2
Constructing a Responsive Portfolio Page with Skeleton

In our previous chapter, we discussed responsive web design and had a first look at the frameworks that make it possible for us to create a responsive website more quickly.

In this chapter, we will create a simple responsive portfolio website with Skeleton. So, if you are a creative person who wants to showcase your own work on your own website, this could be a perfect chapter to work through.

To sum it up, here is what we will focus on in this chapter:

- Digging into the Skeleton components
- Utilizing the Skeleton components
- Setting up a project with Skeleton
- Preparing the project assets
- Constructing a website with HTML5

So, let's get started.

Getting started with Skeleton

As mentioned in the previous chapter, one of the disadvantages of using a framework is the learning curve; we need to spend some time to learn how to use the framework, particularly if this is the first time using it. So, before we build our responsive portfolio website with Skeleton, it is a good idea to unpack and take a look at what is included in Skeleton.

Time for action – creating a working directory and getting Skeleton

Perform the following steps for creating a working directory and getting Skeleton:

1. First, create a folder named `portfolio`. This should be our working directory for the responsive portfolio website.
2. Under this `portfolio` folder, create two folders named `html` and `psd`.
3. Now it is time to get Skeleton. So, let's go to the Skeleton website (`www.getskeleton.com`).
4. Go to the **Download** section and download the Skeleton package. At the time of writing, the latest version of Skeleton is Version 1.2.
5. Save the downloaded file in the `html` folder.
6. This downloaded file is in the `tar.gz` format. Let's extract it to retrieve the files inside the downloaded file.
7. After extracting, you should find two new folders named `stylesheet` and `images`, and an HTML document named `index.html`. This is optional, but we can now safely remove the `.tar.gz` file.
8. Lastly, from the **Download** section on `www.getskeleton.com`, download the Skeleton PSD template, save it in the `psd` folder, and unpack it.

What just happened?

We have just created a working directory. We have also downloaded the Skeleton package as well as the PSD template, and placed it in the appropriate folder to work on this project.

What is included in Skeleton?

Compared to other frameworks that we have mentioned in this book, Skeleton is the simplest. It is not overstuffed with heavy styles or additional components, such as jQuery plugins, which we may not need for the website. Skeleton comes only with an `index.html` file, a few stylesheets containing the style rules, a few images, and a PSD template. Let's have a look at each of these.

Starter HTML document

Skeleton comes with a starter HTML template named `index.html`, so we don't have to worry about writing the basic HTML document. The author of Skeleton has added the essential elements in this template, including the parts discussed in the following sections.

The viewport meta tag

The viewport meta tag in this HTML starter template is set to `1` for both `initial-scale` and `maximum-scale`, as shown in the following code snippet:

```
<meta name="viewport" content="width=device-width, initial-scale=1,
maximum-scale=1">
```

As we mentioned in the first chapter, setting `initial-scale` to `1` will set the web page to be 100 percent of the viewport size, when we open the web page for the first time.

However, one thing that should be noted when setting `maximum-scale` to `1` is that it will prevent the zooming ability. Thus, it is suggested to ensure that the users, later on, can clearly see the content, text, or images, without zooming the web page.

HTML5 Shim

Since we will be using the HTML5 elements in our document, we need to include the HTML5 Shim JavaScript Library so that Internet Explorer 8 and its earlier versions recognize the new elements from HTML5.

HTML5 Shim, by default, has also been included in the Skeleton starter HTML document; you should find the following line inside the <head> section:

```
<!--[if lt IE 9]>
<script src="http://html5shim.googlecode.com/svn/trunk/html5.js"></
script>
<![endif]-->
```

The preceding HTML5 Shim script is wrapped within the conditional comment tag that is designated for Internet Explorer. The comment `<!—-[if lt IE 9]>` stated "if less than Internet Explorer 9", which means the script within will only apply to Internet Explorer 8 and its earlier versions where new HTML5 elements are not recognized. Other browsers will simply ignore this comment tag.

You can read a post by Paul Irish (`http://paulirish.com/2011/the-history-of-the-html5-shiv/`) for the history behind HTML5 Shim and about how it was invented and developed.

Responsive Grid

Skeleton is equipped with Responsive Grid to quickly build responsive layout. The Skeleton's grid system is 960 px wide and is made up from sixteen columns of grid that are defined in a very logical naming system.

