success">Middle 3</button>
</div>
<div class="btn-group">
  <button type="button" class="btn btn-success">Middle 4</button>
</div>
<div class="btn-group">
  <button type="button" class="btn btn-success">Right</button>
</div>
</div>

```

Left		Middle			Right	
Left		Middle 1	Middle 2		Right	
Left	Middle 1	Middle 2	Middle 3	Middle 4	Right	

Figure 29: Output produced by sample 36

The golden rule with the markup is that each individual button has to be wrapped in its own `<div>` with a `btn-group` class applied to it. Each of these single groups then need to be wrapped in an outer `div`, which has both `btn-group` and `btn-group-justified` applied to it.

There's one thing to note if you're using justified groups: because the buttons are rendered out as a block element, the button group-sizing classes won't work as expected. This shouldn't really come as a surprise, because they won't work on single buttons that have the `btn-block-level` class added either, just as inline elements marked and styled using block can't be fluidly controlled in general.

If your buttons are made up of `<a>` tags, then you don't need all this extra mark up—you just need the outer `div` with the `btn-group-justified` class added to it, and regular buttons inside of it.

A drop-down button with a menu attached is still created in the exact same manner as it is in BS2. First, you create an outer `<div>` with a `btn-group` class applied to it. Immediately inside this, you create a regular set of button styling classes applied to it, along with a class of `dropdown-toggle` and a `data` attribute called `toggle` with the value of `dropdown` applied to it. Immediately after the button element, but before you close the group `<div>`, you then define the actual menu using a standard, unordered list with a class of `dropdown-menu` and a role or menu applied to it.



**Note:** One of the new features that the HTML 5 specification brings to the table is something called *data attributes*. BS3 (and for that matter BS2) uses these attributes in many different places to allow elements and JavaScript functions to be joined together without the developer ever having to write a single line of JavaScript. Data attributes always take the form of *data-name*, where *name* is the name the developer wishes to assign to that attribute. Using *data-* ensures that the attributes you define will never interfere with anything the WHATWG standards body adds to the spec, giving developers utmost flexibility in what they wish to use them for.

*In most cases, these data attributes are used the way BS3 uses them: in order to pass information from the element into the JavaScript routines working on it. They can be used to pass in ID names, options, parameters for sizes, and many other things, meaning that you often don't actually have to write any boilerplate code in your app. Throughout this chapter, you'll be introduced to many more of these data attributes.*

In order for the drop-down buttons to work, you **MUST** have the drop-down JavaScript plug-in added to your BS3 build. If you've just downloaded the default set, then this won't be a problem, as `dropdown.js` is already part of the main `bootstrap.js` file. If you've done a custom build, or even compiled from the Less sources yourself, you'll need to make sure it's present.

The following code shows a basic drop-down button example:

*Code Sample 37: Dropdown Button Example*

```
<div class="btn-group">
  <button type="button" class="btn btn-success dropdown-toggle" data-
toggle="dropdown">
    A dropdown button <span class="caret"></span>
  </button>
  <ul class="dropdown-menu" role="menu">
    <li><a href="#">1st Link</a></li>
    <li><a href="#">2nd Link</a></li>
    <li><a href="#">3rd Link</a></li>
    <li class="divider"></li>
    <li><a href="#">1st Link after divider</a></li>
  </ul>
</div>
```

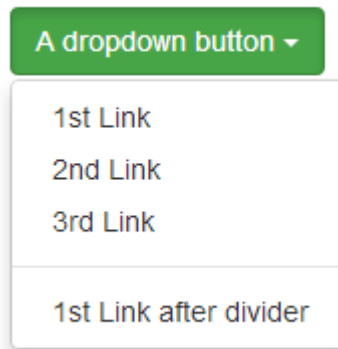


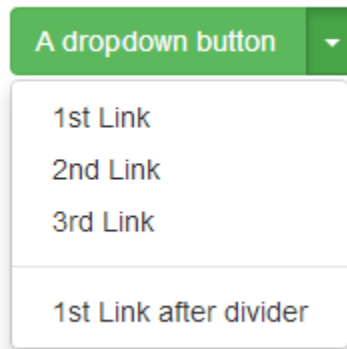
Figure 30: Output generated by code sample 37

By making a simple modification to code sample 37, you can easily turn the button into a split drop-down menu, where the action of the button remains a single action in its own right, but still allowing the menu to appear when the caret is clicked.

To do this, simply add another button element just before the button element, then move the text (but NOT the caret) from the (now) second button into the first, so the new button tag looks like this:

```
<button type="button" class="btn btn-success">A dropdown button
  <button type="button" class="btn btn-success dropdown-toggle" data-toggle="dropdown">
    <span class="caret"></span>
  </button>
  <ul class="dropdown-menu" role="menu">
    <li><a href="#">1st Link</a></li>
    <li><a href="#">2nd Link</a></li>
    <li><a href="#">3rd Link</a></li>
    <li class="divider"></li>
    <li><a href="#">1st Link after divider</a></li>
  </ul>
</button>
```

If you then re-render the output in your browser, you should see the following:



*Figure 31: Output generated by code sample 37 after it's been amended to produce a split drop-down menu*

The final thing to mention before moving on are the sizing and drop-up variations.

The sizing is simple; the sizing classes mentioned in the CSS chapter work just as expected with drop-down buttons, in exactly the same manner as they do on normal buttons (**lg** for large, no **md** since medium is the default, **sm** for small, and **xs** for extra small).

**Drop-up** is a class name that you apply to the outer element with the **btn-group** class when constructing a drop-down or split drop-down button, which, as its name implies, causes the menu to pop upwards rather than downwards.

### **A final note on drop-down menus**

Drop-down menus aren't only attached to buttons—ANY element that can be wrapped in an outer parent can be used.

You can apply a class of **dropdown** to the wrapped element with a **data** attribute of **toggle**, and the value of **dropdown**, which will cause that element to become the trigger for the drop-down menu to display.

## **Input Group Changes**

In short, there have been no changes to the input group functionality between BS2 and BS3.

Input groups are still marked up exactly as before, with the same caveats, namely:

You cannot add multiple additions on one side, only one on each end.

Multiple elements in one input group are not supported.

You cannot mix input groups with other components; you must nest them.

If you're attaching tooltips and popovers, you must add a container option, as previously mentioned under button changes

Where possible, avoid using input groups with select lists, especially when targeting WebKit-based browsers

To create an input group, simply wrap your input control and its [pre|suff]fix (which will generally be a `<span>`) in a `<div>` or other block-level element with a class type of `input-group` applied to it. Next, make sure that the additional `<span>` has a class of `input-group-addon` applied to it.

For example, the following code will create an input box for a Twitter handle, and an input box for money where you're not entering a decimal part.

Code Sample 38: BS3 Input Addons Using Input Groups

```
<div class="input-group">
  <span class="input-group-addon">@</span>
  <input type="text" class="form-control" placeholder="Username">
</div>
<br/>
<div class="input-group">
  <input type="text" class="form-control">
  <span class="input-group-addon">.00</span>
</div>
```



Figure 32: The output produced by code sample 38

The normal sizing classes also apply to these, just as they apply to other form inputs, so adding `input-group-lg` and `input-group-sm` (again, there is no `md`) to the outer input group will size the entire control set accordingly.

One thing that's not new in BS3, but has been made easier to do, is adding checkboxes, radio buttons, regular buttons, and drop-down menus to input groups.

Each of these options require a bit more markup than when you're using a regular `<span>`, but the resulting effect is that you can add interactivity to your fields. For example, you might use these inputs to create your own file upload control, or a button that creates a random password.

The following code shows how to achieve this:

Code Sample 39: Input Groups Using Buttons and Radios

```
<div class="input-group">
  <span class="input-group-addon">
```



```

    <input type="checkbox"> Keep Email Private
  </span>
  <input type="email" class="form-control" placeholder="Please enter email address">
</div>
<br/>
<div class="input-group">
  <span class="input-group-addon">
    <input type="radio"> Make Default
  </span>
  <input type="text" class="form-control" placeholder="Enter an address here">
</div>
<br/>
<div class="input-group">
  <span class="input-group-btn">
    <button class="btn btn-default" type="button">Check User Name</button>
  </span>
  <input type="text" class="form-control" placeholder="Please enter a user name
here">
</div>
<br/>
<div class="input-group">
  <input type="text" class="form-control" placeholder="Enter the password you would
like to use">
  <span class="input-group-btn">
    <button class="btn btn-success" type="button">Create Random Password</button>
  </span>
</div>
</div>

```

The rendered form consists of four input groups stacked vertically. Each group is enclosed in a light gray border. The first group contains a checkbox labeled 'Keep Email Private' and an email input field with the placeholder text 'Please enter email address'. The second group contains a radio button labeled 'Make Default' and a text input field with the placeholder text 'Enter an address here'. The third group contains a button labeled 'Check User Name' and a text input field with the placeholder text 'Please enter a user name here'. The fourth group contains a text input field with the placeholder text 'Enter the password you would like to use' and a green button labeled 'Create Random Password'.

Figure 33: Output produced by code sample 39

## Navigation Changes

The navigation components have had their own share of changes, but much like the other groups, this is mostly to tie common class names together and basically tidy things up so they are more meaningful.

There are two main sections when it comes to the navigation components: basic navigation and nav-bars. Of these two components, nav-bars have changed the most.

To begin with, in BS2, all you really had were nav-bars; there were no button classes or text classes—only the brand-label. This led to much confusion among developers and many questions, such as "How do I centrally line text up in a nav-bar?" and "How do I stop my buttons from increasing the height of my nav-bar?"

BS3 has now introduced many new classes designed especially for these kinds of scenarios, which we'll introduce soon. But first, let's look at what's different in the basic navigation components.

## Basic Navigation

First things first: the navigation list component that was in BS2 is gone. Not just deprecated, but taken away never to be seen again—and this is a good thing.

While it was easy to work with (you created an unordered list, added a sprinkle of bootstrap magic, and were good-to-go), it was messy, didn't work consistently in all browsers, and had a few rendering problems.

What do we have in its place?

We have *List groups*, which we'll cover in detail later on. For now, let's talk about what remains, as it will set the groundwork for using List groups a little later on.

So what is the base class then? It may surprise you to learn that it's simply a class called **nav**, which along with **nav-tabs**, **nav-pills**, and **nav-stacked**, can be used to make tab, pill, and sidebar-based navigation structures in the same way you used to make **nav** lists in BS2.

The markup is now much simpler. You don't need to worry about any extra padding or browser quirks, and you certainly don't need any custom stuff to get things done.

To use tabs, simply take an unordered list, and add a **nav** and **nav-tabs** class, as the following code shows:

*Code Sample 40: Tab Based Navigation*

