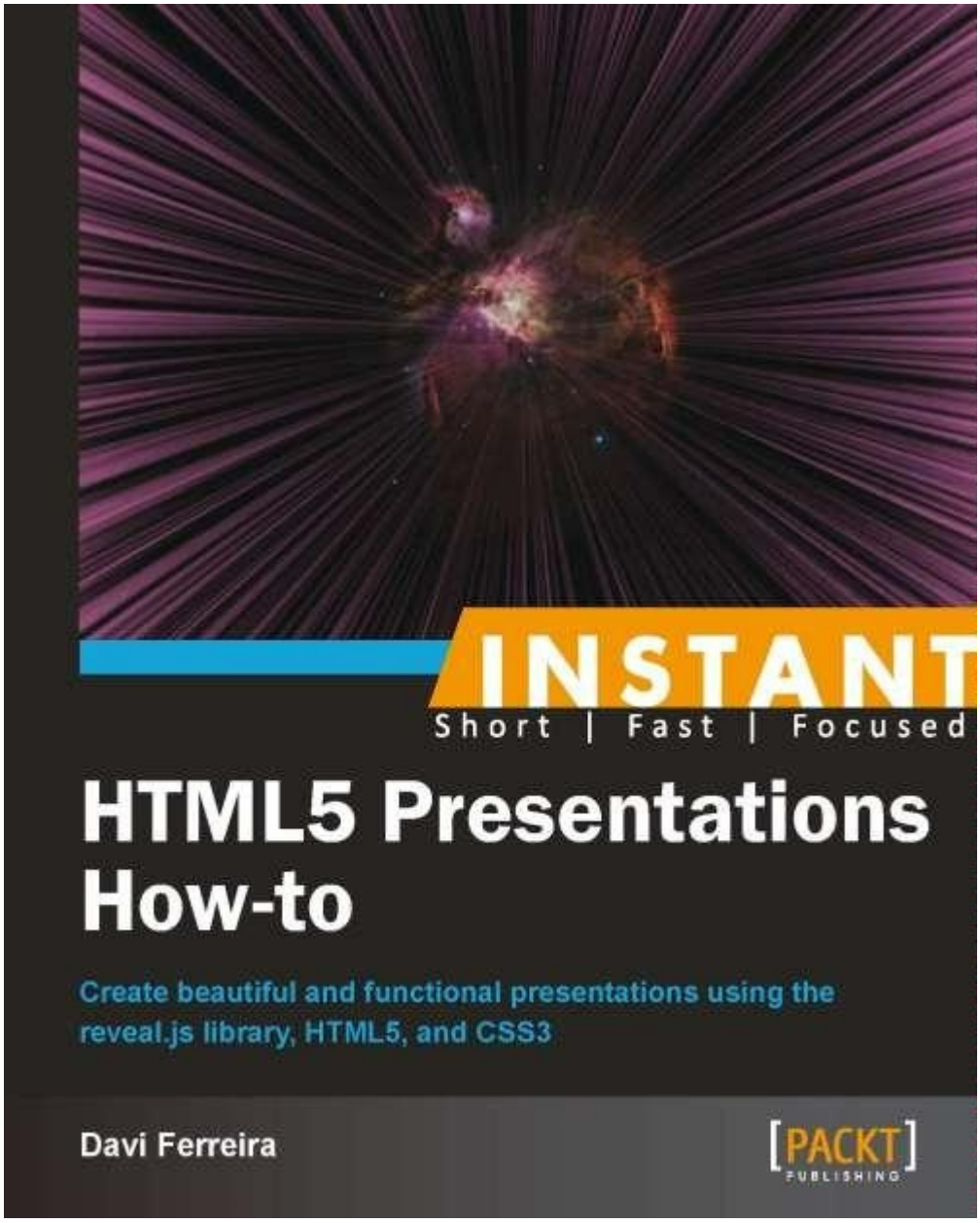

INSTANT
Short | Fast | Focused
HTML5 Presentations How-to
Create beautiful and functional presentations using the reveal.js library, HTML5, and CSS3
Davi Ferreira
[PACKT]
PUBLISHING

Table of Contents

Instant HTML5 Presentations How-to
Credits
About the Author
About the Reviewer
www.PacktPub.com
    Support files, eBooks, discount offers and more
        Why Subscribe?
        Free Access for Packt account holders
Preface
    What this book covers
    What you need for this book
    Who this book is for
    Conventions
    Reader feedback
    Customer support
        Downloading the example code
        Errata
        Piracy
        Questions
1. Instant HTML5 Presentations How-to
    Creating our initial presentation (Simple)
        Getting ready
        How to do it...
        How it works...
        There's more...
            Dependencies
            Overview and blackout modes
            Browser support
    Customizing our presentation (Simple)
        How to do it...
        How it works...
    Adding speaker notes (Simple)
        How to do it...
        How it works...
        There's more...
            Presentation control
            Pop up
            Enabling file access from files
    Installing and using Sass (Advanced)
        Getting ready
        How to do it…
        How it works…
        There is more...
            Imports

        Mixins
        Compass
Theming (Medium)
    Getting ready
    How to do it...
    How it works...
Setting up slide transitions (Simple)
    How to do it...
    How it works...
Connecting slides (Simple)
    How to do it...
    How it works...
    There's more...
        External links
Creating vertical/nested slides (Simple)
    How to do it...
    How it works...
    There's more...
        Linking nested slides
        Nested slides transition
Using fragmented views (Simple)
    How to do it...
    How it works...
Adding Markdown content (Simple)
    How to do it...
    How it works...
Customizing slides with global state (Medium)
    How to do it...
    How it works...
Working with custom events (Advanced)
    How to do it...
    How it works...
    There's more...
        Fragment events
Using reveal.js JavaScript API (Advanced)
    How to do it…
    How it works...
Plugins and extensions (Medium)
    How to do it...
    How it works...
Exporting our presentation as PDF (Simple)
    How to do it...
    How it works...
    There is more...
        Printing
        URL parameter
Server-side speaker notes (Advanced)
    Getting ready

How to do it...
How it works...

Instant HTML5 Presentations How-to

Instant HTML5 Presentations How-to

Copyright © 2013 Packt Publishing

All rights reserved. No part of this book may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, without the prior written permission of the publisher, except in the case of brief quotations embedded in critical articles or reviews.

Every effort has been made in the preparation of this book to ensure the accuracy of the information presented. However, the information contained in this book is sold without warranty, either express or implied. Neither the author, nor Packt Publishing, and its dealers and distributors will be held liable for any damages caused or alleged to be caused directly or indirectly by this book.

Packt Publishing has endeavored to provide trademark information about all of the companies and products mentioned in this book by the appropriate use of capitals. However, Packt Publishing cannot guarantee the accuracy of this information.

First published: February 2013

Production Reference: 1180213

Published by Packt Publishing Ltd.

Livery Place

35 Livery Street

Birmingham B3 2PB, UK.

ISBN 978-1-78216-478-4

www.packtpub.com

Credits

Author

Davi Ferreira

Reviewers

Ben Fhala

Acquisition Editor

Erol Staveley

Commissioning Editor

Ameya Sawant

Technical Editor

Devdutt Kulkarni

Project Coordinator

Michelle Quadros

Proofreader

Aaron Nash

Production Coordinator

Prachali Bhiwandkar

Cover Work

Prachali Bhiwandkar

Cover Image

Manu Joseph

About the Author

Davi Ferreira is a passionate web developer with 15 years of experience. He has worked with HTML, CSS, and JavaScript since the very beginning of those technologies.

He is currently working as a Senior Software Developer at globo.com, Internet branch of Brazil's biggest media conglomerate Globo, the home for shows such as Big Brother and The Voice.

Davi is also a resident writer of Tableless.com.br, one of the most important websites about frontend development in Brazil. He writes mostly about JavaScript and HTML5.

I would like to thank my beautiful wife, Leticia, for being patient and helping me out through the whole editing process.

About the Reviewer

Ben Fhala is a leading voice in the technology advertising world of New York. He is the owner of the online video training school 02geek.com, an Adobe ACP. He enjoys spending most of his time learning and teaching and has a love for visual programming and visualization in general.

Ben has had the honor to develop applications for US congress members, prime ministers, and presidents around the world. He has built many advertising experiences for companies such as Target, AT&T, Crayola, Marriott, Neutrogena, and Nokia. He has technically directed and led many award winning projects and so far has helped three agencies receive an Agency of the year award (not alone: he helped lead the technical side to produce award-winning quality work).

Ben has recently written his first book with Packt Publishing called *HTML5 Graphing and Data Visualization Cookbook* and was the technical reviewer for *Learning Modernizr, Adam Watson, Packt Publishing* and this book. You can find many other publications from Ben in the video world mainly at 02geek.com.

Thanks for giving me a chance to be part of this book. I've enjoyed reading it and giving my few small pointers. I'm honored to be included in this project and I want to thank the readers for joining in and wish them a great read.

www.PacktPub.com

Support files, eBooks, discount offers and more

You might want to visit www.PacktPub.com for support files and downloads related to your book.