The columns are defined with the `.columns` class coupled with the respective column numbers `.one`, `.two`, `.three`, `.four`, and so on, to define the column width. These classes can be found in the `skeleton.css` file. The following code snippet shows the definitions of the column numbers and column width in the stylesheet:

```
.container .one.column,
.container .one.columns { width: 40px; }
.container .two.columns { width: 100px; }
.container .three.columns { width: 160px; }
.container .four.columns { width: 220px; }
.container .five.columns { width: 280px; }
.container .six.columns { width: 340px; }
.container .seven.columns { width: 400px; }
.container .eight.columns { width: 460px; }
.container .nine.columns { width: 520px; }
.container .ten.columns { width: 580px; }
.container .eleven.columns { width: 640px; }
.container .twelve.columns { width: 700px; }
.container .thirteen.columns { width: 760px; }
.container .fourteen.columns { width: 820px; }
.container .fifteen.columns { width: 880px; }
.container .sixteen.columns { width: 940px; }
```

If you are not familiar with this practice or you don't know how it works, take a look at the following example.

In this example, we have three `div` elements; one of those is for the container. Inside this container, we will have a `div` element to contain a main area and an `aside` element to contain the sidebar area. The following code snippet shows how our markup looks in the code editor:

```
<div>
  <div>
    <h3>Main Content</h3>
    <p>Lorem ipsum dolor sit amet, consectetur adipiscing elit.
    Aenean consequat porttitor elementum. Mauris pulvinar semper
    lobortis. […]</p>
  </div>
  <aside>
    <h3>Sidebar</h3>
    <p>Lorem ipsum dolor sit amet, consectetur adipiscing elit.
    Aenean consequat porttitor elementum. Mauris pulvinar semper
    lobortis.[…]</p>
  </aside>
</div>
```

Since all the styling rules for the columns are predefined, we simply need to add the appropriate classes into these elements, as follows:

```
<div class="container">
  <div class="ten columns">
    <h3>Main Content</h3>
    <p>Lorem ipsum dolor sit amet, consectetur adipiscing elit.
    Aenean consequat porttitor elementum. Mauris pulvinar semper
    lobortis. […] </p>
  </div>
  <aside class="six columns">
    <h3>Sidebar</h3>
    <p>Lorem ipsum dolor sit amet, consectetur adipiscing elit.
    Aenean consequat porttitor elementum. Mauris pulvinar semper
    lobortis. […]</p>
  </aside>
</div>
```

Then, if we view the document in the browser, we will see something as shown in the following screenshot:

Yes, it's that simple. But just to remember, in Skeleton, the columns should be nested inside an element with the `.container` class, otherwise the column styles will not be applied.

Clearing styles

The column's elements are defined by using the CSS float property definition, which causes the column's parent element to collapse. To solve it, Skeleton provides special classes; we can use either the `.row` class or the `.clearfix` class to clear things around the columns. The following code snippet shows the clearing styles' definitions, which can be found in `skeleton.css`:

```
.container:after { content: "\0020"; display: block; height: 0; clear:
both; visibility: hidden; }
.clearfix:before, .clearfix:after,
.row:before,
.row:after { content: '\0020'; display: block; overflow: hidden;
visibility: hidden; width: 0; height: 0; }
.row:after,
.clearfix:after { clear: both; }
.row, .clearfix { zoom: 1; }
.clear { clear: both; display: block; overflow: hidden; visibility:
hidden; width: 0; height: 0; }
```

On the Smashing Magazine website, Louis Lazaris has thoroughly discussed the CSS float property and how it affects the elements around it in the post available at `http://coding.smashingmagazine.com/2009/10/19/the-mystery-of-css-float-property/`.