```
<ul class="nav nav-tabs">
  <li class="active"><a href="#">Home</a></li>
  <li><a href="#">Profile</a></li>
  <li><a href="#">Messages</a></li>
</ul>
```



*Figure 34: Output produced by code sample 40*

The markup needed now consists of nothing more than a handful of `<li>` elements; the only place the classes need to be applied is on the outer `<ul>` itself. You'll also see that we have an **active** class on the first `<li>` in the list, to show us which tag is the active tag.

If you want BS3 to provide the entire tab's experience (that is, to actually change the content as required), then there's more work you need to do. For that, as we'll see in the JavaScript section, you need to use one of the JavaScript add-ons. If, however, you want to handle the tab switching and content yourself, all you need to do to switch tabs is move the active class from one `<li>` to the next.

Things are no more complicated with pills: just exchange the **nav-tabs** in code sample 40 for **nav-pills** and you should see your output change to this:



Figure 35: Output from code sample 40 when changed to "pill" mode

and if we add **nav-stacked** to either of them, our navigation will stack neatly one on top of each other:



Figure 36: Code sample 40, changed to "pill" mode, and with `nav-stacked` added

As with most other components and changes in the framework, the output now renders as a block element occupying all of the available space. Again, you just need to use the grid system and other available classes to make sure it takes up only the space you need. Now, any parent container that changes size because of differing screen resolutions will now cause its child elements to responsively resize as needed.

By the way, this also works with tabbed navigation, but the output is most likely not what you would expect:

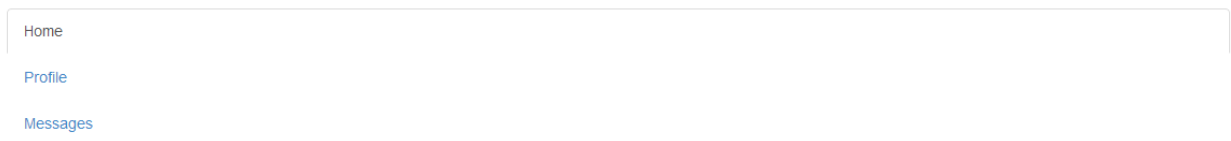
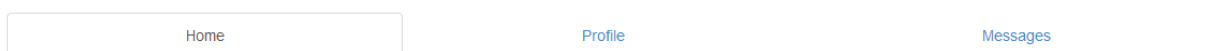


Figure 37: Code sample 40, rendered as stacked tabs

In fairness, though it wasn't really designed to work that way: tabs generally run left to right, and while BS2 did have some classes for putting tabs on the side and along the bottom of elements, these have been removed in BS3. The JavaScript plug-in for making full, tab-based outputs still has the ability to position the tabs, but the general navigation no longer does.

As with the changes made in the button components, the navigation aids also now have a new justified class added to them, called **nav-justified**, and just as with the button version, it will cause a navigation class to span the full width of a container, giving each sibling an equal size.

If we take the code in sample 40 and add **nav-justified** to the two classes already there, we get the following result:



*Figure 38: Navigation tabs from code sample 40 with full-width justification applied*

The navigation aids have another thing in common with the button classes and components: the state-based changes, which are used to show a given state, as well as the ability to take drop-down menus and sub navigation.

Simply adding a class of **disabled** to the **<li>** inside a navigation set will disable and dim that link with the **muted-text** look. Remember though, it does not actually disable the link—the link will still be clickable, and it's up to you to code things so the link does not react.

To add a drop-down menu to any pill- or tab-based navigation item, just use the same layout and classes as shown in the section on buttons. Nest an **<a>** inside an **<li>**, followed by another **<ul>**, with the appropriate classes and data attributes to make it all work.

In a bit of a cross-over between the basic navigation and navigation bars, we also have breadcrumbs and page/pagination navigation aids.

Breadcrumbs are typically used to mark your position in a site's hierarchy, and as we've seen above, using them is simply a case of adding them to an unordered list, as the following code shows:

*Code Sample 41: BS3 Breadcrumb Navigation Example*

```
<ul class="breadcrumb">
  <li><a href="#">Home</a></li>
  <li><a href="#">Library</a></li>
  <li class="active">Data</li>
</ul>
```

*Figure 39: Output produced by code sample 41*

The final controls, before we move onto full navigation bars, are the pagination controls. These controls take a huge amount of the grunt work out of lining up a bar with previous/next and before/after, and a sequence of numbers in the middle.

Typically used where you have a data-heavy site and want to display records one page at a time, rather than all at once, pagination bars are both simple to use and easy to line up with the rest of your output.

Just as with the other basic elements, creating pagination bars is as simple as adding the appropriate classes to an unordered list, as the following code shows:

Code Sample 42: BS3 Pagination Bar

```
<ul class="pagination">
  <li><a href="#">&laquo;</a></li>
  <li><a href="#">1</a></li>
  <li><a href="#">2</a></li>
  <li><a href="#">3</a></li>
  <li><a href="#">4</a></li>
  <li><a href="#">5</a></li>
  <li><a href="#">&raquo;</a></li>
</ul>
```

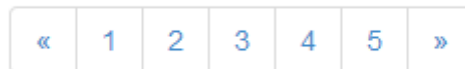


Figure 40: Output produced by code sample 42

Just as with the other basic navigation aids, applying **active** and **disabled** classes where needed to individual `<li>` elements within the list enables you to mark links as disabled and selected.

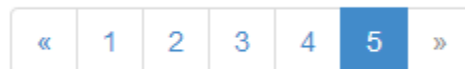


Figure 41: Output produced by code sample 42 with active and disabled classes added

Pagination controls can be sized just as easily as the other controls using the new sizing options, and like the others, they now all follow the same naming scheme: **pagination-lg** and **pagination-sm**. As with some of the others, there is no **pagination-xs**, and no hint that there should be one; likewise there's no **-md** size either, since medium is the default.

To use the pagination controls, just add the appropriate sizing class alongside the pagination class on the parent `<ul>` holding the list, as the following code shows:

Code Sample 43: Using Different Sizes for the Pagination Component

```
<ul class="pagination pagination-lg">
  <li><a href="#">&laquo;</a></li>
  <li><a href="#">1</a></li>
  <li><a href="#">2</a></li>
  <li><a href="#">3</a></li>
  <li><a href="#">4</a></li>
  <li class="active"><a href="#">5</a></li>
  <li class="disabled"><a href="#">&raquo;</a></li>
</ul>
<br/>
<ul class="pagination">
  <li><a href="#">&laquo;</a></li>
  <li><a href="#">1</a></li>
  <li><a href="#">2</a></li>
  <li><a href="#">3</a></li>
  <li><a href="#">4</a></li>
  <li class="active"><a href="#">5</a></li>
  <li class="disabled"><a href="#">&raquo;</a></li>
</ul>
<br/>
<ul class="pagination pagination-sm">
  <li><a href="#">&laquo;</a></li>
  <li><a href="#">1</a></li>
  <li><a href="#">2</a></li>
  <li><a href="#">3</a></li>
  <li><a href="#">4</a></li>
  <li class="active"><a href="#">5</a></li>
  <li class="disabled"><a href="#">&raquo;</a></li>
</ul>
```

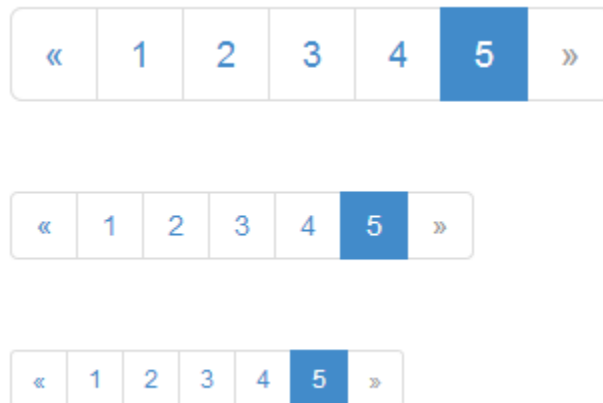


Figure 42: Output produced by code sample 43

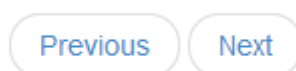
The pagination classes have one final trick up their sleeve—the generation of Previous and Next page buttons.

You'll see that in many wiki- or blog-style sites, there is usually an Older/Newer, or Previous/Next button pair at the foot of a page.

To make the default version of this, simply create a `<ul>` as we have throughout this entire section, and add a class of `pager` to it like so:

*Code Sample 44: Simple BS3 Pager*

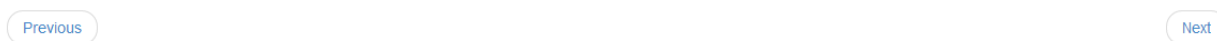
```
<ul class="pager">
  <li><a href="#">Previous</a></li>
  <li><a href="#">Next</a></li>
</ul>
```



*Figure 43: Output from code sample 44*

A pager will automatically place its buttons in the center of the available space, as seen in Figure 44, but you can also justify them left and right by adding the `previous` and `next` classes to the inner `<li>` elements.

If you change code sample 44 so that the `<li>` holding the previous button has a class of `previous` assigned, and the `<li>` holding the next button has a class of `next`, and then re-render the output, you should see your browser change to the following:



*Figure 44: Code sample 44 with optional `previous` and `next` classes applied*

You'll see that, again, the output expands responsively to fill all of the available space, and as before, you can easily control this using grids, spans, and other containers as needed.

## Navbar Navigation

If there's one thing that BS3 does way better than any of the other frameworks I've seen out there, it's navigation bars. Whether it's a drop-down menu, a title with sign-in/out controls, or just a decoration, there's no denying that navigation bars in Bootstrap are powerful.

Unfortunately, in BS2 there was quite a bit lacking. There were many hacks published to allow things to be lined up, but some of the CSS used was a little rough around the edges. BS3 changes all of this.

The basic navigation bar now starts its life as a collapsed item in your pages display. This means that if the first display is on a mobile device, your design will start out with a nicely collapsed menu bar, ready to be expanded exactly as a mobile user would expect. In BS2, the opposite was the case, and you had to take extra steps to collapse the display before making it visible.

The navigation bar will progressively become visible as the display width increases, until the entire unit in full has enough space to render horizontally across the page.

Be careful though—even with a non-fluid column width of 1024px, and a fluid width of your entire display, it's still possible to run out of space. When this happens, you could end up with some major content-overflow scenarios. For example, it's not difficult to push your navigation bar two or more rows in height.

The recommended way to ensure this doesn't happen is to use custom styling to control the width of your elements, and/or to use the responsive visibility classes to control what is and is not shown for different screen sizes.

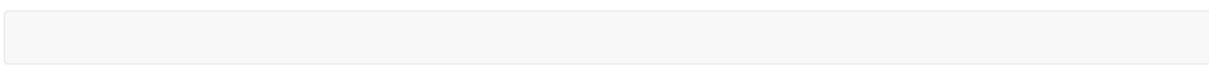
If you're using Less, you can now also customize the `@grid-float-breakpoint` variable to control the trigger point where your navigation bar collapses, or you can just customize the appropriate media queries in the base CSS.

Navigation bars require JavaScript to be enabled to work correctly; if it's not enabled, the bar should still display, but the collapsing won't work correctly. You also need to ensure that if you've done a custom build, you've also included the collapse JavaScript plug-in.

The following code sample shows how to create a basic empty, collapsible navigation bar:

*Code Sample 45: Simple Empty BS3 Collapsible Navigation Bar*

```
<nav class="navbar navbar-default" role="navigation">
  <div class="container-fluid">
    <div class="navbar-header">
      <button type="button" class="navbar-toggle" data-toggle="collapse" data-
target="#bs-example-navbar-collapse-1">
        <span class="sr-only">Toggle navigation</span>
        <span class="icon-bar"></span>
        <span class="icon-bar"></span>
        <span class="icon-bar"></span>
      </button>
    </div>
    <div class="collapse navbar-collapse" id="bs-example-navbar-collapse-1">
    </div><!-- /.navbar-collapse -->
  </div><!-- /.container-fluid -->
</nav>
```



*Figure 45: Blank nav bar produced by code sample 45*



It may not look like much, but this bar will collapse and expand as needed for different sizes, and all we need to do now is to add some content to it. Let's start with branding.

If you're going to brand your navigation using the **navbar-brand** class on an `<a>` link, then the BS3 best practice is to group this element with the **navbar-toggle** element that builds the three-line collapse icon. This ensures that, when collapsing, the text shrinks down as far as it can before becoming invisible and stacking vertically.

To make this change, in code sample 45, simply add:

```
<a class="navbar-brand" href="#">BS3</a>
```

Immediately after the closing button tag that holds the three `<span>` elements with a class of **icon-bar**, and then refresh your browser. The result should now look like this:

A screenshot of a browser showing the text "BS3" in a navigation bar. The text is centered and appears to be part of a larger navigation structure.

*Figure 46: Blank nav bar with a brand added*

If you hover over the brand text, you'll see that it's also an active link, generally used to return to a site's home page. You can also put an image in here too, but there are no specific classes here to help you. I've done this in some designs, but unfortunately, it still takes a bit of manual work. If you make the image about the same height as the `<H1>` element, things stay lined up nicely. Otherwise, the best way I've found to make it work is to put some padding in front of the element holding the brand name, then absolute-position your logo with suitable bottom margins/padding so that it does not hide any content below.

Moving on from the brand class, we also have specific classes for forms, buttons, and text, as follows:

**navbar-nav**: for creating a main `<ul>` of navigation links and dropdowns

**navbar-form**: for creating inline miniature forms

**navbar-btn**: for marking button components up to be navbar friendly

**navbar-text**: for including independent lines of text in your bar

**navbar-link**: for adding independent links that are not part of the regular navigation items

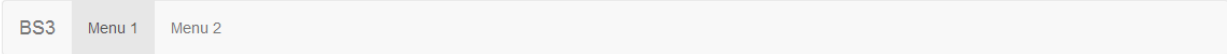
You can use them all together, on their own as needed, or combine them with the new emerging web components specification and create a single bar that takes a configuration and draws what it needs, as it needs.

The following code sample expands sample 45 to include some navigation links:

Code Sample 46: Previous Example Expanded to Include Basic Links

```
<nav class="navbar navbar-default" role="navigation">
  <div class="container-fluid">
    <div class="navbar-header">
      <button type="button" class="navbar-toggle" data-toggle="collapse" data-
target="#bs-example-navbar-collapse-1">
        <span class="sr-only">Toggle navigation</span>
        <span class="icon-bar"></span>
        <span class="icon-bar"></span>
        <span class="icon-bar"></span>
      </button>
      <a class="navbar-brand" href="#">BS3</a>
    </div>

    <div class="collapse navbar-collapse" id="bs-example-navbar-collapse-1">
      <ul class="nav navbar-nav">
        <li class="active"><a href="#">Menu 1</a></li>
        <li><a href="#">Menu 2</a></li>
      </ul>
    </div><!-- /.navbar-collapse -->
  </div><!-- /.container-fluid -->
</nav>
```



BS3 Menu 1 Menu 2

Figure 47: Output produced by code sample 46

As with all the other navigation, link classes, and elements, you'll see we've employed the **active** class to mark the first link as active, and like the others, you can apply the **disabled** and other similar classes to mark up items as needed.

It's just as easy to create a menu that expands down from any of the buttons. All you have to do is use the same pattern of markup as you do when creating drop-down menus on buttons, except you wrap it in a **navbar-nav** parent element like so:

Code Sample 47: Navigation Bar Expanded to Hold a Drop-Down Menu Button

```
<nav class="navbar navbar-default" role="navigation">
  <div class="container-fluid">
    <div class="navbar-header">
      <button type="button" class="navbar-toggle" data-toggle="collapse" data-
target="#bs-example-navbar-collapse-1">
        <span class="sr-only">Toggle navigation</span>
        <span class="icon-bar"></span>
        <span class="icon-bar"></span>
        <span class="icon-bar"></span>
      </button>
      <a class="navbar-brand" href="#">BS3</a>
    </div>
```

```

<div class="collapse navbar-collapse" id="bs-example-navbar-collapse-1">
  <ul class="nav navbar-nav">
    <li class="dropdown">
      <a href="#" class="dropdown-toggle" data-toggle="dropdown">Menu 1 <b
class="caret"></b></a>
      <ul class="dropdown-menu">
        <li><a href="#">Sub Menu 1</a></li>
        <li><a href="#">Sub Menu 2</a></li>
        <li><a href="#">Sub Menu 3</a></li>
        <li class="divider"></li>
        <li><a href="#">Sub Menu 4</a></li>
        <li class="divider"></li>
        <li><a href="#">Sub Menu 5</a></li>
      </ul>
    </li>
  </ul>
</div><!-- /.navbar-collapse -->
</div><!-- /.container-fluid -->
</nav>

```

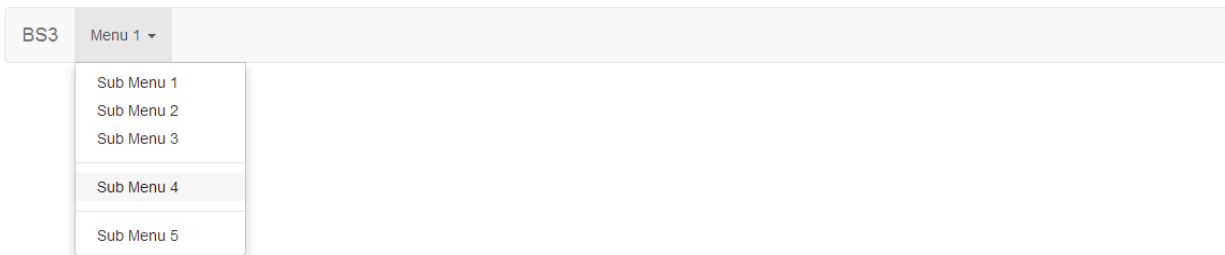


Figure 48: Drop-down menu produced by code sample 47

You can combine each of these, so it's simple to have drop-down and non-drop-down links in one `navbar-nav` `<ul>` just by marking up each `<li>` element as needed.

Another common requirement in navigation bars is the obligatory sign-in form, generally used to sign in to a protected area on a site.

BS3 makes these forms even easier to construct using the `navbar-form` class. So far, we've seen that all our elements are left-aligned in the navigation bar's inner-content area. Since sign-in forms are usually right-aligned, the BS3 team has provided the `navbar-right` class to allow you to do just such a thing. The following code illustrates an example:

Code Sample 48: BS3 Navigation Bar with a Sign-In Form Right Aligned

```

<nav class="navbar navbar-default" role="navigation">
  <div class="container-fluid">
    <div class="navbar-header">
      <button type="button" class="navbar-toggle" data-toggle="collapse" data-
target="#bs-example-navbar-collapse-1">
        <span class="sr-only">Toggle navigation</span>
        <span class="icon-bar"></span>
        <span class="icon-bar"></span>

```

```

    <span class="icon-bar"></span>
  </button>
  <a class="navbar-brand" href="#">BS3</a>
</div>
<div class="collapse navbar-collapse" id="bs-example-navbar-collapse-1">

  <form class="navbar-form navbar-right">
    <div class="form-group">
      <input type="text" class="form-control" placeholder="Sign-in
name">
    </div>
    <div class="form-group">
      <input type="password" class="form-control"
placeholder="Password">
    </div>
    <button type="submit" class="btn btn-default">Sign In</button>
  </form>

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

```

BS3

Sign-in name

Password

Sign In

Figure 49: Navigation bar with sign-in form, produced by code sample 48

Along the same lines, it's often also required that when a person is signed in to your application, that they have their name shown instead of the form, and optionally, a link and/or button to sign-out, or change options.

The button, text, and link classes have all of this covered, as shown in the following code:

Code Sample 49: Swapping the Sign-In Form for an Information Display

```

<nav class="navbar navbar-default" role="navigation">
  <div class="container-fluid">
    <div class="navbar-header">
      <button type="button" class="navbar-toggle" data-toggle="collapse" data-
target="#bs-example-navbar-collapse-1">
        <span class="sr-only">Toggle navigation</span>
        <span class="icon-bar"></span>
        <span class="icon-bar"></span>
        <span class="icon-bar"></span>
      </button>
      <a class="navbar-brand" href="#">BS3</a>
    </div>
    <div class="collapse navbar-collapse" id="bs-example-navbar-collapse-1">
      <div class="navbar-right">
        <p class="navbar-text">Signed in as <strong>Peter Shaw</strong></p>
        <a href="#" class="navbar-link">Options</a>&nbsp;&nbsp;&nbsp;
        <button class="btn btn-primary navbar-btn">Sign Out</button>
      </div>
    </div>
  </div>
</nav>

```

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

BS3

Signed in as Peter Shaw Options Sign Out

Figure 50: Navigation bar with information display produced by code sample 49

The navigation bar itself also has a few clever tricks, you can place your navigation bar inside a **container** or other alignment class, then add **navbar-fixed-top** to it, your navigation bar will stick to the top of its container, and align itself perfectly with it as in the following image:

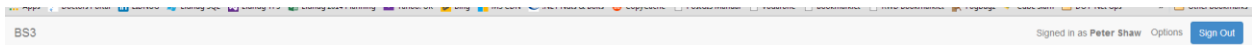


Figure 51: Navigation bar from sample 49 with the *navbar-fixed-top* class applied to it

You can also add **navbar-fixed-bottom** class, which will stick your navigation bar to the bottom of the page, and the **navbar-static-top** class, which will stick the bar to the top of the container, but will allow it to scroll with the page content.

Finally, if you want a darkened navigation bar, then you can use the **navbar-inverse** class, which will invert the color set used by the entire component:

BS3

Signed in as Peter Shaw Options Sign Out

Figure 52: Navbar from sample 49 with the *inverse* class set

## Label and Badge Changes

This is going to be a very short section, because there have only been two changes to the labels, and one big one to the badges.

To make a label in BS2, we simply gave it a **label-xxxx** class name, where **xxxx** represents the state the label was to portray. Under BS3, we now have a two-part class definition, and a rename from **error** to **danger** for the red color class, so that the naming now matches everything else, and that's pretty much it.

There are also no sizing classes/options for these in BS3, as labels take on the size of the surrounding container. So, if you create a **<span>** with an appropriate label class, and then wrap it in a **<H1>**, the size of that label will be much larger than the rest.

The following code sample demonstrates this:

Code Sample 50: BS3 Label Example with Different Sizes

```
<h1><span class="label label-default">Default</span></h1>
```

```
<h2><span class="label label-primary">Primary</span></h2>
<h3><span class="label label-success">Success</span></h3>
<h4><span class="label label-info">Info</span></h4>
<h5><span class="label label-warning">Warning</span></h5>
<h6><span class="label label-danger">Danger</span></h6>
```

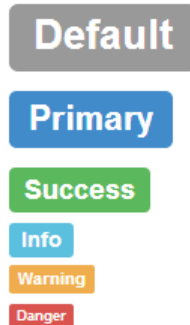


Figure 53: Output produced by code sample 50

Badges, on the other hand, have undergone a bigger change.

In BS3, badges no longer have contextual color classes. That is, there is no longer a **label-success** or **label-warning**, as there is with the previous labels and with other elements.

This means you can't color badges in the same way you could in BS2, and any label you create can only be rendered using the standard grey color. Well, at least officially you can't.

If you apply an **alert-xxxxx** alongside a **label** class on the **<span>** being used for your label, then the label will take on the coloring of that alert class, and while this is not a supported way of doing things, it does work, and allows you to use the labels as you did in BS2.

To mark up a standard label, all you need to do is add a **label** and **'label-default'** class to a **<span>**, and you're ready to go:

Code Sample 51: Standard Label Example