Did you know that Packt offers eBook versions of every book published, with PDF and ePub files available? You can upgrade to the eBook version at www.PacktPub.com and as a print book customer, you are entitled to a discount on the eBook copy. Get in touch with us at <service@packtpub.com> for more details.

At www.PacktPub.com, you can also read a collection of free technical articles, sign up for a range of free newsletters and receive exclusive discounts and offers on Packt books and eBooks.



http://PacktLib.PacktPub.com

Do you need instant solutions to your IT questions? PacktLib is Packt's online digital book library. Here, you can access, read and search across Packt's entire library of books.

Why Subscribe?

- Fully searchable across every book published by Packt
- Copy and paste, print and bookmark content
- On demand and accessible via web browser

Free Access for Packt account holders

If you have an account with Packt at www.PacktPub.com, you can use this to access PacktLib today and view nine entirely free books. Simply use your login credentials for immediate access.

To Leticia – my favorite person in the world

Preface

Presentations are no longer limited to platform-specific software. You can now create stunning slides and show them on a web browser using modern technologies such as HTML5 and CSS3.

Instant HTML5 Presentations How-to will show you, through hands-on exercises, how to create beautiful and functional presentations to be shown on web browsers. You will learn how to customize and add special effects using HTML5, CSS3, and JavaScript.

What this book covers

Creating our initial presentation (Simple), explains how to create a basic HTML5 presentation that will be used throughout the rest of the book.

Customizing our presentation (Simple), introduces reveal.js look and feel options.

Adding speaker notes (Simple), covers how to add speaker notes to our presentation.

Installing and using Sass (Advanced), explains how to install, configure, and use Sass features.

Theming (Medium), introduces advanced customization options by creating a presentation theme from scratch.

Setting up slide transitions (Simple), explains how to use different transition effects on our presentation.

Connecting slides (Simple), covers how to interconnect slides inside our presentation and how to access them directly using a hyperlink.

Creating vertical/nested slides (Simple), explains how to create a vertical flow in our presentation.

Using fragmented views (Simple), covers the use of fragments to show the content step by step.

Adding Markdown content (Simple), discusses the use of Markdown syntax instead of HTML.

Customizing slides with global state (Medium), introduces the use of special states to set different colors and features for specific slides.

Working with custom events (Advanced), covers reveal.js JavaScript events by implementing and elapsed time clock.

Using reveal.js JavaScript API (Advanced), explores Reveal.js JavaScript API by creating a custom navigation bar.

Plugins and extensions (Medium), covers how to extend reveal.js by using plugins and extensions.

Exporting our presentation as PDF (Simple), explains how to save our HTML5 presentation as a PDF file using a custom stylesheet.

Server-side speaker notes (Advanced), explains how to use a node.js plugin to create a local server and use server-side speaker notes.

What you need for this book

We assume that you know how to create and edit HTML documents. You also must know how to link CSS files and have some minor knowledge of JavaScript. For some advanced recipes, readers are expected to know how to run commands on a terminal.

Who this book is for

This book is for anyone who has basic frontend development skills and wants to create beautiful HTML5 presentations using the reveal.js library.

Conventions

In this book, you will find a number of styles of text that distinguish between different kinds of information. Here are some examples of these styles, and an explanation of their meaning.

Code words in text are shown as follows: " The first step is to create an HTML file and save it as `index.html`."

A block of code is set as follows:

```
<!doctype html>
<html>
  <head>
    <meta charset="utf-8">
    <title>HTML5 Presentations How-To</title>
  </head>
  <body>
    <div class="reveal">
      <div class="slides">
      </div>
    </div>
  </body>
</html>
```

When we wish to draw your attention to a particular part of a code block, the relevant lines or items are set in bold:

```
Reveal.initialize({
  controls: false,
  keyboard: false,
  center: true,
  transition: 'default',
  mouseWheel: true
});
```

Any command-line input or output is written as follows:

```
node plugin/notes-server
```

New terms and **important words** are shown in bold. Words that you see on the screen, in menus or dialog boxes for example, appear in the text like this: " To install it simply visit http://nodejs.org and click on the **INSTALL** button."

> **Note**

Warnings or important notes appear in a box like this.

Tip

Tips and tricks appear like this.

Reader feedback

Feedback from our readers is always welcome. Let us know what you think about this book—what you liked or may have disliked. Reader feedback is important for us to develop titles that you really get the most out of.

To send us general feedback, simply send an e-mail to <feedback@packtpub.com>, and mention the book title via the subject of your message.

If there is a book that you need and would like to see us publish, please send us a note in the **SUGGEST A TITLE** form on www.packtpub.com or e-mail <suggest@packtpub.com>.

If there is a topic that you have expertise in and you are interested in either writing or contributing to a book, see our author guide on www.packtpub.com/authors.

Customer support

Now that you are the proud owner of a Packt book, we have a number of things to help you to get the most from your purchase.

Downloading the example code

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

Errata

Although we have taken every care to ensure the accuracy of our content, mistakes do happen. If you find a mistake in one of our books—maybe a mistake in the text or the code—we would be grateful if you would report this to us. By doing so, you can save other readers from frustration and help us improve subsequent versions of this book. If you find any errata, please report them by visiting http://www.packtpub.com/support, selecting your book, clicking on the errata submission form link, and entering the details of your errata. Once your errata are verified, your submission will be accepted and the errata will be uploaded on our website, or added to any list of existing errata, under the Errata section of that title. Any existing errata can be viewed by selecting your title from http://www.packtpub.com/support.

Piracy

Piracy of copyright material on the Internet is an ongoing problem across all media. At Packt, we take the protection of our copyright and licenses very seriously. If you come across any illegal copies of our works, in any form, on the Internet, please provide us with the location address or website name immediately so that we can pursue a remedy.

Please contact us at <copyright@packtpub.com> with a link to the suspected pirated material.

We appreciate your help in protecting our authors, and our ability to bring you valuable content.

Questions

You can contact us at <questions@packtpub.com> if you are having a problem with any aspect of the book, and we will do our best to address it.

Chapter 1. Instant HTML5 Presentations How-to

Presentations are no longer limited to platform-specific software. You can now create stunning slides and show them on a web browser using modern technologies such as HTML5 and CSS3 3D transitions. This book will show you, through hands-on exercises, how to quickly create beautiful and functional presentations using the `reveal.js` JavaScript library.

Creating our initial presentation (Simple)

This recipe will teach you how to create a basic HTML5 presentation using `reveal.js`. We will cover the initial markup and setup. The presentation created here will be used throughout the rest of the book.

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

Getting ready

Before we start, we need to download the `reveal.js` script and assets. To do that, simply visit https://github.com/hakimel/reveal.js/downloads and get the latest version of reveal.

Now open your favorite text editor and let's work on the presentation markup.

How to do it...

To create our initial presentation perform the following steps:

1. The first step is to create an HTML file and save it as `index.html`. Then copy the `reveal.js` files to the same directory of our HTML

file, keeping the following structure (highlighted files are the ones we are using in this recipe):

```
index.html
-css
---reveal.min.css
---print
---shaders
---theme
-----default.css
-----source
-----template
-js
---reveal.min.js
-lib
---css
---font
---js
-----head.min.js
-plugin
---highlight
---markdown
---notes
---notes-server
---postmessage
---remotes
---zoom-js
```

2. Now we need to write our initial HTML5 markup, as follows:

```
<!doctype html>
<html>
  <head>
    <meta charset="utf-8">
    <title>HTML5 Presentations
How-To</title>
  </head>
  <body>
    <div class="reveal">
      <div class="slides">
      </div>
    </div>
  </body>
</html>
```

3. With our initial markup ready we include the `reveal.js` base stylesheet and script file plus the default theme stylesheet. We will also need to include the HeadJS JavaScript loader in order to manage `reveal.js` plugins.

   The stylesheets will go directly below the `title` tag, while the script files must be included before the closing `body` tag:

```
<head>
```

```
  <meta charset="utf-8">
  <title>HTML5 Presentations How-To</title>
  <link rel="stylesheet"
href="css/reveal.min.css">
  <link rel="stylesheet"
href="css/theme/default.css">
</head>
<body>
  ...
  <script
src="lib/js/head.min.js"></script>
  <script src="js/reveal.min.js"></script>
</body>
```

4. The fourth step is to create the elements for our slides. Our first presentation will have four slides, so we must add four section elements to our slides `div`, as follows:

```
<div class="reveal">
  <div class="slides">
    <section>
      <h1>HTML5 Presentations</h1>
      <h3>How to create presentations for
web browsers</h3>
    </section>
    <section>
      <h1>About this presentation</h1>
      <p>This is the example presentation
for the book HTML5 Presentations How-To</p>
    </section>
    <section>
      <h1>reveal.js</h1>
      <ul>
        <li>created by Hakim El Hattab</li>
        <li>open source</li>
      </ul>
    </section>
    <section>
      <h1>References</h1>
      <ol>
        <li><a
href="http://hakim.se/">http://hakim.se/</a></li>
        <li><a
href="https://github.com/hakimel/reveal.js">https://github.com/ha
      </ol>
    </section>
  </div>
</div>
```

5. After that, we initialize the `reveal.js` JavaScript object, right after the `reveal.js` script tag, as follows:

```
<body>
  ...
  <script src="js/reveal.min.js"></script>
```

```
    <script>
    Reveal.initialize({
      controls: true,
      keyboard: true,
      center: true,
      transition: 'default'
    });
    </script>
  </body>
```

6. For the last step, open `index.html` using the browser of your choice and check out how our initial HTML5 presentation looks, as shown in the following screenshot:



How it works...

