In this package, you will find:

- The author biography
- A preview chapter from the book, Chapter 1 'Improving Web Development with Less'
- A synopsis of the book’s content

About the Author

**Bass Jobsen** has been programming for the Web since 1995, covering everything from C to PHP, and is always on the hunt for the most accessible interfaces. Based in Orthen, the Netherlands, he was the coauthor of *Less Web Development Cookbook, Packt Publishing*, which contains over 110 practical recipes to help you write leaner, more efficient CSS code using Less.

He uses Less in his daily job for web design tasks, WordPress theme development, the awesome StreetArt.nl project, and other Twitter Bootstrap apps.


Also, check out his Bootstrap WordPress Starter Theme (JBST) and other projects on GitHub at [https://github.com/bassjobsen](https://github.com/bassjobsen).
Less (Leaner CSS) is a preprocessor, which means Less code compiles into static CSS code. Less changes the way you write and maintain your CSS code; the final CSS code will be the same, but better organized in many situations.

In your hands is the *Less Web Development Essentials, Second Edition* book. Since the successful first edition of this book, Less has been improved still further. This book has been updated for the all new features that come with the release of Less Version 2. The Less Version 2 release not only has many improvements, but has also introduced the possibility to use and write Less plugins. Less plugins will enable you to customize Less for your own unique needs.

In the semantic sense, valid CSS code is also valid Less code. Less tries to stay as close as possible to the declarative nature of CSS. Coding Less is intuitive and easy to learn for those who are already familiar with CSS.

Since the introduction of CSS3 in 2012, the role of CSS in modern web development has become more and more important. Nowadays, CSS3 is not only used to style your HTML documents, but also plays an important role in the responsibility of your designs. Last but not least, CSS3 extends CSS with features such as animations and transitions.

Writing correct functional CSS code will be the first thing—and keeping this code readable—working on all major browsers will be the second. CSS files grow and become untidy during the development and maintenance processes. CSS also doesn't have the ability to modify existing values or reuse common styles. Also, doing math or defining variables is not possible in CSS. This is where Less comes into the frame.

Less is a dynamic stylesheet language designed by Alexis Sellier. Started in 2010 and now maintained and extended by the Less core team, it helps you make your CSS code maintainable, reusable, and also prevents code duplications. It also enables you to write your CSS code DRY (Do not Repeat Yourself), extending CSS syntax with variables, mixins, functions, and many other techniques. Less ensures that the CSS language is more maintainable, themable, and extendable.

In this book, you will learn to write, compile, and understand Less. We will help you to perform faster and more cost effective web development. You will get practical tips to integrate Less into your current and new projects. After reading this book, you will write clear and readable CSS3 with Less. Instead of spending your time debugging your complex CSS code for a specific device or browser, you can pay more attention to your real-design tasks.
Your clients will be happy with your advanced and stable designs.
Development and maintenance time, along with expenditure, will decrease.

Also, other projects know the power of Less. Projects such as Twitter Bootstrap and the WordPress Roots theme rely on Less. These projects build clear and extendable frameworks with Less.

Less is open source and licensed under Apache License. The source code of Less is maintained on GitHub. Everybody will be allowed to contribute to it. You can use Less free of charge.

**What This Book Covers**

*Chapter 1, Improving Web Development with Less,* talks about how CSS3 brought web designers advanced functions such as gradients, transitions, and animations; however, it also stresses how CSS code can become more complex and difficult to maintain. Less helps you to make your CSS maintainable, reusable, and also prevents code duplications.

*Chapter 2, Using Variables and Mixins,* explains why variables allow you to specify widely used values in a single place and then reuse them throughout the style sheet, making global changes as easy as changing one line of code. Mixins allow you to embed all the properties of a class into another class by simply including the class name as one of its properties. It also explains what parametric mixins are and how to use them.

*Chapter 3, Nested Rules, Operations, and Built-in Functions,* explains how to use nested rules for making inheritance clear and creating shorter style sheets. It also shows you how to create complex relationships between properties and use the built-in functions of Less.

*Chapter 4, Testing Your Code and Using Prebuilt Mixins Libraries,* explains how to use well-written and tested Less code of third parties for your projects. Pre-built mixins and other sources help you to (re)use them.

*Chapter 5, Integrating Less in Your Own Projects,* explains how to organize your files for new projects or get the projects you maintain ready for using Less.

*Chapter 6, Using the Bootstrap 3 Frontend Framework,* explains how to use, customize, and extend Bootstrap with Less. Bootstrap is a popular CSS, HTML, and JavaScript framework to build mobile-first responsive designs.

*Chapter 7, Less with External Applications and Frameworks,* explains how to use other frameworks and grid systems to build your HTML designs with Less, and with greater ease, including the integration of Less into WordPress.
Improving Web Development with Less

It is impossible to imagine modern web design without CSS. With CSS3, web designers are able to rely on advanced functions such as gradients, transitions, and animations. On the other hand, with these, the CSS code becomes more complex and difficult to maintain. Less is a CSS preprocessor that extends CSS with modern programming-language concepts. It enables you to use variables, functions, operations, and nesting (rule or selector) while coding your CSS. It also helps you write CSS with the don't repeat yourself (DRY) principle. The DRY principle prevents you from repeating any kind of information in your code.

This chapter will cover the following topics:

- Introduction to CSS3
- Compiling Less into CSS
- Vendor-specific rules
- CSS3 rounded corners, animations, and gradients
- Using box-sizing border-box
- Server-side compiling, setting up a build process, and using GUIs
- The OOCSS, SMACSS, and BEM methodologies
Improving Web Development with Less

Using CSS3 for styling your HTML
In web design, you will use HTML to describe the structure of your documents and the CSS language to describe their presentation, including fonts, colors, and layout. The current standard HTML5 and CSS3 versions work on most modern browsers and mobile devices. CSS3 extends the old CSS with new selectors, text effects, background gradients, and animations. The power of CSS3, its new functionalities, and high acceptance on mobile devices using HTML5 and CSS3, makes it the standard for modern web design. The combination of HTML5 and CSS3 is ideal for building responsive websites because of their high acceptance on mobile phones and other devices.

Together, HTML5 and CSS3 introduce many new features. In this book, you will be shown and taught about the concepts of the most significant ones.

Using the CSS selectors to style your HTML
With Less (and CSS), you can style your HTML code using selectors. The CSS selectors are patterns or names that identify which HTML elements of the web page should be styled. The CSS selectors play an important role in writing the Less code.

For body p.article {color:red}, the selector here is body p.article. Selectors don't refer exclusively to one element. They can point to more than one element and different ones can refer to the same element. For instance, a single p selector refers to all the p elements, including the p elements with a .article class. In case of conflicts, cascade and specificity determine the styles that should be applied. When writing the Less code, we should keep the aforementioned rules in mind. Less makes it easier to write complex CSS without changing how your website looks. It doesn't introduce any limitations on your final CSS. With Less, you can edit well-structured code instead of changing the effect of the final CSS.

CSS3 introduces many new and handy selectors. One of them is :nth-child(n), which makes it possible to style, for example, every fourth paragraph's p tag in an HTML document. The CSS code for the preceding description will look as follows:

```css
p:nth-child(4n) {
  color:red;
}
```

Downloading the example code
You can download the example code files from your account at http://www.packtpub.com for all the Packt Publishing books you have purchased. If you purchased this book elsewhere, you can visit http://www.packtpub.com/support and register to have the files e-mailed directly to you.
Such selectors add powerful functions to CSS3. Nowadays, we are able to perform operations with CSS alone. However, in the past, we needed JavaScript or hardcoded styles (or classes at the very least). Less helps you organize and maintain these new selectors well and this is one of the reasons to learn it. Powerful selectors make CSS more important, but the CSS code also becomes cumbersome and difficult to maintain. Less will prevent this problem in CSS, even making complex code flexible and easy to maintain.

Visit [http://www.w3.org/TR/selectors/#selectors](http://www.w3.org/TR/selectors/#selectors) for a complete list of the CSS selectors.