Media queries

Skeleton has provided CSS3 media queries to apply specific style rules for standard viewport size and also making the grid responsive. For example, the following media query will specify the styles for 959 px viewport size and less:

```
@media only screen and (max-width: 959px) {
  …
}
```

Remember that Skeleton is a 960 grid-based framework, which means the maximum width of the web page would only be 960 px. So when the viewport is 959 px wide or less, in other words, smaller than the base size, the styles under this media query will be applied. The same idea also applies to the other defined media queries for example:

```
/* Tablet Portrait size to standard 960 (devices and browsers) */
@media only screen and (min-width: 768px) and (max-width: 959px) { }
/* All Mobile Sizes (devices and browser) */
@media only screen and (max-width: 767px) { }
/* Mobile Landscape Size to Tablet Portrait (devices and browsers) */
@media only screen and (min-width: 480px) and (max-width: 767px) { }
/* Mobile Portrait Size to Mobile Landscape Size (devices and
browsers) */
@media only screen and (max-width: 479px) { }
```

These media query definitions can be found in the `skeleton.css` and `layout.css` stylesheet.

Referring to our previous example, the web page is already responsive, as the column classes and the styles are predefined under the media queries in the `skeleton.css` stylesheet.

Thus, when we view it in a much smaller viewport with—in this example it as 320 px—we will get the result as shown in the following screenshot:

Typography styles

Typography styles have a key role in making a website readable. While the browsers have default styles for typography, Skeleton provides an improvement in this area for some elements, including headings, paragraphs, and pull-quotes. In Skeleton, these typography styles are available in the `base.css` stylesheet.

Button styles

Skeleton provides basic styles for buttons, which are applied by adding the `.button` class to some elements, such as the <button> or <a> elements, as shown in the following code snippet:

```
<button class="button" type="submit">Button Element</button>
<a href="#" class="button">Anchor Tag</a>
```

The result of the preceding code snippet is rendered, as shown in the following screenshot:

Form styles

Styling form elements can be complicated. But, Skeleton simplifies the process with its default styles. We simply need to structure the markup properly, without adding any special classes, as shown in the following code snippet:

```
<form>
  <label for="name">Name</label>
  <input type="text" id="name">
  <label for="message">Message</label>
  <textarea id="message"></textarea>
  <button type="submit">Submit Form</button>
</form>
```

In the browsers, we will get the result as shown in the following screenshot:

Apple icon devices

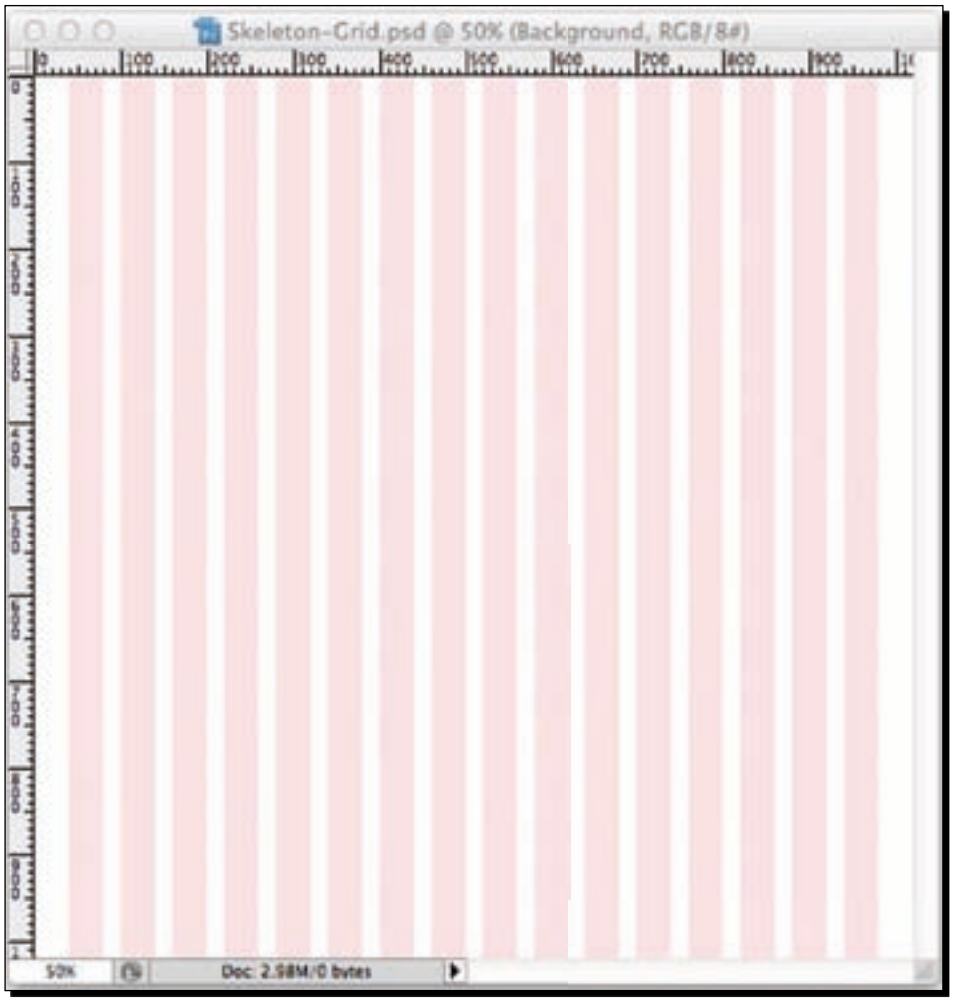
Skeleton comes with favicon and iOS icons, which we can easily replace with our own custom icons, if needed. The following screenshot shows these images in different sizes for different devices and resolutions:

The first one, which is the smallest, is the icon for iPhone. The second one, which is bigger than the first one, is to serve the iPad, while the biggest one will be displayed for Apple devices with higher resolution Retina Display.

You can read the documentation available at Apple Dev Center (`http://developer.apple.com/library/safari/#documentation/AppleApplications/Reference/SafariWebContent/ConfiguringWebApplications/ConfiguringWebApplications.html#//apple_ref/doc/uid/TP40002051-CH3-SW3`) for more details on the use of these icons.

Photoshop template

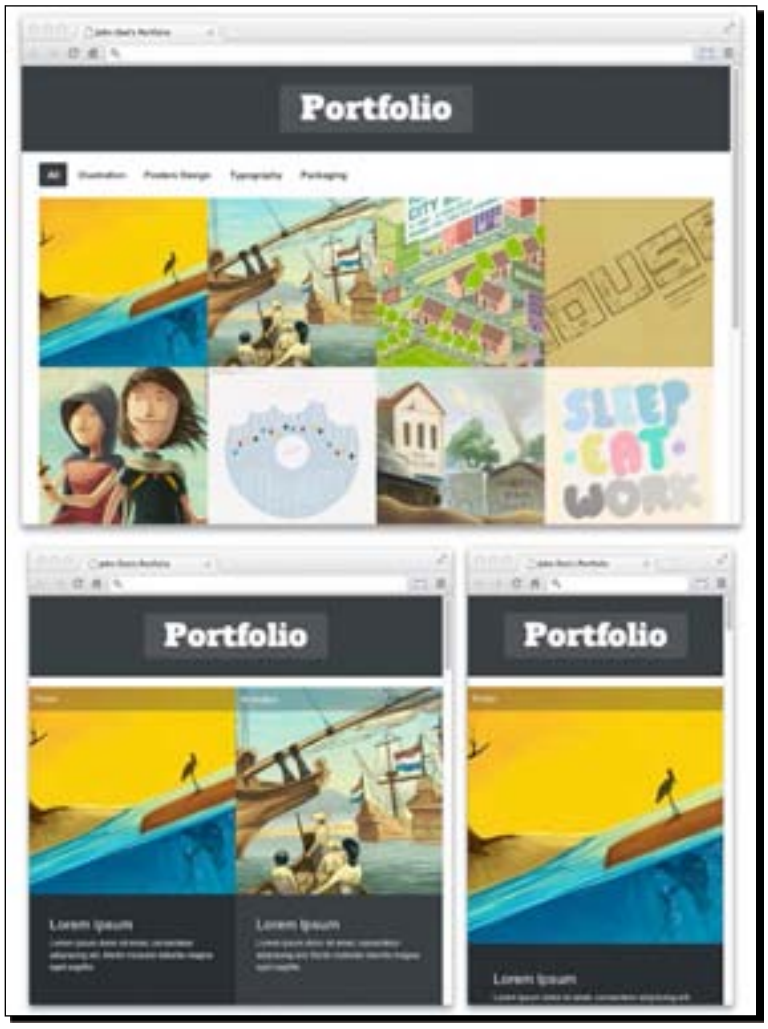
We have downloaded a PSD template earlier in this chapter. This template contains only one extra layer. **Layer** is a semi-transparent overlay showing the 16 columns of the grid, as shown in the following screenshot:



This grid is useful as a visual helper to design the website. So later on, when we translate the design into a web document, we will know the appropriate grid number for the translated elements.