```
<span class="label label-default">This is a label</span>
```

A single grey label with the text 'This is a label' centered inside.

Figure 54: Label produced by code sample 51

If you want to hack your labels to use the different colors, then you can do so as follows:

Code Sample 52: Label Example Hacked to Use Non-Standard Coloring

```
<span class="badge">Normal</span>
<span class="badge alert-success">Success</span>
```

```
<span class="badge alert-info">Info</span>
<span class="badge alert-warning">Warning</span>
<span class="badge alert-danger">Danger</span>
```



Figure 55: Output produced by code sample 52

One nice feature the badge component does still have, however, is its use of the **empty:** pseudo-selector, allowing it to automatically vanish from display if its inner text is empty. This helps with the internal changes designed to work on List groups and Pill lists, because it now means that it's immensely easy to create things like inbox notification lists, where the values disappear when the **<span>** contents are removed. Here's an example:

Code Sample 53: BS3 List Styling for Badges

```
<div class="col-md-4">
  <ul class="nav nav-pills nav-stacked">
    <li class="active">
      <a href="#">
        <span class="badge pull-right">42</span>
        Inbox
      </a>
    </li>
    <li class="active">
      <a href="#">
        <span class="badge pull-right">10000</span>
        Spam
      </a>
    </li>
    <li class="active">
      <a href="#">
        <span class="badge pull-right"></span>
        Lottery Wins
      </a>
    </li>
  </ul>
</div>
```



Figure 56: Output produced by code sample 53

If you examine the code, you'll see that the last option in the list has no value inside its span tag, which, because of the **empty:** pseudo-selector, causes the browser not to render it. However, the second you put anything in there using JavaScript or any method that can manipulate the DOM, that badge will pop right back into existence without missing a beat. Notice also the use of the **pull-right** alignment class to make sure the label sits on the right side of the element, making everything look nice and aligned.

## List Group Changes

List groups are a new thing added in BS3, designed to replace the BS2 navigation lists, and have far more capabilities than their BS2 counterpart.

Once you start to use them, you'll realize that list groups are more like fully styleable list-item boxes, and to be honest, all it would take to create a fully styleable list box would in fact be to wrap them in their own **div** and set the overflow to scroll appropriately.

A basic list group component can be created with markup similar to the following:

*Code Sample 54: Basic BS3 List Group*

```
<ul class="list-group">
  <li class="list-group-item">Cheese</li>
  <li class="list-group-item">Burger</li>
  <li class="list-group-item">Bun</li>
  <li class="list-group-item">Pickles</li>
  <li class="list-group-item">Tomato</li>
</ul>
```

Cheese
Burger
Bun
Pickles
Tomato

*Figure 57: Output produced by code sample 54*

As you saw previously in the badges section, you can add badges to lists and other objects, and if you add them to a list group, they line up perfectly:

*Code Sample 55: List Group with Badges Attached*

```
<ul class="list-group">
  <li class="list-group-item"><span class="badge pull-right">4</span>Cheese</li>
  <li class="list-group-item"><span class="badge pull-right">2</span>Burger</li>
  <li class="list-group-item"><span class="badge pull-right">1</span>Bun</li>
  <li class="list-group-item"><span class="badge pull-right"></span>Pickles</li>
</ul>
```



```
<li class="list-group-item"><span class="badge pull-right">2</span>Tomato</li>
</ul>
```

Cheese	4
Burger	2
Bun	1
Pickles	
Tomato	2

Figure 58: Output from code sample 55

The most observant of you may be thinking, “Ok, so list groups look good, but there just still `<ul>`s underneath the surface, there's really nothing that special about them.”

Well, maybe you're right about that...or maybe we need to experiment a bit more.

List groups, unlike the classes we saw in the navigation sections, cannot just be applied to an unordered list. A list group can be applied to any parent container, and when done so, the styling will cause all of the children of that container to become linked. Take a look at the following example:

Code Sample 56: List Group Created Using `<a>` Tags

```
<div class="list-group">
  <a class="list-group-item">Cheese</a>
  <a class="list-group-item">Burger</a>
  <a class="list-group-item">Bun</a>
  <a class="list-group-item">Pickles</a>
  <a class="list-group-item">Tomato</a>
</div>
```

Cheese
Burger
Bun
Pickles
Tomato

Figure 59: Output produced by code sample 56

As code sample 56 shows, we can create a list of links using nothing more than a `<div>` surrounding the list, and that list then instantly becomes a list-style menu with a light grey hover effect. However, because the inner tags are `<a>` tags, we can now go even further:

Code Sample 57: List Group Expanded to Use More than Just Simple Lists

```
<div class="list-group">
  <a class="list-group-item active">
```

```

<strong>Cheese</strong>
<p>A delicious slice of ...</p>

</a>
<a class="list-group-item">
  <strong>Burger</strong>
  <p>Prime Aberdeen angus beef ...</p>
  
</a>
<a class="list-group-item">
  <strong>Bun</strong>
  <p>A freshly baked soft sesame seed bun ...</p>
  
</a>
<a class="list-group-item">
  <strong>Pickles</strong>
  <p>The finest pickles from the finest purveyors ...</p>
  
</a>
<a class="list-group-item">
  <strong>Tomato</strong>
  <p>Fresh organic tomatoes picked directly from the vine ...</p>
  
</a>
</div>

```

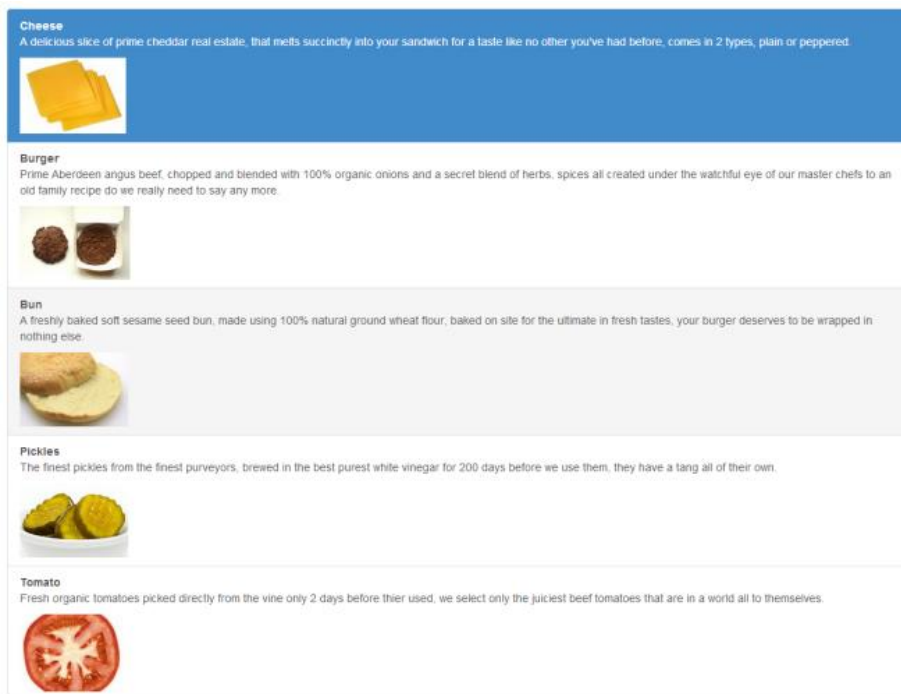


Figure 60: Output produced by code sample 57

Notice that we still get the full, cell-hover effect when your mouse hovers over a link, and by simply adding the **active** class, as we have in other examples, we can mark an item as being the active one, and all while maintaining a single set of borders and the appearance of each item behaving just like a regular `<ul>` element.

But why stop there?

Just as with any other element in BS3, list groups also have their own contextual classes to give them meaning in the standard color set:

*Code Sample 58: BS3 List Group with Contextual Color Classes*

```
<div class="list-group">
  <a class="list-group-item list-group-item-success">
    <strong>Cheese</strong>
    <p>...</p>
    
  </a>
  <a class="list-group-item list-group-item-success">
    <strong>Burger</strong>
    <p>...</p>
    
  </a>
  <a class="list-group-item list-group-item-success">
    <strong>Bun</strong>
    <p>...</p>
    
  </a>
  <a class="list-group-item list-group-item-danger">
    <strong>Pickles</strong>
    <p>...</p>
    
  </a>
  <a class="list-group-item list-group-item-warning">
    <strong>Tomato</strong>
    <p>...</p>
    
  </a>
</div>
```



Figure 61: Output produced by code sample 58

As you can see, what use is a burger without cheese and a bun? Tomatoes I can take or leave, but please hold the pickle!

## Media Objects and Custom Thumbnail Changes

We covered the basic thumbnail changes for the styling back in the chapter on CSS, but what we didn't get into are the changes in the surrounding components.

The amalgamation that now exists in BS3 allows us to combine the thumbnail classes with the grid classes to create easy-to-use image lists. The most basic way we can use them is to use the **row** class and grid spans, along with the basic thumbnail classes, to make perfectly lined-up thumbnails, as the following shows:

Code Sample 59: Basic Thumbnails with Grid Classes

```
<div class="row">
  <div class="col-md-3">
    <a href="#" class="thumbnail">
      
    </a>
```

```

</div>
<div class="col-md-3">
  <a href="#" class="thumbnail">
    
  </a>
</div>
<div class="col-md-3">
  <a href="#" class="thumbnail">
    
  </a>
</div>
<div class="col-md-3">
  <a href="#" class="thumbnail">
    
  </a>
</div>
</div>
</div>

```

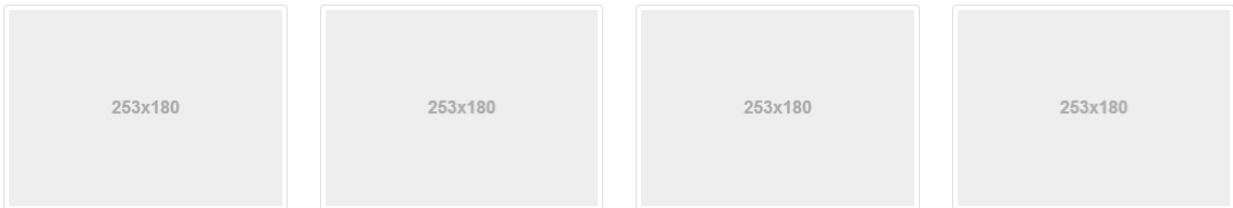


Figure 62: Thumbnails produced by code sample 59

For those who are wondering, I'm using **holder.js** to produce the image place holders. I'll let you Google the location (it'll come up as the first available); it's a great tool for reserving image space, especially when doing mockups for clients.

This example is nothing really special; if you've been reading the rest of this book, you've likely already figured out how easy it is to do this. However, if we add a little more markup and work with the same idea, we can easily produce something like the following:

Code Sample 60: BS3 Thumbnails with Extra Content

```

<div class="row">
  <div class="col-md-3">
    <div class="thumbnail">
      
      <div class="caption">
        <h3>Ecstasy requires exploration</h3>
        <p>The planet is radiating pulses. This life is nothing short of a blossoming
oasis of transformative potentiality outside of the being.</p>
        <p>
          <a href="#" class="btn btn-success" role="button">Like</a>
          <a href="#" class="btn btn-danger" role="button">Dislike</a>
        </p>
      </div>
    </div>
  </div>
</div>

```