**Specificity, inheritance, and cascade in CSS**

In most cases, many CSS styles can be applied on the same HTML element, but only one declaration for each individual property will win. The properties of an element may come from declarations in different rules. The W3C specifications describe the rules for which CSS styles will get the most precedence and will ultimately be applied. You can find these specifications in the next section.

The rules regarding the order of importance have not significantly changed with CSS3. They are briefly mentioned to help you understand some of the common pitfalls with Less/CSS and how to solve them. Sooner or later, you will be in a situation where you're trying to apply a CSS style to an element, but its effect stays invisible. You will reload, pull out your hair, and check for typos again and again, but nothing will help. This is because in most of these cases, your style will be overruled by another style that has a higher precedence.

The global rules for cascade in CSS are as follows:

- Find all the CSS declarations that apply to the element and property in question
- Inline styles have the highest precedence, except for `!important`

The `!important` statement in CSS is a keyword used to add weight to a declaration. The `!important` statement is added at the end of a CSS property value. After this, check who set the declaration; styles set by the author get a higher precedence than the styles defined by the user or browser (default). Author styles are defined by CSS in the web page; user styles are set by the user via the settings of his or her web browser; and default styles are set by the web browsers. The importance of the user is higher than the default, and the code with the `!important` statement (refer to Chapter 2, *Using Variables and Mixins*, for its meaning in Less) will always get the highest precedence.
Note that browsers such as Firefox have options to disable pages in order to use other alternatives to user-defined fonts. Here, the user settings overrule CSS of the web page. This way of overruling the page settings is not part of the CSS precedence unless they are set using `!important`.

- Calculate the specificity, which is discussed in the next section.
- If two or more rules have the same precedence and specificity, the one declared last wins.

As a Less/CSS designer, you will be making use of the calculated CSS specificity in most cases.

**How CSS specificity works**

Every CSS declaration gets a specificity, which will be calculated from the type of declaration and the selectors used in its declaration. Inline styles will always get the highest specificity and will always be applied (unless overwritten by the first two cascade rules). In practice, you should not use inline styles in many cases as it will break the DRY principle. It will also disable you from changing your styles only on a centralized location and will prevent you from using Less for styling.

An example of an inline style declaration is shown as follows:

```html
<p style="color:#0000ff;">
</p>
```

After this, the number of IDs in the selector will be the next indicator to calculate specificity. The `#footer #leftcolumn {}` selector has two IDs, the `#footer {}` selector has one ID, and so on.

Note that in this case, an ID is a unique selector starting with #. The `[id=]` selector for the same HTML element counts as an attribute. This means that `div#unique {}` has one ID and `div[id="unique"] {}` has zero IDs and one attribute.

If the number of IDs for two declarations is equal, the number of classes, pseudo classes, and attributes of the selector will be of importance. Classes start with a dot. For example, `.row` is a class. Pseudo classes, such as `:hover` and `:after`, start with a colon, and attributes, of course, are `href`, `alt`, `id`, and so on.

The `#footer a.alert:hover {}` selector scores two (one class and one pseudo class) and the `#footer div.right a.alert:hover {}` selector scores three (two classes and one pseudo class).
If this value is equal for both declarations, we can start counting the elements and pseudo elements. The pseudo elements are defined with a double colon (::) and allow authors to refer to otherwise inaccessible information, such as ::first-letter. The following example shows you how this works.

The \#footer div a{} selector scores two (two elements) and the \#footer div p a {} selector scores three (three elements).

You should now know what to do when your style isn't directly applied. In most cases, make your selector more specific to get your style applied. For instance, if \#header p{} doesn't work, you can try adding a \#header #subheader p{} ID, a \#header p.head{} class, and so on.

When the cascade and !important rules do not give a conclusive answer, specificity calculation seems to be a difficult and time-consuming job. Although Less won't help you here, tools such as Firebug (and other developer tools) can make the specificity visible. An example using Firebug is shown in the following screenshot, where the selector with the highest specificity is displayed at the top of the screen and the overruled styles are struck out:

<table>
<thead>
<tr>
<th>Inherited from</th>
<th>Inherited from</th>
<th>Inherited from</th>
</tr>
</thead>
<tbody>
<tr>
<td>ul.nav</td>
<td></td>
<td>body</td>
</tr>
<tr>
<td>.bs.sidenav</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>.nav</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>list-style: none outside none;</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>body</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>color: #333333;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>font-family: &quot;Helvetica Neue&quot;,Helvetica,Arial,sans-serif;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>font-size: 14px;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>line-height: 1.42857;</td>
</tr>
<tr>
<td></td>
<td>html</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>font-size: 62-5%;</td>
</tr>
<tr>
<td></td>
<td>html</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>font-family: sans-serif;</td>
</tr>
</tbody>
</table>

An example of specificity in Firebug
Building your layouts with flexible boxes

The flexbox layout (also called flexible boxes) is a new feature of CSS3. It is extremely useful in creating responsive and flexible layouts. Flexbox provides the ability to dynamically change the layout for different screen resolutions. It does not use floats and contains margins that do not collapse with their content. The latest versions of all major browsers, except from Opera mini, now support the flexbox layouts. Unfortunately, this support is not provided by many older browsers. Information about browser support can also be found at http://caniuse.com/#feat=flexbox.

We will focus on flexbox, due to its power. Also, as it is an important feature of CSS, we can also produce and maintain it using Less. You can access a set of Less mixins for CSS3 flexbox at https://gist.github.com/bassjobsen/8068034. You can use these mixins to create the flexbox layouts with Less, without using duplicate code.

These mixins will not be explained in great detail right now, but the following example shows how Less reduces the code needed to create a flex container. Using CSS, you might use the following code:

```css
div#wrapper {
  display: -webkit-flex;
  display: -moz-flex;
  display: -ms-flexbox;
  display: -ms-flex;
  display: flex;
}
```

The preceding example uses the .flex-display(); mixin to set the vendor prefixes for the flex value of the display property. In the Vendor-specific rules section, you will learn more about vendor prefixes and the Less autoprefix plugin. The Less autoprefix plugin makes it unnecessary to use vendor prefixes in your Less code. You can find the Less autoprefix plugin at https://github.com/less/less-plugin-autoprefix. Also, the .flex-display(); mixin becomes unnecessary. Some ancient browsers use an older syntax for the flexbox layout, autoprefixing does not fix old syntaxes and polyfills.

The flexbox grid is a grid system based on the flex display property. This grid system can be used with the Bootstrap grid as a fall back for older browsers. The Less code to build the flexbox grid can be found at https://github.com/bassjobsen/flexboxgrid and the official website of the flexbox grid is http://flexboxgrid.com/.
Flexboxes have been mentioned because they have the potential to play an important role in the future of web design. In Chapter 5, Integrating Less in Your Own Projects, you can read about the flexboxgrid.com and flexible.gs grid systems, which are built with flexboxes. However, this book will mainly focus on creating responsive and flexible layouts with Less using the CSS media queries and the float property.


Compiling Less

After delving into the theory of CSS, you can finally start using Less. As mentioned earlier, the syntax of Less is very similar to the syntax of CSS. More precisely, Less extends the CSS syntax. This means that any valid CSS code is, in fact, a valid Less code too. With Less, you can produce the CSS code that can be used to style your website. The process used to make CSS from Less is called compiling, where you can compile the Less code via server side or client side. The examples given in this book will make use of client-side compiling. Client side, in this context, means loading a Less file in the browser and using JavaScript on the client machine to compile and apply the resulting CSS. Client-side compiling is used in this book because it is the easiest way to get started, while still being good enough for developing your Less skills.

It is important to note that the results from client-side compiling serve only demonstration purposes. For production, and especially when considering the performance of an application, it is recommended that you use server-side precompiling. Less bundles a compiler based on Node.js, and many other GUIs are available to precompile your code. These GUIs will be discussed toward the end of this chapter.