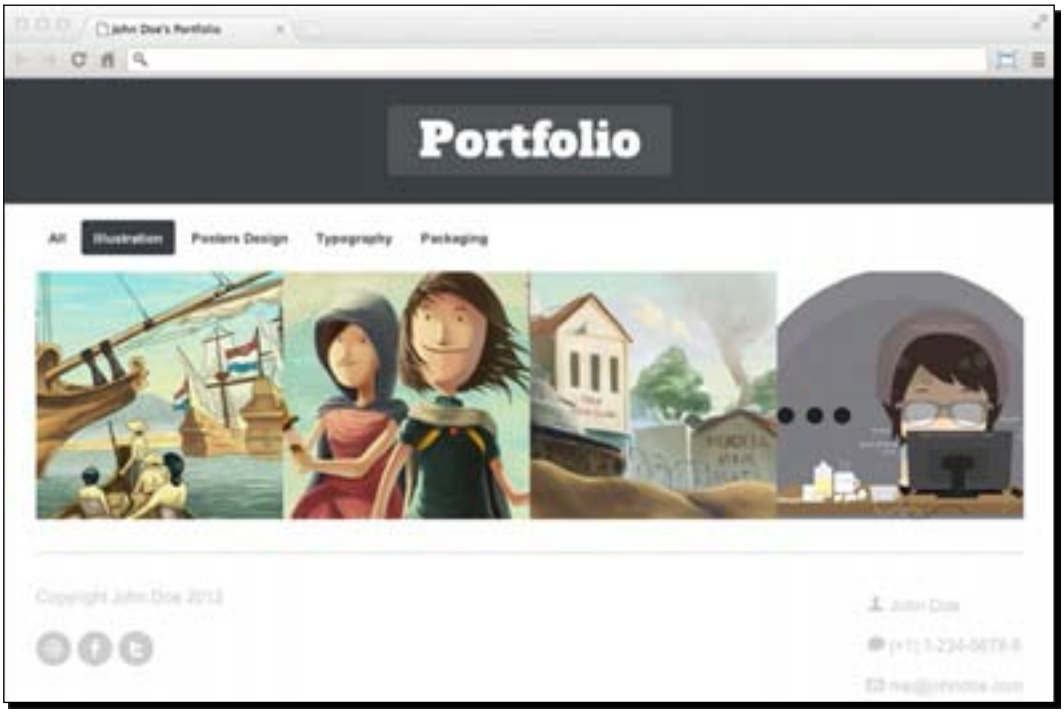
How will the website look?

At this point, you may wonder how our first website will look. It will be really simple with only three sections: the header, the main content area that displays the portfolio, and the footer. The following screenshot shows three different views of the website with respect to the different viewport sizes:



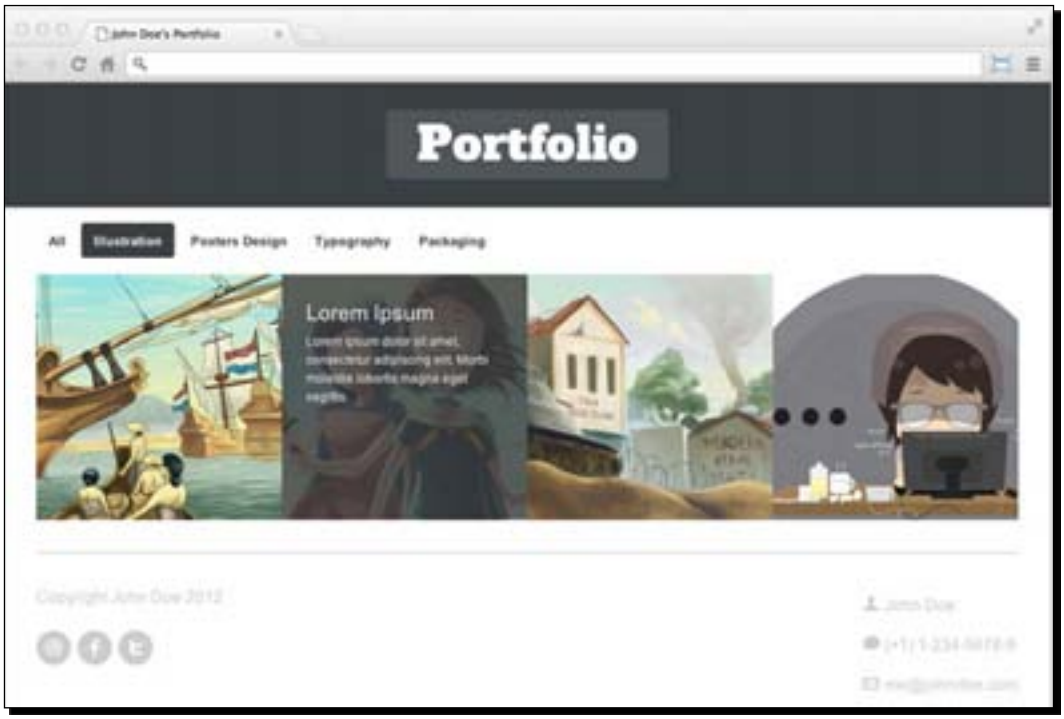
Website navigation

Our website's navigation will be somewhat unusual; rather than being used to move between pages, it will be used to sort the portfolio. We have several categories of portfolios: they are **Illustration**, **Poster Design**, **Typography**, and **Packaging**. The following screenshot shows the result of selecting the **Illustration** category:



Thumbnail hover effect

We will also add a fancy effect to make our website more attractive. When we hover over one of the portfolio thumbnails, the description of that portfolio will be revealed. The following screenshot shows this effect:



Setting up the Skeleton document

Now, it is time to set up the Skeleton document. It is important to note that when we are working on a framework, it is best not to alter the codes in the core files, which are the original files from the downloaded package. If we change these files, it may make our website less maintainable, and our changes may be overwritten if the framework is upgraded later. Thus we need to add a CSS file for our own.

Time for action – adding an extra CSS file

Perform the following steps for adding an extra CSS file:

1. Go to our working directory, `portfolio`.
2. Then go to the `stylesheets` folder and create a new file.

3. Rename this new file as `styles.css`.
4. Open the `index.html` file.
5. Add the following lines inside the <head> tag, right after the default Skeleton styles `base.css`, `skeleton.css`, and `layout.css`:

```
<link rel="stylesheet" href="stylesheets/base.css">
<link rel="stylesheet" href="stylesheets/skeleton.css">
<link rel="stylesheet" href="stylesheets/layout.css">
<link rel="stylesheet" href="stylesheets/styles.css">
```

What just happened?

We have just created a new stylesheet named `styles.css`, which we will be using for our own styles apart from the default Skeleton styles. Then, we called this stylesheet in our HTML document so that the styles within this stylesheet show their effect.

The reason we added this stylesheet after the other stylesheet links is because we want our styles to take place over the other style definitions.

You can read about CSS Specificity at `http://coding.smashingmagazine.com/2007/07/27/css-specificity-things-you-should-know/`.

Adding custom fonts

Earlier we were limited to fonts that were installed on a given user's machine, which meant that the only practical fonts were those with a broad installed base, such as Arial, Times, and Georgia. Today, we are able to embed font families for websites apart from the ones in the user's machine.

If you look at our design's header section, you can see that the main Porfolio heading uses an uncommon font—in this case, Alfa Slab One.

There are several options for embedding fonts. For this website we will use Google Web Fonts. In Google Web Fonts, we can find various font types that are allowed to be embedded on websites for free.

Time for action – embedding Google Web Fonts

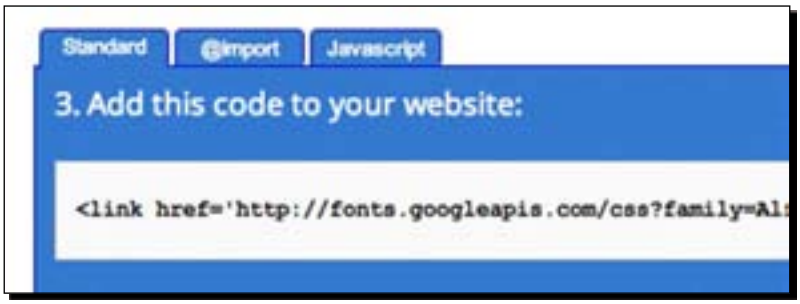
Perform the following steps for embedding Google Web Fonts:

1. First, go to the Google Web Font website (`http://www.google.com/webfonts`).
2. Find a **Search** box and type `Alfa Slab One`; this is the name of the font that we are going to use for the website logo.