`reveal.js` is a powerful JavaScript library that allows us to create presentations using a web browser. It was created by Hakim El Hattab, a web developer known for his experiments with CSS3 transitions and JavaScript animations.

By performing the preceding steps we created a regular HTML5 presentation. We have used `reveal.js` default themes and some minor configurations just to get started.

We started with a basic HTML5 structure. It is important to bear in mind that our presentation is, before anything else, an HTML and it should be rendered normally in the browser.

The presentation must follow a structure expected by the `reveal.js` script. First, our HTML must have one `div` with `reveal` as a class attribute. That `div` must contain another `div` with `slides` as a class attribute.

The slides `div` must contain one or more section elements, each one representing a slide. As we will see in the *Creating vertical/nested slides (Simple)* recipe, it is possible to nest slides, creating a vertical navigation.

Each slide can contain any HTML or markdown content. We started with just a few elements but we will enrich our presentation through each recipe.

`reveal.js` comes with multiple themes, available under the `css/theme` directory. For our initial presentation, we are using the `default.css` theme. Other options would be `beige`, `night`, `serif`, `simple`, and `sky`. Theming will be covered in the *Theming (Medium)* recipe.

With all the markup and styles set, we initialized the `reveal.js` JavaScript object by using the `initialize` method. We used only four options, as follows:

```
Reveal.initialize({
  controls: true,
  keyboard: true,
  center: true,
  transition: 'default'
});
```

First we enabled the navigation controls at the user interface, setting the `controls` option to `true`. We activated keyboard navigation using the arrow keys and also centered the slides' content. The last option sets the transition between slides to default. The next recipe will cover the full list of available options.

There's more...

`reveal.js` will work on any browser, but modern browsers are preferred due to animations and 3D effects. We will also need to use external plugins in order to extend our presentation features.

Dependencies

As previously mentioned, `reveal.js` uses the HeadJS library to manage plugins and dependencies. You can use only `reveal.js` if you want, but some useful resources (such as speaker notes and zoom) are only enabled using plugins.

The plugins can be loaded through the `initialize` method and we will see more about them in the *Plugins and extensions (Medium)* recipe.

Overview and blackout modes

While on our presentation, pressing the *Esc* key will switch to overview mode where you can interact with every slide.

Another useful resource is the blackout mode. If you want to pause and draw attention from your audience, you can press the *B* key to alternate between normal and blackout modes, and hide your current slide data.

Browser support

`reveal.js` will work better on browsers with full support for CSS 3D animations (Chrome, Safari, Firefox, and Internet Explorer 10), but fallbacks (such as 2D transitions) are available for older browsers.

Presentations will not work on Internet Explorer 7 or lower.

In this recipe we have created our initial HTML5 presentation using `reveal.js` default options and theme. The next recipes will demonstrate how to fully customize our slides' appearance and enhance our presentation experience.

Customizing our presentation (Simple)

Now that we have created our initial presentation, let's learn how to customize it via `reveal.js` options.

How to do it...

To customize our presentations perform the following steps:

1. First, we need to open `index.html` and change the options used on our `reveal.js` initialization. Let's start by setting the `mouseWheel` option to `true`:

```
Reveal.initialize({
  controls: true,
  keyboard: true,
  center: true,
  transition: 'default',
  mouseWheel: true
});
```

2. We can also remove navigation controls by changing the option initialization value to `false`:

```
Reveal.initialize({
  controls: false,
  keyboard: false,
  center: true,
  transition: 'default',
  mouseWheel: true
});
```

3. How about setting up our presentation to automatically switch between slides after a certain amount of time? To accomplish that we will use the `autoSlide` option. This option expects the number of milliseconds to wait for before changing to the next slide.

```
Reveal.initialize({
  controls: false,
  keyboard: true,
  center: true,
  transition: 'default',
  mouseWheel: true,
  autoSlide: 6000
});
```

4. It is also possible to remove the 3D effect on links inside the presentation using the `rollingLinks` option:

```
Reveal.initialize({
  controls: false,
  keyboard: true,
  center: true,
  transition: 'default',
  mouseWheel: true,
  autoSlide: 6000,
  rollingLinks: false
});
```

How it works...

`reveal.js` comes with a handful of options to customize our presentation's look and feel without too much work. Later on we will see some more advanced customizing options, but for now we will work on all the default options' possible values and customizations.

Using the preceding steps we customized some aspects of our presentation, such as transition effect and controls.

The option argument, which receives as a parameter an object containing the options and their values, must be set at the `reveal.js` initialize method call.

Now let's take a look at the full list of `reveal.js` initialization options and their default values:

Option	Description	Default value
`controls`	Hides or displays the navigation controls positioned at the lower-right corner.	`true`
`progress`	Shows a smooth progression bar of our presentation.	`true`
`history`	Enables history support to use the back and forward buttons of the browser.	`true`
`keyboard`	Turns on keyboard navigation between slides using the arrow keys.	`true`

Option	Description	Default value
`overview`	Enables/disables the overview mode using the *Esc* key.	`true`
`center`	Centers the presentation slides vertically according to the browser viewport dimensions.	`true`
`loop`	Enables/disables presentation loop.	`false`
`rtl`	Switches the presentation direction to be **RTL** (**right-to-left**).	`false`
`autoSlide`	Implements a timer in milliseconds to enable automatic presentation.	0
`mouseWheel`	Enables/disables slides navigation using the mouse wheel.	`false`
`rollingLinks`	Sets a 3D roll effect to link elements included in the presentation.	`true`
`transition`	Determines the transition effect between slides. Possible values are `cube`, `page`, `concave`, `zoom`, `linear`. You can also turn off any transition effect by using `'none'` as value.	`'default'`

Here we have covered `reveal.js` initialization options allowing us to customize our presentation experience, including slide transition style, keyboard navigation, and browser history.

Adding speaker notes (Simple)

Speaker notes are powerful resources for any presentation. In this recipe we will learn how to add and work with notes using `reveal.js`.

How to do it...

To add speaker notes perform the following steps:

1. First we need to change the `reveal.js` initialization to include the speaker notes dependency:

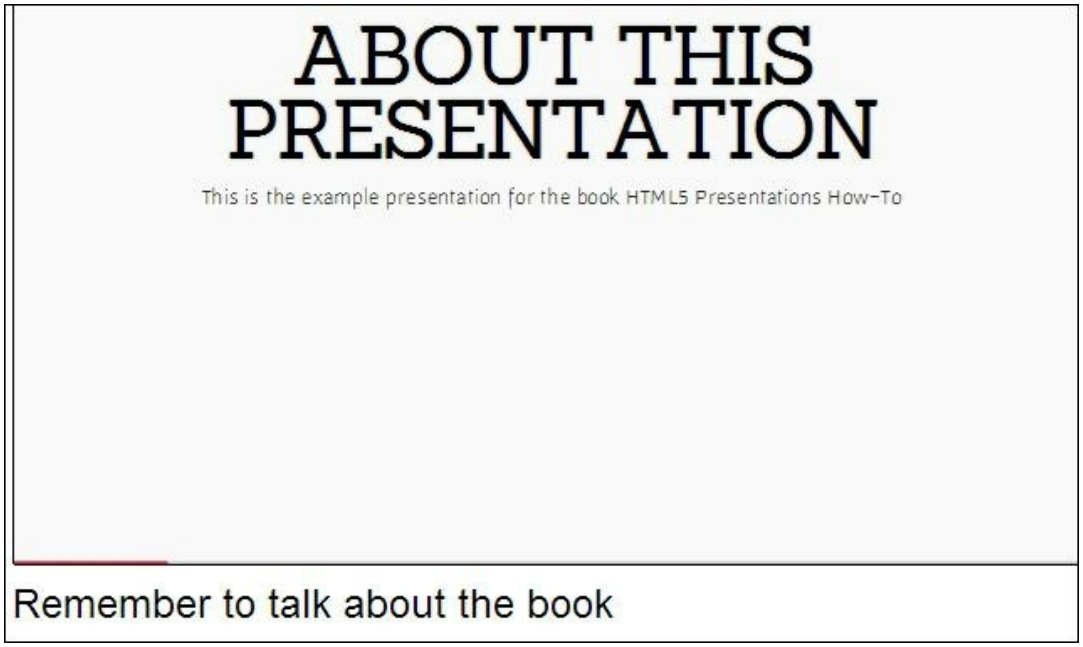
```
Reveal.initialize({
  controls: false,
  keyboard: true,
  center: true,
  transition: 'default',
  mouseWheel: true,
  autoSlide: 6000,
  rollingLinks: false,
  dependencies: [{
    src: 'plugin/notes/notes.js',
    async: true
  }]
});
```

2. To attach speaker notes to our presentations we must use the HTML5 tag `aside` with notes as a class attribute value, as follows:

```
<section>
  <h1>HTML5 Presentations</h1>
  <h3>How to create presentations for web
browsers</h3>
  <aside class="notes">Remember to talk
about the book</aside>
</section>
```

How it works...