```

<div class="col-md-3">
  <div class="thumbnail">
    
    <div class="caption">
      <h3>We exist as frequencies</h3>
      <p>This life is nothing short of a blossoming uprising of frequency
aspiration. Potential is the richness of conscious living, and of us.</p>
      <p>
        <a href="#" class="btn btn-success" role="button">Like</a>
        <a href="#" class="btn btn-danger" role="button">Dislike</a>
      </p>
    </div>
  </div>
</div>
<div class="col-md-3">
  <div class="thumbnail">
    
    <div class="caption">
      <h3>To traverse the myth is to become one</h3>
      <p>It can be difficult to know where to begin. The totality is calling to you
via sub-atomic particles. Can you hear it?</p>
      <p>
        <a href="#" class="btn btn-success" role="button">Like</a>
        <a href="#" class="btn btn-danger" role="button">Dislike</a>
      </p>
    </div>
  </div>
</div>
<div class="col-md-3">
  <div class="thumbnail">
    
    <div class="caption">
      <h3>We reflect, we heal, we are reborn</h3>
      <p>Through reiki, our essences are nurtured by purpose. You will soon be
guided by a power deep within yourself.</p>
      <p>
        <a href="#" class="btn btn-success" role="button">Like</a>
        <a href="#" class="btn btn-danger" role="button">Dislike</a>
      </p>
    </div>
  </div>
</div>
</div>
</div>

```

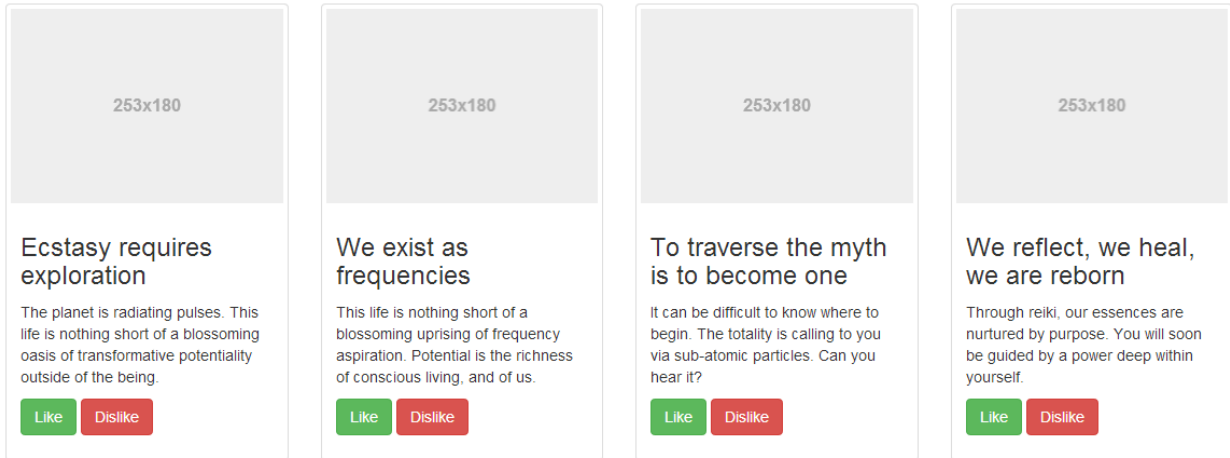


Figure 63: Output produced by code sample 60

Creating vertical lists is just as easy, but instead of using the `thumbnail` classes, we now need to use the `media` object classes. These classes and the associated markup create a layout similar to the previous example, but instead of appearing underneath the thumbnail, the text is lined up to the right.

The primary design motive here is in comment and message lists with an avatar image, but they can be used for news items, product lists, and many other things.

To mark up a media object, simply use an outer `div` with a class of `media`. Then, inside of that, use the `media-object`, `media-body`, and `media-heading` classes to mark up the individual bits, as the following example shows:

Code Sample 61: Mark Up for a Single BS3 Media Object

```
<div class="media">
  <a class="pull-left" href="#">
    
  </a>
  <div class="media-body">
    <h4 class="media-heading">To follow the journey is to become one with it</h4>
    <p>The goal of ultrasonic energy is to plant the seeds of awareness rather than
    stagnation. You and I are beings of the quantum matrix. Purpose is a constant.</p>
  </div>
</div>
```

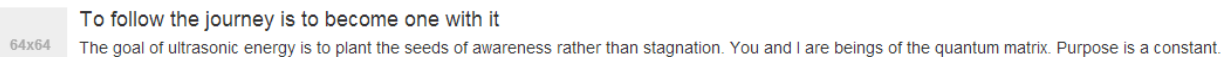


Figure 64: Media object produced by code sample 61

As you can probably already imagine, it wouldn't take much to add these into a List group, or some other structure built up using the BS3 grid system. BS3 actually makes it much easier than that by providing a dedicated media object list class and supported markup.

By changing your markup just a little, you can generate full lists of media objects, as follows:

Code Sample 62: Full Media Object List

```
<ul class="media-list">
  <li class="media">
    <a class="pull-left" href="#">
      
    </a>
    <div class="media-body">
      <h4 class="media-heading">Humankind has nothing to lose</h4>
      <p>Awareness is the growth of synchronicity, and of us. This life is nothing
short of an unfolding explosion of infinite truth. We exist as sonar energy.</p>
    </div>
  </li>
  <li class="media">
    <a class="pull-left" href="#">
      
    </a>
    <div class="media-body">
      <h4 class="media-heading">It is a sign of things to come</h4>
      <p>Humankind has nothing to lose. We are at a crossroads of rebirth and
discontinuity. We are in the midst of a consciousness-expanding maturing of guidance
that will clear a path toward the quantum soup itself.</p>
    </div>
  </li>
  <li class="media">
    <a class="pull-left" href="#">
      
    </a>
    <div class="media-body">
      <h4 class="media-heading">Only a traveller of the galaxy may generate this
canopy of life</h4>
      <p>We must learn how to lead unified lives in the face of delusion. Eons from
now, we warriors will reflect like never before as we are reborn by the quantum soup.
We must beckon ourselves and fulfill others.</p>
    </div>
  </li>
</ul>
```




-  **Humankind has nothing to lose**  
Awareness is the growth of synchronicity, and of us. This life is nothing short of an unfolding explosion of infinite truth. We exist as sonar energy.
-  **It is a sign of things to come**  
Humankind has nothing to lose. We are at a crossroads of rebirth and discontinuity. We are in the midst of a consciousness-expanding maturing of guidance that will clear a path toward the quantum soup itself.
-  **Only a traveller of the galaxy may generate this canopy of life**  
We must learn how to lead unified lives in the face of delusion. Eons from now, we warriors will reflect like never before as we are reborn by the quantum soup. We must beckon ourselves and fulfill others.

Figure 65: Full media object list generated by code sample 62



If you nest the `<li>`s and `<ul>`s inside the media list, BS3 will automatically indent them to give the effect of a parent/child hierarchy. You can also use different combinations of **pull-right** and **pull-left** where needed to position the image on the opposite side, for example. Or, to line things up slightly differently, you can combine them with panels (which we'll see shortly) and other similar structures to put borders and other contextual hints around the entire list and wrap them inline to produce composite components.

## Panel Changes

Like list groups, panels are a new addition to BS3—and what an addition they are!

We're all familiar with things like forms, for example, being in group boxes, or sections of a website being logically divided up from each other. BS3 panels allow you to easily control this logical separation.

You can have plain old panels, panels with headers, panels with titles in headers, panels with footers, and panels with their own set of contextual colors.

A basic panel is constructed using just two `<div>` elements and three CSS classes:

*Code Sample 63: Basic Panel Object*

```
<div class="panel panel-default">
  <div class="panel-body">
    Panel content goes here
  </div>
</div>
```

However, it's not much good without a title, and really as it stands in code sample 63, it's nothing more than a `div` with padding and a border around it.

If we add a third `<div>` element, however, and give that `<div>` a class of **panel-heading**, then we start to get something a bit more interesting:

*Code Sample 64: Basic Panel Object with a Panel Header*

```
<div class="panel panel-default">
  <div class="panel-heading">Panel Header</div>
  <div class="panel-body">
    Panel content goes here
  </div>
</div>
```

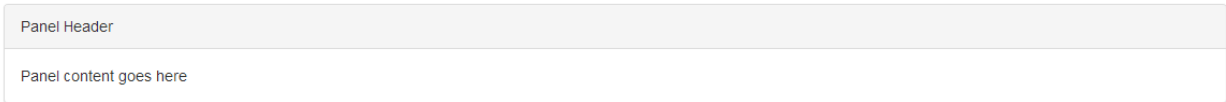


Figure 66: Panel produced by code sample 64

Your panel header doesn't have to be just a simple `<div>`, either; you can include `<hx>` tags along with any other regular tag, such as an `<img>`, and things will resize as required:

Code Sample 65: Panel with an H1 Tag as a Panel Title

```
<div class="panel panel-default">
  <div class="panel-heading">
    <h1 class="panel-title">Panel Header</h1>
    <p>Panel sub header</p>
  </div>
  <div class="panel-body">
    <p>Panel content goes here</p>
    <p>Panel content goes here</p>
    <p>Panel content goes here</p>
  </div>
</div>
```

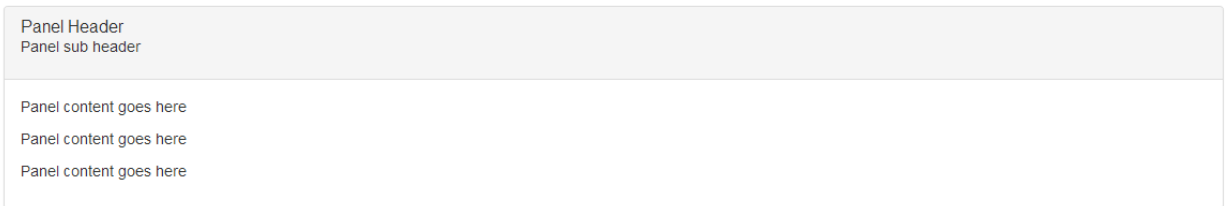


Figure 67: Output produced by code sample 65

You can also apply a panel footer by adding a `<div>` element after your panel body and applying the class `panel-footer`. If you want to give your panels meaning, you can also use the contextual colors that are used for everything else. You do this simply by replacing the `panel-default` class in the previous code samples with `panel-primary`, `panel-success`, `panel-info`, `panel-warning`, or `panel-danger`.

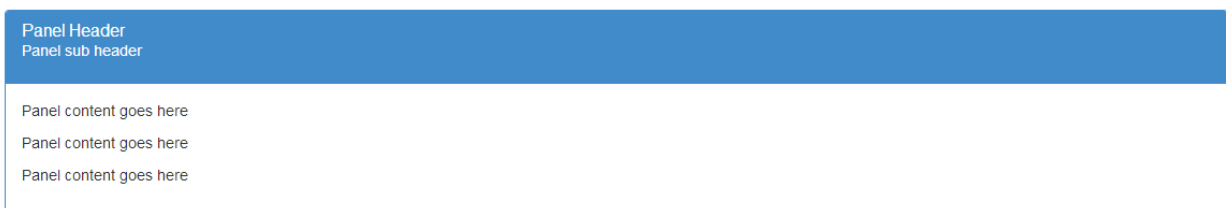


Figure 68: Output produced by code sample 65, with the type set to `panel-primary`

One thing to be careful of, though—if you're using **panel-footer** and contextual colors, the footer will NOT adopt the panel's color scheme. This is deliberate, as the BS3 developers felt that contextually, only the contents of the panel were important. Any buttons, controls, or footer information to respond to that information did not need to be highlighted in the same way, thus allowing the developer freedom to add different contextual colors for different controls as required.

Panels can also work fluidly with list groups and tables. If you join a table with a panel inside of the parent **div**, you get a panel as shown previously, and a seamless join to the table below it, as the following shows:

*Code Sample 66: Panel from Sample 65 Colored as Primary and with a Table Applied*