Getting started with Less

You can finally start using Less. The first thing you have to do is download Less from http://www.lesscss.org/. In this book, Version 2 of less.js will be used. After downloading it, an HTML5 document will be created. It will include less.js and your very first Less file. Instead of a local version of the less.js compiler, you include the latest version from a content delivery network (CDN). Flexboxes have been mentioned because they have the potential to play an important role in the future of web design. For now, they are beyond the scope of this book. This book will focus on creating responsive and flexible layouts with Less using CSS media queries and grids.
The client-side examples in this book load from the less.js compiler CDN by using the following code in the section HEAD of the HTML files:

```
<script
    src="//cdnjs.cloudflare.com/ajax/libs/less.js/2.x.x/less.min.js"></script>
```

Replace 2.x.x in the preceding code with the latest version available.

Note that you can download the examples from the support files for this chapter in the downloadable files for the book at www.packtpub.com.

**Downloading the example code**

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

To start with, have a look at this plain yet well-structured HTML5 file:

```
<!doctype html>
<html lang="en">
    <head>
        <meta charset="utf-8">
        <title>Example code</title>
        <meta name="description" content="Example code">
        <meta name="author" content="Bass Jobsen">

        <link rel="stylesheet/less" type="text/css" href="less/styles.less" />
        <script
            src="//cdnjs.cloudflare.com/ajax/libs/less.js/2.2.0/less.min.js"></script>
    </head>

    <body>
        <h1>Less makes me Happy!</h1>
    </body>
</html>
```
As you can see, a Less file has been added to this document by using the following code:

```
<link rel="stylesheet/less" type="text/css"
  href="less/styles.less" />
```

When rel="stylesheet/less" is used, the code will be the same as for a style sheet. After the Less file, you can call less.js by using the following code:

```
<script
  src="//cdnjs.cloudflare.com/ajax/libs/less.js/2.2.0/
  less.min.js"></script>
```

Or alternatively, load a local file with a code that looks like the following:

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

In fact, that's all that you need to get started!

To keep things clear, html5shiv (which you can access at http://code.google.com/p/html5shiv/) and Modernizr (which you can access at http://modernizr.com/) have been ignored for now. These scripts add support and detect the new CSS3 and HTML5 features for older browsers such as IE7 and IE8. It is expected that you will be using a modern browser such as Mozilla Firefox, Google Chrome, or any version of Internet Explorer beyond IE8. These will offer full support of HTML5, CSS3, and media queries, which you will need when reading this book and doing the exercises.

You already know that you should only use less.js for development and testing in most of the cases. There can still be use cases, which do justice to the client-side use of less.js in production. To support less.js for older browsers, you could try es5-shim (https://github.com/es-shims/es5-shim/).

Now open http://localhost/index.html in your browser. You will see the Less makes me happy! header text in its default font and color.

It is not necessary to have a local web server running. Navigating to the index.html file on your hard drive with your browser should be enough. Or double click on the index.html file to open it in your default browser. Unfortunately, this won't work for all browsers, so use Mozilla Firefox in order to be sure. The examples in this book use http://localhost/map/, but this can be replaced with something similar to file:///map/ or c:\map\, depending on your situation. Note that you cannot run less.js from CDN when using the file:// protocol in Google Chrome. You can not use the less.js compiler from CDN. You should replace it with a local version.
After this, you should open `less/styles.less` in your favorite text editor. The syntax of Less and CSS doesn't differ here, so you can enter the following code into this file:

```css
h1{color:red;}
```

Following this, reload your browser. You will see the header text in red.

From the preceding code, `h1` is the selector that selects the HTML `H1` attribute in your HTML. The `color` property has been set to `red` between the accolades. The properties will then be applied onto your selectors, just like CSS does.

**Using the watch function for automatic reloading**

The `less.js` file has a `watch` function, which checks your files for changes and reloads your browser views when they are found. It is pretty simple to use. Execute the following steps:

1. Add `#!watch` after the URL you want to open, and then reload your browser.
2. Open `http://localhost/index.html#!watch` in your browser and start editing your Less files. Your browser will reflect your changes without having to reload.
3. Now, open `less/styles.less` in your text editor. In this file, write `h1{color:red;}`, and then save the file.
4. You should now navigate to your browser, which will show **Less makes me Happy!** in red.
5. Rearrange your screen in order to see both the text editor and browser together in the same window.
6. Furthermore, if you change `red` to `blue` in `less/styles.less`, you will see that the browser tracks these changes and shows **Less makes me Happy!** in blue, once the file is saved.

Pretty cool, isn't it?

The preceding examples used color names instead of hexadecimal values. For instance, `red` instead of `#ff0000`. In this book, named colors are used throughout.
Debugging your code

As we are only human, we are prone to making mistakes or typos. It is important to be able to see what you did wrong and debug your code. If your Less file contains errors, it won't compile at all. So one small typo breaks the complete style of the document.

Debugging is also easy with less.js. To use debugging or allow less.js to display errors, you can add the following line of code to your index.html file:

```html
<link rel="stylesheet/less" type="text/css" href="less/styles.less" />
<script type="text/javascript">var less = { env: 'development' };</script>
<script src="//cdnjs.cloudflare.com/ajax/libs/less.js/2.2.0/less.min.js"></script><script src="less.js" type="text/javascript"></script>
```

As you can see, the line with `var less = { env: 'development' }` is new here. This line contains `less` as a JavaScript variable used by less.js. In fact, this is a global Less object used to parse some settings to less.js. The only setting that will be used in this book is `env: 'development'`. For more settings, you can check http://lesscss.org/#client-side-usage-browser-options.

The `env: 'development'` line also prevents Less from caching. Less doesn't cache files in the browser cache. Instead, files are cached in the browser's local storage. If `env` is set to `production`, this caching could yield unexpected results as the changed and saved files are not compiled.

Since Version 2 of Less, options can also be specified on the `script` tag, as shown in the following code:

```html
<script src="less.js" data-env="development"></script>
```

Alternatively, you can use the `link` tag that refers to your source file, as follows:

```html
<link data-env="development" rel="stylesheet/less" type="text/css" href="less/styles.less" />
```

To try this new setting, edit `less/styles.less` again and remove an accolade curly bracket to create an invalid syntax of the `h1{color:red` form, and then save the file.
In your browser, you will see a page like the following screenshot:

![ParseError: missing closing `}`]

Besides syntax errors, there will also be name errors displayed. In the case of a name error, an undeclared function or variable would have been used.

It is possible to set other settings for debugging, either in the global Less object or by appending the setting to the URL. For example, you can specify the dumpLineNumbers setting by adding the following lines of code to your HTML file:

```javascript
<script type="text/javascript">less = { env: 'development', dumpLineNumbers: "mediaQuery" };</script>
```

Alternatively, you can add `!dumpLineNumbers:mediaQuery` to the URL. You can, for instance, open `http://localhost/index.html#!dumpLineNumbers:mediaQuery` in your browser to enable the setting. This setting enables other tools to find the line number of the error in the Less source file. Setting this option to `mediaQuery` makes error reporting available for the Firebug or Chrome development tools.

Similarly, setting this to `comments` achieves the same for tools such as FireLESS. For instance, using FireLESS allows Firebug to display the original Less filename and the line number of CSS styles generated by Less. Older and experimental versions of Chrome did support this `mediaQuery` format. Currently, most browsers offer support for the CSS source maps, as described in the next section, in favor of other source mapping formats. The current version of Less does not support CSS sourcemap for in-browser usage. When developing in browser with the `less.js` compiler, you can use the FireLESS add-on for Firefox.
The https://addons.mozilla.org/en-us/firefox/addon/firebug/ path shows that the Firebug add-on integrates with Firefox to put additional development tools, and the FireLESS add-on allows Firebug to display the original Less file name and line number of the Less-generated CSS styles. More information of using FireLESS with the latest version of Less can be found at http://bassjobsen.weblogs.fm/fireless-less-v2/.

Tools like Firebug, or the default Chrome browser development tools, or the default browser used to inspect the elements and functions of your webpage (which you can access by right-clicking on your browser screen) can also be used to see and evaluate the compiled CSS code. The CSS code is displayed as the inline CSS code wrapped inside a `<style type="text/css" id="less:book-less-styles">` tag. In the example given in the following screenshot, you will see an ID with the value `less:book-less-styles`. The values of this ID have been automatically generated by Less, based on the path and name of the `book/less/styles.less` Less file:

```
<
  p {
    color: navy;
  }
</style>
```

The Less-generated CSS styles

If you set the `dumpLineNumbers` option to comments in the code shown in the preceding screenshot, you will see the file names and linenumber of the origin too.