Speaker notes will only be available if we open our presentation with the `?notes` parameter on the URL or when we press the *S* key.



The dependencies initialization option takes a list as an argument. The list must contain one or more dependencies to inject in our presentation. For this recipe, we only need the notes plugin dependency.

Each dependency object has the following four properties:

- `source`: This decides the relative path to the script
- `async` (optional): If this property is set to `true`, the script will be loaded only after `reveal.js` has been loaded
- `callback` (optional): `callback` is a function to be executed when the dependency is loaded
- `condition` (optional): This parameter is a condition that must be matched for the plugin to be loaded (usually a function that returns `true` or `false`)

`reveal.js` will search for `aside` elements inside our slides and will show their contents along with the slide on the speaker notes panel.

You can add as many notes and content as you wish, since they will not be visible on the presentation window.

For instance, the following URL would open a speaker notes window along with the presentation window:

http://www.daviferreira.com/html5-presentations-how-to/index.html?notes

There's more...

The speaker notes interface depends on some extra configurations on your browser, such as disabling pop-up blockers.

Presentation control

Ideally, we should be using two monitors – one with the presentation itself and the other with the speaker notes panel. The speaker notes panel is also a nice way to control our presentation flow, since we can check on upcoming slides.

Pop up

Since the speaker notes window opens as a new window, we need to set up our browser not to block pop ups.

Enabling file access from files

Opening the presentation using the file protocol may cause a security warning on Google Chrome. You can fix that by using a local web server or by starting Chrome with the following initialization parameter:

```
--allow-file-access-from-files
```

Installing and using Sass (Advanced)

We need to have Sass installed and configured in order to create and customize `reveal.js` presentation themes.

Sass (**Syntactically Awesome Stylesheets**) is an improved version of CSS3. It adds new features such as variables, functions, and nested rules. Sass has its own syntax which is very similar to CSS declarations. Using the command line or a graphical tool, the styles you write using Sass will be converted to regular CSS rules. Sass files use the `.scss` extension.

Getting ready

There are two ways to use Sass. The first one is using the terminal (or command prompt if you are using Windows) by installing Sass as Ruby gem. That method requires you to have the Ruby language installed and configured on your system.

If you are using a Mac or are on a Linux computer you probably already have the Ruby language installed. To check, type the following command on a terminal:

```
ruby --version
```

The output should be something like the following:

```
Ruby 1.9.3p362 (2012-12-25 revision 38607)
[x86_64-darwin10.0.0]
```

Windows users can visit http://rubyinstaller.org/downloads/ and download the latest installer.

Once you have the Ruby language installed on your system, you can install Sass by typing the `gem install` command on a terminal (command prompt on Windows):

```
gem install sass
```

The preceding command will install the latest Sass version. To check if it was correctly installed, type in the following command on a terminal:

```
sass --version
```

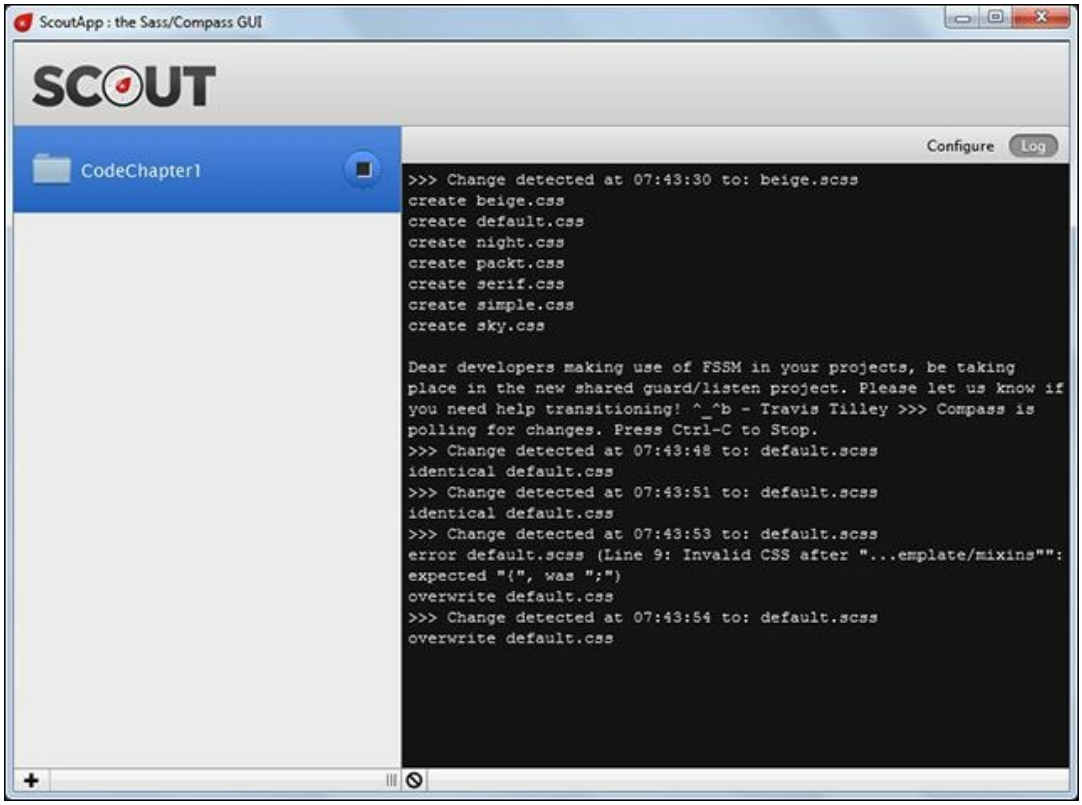
The output should look like the following:

```
Sass 3.2.5 (Media Mark)
```

The second way is to use a tool that can compile Sass using a graphical interface. If you are not a fan of the command line, you can use some applications to generate CSS files from Sass. The following is a list of available software:

- **ScoutApp (PC and Mac)**: http://mhs.github.com/scout-app/
- **Compass.app (PC, Mac, and Linux)**: http://mhs.github.com/scout-app/
- **CodeKit (Mac)**: http://incident57.com/codekit/
- **LiveReload (Mac)**: http://livereload.com/

The following is the screenshot showing the command line in ScoutApp:



How to do it…

To learn how to use Saas perform the following steps:

1. Now that we have Sass installed let's create a new stylesheet. Using your favorite editor create a file called `sass-example.scss` and save it inside the `css` directory of our presentation.
2. Let's start by using variables, one of the coolest features of Sass. A variable saves a value that can be reused on different stylesheets. It is ideal for colors, fonts, and backgrounds. Sass variables are declared using the `$font_size` character, as follows:

```
$text_color: #000;
$font_size: 12px;

body {
  font-size: $font-size;
  color: $text_color;
}

h1 {
  color: $text_color;
}
```

   The good thing about using variables is that we need to change the value only at one place (the variable declaration).

3. Another cool feature of Sass is the possibility of using nested declarations. For example, let's say we have `span` inside our `h1` tag. We could use a nested declaration to delimit the CSS rule scope. In that case, the rule will be applied only to `span` tags inside `h1` tags:

```
h1 {
  color: $text_color;
  span {
    font-size: 10px;
  }
}
```

4. It is time to check the CSS result of our Sass code. At the command line, inside the `css` directory, type in the following command to compile the Sass declarations and generate our `sass-example.css` file:

```
sass sass-example.scss sass-example.css
```

5. If you open the `sass-example.css` file, you will see that the following CSS rules were generated from our variables and nested declarations:

```
body {
font-size: 12px;
color: black;
```

```
}

h1 {
color: black;
}
h1 span {
font-size: 10px;
}
```

How it works…

The main goal of Sass is to make CSS more fun to write by giving it new capabilities. Using the command line compiler or a graphical tool you can write style declarations with variables, nested rules, functions, and many other features. The generated CSS file can be directly attached to any HTML file.

One of the best parts of Sass is allowing the reuse of rules and declarations, avoiding repetitions and making stylesheets maintenance much easier. Using variables, for example, you can concentrate all your default values for text colors, font sizes, links behaviors, and more in a single place.

There is more...