```
<div class="panel panel-primary">
  <div class="panel-heading">
    <h1 class="panel-title">Panel Header</h1>
    <p>Panel sub header</p>
  </div>
  <div class="panel-body">
    <p>Panel content goes here</p>
    <p>Panel content goes here</p>
    <p>Panel content goes here</p>
  </div>
  <table class="table">
    <thead>
      <tr>
        <th>#ID</th>
        <th>Name</th>
        <th>Twitter Handle</th>
      </tr>
    </thead>
    <tbody>
      <tr>
        <td>1</td>
        <td>Peter Shaw</td>
        <td>@shawty_ds</td>
      </tr>
      <tr>
        <td>2</td>
        <td>Digital Solutions UK</td>
        <td>@digitalsolut_uk</td>
      </tr>
    </tbody>
  </table>
</div>
```

Panel Header		
Panel sub header		
Panel content goes here		
Panel content goes here		
Panel content goes here		
#ID	Name	Twitter Handle
1	Peter Shaw	@shawty_ds
2	Digital Solutions UK	@digitalsolut_uk

Figure 69: Output produced by code sample 66

If you remove the `<div>` containing the panel body from your markup, the whole thing will collapse down, leaving just the panel header and border, and leading straight into the table—great for just providing tables with headers, if that’s what you need.

Adding a list group into your panel is also just as easy, and performed in exactly the same way. I’ll leave that as an exercise to the reader to play with, however.

## Other changes

Before we finally leave the chapter on components behind, there are a few more things to cover: the jumbotron, alert boxes, progress bars, and wells.

The jumbotron is not a new element, but its use in BS2 was slightly more messy than it now is in BS3. Creating a jumbotron is very simple; you simply use a `<div>` with the class `jumbotron` applied, and then add some optional markup inside. What markup you use is entirely up to you, but to get the intended effect, the recommended markup is:

Code Sample 67: BS3 Jumbotron Example

```
<div class="jumbotron">
  <h1>Bootstrap 3</h1>
  <p>Don't you just love this framework? It's elegant, simple to use, and frees you
from so much boilerplate code.</p>
  <p><a class="btn btn-primary btn-lg" role="button">Heck yes I do</a></p>
</div>
```

# Bootstrap 3

Don't you just love this framework? It's elegant, simple to use, and frees you from so much boilerplate code.

Heck yes I do

Figure 70: Jumbotron produced by code sample 67

And that's it. You can remove the rounded corners and make it full-width by swapping the order and moving it outside its container, but there are no special classes or optional colors for this—it's designed to be big and bold, and catch your attention.

Speaking of catching attention, we also have a number of alert classes, and the only real change between the BS2 version and BS3 is the renaming of the class name **alert-error** to **alert-danger**. Other than that, the markup to produce alerts is still as simple as using a standard `<div>` with the appropriate contextual classes added:

Code Sample 68: Contextual Alert Classes

```
<div class="alert alert-success">You did something ...</div>
<div class="alert alert-info">Something happened you ...</div>
<div class="alert alert-warning">Something happened that wasn't ...</div>
<div class="alert alert-danger">You did something that was not liked ...</div>
```

You did something that was immensely successful.

Something happened you should be informed about, but you don't need to do anything.

Something happened that wasn't really good, perhaps you should take a look just in case.

You did something that was not liked at all and for that you need to be told about it.

Figure 71: Alert boxes produced by code sample 68

If you want to add links in your alerts using a standard `<a>` tag, make sure you apply the class **alert-link** to the anchor tag; this will ensure that the link coloring remains consistent with the contextual color class being used.

You can also add a cross/dismiss icon to the alert, allowing the user to close it and make it vanish from the display. To do this, we need to add a `<button>` element marked up with a sprinkle of data attributes and extra classes as follows:

Code Sample 69: A Dismissible Alert Box

```
<div class="alert alert-danger alert-dismissible">
  <button type="button" class="close" data-dismiss="alert" aria-
hidden="true">&times;</button>
  <strong>YO!</strong> You got that operation very, very wrong indeed.
</div>
```




Figure 72: Dismissible alert produced by code sample 69

We added the extra class **alert-dismissible** to the outer **<div>**, and then used a **dismiss** data attribute to connect the JavaScript action to the alert's **dismiss** method, using a simple button.

If alerts provide operational feedback, then there's one other feedback-related component that's a must-have—the progress bar.

No feedback element has produced so much hatred or salutation as this humble, little colored bar. From BS2 to BS3, the only changes made here, like with the alert classes, are the renaming of the contextual colors to match the overall scheme renames elsewhere in the framework.

The base HTML markup to produce them remains exactly the same:

Code Sample 70: Basic Progress Bar

```
<div class="progress">
  <div class="progress-bar" role="progressbar" aria-valuenow="60" aria-valuemin="0"
aria-valuemax="100" style="width: 60%;">
    <span class="sr-only">60% Complete</span>
  </div>
</div>
```



Figure 73: Progress bar produced by sample 70

One thing you really should pay attention to, however (and something that I have reiterated at various points in this book), is the group of attributes and extras used to make this element friendly to screen readers and similar equipment.

Since a progress bar often tends to be purely graphical in nature, it means nothing to someone who's unable to read the screen. Most other elements have at least enough text to give the reader an idea of what's there; progress bars need all the help they can get. You'll see from the previous example that we've added extra aria values to report on what percentage the value is at, and we've also marked up a span that's for screen readers only, specifically to give an audible report.

If we remove the **sr-only** class, however, it makes our bar look a bit better to those who can see it too:



Figure 74: Progress bar with the *sr-only* class removed from the inner span

It goes without saying that you can also add **progress-bar-success**, **progress-bar-info**, **progress-bar-warning**, and **progress-bar-danger** to the inner `<div>` with the **progress-bar** class, in order to take advantage of the contextual colors available.

You can also add **progress-striped** and/or **active** to the outer `<div>` alongside the **progress** class in order to get striped and animated effects on your progress bars. A stacked progress bar effect can be obtained by placing multiple **progress-bar** `<div>`s inside of the outer **progress** container and setting their values appropriately.

The final element that's left, is the humble **well**. No changes have been made to this since BS3, but it does now have an extra class called **well-sm**. There's no magic involved here— if you want a simple, boxed-off area with a shaded background, simply create a `<div>` element, add the **well** class to it, and then add your content inside. Wells are useful for sidebars and/or footers, or anything too simple to warrant being put in a full panel or other fenced off area. There are no contextual colors or special actions, either; it's simple, effective, and easy to use:

Code Sample 71: Simple Well Example

```
<div class="well">Hello World!</div>
```

Figure 75: Well example produced by sample 71

# Chapter 5 Changed JavaScript Features

When it comes to JavaScript in BS3, not much has changed; the vast majority of the changes we've seen so far have revolved around the CSS and components sections. There's a reason for this.

Most of BS2's (and for that matter, BS3's) JavaScript functionality comes in the form of data attributes. In most of the cases, we've already seen how to use these in the various sections on components, which really leaves very little that's specific to JavaScript only.

In this chapter, therefore, I'll just briefly run through most of what's available, and where there is no other description elsewhere in the book, show a brief example on how to use the API available.

The JS facilities available in BS2 and BS3 are very extensible, and even a full book probably couldn't cover everything that's possible. I therefore strongly encourage you to go to [www.getbootstrap.com](http://www.getbootstrap.com) and read through the section on JavaScript.

## Modals

The first thing that anyone mentions when the subject of JavaScript comes up in Bootstrap is the modal dialog boxes—and it's little surprise.

BS3's modal boxes are one of the easiest implementations (and one of the richest) seen in any of the modern browser HTML5 frameworks.

Using them is easy, but unfortunately, does require quite a lot of markup.

The following code gives you a very basic example:

*Code Sample 72: Basic Modal Example*

```
<button class="btn btn-primary btn-lg" data-toggle="modal" data-target="#myModal">
  Show Modal Dialog
</button>

<div class="modal fade" id="myModal" tabindex="-1" role="dialog" aria-labelledby="myModalLabel" aria-
hidden="true">
  <div class="modal-dialog">
    <div class="modal-content">
      <div class="modal-header">
        <button type="button" class="close" data-dismiss="modal" aria-hidden="true">&times;</button>
        <h4 class="modal-title" id="myModalLabel">Modal title</h4>
      </div>
      <div class="modal-body">Modal Body</div>
      <div class="modal-footer">
        <button type="button" class="btn btn-default" data-dismiss="modal">Close</button>
        <button type="button" class="btn btn-primary">Some Action</button>
      </div>
    </div>
  </div>
</div>
```



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

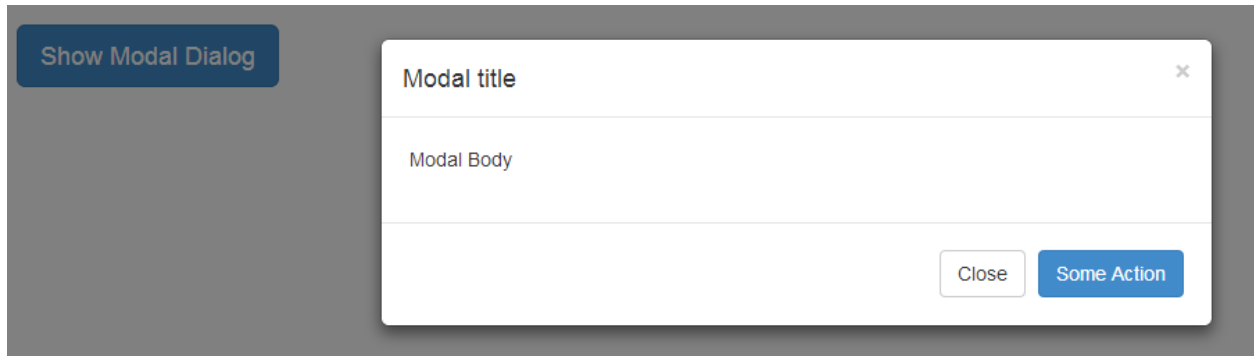


Figure 76: Modal example produced by sample 72

In order to show a modal, you first need to have a trigger target. In sample 72, this is the button marked **Show Modal Dialog**. For a trigger action to work it must have a toggle and target data attribute assigned to it, and the toggle must have the value "modal" to show that it targets a modal dialog. The target must have the ID selector of the outermost `<div>` assigned to it.

In sample 72, the outermost `<div>` has an `ID="myModal"` on it, which means the data attribute for `target` should have `#myModal` as its value.

Your trigger doesn't have to be a button; it can be anything that can accept (or is set up to accept) a mouse click, as long as the toggle and target data attributes are provided.

Once we get into the modal itself, you'll see the structure consists of a number of quite deeply nested `<div>`s. Because of this nesting, it's recommended that you create and place your dialog/modal definitions as close to the body root as possible; if you do not, then there is a chance that other components and HTML structures could cause layout problems that you did not anticipate.

You'll also notice that again, there is a common theme of marking things up to make them friendly to screen readers, and again, I can't stress this enough: you should make every effort to make sure your markup is as friendly to accessibility tools as possible.

A modal starts with an outer `<div>` and the class `modal` applied to it. Optionally, you can also add `fade`, which will give the modal a nice smooth transition when showing and hiding. This outer `<div>` should be the one on which you set your **Z-Order** and anything else in the way of global modal customizations you wish to make.

The next `<div>` in should have a class of `modal-dialog` added to it. That `<div>` should then be followed immediately by a third `<div>` with the class of `modal-content` assigned to it. It's inside this third `<div>` where you actually place your modal content definition.

Once you have the modal content shell defined, you can then place in three further `<div>` elements with the following classes: **modal-header**, **modal-body**, and **modal-footer**. These three inner sections should NOT be nested, but rather added to the markup as siblings of each other, and are used to define the content for the three main sections of the dialog.

You can see from the code in sample 72 that we include a closing cross, as we did for alert boxes. The only difference between this closing cross and the one we saw previously is that the **dismiss** data attribute has a value of **modal** and not **alert**. Any clickable element placed within the inner modal markup that has this data attribute, with this value, will close the dialog when clicked.

Apart from the close icon, the rest of the modal's inner content is just normal BS3 markup and CSS. Anything you can use elsewhere you can use inside a modal, and if it's too tall for the screen, you'll get an inner container that automatically switches to a scrollable element.

There are also two optional width sizes; these are added to the inner **modal-dialog** `<div>` and are **modal-lg** and **modal-sm**. The large size class expands the width of the modal to half the screen width (ideal for tables and lists), whereas the small size shrinks the default width to approximately half of its original size (ideal for things like yes/no prompts).

You can also initialize the modal using the JavaScript API in a standard jQuery fashion; if you want to alter the default option's behavior, then using the JQ constructor is the only way to do it.

*Code Sample 73: Setting the Default Options on a Modal Using JavaScript*