**Plugins**

Plugins have been used since Version 2 of Less. They can be used with both the client-side and the server-side compiler. The current list of plugins can be found at http://lesscss.org/usage/#plugins-list-of-less-plugins. Writing a plugin yourself is not that difficult, but beyond the scope of this book. You can be sure that the list of useful plugins will grow over time. Less plugins described in this book only work in a node.js application (including Grunt and Gulp) and sometimes also for in browser usage.

The Less plugins can be grouped into the following four or five types:

- The first type, the so-called visitors, enable you to change the compile process. The inline URLs plugin converting `url()` to a call to `data-uri()` is an example of a visitor plugin.
• The second type are file manager plugins, which help you to set the files that should be compiled. Examples of file manager plugins are the npm import (https://github.com/less/less-plugin-npm-import) and bower resolve (https://github.com/Mercateo/less-plugin-bower-resolve) plugins.

• The post-process plugins form the third type. The autoprefixer (https://github.com/less/less-plugin-autoprefix) and clean-css (https://github.com/less/less-plugin-clean-css) plugins are examples. They are able to change and modify the already compiled CSS code.

• The fourth type are the pre-process plugins. Pre-process plugins can automatically add the Less code before your custom code. You can use this type of plugin to load a complete library before your custom code. The Bootstrap plugin loads Bootstrap's Less code. You can find the Bootstrap plugin at https://github.com/bassjobsen/less-plugin-bootstrap.

• Finally, plugins can also be used to extend Less with your own custom functions. Plugins that extend Less with custom functions can be grouped in the fifth type, although these plugins, just like the visitor plugins, change the compile process. An example of a plugin, which extends Less with some custom functions, can be found at https://github.com/less/less-plugin-advanced-color-functions.

To use a plugin with Less in browser, you should first include the plugin source, and then set the plugin option of the less.js compiler. Your code will look like that shown in the following code snippet:

```html
<script src="plugin.js"></script>
<script>
  var less = {
    env: "development",
    plugins: [Plugin]
  };
</script>
<script src="//cdnjs.cloudflare.com/ajax/libs/less.js/2.2.0/less.min.js"></script>
```

In this book, you will find many code examples. Unless explicitly mentioned, the format of these examples always shows the Less code first, followed by the compiled CSS code. For instance, you can write the following lines of code in Less:

```less
mixin() {
  color: green;
}
```
This code will be compiled into the following CSS syntax:

```css
p {
  .mixin();
}
```

Your first layout in Less

You must first open `first.html` (from the downloadable files for the book) in your browser, and then open `less/first.less` in your text editor. In your browser, you will see a representation of a header, body, and footer.

As expected, `less/first.less` contains the Less code that will be converted into valid CSS by the `less.js` compiler. Any error in this file will stop the compiler and throws up an error message. Although the Less code shows some similarities to the plain CSS code, the process described here totally differs from editing your CSS directly.

The following screenshot shows you how this layout will look when opened in your web browser:
Vendor-specific rules

CSS3 introduced vendor-specific rules, which offer you the possibility of writing some additional CSS applicable for only one browser. At first sight, this seems the exact opposite of what you want. What you want is a set of standards and practicalities that work the same with every browser, and a standard set of HTML and CSS that has the same effect and interpretation for every browser. Actually, these vendor-specific rules are intended to help us reach this utopia. Vendor-specific rules also provide us with early implementations of standard properties and alternative syntax. Last but not least, these rules allow browsers to implement proprietary CSS properties that would otherwise have no working standard (and may never actually become the standard).

For these reasons, vendor-specific rules play an important role in many new features of CSS3. For example, animation properties, border-radius, and box-shadow, depend on vendor-specific rules. You can easily see that some properties may evolve from vendor prefixes to standard. Currently, most browsers support the border-radius and box-shadow properties without any prefix.

Vendors use the following prefixes:

- **WebKit**: -webkit
- **Firefox**: -moz
- **Opera**: -o
- **Internet Explorer**: -ms

Autoprefixing

Vendor prefixes make writing the CSS code, and therefore the Less code, more complex. Also, you will have to change your code when your requirements change. In this book, you will find the code examples that use mixins for prefixing your CSS properties. Autoprefixing your code will be a better practice in many cases by consensus of the community. Examples of mixins used for prefixing can help you understand why mixins are easier to maintain and how they prevent code duplication. Even when you autoprefix your code, you will need mixins for non-standard code.

To autoprefix your code, you can use the Less autoprefix plugins. Note that you can read more about plugins in general in the Server-side compiling section.
With the autoprefixer plugin, you can compile the following Less code:

```less
#grad {
    background: linear-gradient(red, blue);
}
```

The preceding Less code compiles into the following CSS code:

```css
#grad {
    background: -webkit-gradient(linear, left top, left bottom, from(red), to(blue));
    background: -webkit-linear-gradient(red, blue);
    background: -moz-linear-gradient(red, blue);
    background: -o-linear-gradient(red, blue);
    background: linear-gradient(red, blue);
}
```

The autoprefixer plugin has got a browser's option. This option enables you to specify your target browsers. The Less autoprefixer plugin is based on the autoprefixer plugin for PostCSS and gets its information about the browsers' features and usage statics from the Can I use (http://caniuse.com/) database. You can find the documentation for the Less plugin at https://github.com/less/less-plugin-autoprefix. The documentation of the Less plugin also refers to the PostCSS autoprefixer documentation to show you how to set your target browsers. The PostCSS autoprefixer, including documentation, can be found at https://github.com/postcss/autoprefixer.

Note that the autoprefixer only adds prefixes; adding polyfill or other alternative syntaxes is beyond the scope of the autoprefixer. Later on in this chapter, we will discuss how to create a background gradient. The background gradient code has an alternative fallback syntax for Internet Explorer 7 and 8. The autoprefixer does not add this alternative syntax to your CSS code.

In many situations, you can also consider a strategy of graceful degradation for older syntaxes. Graceful degradation means that you only add more user experience for modern browsers, without breaking the functionality of your design. One of the advantages of a graceful degradation strategy is that you do not have to try to run complex code on old and slow hardware, which can slow down your website. You can read more about graceful degradation at http://www.w3.org/wiki/Graceful_degradation_versus_progressive_enhancement.
The -prefix-free library

Unfortunately, the current version of the Less autoprefix plugin does not work when compiling your Less code in browser. The -prefix-free library offers an alternative for prefixing your code. The -prefix-free library runs client side in your browser and requires JavaScript. Client-side JavaScript can be a reason to not use it in production. The -prefix-free library only adds prefixes when needed, so it keeps your CSS as clean as possible. More information about -prefix-free can be found at https://github.com/LeaVerou/prefixfree.

To use prefixfree.js, you will just have to include prefixfree.js anywhere in your page. It is recommended to put it right after the style sheets, to minimize Flash of unstyled content (FOUC). FOUC means that your visitor sees your content before all styles have been loaded and applied. When you are using the -prefix-free library, your HTML should look as follows:

```html
<link rel="stylesheet/less" type="text/css" href="less/project.less">
<script src="../../less.js" type="text/javascript"></script>
<script src="prefixfree.js"></script>
```

Build rounded corners with border-radius

The border-radius property is a new CSS3 property, which will make many web developers happy. With border-radius, you can give the HTML elements a rounded corner. In previous years, many implementations of rounded corners using images and transparency have been seen. However, these were inflexible (not fluid) and difficult to maintain.

Vendor-specific rules are required for implementation, so you will have to compile your code with the Less autoprefix plugin.

To give an element rounded corners with a radius of 10 pixels, you can use the CSS code with vendor-specific rules as follows:

```css
-webkit-border-radius: 10px;
-moz-border-radius: 10px;
border-radius: 10px;
```

For rounded corners with different radii, use a list with values separated by spaces: 10px 5px 20px 15px; . The radii are given in the following order: top-left, top-right, bottom-right, and bottom-left. By keeping these rules in mind, you will see how Less can keep your code clean.
You can open roundedcornersbordered.html from the download section of this chapter in your browser, and open less/roundedcornersbordered.less in your text editor. In your browser, you will see a representation of a header, body, and footer with a dark blue border and rounded corners.