There is much more coming from Sass than variables and nested rules. This recipe covered only the installation and basic usage, but with Sass you can also extend your CSS declarations by using external files and mixins.

Imports

Sass allows you to reference external files inside any `.scss` file. For example, we could have created a `variables.scss` file instead of declaring our variables directly inside our stylesheet and then referenced it via the `@import` declaration:

```
@import "variables"

body {
  font-size: $text_size;
  color: $text_color;
}
```

This increases the reuse of our variables, making it possible to use them across different stylesheets.

Mixins

Another way to increase the reuse of our CSS rules is by using mixins.

Mixins are a set of CSS declarations that can be called inside any rule using the `@include` declaration. For example, in order to avoid typing cross-browser rules for `border-radius`, we could use the following mixin:

```
@mixin border-radius {
  -moz-border-radius: 3px;
  -webkit-border-radius: 3px;
  border-radius: 3px;
}

#navbar {
  @include border-radius;
}

#button {
  @include border-radius;
}
```

But what if we need different values for the border radius? Sass mixins can have arguments for different values. Optionally, a mixin argument can have a default value:

```
@mixin border-radius($radius: 3px) {
  -moz-border-radius: $radius;
  -webkit-border-radius: $radius;
  border-radius: $radius;
}

#nav {
  @include border-radius(10px);
}

#button {
  @include border-radius;
}
```

Like variables, we can create mixins inside a separated file and reference them by using the `@import` declaration.

Compass

Just like Sass is an extension of CSS, Compass is an extension of Sass. Compass is a set of mixins and pre-defined variables, mostly CSS3 declarations such as `border-radius`, `box-shadow`, `text-shadow`, and `transition`.

You can learn more about it on http://compass-style.org.

Theming (Medium)

Theming is an important part of creating presentations, as it will give our slides an appropriate look and feel according to our presentation subject. In this recipe we will cover how to use the default themes and how to create a new one using a `reveal.js` Sass template.

Getting ready

To create a new theme you must have Sass installed on your computer. Sass is a Ruby gem that improves the power of CSS stylesheets. It implements variables, mixins, and functions using a pre-compiled language that will result in regular CSS flavored files.

To check if Sass is installed, type in the following command on a terminal:

```
sass --version
```

If you don't like the command line, you will find several applications and plugins that compile Sass to CSS. For more information, visit http://sass-lang.com/.

How to do it...

To learn theming perform the following steps:

1. As mentioned previously, besides the default theme, `revel.js` comes with five different themes for changing the appearance of our presentations. To use them, we simply need to change the stylesheet reference. In the following code snippet we are changing our presentation to use the serif theme:

```
<head>
  <meta charset="utf-8">
  <title>HTML5 Presentations How-To</title>
  <link rel="stylesheet"
href="css/reveal.min.css">
  <link rel="stylesheet"
href="css/theme/serif.css">
</head>
<body>
  ...
  <script
src="lib/js/head.min.js"></script>
  <script src="js/reveal.min.js"></script>
</body>
```

2. Now open your web browser and check the new look of our slides. Try all the available values before we move on – `beige`, `night`, `serif`, `simple`, and `sky`.
3. That being done, it is time to create a new theme using the `reveal.js` base template, which can be found at the `css/theme/template` directory.

   Our new theme will be called `packt` and we will save it at the `css/theme/source` directory, where all the `reveal.js` default themes are stored.

   Open your favorite text editor and let's start to customize our theme. We will basically change colors and fonts, overriding the default theme settings. The following is our theme code:

```
@import "../template/mixins";
@import "../template/settings";

@import
url(http://fonts.googleapis.com/css?
family=Sanchez
);
@import
url(http://fonts.googleapis.com/css?
family=Gafata);

$backgroundColor: #f9f9f9;
$mainFont: 'Gafata', Helvetica, Arial,
sans-serif;
$mainColor: #333;
$linkColor: #df0d32;
$linkColorHover: darken( $linkColor,
50% );
$headingFont: 'Sanchez', serif;
$headingColor: #000;
$headingLetterSpacing: -0.05em;
$headingTextTransform: uppercase;
$selectionBackgroundColor: #4ca5fe;
$mainFontSize: 40px;

@import "../template/theme";
```

4. Save it as `packt.scss` inside the `css/theme/source` directory and run the following command to compile our theme:

```
sasspackt.scss ../packt.css
```

5. The preceding command will generate a file called `packt.css` inside the `css/theme` directory and we can now use it as our presentation stylesheet:

```
<head>
```

```
  <meta charset="utf-8">
  <title>HTML5 Presentations How-To</title>
  <link rel="stylesheet"
href="css/reveal.min.css">
  <link rel="stylesheet"
href="css/theme/packt.css">
</head>
<body>
  ...
  <script
src="lib/js/head.min.js"></script>
  <script src="js/reveal.min.js"></script>
</body>
```

6. Reload the presentation and check how our custom theme looks.

How it works...

`reveal.js` uses Sass to modularize its theme files. Editing only the theme variables you can quickly generate new looks for your presentations.

Open the `settings.scss` file located at the `css/themes/template` directory and take a look at all the available options for customizing new themes. Besides the presentation's background, the settings file is divided into four other sections, as follows:

- Primary/body text
- Headings
- Links and actions
- Text selection

The base theme also uses some Sass functions such as `rgba` and `lighten`. The first one creates an `rgba` CSS declaration for any given color, in any format (`rgb`, `hexa`, and so on). The `lighten` function returns a lighter version of the given color. In our `packt` theme we use its counterpart, the `darken` function, to decrease the lightness of the link color.

To check all Sass options and functions, visit the language reference at http://sass-lang.com/docs/yardoc/file.SASS_REFERENCE.html.

The settings file also includes a body background mixin to generate a background for our presentation. You can overwrite it, and use images and gradients if you want.

On a side note, there is nothing to stop you from creating your theme using plain CSS, but using Sass will make this a much easier task.

Setting up slide transitions (Simple)

In the *Customizing our presentations (Simple)* recipe we learned how to set up `reveal.js` initialization options, including slides transition effects. In this recipe we will learn more about the different types of transitions and how to customize them between slides.

How to do it...

To set up slide transitions perform the following steps:

1. To change `reveal.js` transition we need to update our object initialization. Let's start with the true transition:

```
Reveal.initialize({
  controls: false,
  keyboard: true,
  center: true,
  transition: 'cube',
  mouseWheel: true,
  autoSlide: 6000,
  rollingLinks: false,
  dependencies: [{
    src: 'plugin/notes/notes.js',
    async: true

  }]
});
```

2. Open your browser and check how the transition works. Let's try different values. Change our transition initialization options to `concave`:

```
...
  transition: 'concave',
...
```

3. Now, let's try out every possible value left for our slide transitions effect. Change the initialization option to use the values given here and test it on the browser – `page`, `concave`, `zoom`, `linear`, and `none`.
4. It is also possible to create custom transitions. Add the following CSS code to our HTML file, inside the `head` element:

```
<head>
  ...
  <style>
  .reveal.packt .slides>section.past {
```

```
    -webkit-transform: scale(.1);
    -moz-transform: scale(.1);
    -ms-transform: scale(.1);
    -o-transform: scale(.1);
    transform: scale(.1);
  }
  .reveal.packt .slides>section.future {
    -webkit-transform: scale(2.5);
    -moz-transform: scale(2.5);
    -ms-transform: scale(2.5);
    -o-transform: scale(2.5);
    transform: scale(2.5);
  }
  </style>
</head>
```

5. Then, change the `transition` initialization option's value to `packt`:

```
...
  transition: 'packt',
...
```

How it works...

`reveal.js` uses CSS3 animations to implement slide transitions. Using the `transition` initialization option we can set up different kinds of effects when changing slides during our presentation.

The slide transitions are nothing but CSS classes that are applied to our slides. If you want you can create new transitions, with custom effects, and add them to your presentation stylesheet. The transition must have four selectors, each one representing one step of the slide switching effect:

```
.reveal.<transition_name> .slides>section.past
.reveal.<transition_name> .slides>section.future
```

Inside those selectors you can virtually add or alter any CSS property you want (including attributes of child elements).

The `past` class will handle the current slide while the `future` class is responsible for styling the next/previous slide.

Our custom `packt` transition effect increases and decreases the slide content during the animation effect, generating a nice and different slide transition.

It is possible to use different transitions across the presentation using `reveal.js` JavaScript API. We will see more about that in the *Using reveal.js JavaScript API (Advanced)* recipe.

Connecting slides (Simple)

Taking advantage of hyperlinks, it is possible to connect slides in our presentation and access them directly using a mouse click.

How to do it...