```
$('#myModal').modal({
  backdrop: true/false,
  keyboard: true/false,
  show: true/false,
  remote: 'path to url that returns content'
});
```

The options that can be changed are as follows:

**backdrop:** Boolean true or false to include or not include the shaded background on the page when the modal is shown; if the value **static** is specified, then the background is shown but does NOT close the modal when clicked on, as it does if **true** is used.

**keyboard:** Boolean true or false; allows or does not allow the escape key to close the modal.

**show:** Boolean true or false, automatically shows or does not show the dialog as soon as it's initialized.

**remote:** String containing a url to get the inner content for the dialog body; if this is supplied, then the dialog will ask the url to supply a chunk of HTML to be used in the body of the modal.

There are also a number of events that are raised for certain actions, but they are beyond the scope of this chapter.

## Tabs

If you recall, back in the section on basic navigation, I mentioned that the tab component can be wired up with extra markup to actually handle the swapping of content panes for you.

To mark-up a set of tabs that change automatically using JavaScript, you first need to create a `<ul>` in the same manner as shown in the navigation components section. This `<ul>` must have `<a>` elements embedded inside each of its `<li>` elements, with the `href` of each anchor pointing to the `id` of each associated `<div>` set up to hold a tab panel. You also need to make sure that each anchor has a toggle data attribute assigned to it and that its value is set to `tab`.

Once you've created the navigation set, you then need to create an outer `<div>` and assign the class `tab-content` to it. Inside of this `<div>`, you then need to create several separate sibling `<div>`s, each with a class of `tab-pane` and an `id` attribute matching the associated tab in the navigation set. Optionally, you can also add `fade in` to fade tabs when they change, and `active` to mark which of the tabs is currently been shown.

The following code shows an example of this:

Code Sample 74: Automatic Tabs Example

```
<ul class="nav nav-tabs">
  <li class="active"><a href="#tab1" data-toggle="tab">Tab 1</a></li>
  <li><a href="#tab2" data-toggle="tab">Tab 2</a></li>
  <li><a href="#tab3" data-toggle="tab">Tab 3</a></li>
</ul>
<div class="tab-content">
  <div class="tab-pane active" id="tab1">
    <h1>We self-actualize, we believe, we are reborn</h1>
    <p>By refining, we live ...</p>
  </div>
  <div class="tab-pane" id="tab2">
    <h1>We exist as four-dimensional superstructures</h1>
    <p>Today, science tells us that the ...</p>
  </div>
  <div class="tab-pane" id="tab3">
    <h1>The goal of morphogenetic fields is to plant ...</h1>
    <p>You and I are life forms of the quantum soup ...</p>
  </div>
</div>
```

Tab 1 Tab 2 Tab 3

### We exist as four-dimensional superstructures

Today, science tells us that the essence of nature is curiosity. Consciousness consists of bio-feedback of quantum energy. "Quantum" means a refining of the consciousness-expanding

Figure 77: Tab set produced by code sample 74

The tab control doesn't have a constructor that takes options the way modal does, but it does have an API call so that you can tell which tab to show programmatically. To do this, you just need use jQuery to select the appropriate selector, and then call `tab('show')` on it. When this is done, your tabs will automatically make the referenced tab the selected one. As with modals (and others), there are events available to tell you when things change; the details and parameters of each call can be found in the BS3 docs.

## Tooltips and Popovers

Everyone loves tooltips—simple little pop-up tags that can be used for help and many other simple, descriptive tasks.

Using a tooltip in BS3 is incredibly easy. Simply assign a data attribute of `toggle` with the value `tooltip` to any standard HTML element that you wish the tooltip to display for. To define the text for the tooltip, add a title attribute containing the desired text, and optionally, add a data attribute called `placement` containing the value `left`, `top`, `bottom`, or `right` as required, depending on which direction you would like the tooltip to show.

The following code will create a simple button with a tooltip attached to its top:

*Code Sample 75: BS3 Button with a Tool Tip*

```
<button id="mybutton" type="button" class="btn btn-default" data-toggle="tooltip" data-placement="top" title="I am a tooltip!!">The button with a tooltip</button>
```

There is one small caveat that applies to tooltips, but does not apply to any other element: you need to initialize tooltips yourself. You can pass various options into them at the same time (just as with modals), but you **MUST** initialize them, or your tooltips will not appear.

To initialize the button shown in the previous example, place the following line of JavaScript somewhere in your page so that it's run once the DOM is ready and the button is created:

```
$('#mybutton').tooltip();
```

It's entirely up to you how you select each of your buttons. You could, for example, select them all via their element type, but you must call `tooltip()` on every element that has a tooltip attached.

If everything works as expected, you should see something like this:

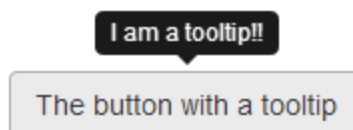


Figure 78: Button with accompanying tooltip

Close behind the humble tooltip comes the popover, and like the tooltip, it must be initialized manually with a call to `popover()`. The main difference between a popover and a tooltip is that popovers can hold more contents than a tooltip.

A tooltip typically only has a simple, single line of text, whereas a popover is larger and can contain multiple HTML elements, ranging from paragraphs to buttons and images.

The second difference is that the element has to be clicked before a popover will display, whereas a tooltip is automatic upon hover.

You create a popover in much the same way as a tooltip, except that the popup content is defined inside a data attribute called `content`, and the `title` attribute is used to give the popover a mini title area (similar to the way the header area is used on a panel component). The following code shows how to define a simple popover:

Code Sample 76: BS3 Button with a Popover

```
<button id="mybutton" type="button" class="btn btn-default" data-container="body" data-toggle="popover" data-placement="bottom" data-content="I am the pop over contents, and I'm awesome." title="Popover Title">
  I am a button with a pop over
</button>
```

As with the tooltip, somewhere in your document start-up, you also need to ensure you initialize the component using something like:

```
$('#mybutton').popover();
```

Also as with the tooltip, you can pass an object containing options in here. There are many available, so again, I'd encourage you to read the BS3 docs to learn them all.

If everything worked, you should be able to render your page and see this:

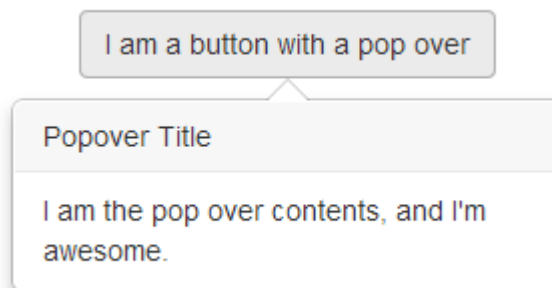


Figure 79: BS3 button with a popover attached

## Collapsible Panels

One of the things removed in BS3, sadly, was the readymade accordion component. In its place, however, is something better: the collapsible panel.

Using these panels, it's still just as easy to create a standard accordion, but they are now also separately useable, standalone components, allowing you to do things like create folding information areas, toolbars, and much more.

One thing to note, however: if you're doing a custom build, you must also make sure that you include the transition helper JavaScript plug-in. The BS3 docs have more information that you'll need if you are doing a custom build.

To create an accordion from collapsible panels, you simply have to create an outer `<div>` with a class of `panel-group` and give it an `id`. Then, inside of that, you need a series of `<div>` tags marked up as shown previously in the section on panel components, with each panel `div` being a single self-contained panel.

Once you have your panels laid out, you just need to add a `panel-title` inside a `panel-header`. This header should contain an `<a>` tag with two data attributes assigned: one called `data-toggle`, and one called `data-parent`.

The toggle attribute should have a value of `collapse`, and the parent attribute should hold the `id` of the outer `<div>` holding the panel group, and an `href` with the `id` of the target panel body that should be the object of the collapsing behavior. Each of the target panels should have the classes `panel-collapse` and `collapse in` assigned to them.

The following code shows how to achieve this:

*Code Sample 77: How to Create an Accordion Replacement Using Collapsible Panels*

```
<div class="panel-group" id="newAccordion">
  <div class="panel panel-default">
    <div class="panel-heading">
      <h4 class="panel-title">
        <a data-toggle="collapse" data-parent="#newAccordion" href="#panelOne">
          Consciousness is the richness of truth, and of us
        </a>
      </h4>
    </div>
    <div id="panelOne" class="panel-collapse collapse in">
      <div class="panel-body">
        Aromatherapy may be the solution to what's ...
      </div>
    </div>
  </div>
  <div class="panel panel-default">
    <div class="panel-heading">
      <h4 class="panel-title">
        <a data-toggle="collapse" data-parent="#newAccordion" href="#panelTwo">
```