3. Click on the **Quick-use** link, as shown in the following screenshot:

This will direct you to a page that contains some additional information about this font, including how to embed it on a web page.

4. There are three ways to embed a Google font: using the standard way, using the `@import` rule, or using JavaScript.



For this website, we will use the standard way. So, let's copy the following line:

```
<link href='http://fonts.googleapis.com/css?family=Alfa+Slab+One'
rel='stylesheet' type='text/css'>
```

5. Open `index.html` and paste the preceding line inside the <head> section directly above the links to other stylesheets, as follows:

```
<link href='http://fonts.googleapis.com/css?family=Alfa+Slab+One'
rel='stylesheet' type='text/css'>
<link rel="stylesheet" href="stylesheets/base.css">
<link rel="stylesheet" href="stylesheets/skeleton.css">
<link rel="stylesheet" href="stylesheets/layout.css">
<link rel="stylesheet" href="stylesheets/styles.css">
```

What just happened?

We have just embedded a new font family in our HTML document from Google Web Fonts.

Alternatively, you can use the `@font-face` rule to embed the font. Font Squirrel provides a handy tool to generate the `@font-face` rule (`http://www.fontsquirrel.com/fontface/generator`). Before embedding the fonts, be sure you agree to the End-users License Agreement of the fonts.

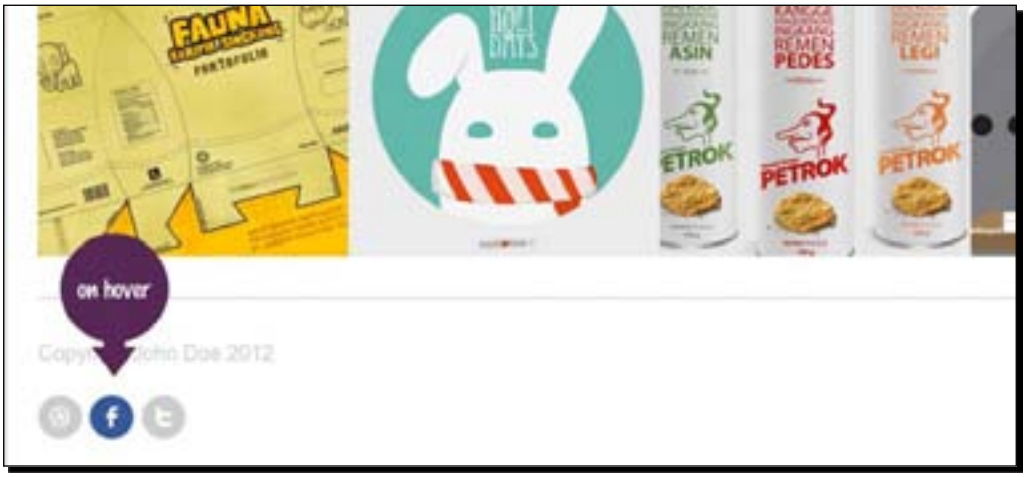
Preparing the images

Since we will be working on a portfolio website, we obviously need some portfolio images to display. I would like to thank two of my artist friends, Ferina Berliani (`http://nantokaa.tumblr.com/`) and Arif Bahari (`http://www.ariefbahari.com`) for letting me use their artwork, and the following images show some of their works that we will be using in this book.

You can use your own images as long as they are sized to a 480 px by 480 px square; you can either use Photoshop or any other image editor of your choice to do so. Then put your images inside the `images` folder under the working directory and name them using this convention: `image-1.jpg`, `image-2.jpg`, `image-3.jpg`, and so on. We have a total of 12 image thumbnails:

Social media icons

In addition, we will place three social media icons in our footer area: one each for Facebook, Twitter, and Dribbble, as shown in the following screenshot:



In the default state, the icons are displayed in grey and then when we hover over these icons, the platform's main brand color will be displayed, such as Facebook's blue and Dribbble's pink. These icons have been provided along with this book.