To connect slides perform the following steps:

1. It is time to increase the number of slides in our presentation. Let's add a new slide after the **References** slide:

```
<div class="slides">
  …
  <section>
    <h1>About the author</h1>
    <p>passionate web developer</p>
    <p>@davitferreira</p>
  </section>
</div>
```

2. Now, in our opening slide, let's add a direct link to our brand new **About the author** slide:

```
<section>
  <h1>HTML5 Presentations</h1>
  <h3>How to create presentations for web
browsers</h3>
  <a href="#/4">About the author</a>
</section>
```

3. By clicking on the **about the author** hyperlink in our opening slide, our presentation will go straight to the fifth slide.

How it works...

`reveal.js` comes with a smart and simple way of connecting slides. Any link started with the hash character followed by a slash and a number will take the presentation to the slide represented by that number in the sequence. The number actually represents the slide index, starting from 0.

For instance, a link with `#/2` as the `href` attribute value will take us to the third slide in our presentations.

There's more...

We can also link to external references in our presentation.

External links

We have already used external links in our **References** slide. Since `reveal.js` presentations are created using HTML5 and CSS3, pretty much anything in this universe is available to attend our creativity.

You can set different link styles (including hover, focus, active, and visited), use different targets, and set titles to any link element in our presentation.

Creating vertical/nested slides (Simple)

One cool resource available in `reveal.js` is the nested slides feature that allows us to create a vertical slide flow inside our presentation.

Nested slides are especially good to present a subset of contents inside a main slide.

How to do it...

To create vertical/nested slides perform the following steps:

1. To create vertical slides we simply need to add multiple section elements inside one parent slide section element. Add a new slide at the end of the presentation with the following markup:

```
<div class="slides">
  …
  <section>
    <section>
      <h1>Why use reveal.js?</h1>
    </section>
    <section>
      <h1>Cross browser and platform</h1>
    </section>
    <section>
      <h1>Open source!</p>
    </section>
  </section>
</div>
```

2. We can add as many slides as we want.

How it works...

Nested slides are child elements inside one main slide. They follow the same rules and options of our main presentation.

Differently from the main flow, vertical slides will animate and change from top to bottom. When we reach the end of the vertical navigation, the presentation continues to the next horizontal slide.

There's more...

Nested slides can have internal links in our presentation just as normal slides. We can also set specific transitions for our vertical slides.

Linking nested slides

We can also create internal links in the presentation to vertical slides. The base link works the same as explained before, with a hash character followed by a slash and the slide index. We will then add the part involving the nested slide using another slash and the index of the vertical slide related to the vertical navigation.

That way, the link to the second slide inside the vertical navigation we created previously would be `#/5/1`.

Nested slides transition

In order to customize nested slides transitions we need to add two selectors to our `packt` CSS class so they will affect the child elements as well:

```
.reveal.packt .slides>section>section.past
.reveal.packt .slides>section>section.future
```

Since they use different selectors, they can have different transitions from the horizontal flow slides.

Using fragmented views (Simple)

This recipe will teach you how to add steps to your slides using the `reveal.js` fragmented views feature.

How to do it...

To use fragmented views open your **About the author** slide and add the following highlighted code:

```
<div class="slides">
  …
  <section>
    <h1>About the author</h1>
    <p>passionate web developer</p>
    <p>@davitferreira</p>
    <p class="fragment">Developer at globo.com</p>
    <p class="fragment">Front-end engineer</p>
    <p class="fragment">Born and based at Rio de
Janeiro, Brazil</p>
  </section>
</div>
```

How it works...

Any element with the `fragment` class will load step by step in our presentation. It is ideal for listing topics without revealing any extra information to the audience.

The default behavior is to hide the fragment and show it when you advance in your presentation.

`reveal.js` makes possible the use of different effects to highlight fragments, as follows:

- `grow`
- `shrink`
- `roll-in`
- `fade-out`
- `highlight-red`
- `highlight-green`
- `highlight-blue`

Try it out, changing the fragments we have created previously. For example:

```
<div class="slides">
```

```
  …
  <section>
    <h1>About the author</h1>
    <p>passionate web developer</p>
    <p>@davitferreira</p>
    <p class="fragment highlight-red">Developer
at globo.com</p>
    <p class="fragment roll-in">Front-end
engineer</p>
    <p class="fragment grow">Born and based at
Rio de Janeiro, Brazil</p>
  </section>
</div>
```

You can also create custom fragment classes using the following selector:

```
.reveal .slides
section .fragment.<custom_fragment_class>
```

Adding Markdown content (Simple)

Markdown is a popular syntax to simplify and write HTML content. It removes the need for tags and implements special patterns that result in HTML.

For example, let's take a look at the following HTML code:

```
<h1>HTML5 Presentations How-to</h1>
<p>This book will teach you how to create
stunning presentations for the browser</p>
<h2>References</h2>
<ul>
<li><a
href="http://hakim.se/">http://hakim.se/</a></li>
<li><a
href="https://github.com/hakimel/reveal.js">https://github.com/hakimel/re
</ul>
```

Using Markdown syntax it could be simply written in the following form:

```
# HTML5 Presentations How-to

This book will teach you how to create stunning
presentations for the browser

## References

* <http://hakim.se/>
* <https://github.com/hakimel/reveal.js>
```

Markdown syntax is really easy to learn and will save you a large amount of time once you master it. It is faster, more flexible and, above all, cleaner than regular HTML tags.

How to do it...

To add Markdown content perform the following steps:

1. The first step is to enable the `markdown` and `showdown` plugins:

```
Reveal.initialize({
  controls: false,
  keyboard: true,
  center: true,
  transition: 'cube',
  mouseWheel: true,
  autoSlide: 6000,
  rollingLinks: false,
```

```
    dependencies: [
      {
        src: 'plugin/notes/notes.js',
        async: true

      },
      {
        src: 'plugin/markdown/showdown.js',
        async: true
      },
      {
        src: 'plugin/markdown/markdown.js',
        async: true
      }
    ]
});
```

2. In order to use Markdown content, we need to add the `data-markdown` attribute to our slides. Let's try it with our last slide, **Why use reveal.js**:

```
<section>
  <section data-markdown>
    <h1>Why use reveal.js?</h1>
  </section>
  …
</section>
```

3. With the `data-markdown` attribute set (note that it doesn't need a value; it just needs to be declared), we can now add Markdown content to the slide. To do that we need to use a `script` tag with the `text/template` type:

```
<section>
  <section data-markdown>
    <script type="text/template">
      # Why use reveal.js?

      Check the full list of features
[here](http://hakim.se/revealjs).
    </script>
  </section>
  …
</section>
```

How it works...

The ability of `reveal.js` to use Markdown syntax comes from a Paul Irish gist (https://gist.github.com/1343518), which uses the `showdown` JavaScript plugin to render the content.

John Gruber created the Markdown syntax, which has as its main goal to reduce the gap between HTML and the final content. It is easy to write and, most important of all, to read.

Our Markdown text sets a header, a paragraph, and a link. It will result in the following HTML code:

```
<h1>Why use reveal.js</h1>
<p>Check the full list of features <a href="
http://hakim.se/revealjs">here</a>.
```

To check the full list of Markdown syntax visit its official site at http://daringfireball.net/projects/markdown/.

Customizing slides with global state (Medium)

With global states it is possible to customize single slides using CSS. This is the easiest way of adding a unique look and feel to specific slides.

How to do it...

To customize slides with global state perform the following steps:

1. Let's set a global state to our **About this presentation** slide. Global states are applied to the `data-state` attribute of the `section` element:

```
…
<section data-state="packt-state">
  <h1>About this presentation</h1>
  <p>This is the example presentation
for the book HTML5 Presentations How-To</p>
</section>
…
```

2. Since we have used a `packt` class name for our custom transitions, here we had to use a different name. `data-state="packt-state"` will add `"packt-state"` as the class attribute of the document element, making it possible to customize everything on the current slide.
3. With that said, we need to create the new styles and add them to our `style` section:

```
<style>
…
  .packt-state .reveal .state-background {
    background: rgba(0, 0, 0, .6);
  }
  .packt-state h1 {
    color: yellow;
  }
  .packt-state p {
    color: #fff;
  }
</style>
```

Now, as soon as we enter the **About this presentation** slide, it will receive a dark background color, its header will be yellow, and its paragraph text will be white.

4. Besides adding a class attribute to the document element, global states also trigger JavaScript events. We can listen to them and implement JavaScript code specific to the active slide.

   Inside our `script` section, right after the `Reveal` initialization, let's add the following code:

```
<script>
…
  Reveal.addEventListener('packt-state',
function() {

 window.open('http://lab.hakim.se/reveal-
js');
  });
</script>
```

   Due to the preceding code, whenever we enter the **About this presentation** slide a new browser window will be opened with the `reveal.js` official website on it.

How it works...

Using global states is the best way of customizing the visual experience of single slides.

`reveal.js` will add the `data-state` value as a class attribute of the document element allowing us to change everything in our current slide using CSS.

Global states will also trigger a JavaScript event with the same name as the state. Your presentation can listen to it, and add JavaScript functions and effects to the current slide.

In order to listen to the event we used the JavaScript native method, `addEventListener` sending two parameters – the name of the event (`'packt-state'`) and the listener function.