```

        The planet is radiating four-dimensional superstructures
    </a>
</h4>
</div>
<div id="panelTwo" class="panel-collapse collapse">
    <div class="panel-body">
        It is time to take chi to the next level ...
    </div>
</div>
</div>
<div class="panel panel-default">
    <div class="panel-heading">
        <h4 class="panel-title">
            <a data-toggle="collapse" data-parent="#newAccordion" href="#panelThree">
                Reality has always been radiating messengers whose souls are opened by
stardust
            </a>
        </h4>
    </div>
    <div id="panelThree" class="panel-collapse collapse">
        <div class="panel-body">
            Self-actualization requires exploration ...
        </div>
    </div>
</div>
</div>
</div>

```

Consciousness is the richness of truth, and of us

Aromatherapy may be the solution to what's holding you back from an untold explosion of potential. You will soon be re-energized by a power deep within yourself - a power that is pranic, sublime. Through crystal healing, our souls are engulfed in awareness.

The planet is radiating four-dimensional superstructures

Reality has always been radiating messengers whose souls are opened by stardust

*Figure 80: Accordion replacement produced by sample 77*

As mentioned, panels don't need to be grouped; they can be used in a singular fashion with just a single element as the trigger for the folding to happen. For example, if you want to collapse a panel using a simple button, just ensure that your button has a data attribute of **toggle** with the value **collapse**, and a data attribute called **target** with the selector for the target panel as its value.

## Carousel

To round this chapter off, the last JavaScript plug-in I'm going to introduce is the newly designed carousel. BS2 had a carousel, but like the accordion, it's now been removed and greatly simplified to make it easier to use.

Typically, the carousel plug-in is used at the top of a page to provide a rotating banner of images, and in BS2, this was the only thing that the carousel could be used for. In BS3, however, any content that can be placed inside the carousel's panels will be rotated, including images, text, svg, and much more.

The following code shows a basic example of how to construct a carousel:

*Code Sample 78: BS3 Carousel*

```
<div id="carousel-example-generic" class="carousel slide" data-ride="carousel">
  <ol class="carousel-indicators">
    <li data-target="#carousel-example-generic" data-slide-to="0"
class="active"></li>
    <li data-target="#carousel-example-generic" data-slide-to="1"></li>
    <li data-target="#carousel-example-generic" data-slide-to="2"></li>
  </ol>
  <div class="carousel-inner">
    <div class="item active">
      
      <div class="carousel-caption">
        <h3>You and I are beings of the planet</h3>
        <p>Spacetime is a constant. Will requires exploration</p>
      </div>
    </div>
    <div class="item">
      
      <div class="carousel-caption">
        <h3>Nothing is impossible</h3>
        <p>Stardust requires exploration. You and I are storytellers of the
cosmos</p>
      </div>
    </div>
    <div class="item">
      
      <div class="carousel-caption">
        <h3>You and I are beings of the planet</h3>
        <p>Spacetime is a constant. Will requires exploration</p>
      </div>
    </div>
  </div>
  <a class="left carousel-control" href="#carousel-example-generic" data-
slide="prev">
    <span class="glyphicon glyphicon-chevron-left"></span>
  </a>
  <a class="right carousel-control" href="#carousel-example-generic" data-
slide="next">
    <span class="glyphicon glyphicon-chevron-right"></span>
  </a>
</div>
```





Figure 81: Carousel produced by code sample 78

There are a few more low-key JavaScript objects, but most of them are not directly usable from normal user code, and are generally only used in special circumstances. The BS3 docs cover everything I've missed, and if you're going to dig deeply into the JavaScript facilities available, a long read and an understanding of how everything is hooked together is definitely a requirement.

One final note: BS3 JavaScript plug-ins are nothing more than regular jQuery plug-ins (BS uses jQuery under the hood). This means that it should be very easy to take your favorite jQuery plug-in from places like unheap.com and adapt them to work with BS3 quite easily. Don't forget there is still a huge number of add-ons available out there already, especially to be used with the framework, most of which are only a Google search away.

# Chapter 6 Customizing Bootstrap 3

And so finally we get to the last chapter in the book, which will show you how to create custom color sets.

If you recall back at the beginning, when we were going through the major changes, I mentioned that the BS authors had provided an additional file to make BS3 look just like BS2, rather than using its flat look.

Throughout this book, we've used the default BS3 flat look, but if you download the prebuilt JavaScript version of BS3 from the get bootstrap site ([www.getbootstrap.com](http://www.getbootstrap.com)) you'll find a new file inside the archive you download called **bootstrap-theme.css**.

If you link this file into your project immediately after your inclusion of the core bootstrap.css file, you'll find that even though you're now using the new BS3, the look and feel of your application still resembles BS2.

The authors realized that one of the main barriers to the adoption of BS3 in new applications was the inability to style it in a custom manner, but with ease.

If you were a user of Less, or even SaSS (as BS3 now has SaSS bindings), then this wasn't an issue. You simply opened up the Less sources, tweaked the variables and Mixins you needed, and ran things through the Less compiler to get your new CSS script.

Unfortunately, not everyone used Less; in fact, many developers and designers only had the ability to download and include the plain old pre-compiled CSS and JavaScript files, so a better way had to be found.

The first change was opening up the entire Less sources to an in-page customization tool directly on the Bootstrap main site, but this isn't a new thing; you were able to use this page before in a limited way. With BS3, however, the Less customization tool has had a complete overhaul, and you can now redefine EVERYTHING that BS3 uses, from font sizes and typefaces, right through to grid sizes, trigger points, and basic contextual color sets.

In fact, there is now nothing that cannot be changed before you decide to download your new customized CSS, as the following image shows:



Figure 82: Screenshot of the top half of the new customization tool

Because of the sheer size of the tool, it's impossible to show the entire thing in this book, but it's easy enough to access. Simply go to [www.getbootstrap.com](http://www.getbootstrap.com) and click **Customize** in the top menu bar. You'll also see that you have many other options, such as which JavaScript plug-ins and tool-kits to include, which components to include, and base style that you may not want.

For example, if all you want to use is the grid system, and nothing else, then you can simply select only the grid system, and unselect all the other components.

The BS3 site will then generate just the required code to include, and no more. This is a boon for those people who complain that all Bootstrap sites look the same, because it means that your site absolutely does not have to look the same as the rest—you can just use the bits you need, and use your own stuff for everything else.

There are two other ways you can customize your build. The first is to take the additional 'bootstrap-theme' CSS style sheet, make a copy, and then change the styles as you see fit. This is not as easy as using the customization tools, but it's also not as difficult as the alternative.

Most of the class names and settings that you'll want to change to stamp your personal mark in BS3 are already separated out in the BS2 theme, so the quickest way to experiment is simply to put together a prototyping page with the main controls and elements on you want to change, and then link in your copy of the BS2 theme.

If you're using Node and something like Bower, it gets even easier, because you can use live reload, then just watch your sample page change in near real-time as you tweak your custom version of the theme sheet.

The second way is slightly more involved, and as described in the BS3 docs, comes in two flavors: light customizations and heavy customizations.

An example of a light customization is adding a contextual color class to the button element. For example, if you wanted to add **btn-sky** alongside the **btn-info**, **btn-primary**, etc. classes, you might define a single style sheet to hold the following rules:

Code Sample 79: Light Customization, Adding a New Contextual Class to the BS3 Button Classes

```
.btn-sky, .btn-sky:hover, .btn-sky:active
{
  color: #000000;
```

```

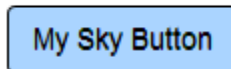
text-shadow: 0 -1px 0 rgba(0, 0, 0, 2);
background-color: #AAD4FF;
border: 1px solid black;
}

.btn-sky
{
background-repeat: repeat-x;
background-image: linear-gradient(top, #00FFFF 0%, #FFFF00 100%);
}

.btn-sky:hover
{
background-position: 0 -10px;
}

```

When added and linked in correctly, it should look something like this:



*Figure 83: Output from code sample 80, when used with the customization in sample 79*

When used with a normal button tag in the following manner:

```
<button class="btn btn-sky">My Sky Button</button>
```

The hardest part of using this method is going through the base CSS style sheets to find the names you wish to override; it's not exactly difficult, just long-winded.

Once you have one such as the button above, or you've found and copied out an alert, panel, list, or other class, then you can easily make a template that can be reused whenever you want to add a custom class of that type.

Heavy customizations are not very different from light customizations; the major difference is that you override the entire class.

So, for example, you find all the classes related to **btn**, copy them, modify them, and include them separately.

The new architecture inside the BS3 framework now means that once you know the target selectors, and their siblings, creating an override is easy. As I've already pointed out, the bootstrap-theme file already contains much of what you might want to modify anyway, and if that's not an option, then try sites such as [Bootswatch.com](http://Bootswatch.com):

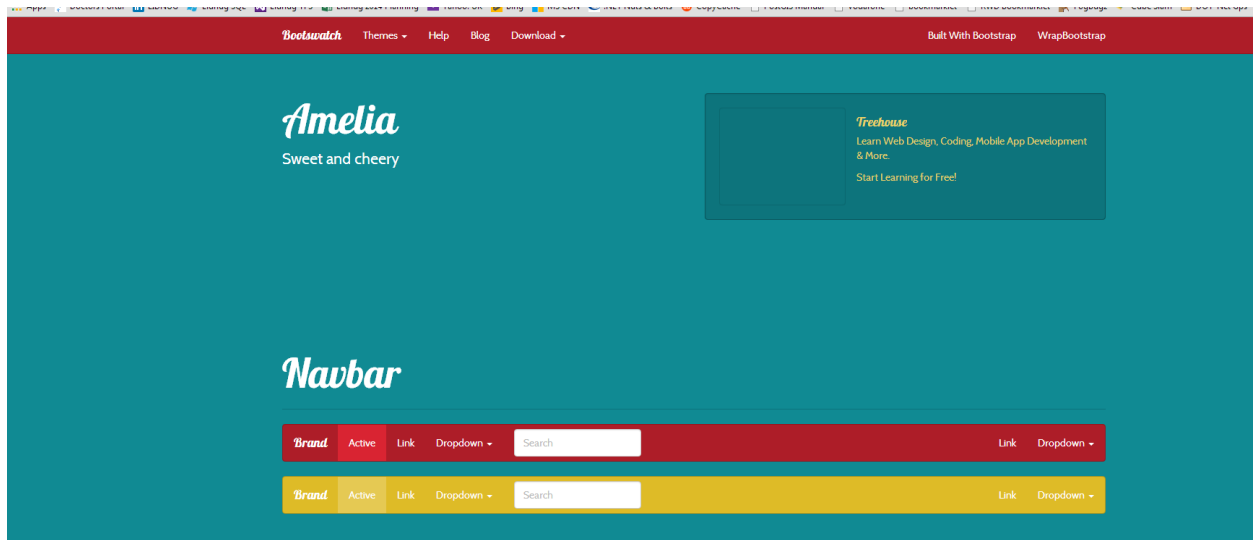


Figure 84: Screen shot of the sweet and cherry theme from Bootswatch.com

Many others have thousands of themes to choose from, both free and paid for, so there's no excuse for your Bootstrap site to look the same as everyone else's.

# Closing Notes

Well that's it folks, that's all I have for this book. I knew when I started writing the BS2 guide that it wouldn't be too long before BS3 went mainstream. It took a little longer than expected to get it out into the wild, but when it was released, it was met with so much enthusiasm that I knew I simply had to start a follow-up covering BS3 straightaway.

After a brief discussion with the marketing team at Syncfusion (Hi guys!), we decided that the way forward was to try and get this written in about a month, and by and large I did it. What you you're seeing now did not exist at the beginning of May 2014, and was nothing more than a few scribbled down ideas on the notepad on my desk.

If you'd like to reach out and ask me any questions about the book, I can generally be found hanging about on twitter as @shawty\_ds. You can also generally find me on [LinkedIn](#), or in the [Linked.NET \(Lidnug\) users group](#) that I help run there.

I hope you enjoy this book and that it helps you become a better developer using Bootstrap 3. In this new world of responsive web design, BS3 is a framework that shouldn't be taken lightly—it represents a lot of research by one of the biggest social media companies in existence, and is used to power most of their public-facing sites.

Until next time, keep calm and carry on bootstrapping.