The CSS code for header in less/bordered.less looks like the following code:

```css
#header{
    background-color: red;
    border-radius: 10px;
    border: 5px solid darkblue;
}
```

You can see that the corners have been created with a radius of 10 pixels, set by the border-radius property, and the border itself has been set with the border property. If you were using CSS, you would have to repeat these properties three times for the header, footer, and body. In order to change these rules or add a vendor, you would also have to change the same code three times. To begin with, you might perhaps think, "Why not group the selectors?" in a fashion similar to the following code:

```css
#header, #content, #footer{
    border-radius: 10px;
    border: 5px solid darkblue;
    -webkit-border-radius: 10px;
    -moz-border-radius: 10px;
    border-radius: 10px;
}
```

The preceding code is syntactically correct in order to write the CSS or Less code. However, as your code base grows, it won't be easy to maintain. Grouping selectors based on properties makes no sense when reading and maintaining your code. Moreover, such constructs will also introduce many duplicated and unstructured usages of the same selectors.

With Less, you are able to solve these problems efficiently. By creating a so-called mixin, you can solve the issues mentioned earlier. For the border radius, you can use the following code:

```css
.bordered(@radius; @border-color) {
    border-radius: @radius;
    border: 5px solid @border-color;
}
```
Improving Web Development with Less

To use the mixin mentioned in the preceding code, you should call it as a property for the selector, using the following code:

```css
#header{
  background-color: red;
  bordered(10px, darkblue);
}
```

The compiled CSS of this Less code will now be as follows:

```css
#header {
  background-color: red;
  border-radius: 10px;
  border: 5px solid darkblue;
}
```

Looking at the original code in the `less/bordered.less` file, you can see that the preceding code wouldn’t be able to work for `#content`. The border radius for the content is 20 pixels instead of 10 pixels, as used for `header` and `footer`. Again, Less helps you solve this problem efficiently. Mixins can be called with parameters in the same way as functions can be called in functional programming. This means that in combination with a value and a reference for this value, mixins can be called in order to set the properties. In the following example, this will change to the following code:

```css
.bordered(@radius: 10px; @border-color: darkblue) {
  border-radius: @radius;
  border: 5px solid @border-color;
}
```

In the `.bordered(@radius: 10px; @border-color: darkblue)` mixin, the `@radius` and `@border-color` variables are your parameters. The default value of the `@radius` parameter will be 10px and the default value of the `@border-color` parameter will be `darkblue`.

From this point onwards, mixins can be used in your code. The `.bordered(50px);` statement will set the corners with a radius of 50px and the `.bordered();` statement will do the same with a radius of 10px and `darkblue` border color (default).

The same is true for the second `@border-color` parameter, which has a default value set to `darkblue`. A `.bordered(10px; white);` call will set a white border with a radius of 10px. Note that Less supports both a comma and a semicolon as separator for the parameters of the mixins. Using commas can become confusing when your arguments are `comma-separated values (CSV)`.
Using this, you can rewrite `less/borderedmixinsroundedcorners.less` so that it changes to the following code:

```less
/* mixins */
.bordered(@radius: 10px; @border-color: darkblue;) {
  border-radius: @radius;
  border: 5px solid @border-color;
}
#header{
  background-color: red;
  .bordered();
}
#content{
  background-color: white;
  min-height: 300px;
  .bordered(20px);
}
#footer{
  background-color: navy;
  .bordered(10px; white);
}
```

The `less/roundedcornersborderedmixins.less` file, which is available in the download section, contains a copy of this code. To use this, you also have to change the reference in your HTML file to `<link rel="stylesheet/less" type="text/css" href="less/groundedcornersborderedmixins.less" />`. Note that this code leaves out the general styling of the `div` and `body` tags in HTML. These styles are only used to make the demo look good and do not actually demonstrate Less in any useful manner.

After rewriting your Less code, reload your browser or watch it if you have applied the `#!watch` trick. You will see that the output will be exactly the same. This shows you how to get the same results with Less, using a more efficiently structured code.
Improving Web Development with Less

The preceding code will result into an example of a simple layout with borders and rounded corners. The final result should look like the following screenshot:

![Layout with borders and rounded corners](image)

**Preventing cross-browser issues with CSS resets**

When talking about cascade in CSS, there will definitely be a mention of the browser default settings getting a higher precedence than the author's preferred styling. When writing the Less code, you will overwrite the browser's default styling. In other words, anything that you do not define will be assigned a default styling, which is defined by the browser. This behavior plays a major role in many cross-browser issues. To prevent these sorts of problems, you can perform a CSS reset. The most famous browser reset is Eric and Meyer's Reset CSS (accessible at [http://meyerweb.com/eric/tools/css/reset/](http://meyerweb.com/eric/tools/css/reset/)).

The CSS reset overwrites the default styling rules of the browser and creates a starting point for styling. This starting point looks and acts the same on all (or most) browsers. In this book, normalize.css V2 is used. The normalize.css file is a modern, HTML5-ready alternative to the CSS resets and can be downloaded from [http://necolas.github.io/normalize.css/](http://necolas.github.io/normalize.css/). It lets browsers render all elements more consistently and makes them adhere to modern standards.
To use a CSS reset, you can make use of the @import directive of Less. With @import, you can include the other Less files in your main Less file. The syntax is @import "{filename}". By default, the search path for the directives starts at the directory of the main file. Although setting alternative search paths is possible (by setting the path’s variable of your Less environment), it will not be used in this book.

The examples of Less files in this book contain @import "normalize.less"; in the first few lines of the code. Again, you should note that normalize.less does contain the CSS code. You should pay particular attention to the profits of this solution!

If you want to change or update the CSS reset, you will only have to replace one file. If you have to manage or build more than one project, which most of you may be doing, then you can simply reuse the complete reset code.

### Creating background gradients

A new feature in CSS3 is the possibility of adding a gradient to the background color of an element. This acts as a replacement for complex code and image fallbacks.

It is possible to define different types of gradients and use two or more colors. In the following screenshot, you will see a background gradient of different colors:

![Gradient Background](image)

A gradient example

The preceding example is taken from [http://www.w3schools.com/](http://www.w3schools.com/).

In the next example, you can use a linear gradient of two colors. The background gradients use vendor-specific rules.

You can use the example code from the rounded corners example to add gradients to it.

The first step is to copy or open `less/gradient.less` and add a new mixin at the start of this file, as shown in the following code:

```less
/* Mixin */
.gradient (@start: black, @stop: white, @origin: left) {
    background-color: @start;
}
```
Improving Web Development with Less

background-image: -webkit-linear-gradient(@origin, @start, @stop);
background-image: -moz-linear-gradient(@origin, @start, @stop);
background-image: -o-linear-gradient(@origin, @start, @stop);
background-image: -ms-linear-gradient(@origin, @start, @stop);
background-image: linear-gradient(@origin, @start, @stop);
}

This will create gradients going from left (@origin) to right with colors from @start to @stop. This mixin has default values.

IE9 (and its earlier versions) do not support gradients. A fallback can be added by adding background-color: @start;, which will create a uniform colored background for older browsers.