Working with custom events (Advanced)

`reveal.js` comes with handy custom events triggered by specific actions, such as initialization and slide change, which can be very useful in certain situations.

In this recipe we will implement an elapsed time clock on our presentation.



How to do it...

To work with custom events perform the following steps:

1. Let's start with the `ready` event, which is triggered when `reveal.js` and all its dependencies are loaded. Our brand new improvement to the presentation is an elapsed time clock to indicate the length of our presentation so far.

   First we need to add some new HTML elements to our `index.html` file. Right after our opening `body` tag, add the following code:

```
<body>
  <div class="timer">
    Elapsed time: <span
id="elapsed-time">00:00:00</span>
  </div>
  …
</body>
```

These new elements include a `span` tag with `elapsed-time` as the value for the `id` attribute. That is the element that will receive the updated elapsed time value.

2. The `timer` element needs some style. We will be positioning it in the upper-right corner of our presentation:

```
<style>
  …
  .timer {
    z-index: 1000;
    position: absolute;
    top: 0;
    right: 0;
    padding: 12px;
    background-color: #df0d32;
    color: #fff;
    font-family: Helvetica, Arial,
sans-serif;
  }
</style>
```

3. If you reload our presentation, you will be able to see the elapsed time clock in the upper-right corner of the presentation. The timing magic will occur with some JavaScript functions. The following is the full `elapsedTime` function (there's more about this in the *How it works…* section):

```
<script>
…
var elapsedTime = function () {
  var  seconds = -1,
    clock =
document.getElementById('elapsed-time'),
    pad = function (n) {
      return (n < 10 ? '0' + n : n);
    },
    elapse = function () {
      var s, h, m;
      seconds++;
      s = seconds;
      h = Math.floor(s / 3600);
      s %= 3600;
      m = Math.floor(s / 60);
      s %= 60;
      clock.innerHTML = pad(h) + ':' +
pad(m) + ':' + pad(s);
    };
  window.timer = setInterval(elapse, 1000);
  elapse();
};
</script>
```

4. Our next step is to add the `elapsedTime` function as a listener to the `'ready'` event of `reveal.js`.

```
<script>
…
Reveal.addEventListener('ready',
elapsedTime);
</script>
```

5. As a final touch, we need to stop the elapsed time clock as soon as we reach the last step. To do that, let's add a `'data-last'` attribute to our last slide:

```
<div class="slides">
  …
  <section>
    <section>
      <h1>Why use reveal.js?</h1>
    </section>
    <section>
      <h1>Cross browser and platform</h1>
    </section>
    <section data-last>
      <h1>Open source!</p>
    </section>
  </section>
</div>
```

6. Another custom event provided by `reveal.js` is the `slidechanged` event, triggered every time we change a slide. This event returns an object containing the horizontal and vertical indexes of our presentation, plus the previous and current slide element objects.

   We will use the current slide element to verify the presence of the `data-last` attribute:

```
<script>
Reveal.addEventListener('slidechanged',
function(e) {
  if
(e.currentSlide.getAttribute('data-last') !
== null) {
    clearInterval(window.timer);
  }
});</script>;
```

7. Now try opening the presentation in the browser and check the timer ticking after it has been loaded.

How it works...

Events are a powerful tool when talking about JavaScript programming. `reveal.js` comes with two basic types of events to listen to – `ready` and `slidechanged`.

These events are triggered according to custom actions executed by the `Reveal` object. The `ready` event will be dispatched when `reveal.js` and all its dependencies have been loaded, while the `slidechanged` event is triggered every time we switch slides in our presentation.

By modifying our presentation HTML structure we have added an elapsed time clock. The real action occurs with some JavaScript code. Let's break it in order to understand each part of the script.

First we created a new global variable called `elapsedTime` with a function as its content. This is called an expression:

```
var elapsedTime = function () {
};
```

Then, inside our `elapsedTime` expression, we have instantiated a couple of variables and expressions:

```
var elapsedTime = function () {
  var  seconds = -1,
    clock =
document.getElementById('elapsed-time'),
    pad,
    elapse;
};
```

Our `seconds` variable starts with a value of `-1`, because when we tick for the first time the clock will be set to 0 seconds. The `clock` variable is saving the clock element so we don't have to look at the DOM tree every time our clock ticks.

The `pad` variable will receive a function to add an extra 0 to numbers less than 10. Finally, the `elapse` variable will receive the function to tick the elapsed time clock.

> **Note**
>
> As good practice, we are using a single `var` declaration for instantiating our variables.

Our `pad` function will simply check if a number is less than 10 and add an extra 0 to its left:

```
var elapsedTime = function () {
  var  seconds = -1,
    clock =
document.getElementById('elapsed-time'),
    pad = function (n) {
      return (n < 10 ? '0' + n : n);
    },
    elapse;
};
```

Our most important actor is the `elapse` function since it is responsible for getting the clock running:

```
var elapsedTime = function () {
  var  seconds = -1,
    clock =
document.getElementById('elapsed-time'),
    pad = function (n) {
      return (n < 10 ? '0' + n : n);
    },
    elapse = function () {
      var s, h, m;
      seconds++;
      s = seconds;
      h = Math.floor(s / 3600);
      s %= 3600;
      m = Math.floor(s / 60);
      s %= 60;
      clock.innerHTML = pad(h) + ':' + pad(m) +
':' + pad(s);
    };
};
```

We will be calling the `elapse` function within an interval of 1 second. What it simply does is to transform the number of elapsed seconds into the `hh:mm:ss` format and apply it to the `clock` element's `innerHTML` property.

Our script's last two lines will set the elapsed time clock to run within an interval of 1 second and immediately run it the first time:

```
var elapsedTime = function () {
  var  seconds = -1,
    clock =
document.getElementById('elapsed-time'),
    pad = function (n) {
      return (n < 10 ? '0' + n : n);
    },
```

```
    elapse = function () {
      var s, h, m;
      seconds++;
      s = seconds;
      h = Math.floor(s / 3600);
      s %= 3600;
      m = Math.floor(s / 60);
      s %= 60;
      clock.innerHTML = pad(h) + ':' + pad(m) +
':' + pad(s);
    };
  window.timer = setInterval(elapse, 1000);
  elapse();
};
```

There's more...

Fragments have their own events that we can listen to and use with some additional JavaScript.

Fragment events

In this recipe we have seen two important events triggered by `reveal.js` – `ready` and `slidechanged`.

But there are two other events triggered by the `Reveal` object – `fragmentshown` and `fragmenthidden`. These events are dispatched when slide fragments are shown or hidden, and you can use them to add some JavaScript magic between fragment transitions.

```
Reveal.addEventListener('fragmentshown',
function(event) {
  …
});

Reveal.addEventListener('fragmenthidden',
function(event) {
  …
});
```

The event received by the function contains the fragment DOM element inside the `event.fragment` object. Using that you can apply some JavaScript code to specific fragments through the `id` element or any other unique attribute.

Using reveal.js JavaScript API (Advanced)

`reveal.js` provides an extended API to control our presentation behavior. With `reveal.js` API methods we can move to previous and next slides, get the current index, toggle overview mode, and more.

How to do it…

To use `reveal.js` JavaScript API perform the following steps:

1. Using `reveal.js` API we will create a navigation bar that will be positioned at the bottom of our presentation screen, containing links to our main slides plus links to the first, next, previous, and last slides.

   We will start by adding some HTML5 markup, including `reveal.js` API method calls inside the `onclick` attribute of our navigation elements:

```
<body>
  <div id="navigation-bar">
    <nav>
      <a href="#"
onclick="Reveal.slide(0);">&laquo;&laquo;
first</a>
      <a href="#"
onclick="Reveal.prev();">&laquo;
previous</a>
      <a href="#"
onclick="Reveal.slide(0);">html5
presentations</a>
      <a href="#"
onclick="Reveal.slide(1);">about this
presentation</a>
      <a href="#"
onclick="Reveal.slide(2);">reveal.js</a>
      <a href="#"
onclick="Reveal.slide(3);">references</a>
      <a href="#"
onclick="Reveal.slide(4);">about the
author</a>
      <a href="#"
onclick="Reveal.slide(5);">why use
reveal.js</a>
      <a href="#"
onclick="Reveal.next();">next &raquo;</a>
```

```
      <a href="#"
onclick="Reveal.slide(5);">last
&raquo;&raquo;</a>
    </nav>
  </div>
  …
</body>
```

2. Our navigation bar will need a CSS of its own. Let's add some new style rules to our `style` tag:

```
<style>
  …
  #navigation-bar {
    position: fixed;
    left: 0;
    bottom: 2px;
    width: 100%;
    height: 50px;
    z-index: 20;
    background-color: #fff;
    border-top: 6px solid #df0d32;
    font-family: Helvetica, Arial,
sans-serif;
  }
  #navigation-bar nav {
    padding: 15px 0;
    text-align: center;
  }
  #navigation-bar a {
    display: inline-block;
    padding: 0 10px;
    border-right: 2px solid #ccc;
    text-decoration: none;
    color: #df0d32;
  }
  #navigation-bar a:last-child {
    border-right: none;
  }
  #navigation-bar a:hover {
    text-decoration: underline;
  }
</style>
```

3. If you reload the presentation now, you will see our brand new navigation bar at the bottom, and if you click on the links, you will be taken directly to the referenced slides or actions (first, next, previous, last).



How it works...

`reveal.js` JavaScript API allows us to control our presentation behavior using JavaScript code. In the preceding example we have used almost all methods available. Let's take a closer look at them:

Option	Description
`Reveal.slide(indexh, indexv, indexf)`	Slides to the specified index (the `indexh` parameter). The first slide will have 0 as index, the second has 1, and so on. You can also specify the vertical index (the `indexv` parameter) in case of nested slides and the fragment index (the `indexf` parameter) inside the slide.
`Reveal.left()`	Goes to the previous slide in the horizontal line of the presentation.
`Reveal.right()`	Goes to the next slide in the horizontal line of the presentation.

Option	Description
`Reveal.up()`	Changes to the following nested slide.
`Reveal.down()`	Returns to the previous nested slide.
`Reveal.prev()`	Goes to the previous slide, horizontal or vertical.
`Reveal.next()`	Goes to the next slide, horizontal or vertical.
`Reveal.prevFragment()`	Switches to the previous fragment inside the current slide.
`Reveal.nextFragment()`	Switches to the next fragment inside the current slide.
`Reveal.toggleOverview()`	Toggles overview mode.
`Reveal.getPreviousSlide()`	Returns the DOM element of the previous slide.
`Reveal.getCurrentSlide()`	Returns the DOM element of the next slide.
`Reveal.getIndices()`	Returns an object with the current indices, for example {h: 2, v: 1}.

The preceding methods are available to use in any JavaScript code after the `reveal.js` object has been initialized. You can either use it directly on elements' attributes (such as `onclick`) or call it directly from JavaScript functions and listeners.

Plugins and extensions (Medium)

We have already seen the use of plugins and extensions in both the *Adding speaker notes (Simple)* and *Adding Markdown content (Simple)* recipes. As stated before, `reveal.js` doesn't rely on any additional library but allows us to implement our own plugins or use some default extensions (like the Markdown and speaker notes). In this recipe we will activate two new extensions in our presentation – zoom and syntax highlighting.

How to do it...

To load plugins and extensions perform the following steps:

1. We will start by adding the two new dependencies to our `reveal.js` object initialization:

```
Reveal.initialize({
  controls: false,
  keyboard: true,
  center: true,
  transition: 'cube',
  mouseWheel: true,
  autoSlide: 6000,
  rollingLinks: false,
  dependencies: [
    {
      src: 'plugin/notes/notes.js',
      async: true

    },
    {
      src: 'plugin/markdown/showdown.js',
      async: true
    },
    {
      src: 'plugin/markdown/markdown.js',
      async: true
    },
    {
      src: 'plugin/zoom-js/zoom.js',
      async: true
    },
    {
      src: 'plugin/highlight/highlight.js',
      async: true
    }
  ]
});
```

2. With the zoom extension enabled we can use the *Alt* + click combination to zoom in and out on our presentation slides. We don't need any additional code to achieve that.
3. As for our syntax highlight extension now, whenever we add a code or a `pre` HTML tag, they will automatically be formatted and highlighted. Let's try it with our **reveal.js** slide, adding a `pre` block with `reveal.js` initialization syntax:

```
<section>
  <h1>reveal.js</h1>…
  <pre>
  Reveal.initialize({
  controls: false,
  keyboard: true
  });
  </pre>
</section>
```

4. Now, open the presentation and navigate to the **reveal.js** slide to check how the code is displayed.

How it works...

Besides the extension we have seen so far, `reveal.js` has two other plugins – one for `classList` method cross-browser compatibility and one for enabling remote control via touch devices.

As for loading the plugins, the `source` parameter is the only one that is required, but there are three optional parameters:

- `src`: This decides the relative path to the script
- `async` (optional): If this is set to `true`, the script will be loaded only after `reaveal.js` has been loaded
- `callback` (optional): `callback` is a function that will be executed when the dependency is loaded
- `condition` (optional): This parameter is a condition that must be matched for the plugin to be loaded (usually a function that returns `true` or `false`)

Following the preceding pattern you can add your own extensions to `reveal.js`.

Exporting our presentation as PDF (Simple)

With `reveal.js` it is possible to export our presentation as a PDF file using a custom stylesheet file.

How to do it...

To export our presentation as a PDF perform the following steps:

1. The first step is to add the link to `reveal.js` PDF stylesheet. Insert the link to `css/print/pdf.css` inside our `head` element:

```
<head>
  …
  <link rel="stylesheet"
href="css/print/pdf.css" media="print">
</head>
```

2. Remember we personalized our presentation with an elapsed time clock and a navigation bar? Since they won't be of any use on paper, let's hide them by editing the `css/print/pdf.css` file. Add the following rule:

```
#navigation-bar, .timer {
  display: none;
}
```

3. Now, when we export our presentation as a PDF, it won't display the navigation bar or the elapsed time clock element.

How it works...

Using the browser print dialogue you can generate a PDF file of our presentation. In the Google Chrome print dialogue you can choose the destination to be a PDF file.

Other browsers will have an option to save as PDF instead of printing. It varies according to browsers and operating systems.

In order to generate the PDF file we have used a custom print stylesheet that is applied only when printing, and ignored when displaying on a screen. Any rule we use there will be applied only on a printed file.

There is more...

We can also directly print our presentation instead of saving it as a PDF file.

Printing

`reveal.js` also comes with a default CSS for printing, available at `css/print/paper.css`. If you want to print directly to the printer, `paper.css` is the recommended and optimized file to use. The link element would be as follows:

```
<head>
  …
  <link rel="stylesheet"
href="css/print/paper.css" media="print">
</head>
```

URL parameter

The `reveal.js` default index file implements an interesting approach for enabling the PDF stylesheet. If you add an extra parameter (`?pdf`) to the presentation URL, the PDF css file will be loaded, otherwise it will be the default print stylesheet.

Just attach the following code to the `head` section of your presentation:

```
<script>
  document.write('<link rel="stylesheet"
href="css/print/' +
( window.location.search.match( /print-
pdf/gi ) ? 'pdf' : 'paper') + '.css"
type="text/css" media="print">');
</script>
```

Now, the export PDF feature will only be available if you go to http://www.daviferreira.com/html5-presentations-how-to/index.html?pdf.

Server-side speaker notes (Advanced)

In the *Adding speaker notes (Simple)* recipe we have seen how to set up notes on the client side, using a pop-up window. `reveal.js` also comes with a `node.js` plugin to show the notes on the server side.

Getting ready

In order to use the speaker notes server plugin you need to have `node.js` installed and configured on your machine. `node.js` is an event-driven, server-side platform built on top of JavaScript.

To install it simply visit http://nodejs.org and click on the **INSTALL** button. After the installation process you can verify if `node.js` was configured correctly by typing in the following command on a terminal:

```
node –version
```

Its output should display the `node.js` version, as follows:

```
$ node --version
v0.6.11
```

How to do it...

To add server-side speaker notes perform the following steps:

1. With `node.js` installed and configured, it is now time to update `reveal.js` dependencies with `socket.io` and the `notes-server` plugin:

```
Reveal.initialize({
  controls: false,
  keyboard: true,
  center: true,
  transition: 'cube',
  mouseWheel: true,
  autoSlide: 6000,
  rollingLinks: false,
  dependencies: [
    {
      src: 'plugin/notes/notes.js',
      async: true
```

```
      },
      {
        src: 'plugin/markdown/showdown.js',
        async: true
      },
      {
        src: 'plugin/markdown/markdown.js',
        async: true
      },
      {
        src: 'plugin/zoom-js/zoom.js',
        async: true
      },
      {
        src: 'plugin/highlight/highlight.js',
        async: true
      },
      {
        src: '/socket.io/socket.io.js',
        async: true
      },
      {
        src: 'plugin/notes-server/client.js',
        async: true
      }
    ]
});
```

2. Then, open up a terminal and type in the following command:

```
npm install
```

3. `npm` is the `node.js` package manager. It will install all `reveal.js` dependencies so that we can run our presentation as a server using the following command:

```
node plugin/notes-server
```

1. This is it. `node.js` will start a server and our presentation will be accessible via `http://localhost:1947`. Every time we reach a slide with speaker notes, the plugin will generate a permalink to them, as follows:

   `http://localhost:1947/notes/10169049212709069`

How it works...

Using `reveal.js` in combination with `node.js` we have not only created permalinks for our speaker notes, but also a fast and simple server to display our presentation.

If you are over a network, you can replace the localhost on the URL with your IP address, and share the notes and presentation links with anyone connected to that network.