After adding the mixin to your code, you can call on it for our #header, #body, and #footer selectors, as shown in the following code:

```
#header{
  background-color: red;
  .roundedcornersmixin();
  .gradient(red, lightred);
}
#content{
  background-color: white;
  min-height: 300px;
  .roundedcornersmixin(20px);
  .gradient();
}
#footer{
  background-color: navy;
  .roundedcornersmixin(20px);
  .gradient(navy, lightblue);
}
```

For example, if you renamed the Less file to less/gradient.less, you would also have had to change the reference in your HTML file to the following code:

```
<link rel="stylesheet/less" type="text/css"
href="less/gradient.less" />
```
If you now load the HTML file in the browser, your results should be like the following screenshot:

Gradients in the header, content, and footer from the example code

**CSS transitions, transformations, and animations**

Another new feature in CSS3 is the presence of transitions, transformations, and animations. These functions can replace the animated images, flash animations, and JavaScript codes in existing or new web pages. The difference between transitions, transformations, and animations isn't trivial. Animations are constructed with a range of `@keyframes`, where each `@keyframes` handles different states of your element in time. Transitions also describe the state of an element between the start and end of an event. They are mostly triggered by the CSS changes, such as a mouse over (hover) of an element.
To make things clear, it is important to keep in mind the button that is about to be pressed. The button will have two states: pressed and not pressed. Without transitions and animations, we are only able to style these states. The color of the button is white, but it becomes red when you hover the mouse over it. (In the CSS terms, its state becomes hovered by adding the :hover pseudo class.) In this case, the transition describes how the hovered button becomes red. For example, the change in color from white to red in two seconds (which makes it pink halfway) shows that the start of the color change is slow and changes faster as time passes. Using animations here enables us to describe the state of the button for every time interval between the start (not hovered) and end (hovered).

Transformations change the position of an element and how it looks. They do not depend on the state of the element. Some of the possible transformations are scaling, translating (moving), and rotating.

In practice, we use a combination of animations, transformations, and/or transitions in most situations. Also, in these cases, vendor-specific rules will play an important role.

Now a transformation will be added to our example.

Using the example code with rounded corners and gradients, copy the following code to less/transition.less or open less/transition.less, and add the following code to the beginning of the file:

```less
/* Mixin */
.transition (@prop: all, @time: 1s, @ease: linear) {
  -webkit-transition: @prop @time @ease;
  -moz-transition: @prop @time @ease;
  -o-transition: @prop @time @ease;
  -ms-transition: @prop @time @ease;
  transition: @prop @time @ease;
}
```

This mixin example has three variables; the first is the property (@prop), which you will change. This can be height, background-color, visibility, and so on. The default value all shouldn't be used in the production code as this will have a negative effect on performance. The @time variable sets the duration in milliseconds or seconds with ms or s respectively appended to it. The last variable, @ease, sets the transition-timing-function property. This function describes the value of a property, given that a certain percentage of it has been completed. The transition-timing-function property describes the completeness of the transition as a function of time. Setting it to linear shows the effect with the same speed from start to end, while ease starts slow and ends slow, having a higher speed in the middle. The predefined functions are ease, linear, ease-in, ease-out, ease-in-out, step-start, and step-end.
Note that these mixins will not be needed when you compile your code with the Less autoprefix plugin. Similarly, the .roundedcornersmixin(); mixin will not be needed and can be replaced with the border-radius property.

Now, you can edit less/transition.less to use this mixin. You can set the background color of the body when you hover over it. Note that you don't need to use the transition to change the gradient color but to rather change the background-color attribute. You are using background-color because transition-duration doesn't have a visible effect on the gradient. The code of the background-color transition is as follows:

```less
#content{
  background-color: white;
  min-height: 300px;
  .roundedcornersmixin(20px);
  transition(background-color,5s);
}
#content:hover{
  background-color: red;
}
```

If you renamed the Less file, for example, to less/transition.less, you would also have to change the reference in your HTML file to the following code:

```html
<link rel="stylesheet/less" type="text/css" href="less/transition.less" />
```

If you load the HTML file in the browser, you will be able to see the results there. Move your mouse over the content and see it change from white to red in five seconds.

Finally, a second example, that rotates the header, can be added. In this example, you will use @keyframes. Using @keyframes will be complex. So, in this case, you can define some vendor-specific rules and add these animation properties to #header as follows:

```less
@-moz-keyframes spin { 100% { -moz-transform: rotate(360deg); } }
@-webkit-keyframes spin { 100% { -webkit-transform: rotate(360deg); } }
@keyframes spin { 100% { -webkit-transform: rotate(360deg); transform: rotate(360deg); } }
#header{
  -webkit-animation:spin 4s linear infinite;
  -moz-animation:spin 4s linear infinite;
  animation:spin 4s linear infinite;
}
```
You can add the preceding code to our example files or open less/keyframes.less. If you renamed the Less file, for example, to less/keyframes.less, you also have to change the reference in your HTML file to the following code:

```html
<link rel="stylesheet/less" type="text/css"
 href="less/keyframes.less" />
```

Now, load the HTML file in the browser and watch your results. Amazing, isn’t it? With a little bit of creative thinking, you will see the possibilities of creating a rotating windmill or a winking owl using only CSS3. However, the first thing that should be done is to explain the code used here in more detail. As mentioned earlier, there are many cases in which you would make combinations of animations and transformations. In this example, you also get to animate a transformation effect. To understand what is going on, we will split the code into three parts.

The first part is @keyframes, shown in the following code, which describes the value of the CSS properties (transformation in this case) as a function of the percentage of the animation completeness:

```css
@keyframes spin {
  100% {
    -webkit-transform: rotate(360deg);
    transform: rotate(360deg);
  }
}
```

The name reference given to these keyframes definitions is my-spin-effect, which is not a special effect but only a chosen name. In the preceding example, a state of 100 percent completeness is described. At this state, the animated element should have made a rotation of 360 degrees.

This rotation is the second part that needs our attention. The transformation describes the position or dimensions of an element in the space. In the preceding example, the position is described by the number of degrees of rotation around the axis: 360 degrees at 100 percent, 180 degrees at 50 percent, 90 degrees at 25 percent, and so on.

The third part is the animation itself, described by the animation, my-spin-effect, which is 4s, linear, and infinite. This is the shorthand notation of settings of the sub properties of the animation property. In fact, you can write this as the following code, without the vendor-specific rules:

```css
animation-name: my-spin-effect;
animation-duration: 4s;
animation-timing-function: linear;
animation-iteration-count: infinite;
```
You can use the three parts of the preceding code to build a complete animation. After doing this, you can extend it. For example, add an extra keyframe, which makes the time curve nonlinear, as follows:

```css
@keyframes my-spin-effect {
  50% { transform: rotate(10deg); }
  100% { transform: rotate(360deg); }
}
```

You can add a second property using `background-color`. Don't forget to remove the gradient to see its effect. This is shown in the following code:

```css
@-moz-keyframes my-spin-effect {
  50% { transform: rotate(10deg); background-color:green; }
  100% { transform: rotate(360deg); }
}
```

```css
//.gradient(red,yellow);
```

You might have noticed that the complete profit of using Less isn't realized here. You will have to write the `@keyframes` definition repeatedly due to the different vendor prefixes its variable animation name. In Chapter 4, Testing Your Code and Using Prebuilt Mixins Libraries, a solution will be provided to you for this. Again, you can use the Less autoprefix plugin to solve this problem.

Unfortunately, browser support for transitions, transformations, and animations is not great and varies between browsers. Google Chrome does support CSS 3D transforms since Version 36. Older versions of Firefox lack support for the CSS filters, and IE9 (and earlier versions) don't support them at all. To solve this, many developers look to jQuery to support their animations. The `jQuery.animate()` function allows us to change the CSS properties of the elements using JavaScript. You can still use Less to set the initial CSS. An alternative for this would be to use `animate.css` (which you can access at https://github.com/daneden/animate.css); this cross-browser library of the CSS animations gets converted into the Less code with a jQuery fallback.

### Box-sizing

The box-sizing property is the one that sets the CSS-box model used for calculating the dimensions of an element. In fact, box-sizing is not new in CSS, but nonetheless, switching your code to `box-sizing: border-box` will make your work a lot easier. When using the `border-box` settings, calculation of the width of an element includes border width and padding. So, changing the border of padding won't break your layouts. You can find a copy of the code used in this section in boxsizing.html from the download files.
Nowadays, most web designs use a grid. Grids split your design into columns of equal size. This helps you make things clear and build responsive interfaces. Depending on the available screen size (or width), you can show your content and navigation with a different representation of the same columns.

To handle different screen sizes, some parts of your website will have fluid width or height. Other elements, such as borders, gutters, and the white space, should have a fixed width. The combination of fluid widths as a percentage of the screen width (or viewport) with fixed widths becomes complex. This complexity will be due to the fact that browsers use different calculations for padding and margins of elements.

In order for you to see this, look at the following example. A container of 5300 pixels width has been created. Inside this container, you can add two rows and split the second row into two parts of 50 percent or half of its width.

```html
<div class="wrapper" style="width:300px;">
  <div style="background-color:red;width:100%;">1</div>
  <div style="background-color:green;width:50%;float:left;">2</div>
  <div style="background-color:blue;width:50%;float:right;">3</div>
</div>
```

The output will now look like the following screenshot:

![An HTML wrapper](image)

The current structure doesn't show a problem until you add some padding, which is used to construct some space or a border between the two columns in the second row (numbers, 2 and 3, in the screenshot of the HTML wrapper). The padding and the border will break our layout as follows:

```html
<div class="wrapper" style="width:300px;">
  <div style="background-color:red;width:100%;">1</div>
  <div style="background-color:green;width:50%;float:left; border:5px solid yellow;">2</div>
  <div style="background-color:blue;width:50%;border:5px solid yellow;float:right;">3</div>
</div>
<br>
```
Finally, the output of this code should look like the following screenshot:

A broken layout due to padding and borders

A similar action can be performed, except that the wrappers can be wrapped inside an extra wrapper. The box-sizing: border-box; declaration can then be applied to this. Now, the results should look like the following screenshot:

A layout with box-sizing: border-box
As you can see, the padding and borders are subtracted by 50 percent from the parent. This will make the calculation a lot easier. Of course, you can do the calculating yourself, once the parent container wrapper has a fixed width. If the parent has 300 pixels, 50 percent of this will be 150 pixels. Taking away the padding and the width of the border will give you the fixed size of a column. This won't work when your parent has a fluid width (the percentage of the viewport). Fluid layouts change their width with the width of the screen. If your screen becomes smaller, all the elements become smaller too and the percentage stays equal. You will quickly realize that doing calculations for all the possible screen sizes to find the real size of a column that allows all of your elements to align, is a long, challenging, and arduous process.

For these reasons, you should make use of `box-sizing: border-box;` for all the examples in this book. Note that `box-sizing` also has to be defined by vendor-specific rules, as follows:

```css
-webkit-box-sizing: border-box;
-moz-box-sizing: border-box;
box-sizing: border-box;
```

In this example, the Less code will be as follows:

```less
// Box sizing mixin
.box-sizing(@boxmodel) {  
  -webkit-box-sizing: @boxmodel;  
  -moz-box-sizing: @boxmodel;  
  box-sizing: @boxmodel;  
}
// Reset the box-sizing
*,  
*:before,  
*:after {  
  .box-sizing(border-box);  
}
```

Note that the `.box-sizing` mixin will become obsolete when you compile your Less code with the Less autoprefix plugin.
With autoprefixing, your Less code can be reduced to the following code:

```less
// Reset the box-sizing
*, *
*::before,
*::after {
  box-sizing: border-box;
}
```

The preceding code has been added into a separate file called `boxsizing.less`. From now on, the basics of our Less files will contain the following code:

```less
@import "normalize.less";
@import "boxsizing.less"
```

In the next chapters, you will learn more about organizing your Less code into files.

**Server-side compiling**

You have already taken the first few steps toward developing Less. As explained earlier, client-side compiling has been used. However, client-side compiling with `less.js` shouldn’t be used on real websites. This is because, despite making your development easy and fast, compiling your Less files for every page request (or in fact, initial page load per user) will actually slow down your website.

For the production environment, it is required that you compile your files and serve the final CSS file to the browser. The term, server side, can be somewhat misleading. Server side in this context means that a compiled CSS code is sent to the client’s browser instead of the Less code, which has to be compiled in the client's browser by `less.js` before it is shown. You should precompile your Less code. By copying and pasting the results of `less.js` to a file and including this as a CSS file in your HTML files, you should get the same effect, except that your CSS is not minimized.

Less bundles a command-line compiler. Installing and using it is simple, using the following command:

```bash
>> npm install -g less
>> lessc styles.less styles.css
```
Node enables you to run the JavaScripts without a browser. Node and npm run on Windows, Mac OS X, and other Unix/*nix machines. You will find the Node.js source code or a prebuilt installer for your platform by visiting http://nodejs.org/download/. The package manager for the Node JavaScript platform is npm. The npm command-line tool is bundled with Node.js, so installing Node.js will install npm at the same time. More information about npm can also be found at https://www.npmjs.com/package/npm.

Use the -help function to get a list of options you can use with the following command-line compiler:

    >> lessc -help

The links to styles.css in your HTML, after successfully compiling it, are then shown as follows:

    <link rel="stylesheet/css" type="text/css" href="styles.css">

**Using CSS source maps for debugging**

When working with the large Less code bases, finding the original source can become complex when viewing your result in the browsers. Since Version 1.5, Less offers support for the CSS source maps. The CSS source maps enable developer tools to map calls back to their location in original source files. This also works for compressed files. Both Chrome and Firefox developer tools support the CSS source maps format. Also, Internet Explorer Version 11 supports this format.

Currently, the CSS source maps debugging won't work for client-side compiling, as used for the examples in this book. However, the server-side lessc compiler can generate the useful CSS source maps.

After installing the lessc compiler, you can run the following command:

    >> lessc --source-map=styles.css.map styles.less > styles.css

The preceding command will generate two files: styles.css.map and styles.css. The last line of styles.css now contains an extra line, which refers to the source map as follows:

    /*# sourceMappingURL=bootstrap.css.map */

In your HTML code, you only have to include the styles.css, as shown in the following code:

    <link href="styles.css" rel="stylesheet"/>
When using the CSS source maps, as described in the preceding code, and inspecting your HTML code with Google Chrome **Developer Tools**, you will see something like the following screenshot:

As you can see, styles now have a reference to their original Less file, such as `grid.less`, including line number, which helps you in debugging. The `styles.css.map` file should be in the same directory as the `styles.css` file. You don’t have to include your Less files in this directory too.

**Plugins**

The Less plugins can be installed with `npm` too. The `npm` plugin is the package manager for `Node.js`.

To install a Less plugin, you should run the following command on your console:

```
npm install less-plugin-{plugin-name}
```

To use an already installed plugin with the command-line compiler, you should use `plugin-name` as an option for the compiler. Options for the plugin can be set by leveraging the `=` sign, followed by the list of options.
Compressing and minimizing your CSS

After compilation, the CSS code is clean and readable. When taking this code into production, you have to compress and minimize it, in order to increase the loading speed and save on the bandwidth as well. The basic steps for compressing and minimizing the CSS code are removing comments, whitespaces, and other unnecessary code. The results won't be easy for a human to read. However, this doesn't matter, because you can use the Less files to update or modify the CSS code.

The Less command-line compiler can compress the output when the `--compress` option has been set. This option removes whitespaces. More advanced minification can be done by using the `clean-css` postprocessor (which you can access at https://github.com/GoalSmashers/clean-css). Before Version 2, the compiler had `clean-css` built-in. Since Version 2, you have to install the `clean-css` plugin.

You can install the `clean-css` plugin by running the following command in the command line compiler:

```
npm install less-plugin-clean-css
```

After installing the plugin, you can run the compiler with the `--clean-css` option as follows:

```
lessc --clean-css file.less
```

With `clean-css`, refer to the following code:

```plaintext
.class1 {
  color: red;
}
.class2 {
  color: red;
}
```

The preceding code will compile into the CSS code, as the following code shows:

```css
.class1, .class2 { color: red }
```

As you can see, `clean-css` also merges the selectors. For advanced optimization of your code, which includes selector merging, you should run the `clean-css` plugin with the advanced option as follows:

```
lessc --clean-css="advanced" file.less
```
Also, Pleeease is a CSS post-processor, which can be used to run different post-process once. Pleeease, among others, runs the autoprefixer plugin, provides a fallback for the em units, and can merge media queries. A Pleeease plugin for Less is available, too.

You can read more about Pleeease at http://pleeease.io/.

Compiling your Less code into CSS automatically
When you are ready to start your first Less project, you should consider creating a build process that compiles your Less files into the CSS code automatically. Grunt and Gulp are task runners for Node.js. You can use these task runners to compile your Less code automatically and run postprocessors for the compiled CSS code.

The task runners can also be configured to detect file change and trigger browser reloads. Your final build system can show your changes directly after saving. Also, in the browser, development will be possible.

Gulp is relatively new in comparison to Grunt, so Grunt has more plugins and wider community support. Currently, the Gulp community is growing fast. The biggest difference between Grunt and Gulp is that Gulp does not save intermediary files, but pipes these files’ content in memory to the next stream. A stream enables you to pass some data through a function, which will modify the data, and then pass the modified data to the next function. In many situations, Gulp requires less configuration settings, and so some people find Gulp more intuitive and easier to learn.

In this section, you will learn how to set up a build system with Gulp. Note that you can use the same strategies to do the same with Grunt. Those who are interested in build systems with Grunt can inspect the default build system of Bootstrap. You can read more about Bootstrap in Chapter 6, Using the Bootstrap 3 Frontend Framework. The Bootstrap source code contains a build system that compiles the code automatically leveraging Grunt.

To use Gulp, you should install Node.js first. As already mentioned, npm is bundled with Node.js. So, after installing Node.js, you can run the following commands in your working directory to install Gulp:

```
npm install -g gulp
npm install --save-dev gulp
```
After installing Gulp, you have to create a Gulpfile.js file in your working directory. The Gulpfile.js file defines the task that it should perform on your files when running the gulp command. Your first Gulp file will look like that shown in the following code:

```javascript
var gulp = require('gulp');
var less = require('gulp-less');

gulp.task('default', function () {
    gulp.src('./less/project.less')
        .pipe(less())
        .pipe(gulp.dest('./css'));
});
```

Now you can run the grunt command in the console, which will compile the Less code of the ./less/project.less into the ./css/project.css file. The basic build system uses the gulp-less plugin to compile your Less code. You can add more plugins such as gulp-sourcemaps and gulp-autoprefixer in order to bring your build system to the next level.

The gulp-less plugin enables you to use the Less plugin, too. Using the Less plugins, instead of the gulp plugins, for tasks will simplify your Gulp configuration. You can use the task configuration shown in the following code to compile your Less code with sourcemaps, and the clean-css and autoprefixer postprocessors:

```javascript
var gulp = require('gulp');
var less = require('gulp-less');
var sourcemaps = require('gulp-sourcemaps');
var LessPluginCleanCSS = require("less-plugin-clean-css"),
    cleancss = new LessPluginCleanCSS({advanced: true});
var LessPluginAutoPrefix = require('less-plugin-autoprefix'),
    autoprefix= new LessPluginAutoPrefix({browsers: ["last 2 versions"]});

gulp.task('default', function () {
    gulp.src('./less/project.less')
        .pipe(sourcemaps.init())
        .pipe(less({
            plugins: [
                cleancss,
                autoprefix
            ]
        }))
        .pipe(sourcemaps.write('.'))
        .pipe(gulp.dest('css'));
});
```
Finally, you can consider extending your build system with gulp-livereload. The gulp-livereload plugin can be used with the livereload Chrome extension. The livereload extension enables you to see the compiled version of your code in your browser directly after saving and opens the way to in browser development too. You can read more about livereload at http://livereload.com/.

**Graphical user interfaces**

Some of you will prefer to use a **Graphical User Interface (GUI)** instead of command-line compiling. There are many GUIs available for different platforms that can edit and compile your Less code. All of them cannot be mentioned here. Instead, here is a list of the most positive noticeable ones:

- **WinLess**: This GUI is a Windows GUI for less.js.
- **SimpLESS**: This GUI is a cross-platform editor and compiler with many functions, including the automatic addition of vendor-specific rules to your code.
- **CodeKIT**: This GUI is for Mac (OS X). It compiles many languages, including Less. It includes optimizations and browser previews.
- **Crunch!**: This GUI is a cross-platform compiler and editor.

When choosing a GUI for Less development, always check which version of less.js it uses. Some GUIs are built on older versions of less.js and don't support the latest features.

Web developers using Visual Studio should check out **Web Essentials**. Web Essentials extends Visual Studio with a lot of new features, including Less. Also, other IDEs such as PHPStorm have the built-in Less compilers. There is also a Less plugin for Eclipse.
The OOCSS, SMACSS, and BEM methodologies

As you already know, Less helps you write a reusable and more maintainable CSS. It also produces a valid CSS code, and features as the nesting of selectors already support you to keep related rules together. Less does not prescribe how to code your CSS. The Object oriented CSS (OOCSS); Scalable and Modular Architecture for CSS (SMACSS), which can be referred at https://smacss.com/; and Block, Element, Modifier (BEM), which can be referred at https://en.bem.info/, methodologies discussed in this section will help you organize your Less code, resulting in well organized CSS too.

These methodologies work well together with Less. Now, we will discuss how to deploy them with Less.

All methodologies aim to split up your code into reusable blocks. Blocks should be independent of the design or page context, as far as possible. Most programmers will already be familiar with this kind of abstractions while for some designers, it will be a newer way of thinking. Imagine that you write down all style rules for the footer of your website in a separate file, and then you can reuse that file in a new project to style the footer again.

Most projects do not focus on only one methodology, but use concepts from different methodologies to build a strategy to organize the Less (or CSS) code. Also note that BEM, is, in fact, nothing more than a concrete application of OOCSS.

The focus of SMACC is the organizable structure of your code. SMACC code should be grouped into five categories: base, layout, modules, states, and theme. You can use the @import feature of Less to create a file structure for each group. When you inspect the Less code of projects such as Bootstrap, as discussed in Chapter 6, Using the Bootstrap 3 Frontend Framework, you will find that this code has also been organized in a similar way. You can read more about SMACC on the official website at https://smacss.com/. Most of the content of this website, the book, and the workshops are not available for free.
OOCSS focuses on the code structure of your code. With OOCSS, you will code reusable objects of styles. With OOCSS you work with reusable objects and each object contains styles rules for a HTML structure which holds a type of content. The basic rules of OOCSS help you to separate the structure from skin and container from content. The OOCSS media object is probably the most discussed example. You can read more about the media object at http://www.stubbornella.org/content/2010/06/25/the-media-object-saves-hundreds-of-lines-of-code/. When we create an object for buttons, it will look as shown in the following code:

```css
.btn {}
.btn .btn-error{}
.btn .active {}
```

OOCSS not only tries to separate the container from the content, but also separates the style from the structure. In the button example, the .btn selector provides the structure while the .btn-error selector provides the styling. Also note that the .btn.active {} selector describes a state of your button object.

Nesting of selectors can be used to organize style rules for the button object together. The following Less code can be used for the button object:

```less
.btn {
  .btn-error{}
  .active {}
}
```

As already mentioned, BEM is an application of OOCSS. The BEM syntax uses the following assumptions:

- CSS selectors are used to describe components (.btn)
- The selector name followed by __ (two underscores) and the descriptive name of the descendant are used for styling of the component (.btn__btn-error{})
- Component's states can be declared by the name followed by – (double dashes) and the state (.btnactive{})

The Less code for the button component using BEM will look like the following code:

```less
.btn {
  &__error {}
  &--active{}
}
```
The preceding Less code makes use of the \& parent selector and nesting, and will compile into the CSS code as follows:

```
.btn {}
.btn-error {}
.btn-active {}
```

**Summary**

In this chapter, you refreshed and extended your knowledge of CSS3. You learned how to compile your Less code on the client side. Furthermore, you wrote the code that allows you to have rounded corners, gradients, and animations in Less. You witnessed the profits of using Less and can now take the crucial initial steps to organize and plan your new projects. You also witnessed why you would want to use the CSS resets, how to compile these into the Less code, as well as how the `border-box` can make your job easier. You also saw what a mixin is, how to use it, and how you can import a Less file with the `import` directive. You learned what is server-side compiling, how to set up a build process, and how to use GUIs. Last but not least, you got introduced to some methodologies that will help you to better organize your CSS code, leveraging Less.

In the next chapter, you will learn how to use variables in Less and how to build and use complex mixins.
Where to buy this book


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

Click here for ordering and shipping details.