However, you can substitute with any social media icons that are available on the Internet for free. Just make sure that it is also available in 48 px by 48 px size. These social icons usually come separately. Thus, we will need to concatenate them into one sprite file.

Time for action – sprite images

In the following steps, we will turn these icons into sprite images with a free CSS Sprite Generator Tool (`http://spritegen.website-performance.org/`):

1. Given the icons proper names, such as `twitter.png` and `twitter-hover.png`, as shown in the following screenshot:

   This naming convention also applies to other icons. You don't have to limit yourself to our example; you can provide more than three icons. After all the images are prepared, add these icons to a ZIP file.

2. Go to the CSS Generator Tool website (`http://spritegen.website-performance.org/`).

3. Upload the ZIP file that we created in Step 2.

4. Under the **Sprite Output Options** section, enter `10` in the **Horizontal Offset** and **Vertical Offset** fields to set them to 10 px:

5. Then, click on the **Create Sprite Image & CSS** button, as shown in the following screenshot:

   This will generate the sprite image as well as the CSS rule to display it.

6. Download the image and save it under the `images` folder in our working directory.

7. Copy the CSS snippet into our `style.css` file. It should resemble the following code snippet:

```
.sprite-dribbble-hover{ background-position: 0 0; width: 48px;
height: 48px; }
.sprite-dribbble{ background-position: 0 -58px; width: 48px;
height: 48px; }
.sprite-facebook-hover{ background-position: 0 -116px; width:
48px; height: 48px; }
.sprite-facebook{ background-position: 0 -174px; width: 48px;
height: 48px; }
.sprite-twitter-hover{ background-position: 0 -232px; width: 48px;
height: 48px; }
.sprite-twitter{ background-position: 0 -290px; width: 48px;
height: 48px; }
```

What just happened?

We concatenated the social media icons into one file. We will display these icons on our website using the CSS rule that we have generated. This practice is known as **CSS Sprite**.

Alternatively, you can also follow a screencast by Chris Coyier available at CSS Tricks to create a sprite image in Photoshop (`http://css-tricks.com/video-screencasts/43-how-to-use-css-sprites/`), and as an addition, you can also follow a screencast by Lynda on how to create a sprite grid to help you in positioning sprite images (`http://www.youtube.com/watch?v=Gq7XCMofxcQ`).

Or else, if you are not familiar with CSS Sprites, Dave Shea has discussed this method thoroughly at A List Apart (`http://www.alistapart.com/articles/sprites`).

Contact icons

Our footer will include contact information, such as name, phone number, and e-mail address, each with its own icon as illustrated in the following screenshot:

These icons have been provided along with the code files available with this book, but you can use other icons in 24 px by 24 px size, which are available on the Internet. Similarly, if the icons come separately, you need to concatenate them in one file and generate the CSS rules, as we have demonstrated in the preceding section.

HTML5 elements

HTML5 introduces many new elements and we will use some of them for this website, such as `<header>`, `<section>`, `<figure>`, `<figcaption>`, and `<footer>`.

Element	Discussion
`<header>`	This is used for defining the head of a section. The `<header>` element can be used for the website's header and also the head of other sections where it is reasonable to add it, such as the article's header.
`<footer>`	The `<footer>` element defines the end or the lowest part of a section. Like the `<header>` element, `<footer>` can also be used for the website's footer or the footer part of other sections.
`<section>`	`<section>` can somehow be confusing. But according to the specifications (`http://www.w3.org/html/wg/drafts/html/master/sections.html#the-section-element`), the `<section>` element represents a generic section of a document or application.
`<figure>`	The `<figure>` element is used to represent the document figure, such as an illustration or an image. It can be used with `<figcaption>` to add the caption, if needed.
`<figcaption>`	As mentioned, `<figcaption>` represents the caption of the document's figure. Thus, it should be used along with the `<figure>` element.

Now, let's add these elements to our document.

HTML5 custom data attributes

There are times when developers need to retrieve data within specific elements for further data processing. In the past, some developers used to rely on the `rel` or `class` attributes to store that data, but that way leads to breaking the validity of the document's structure.

To accommodate that situation, HTML5 introduced a new attribute called custom data attribute. We can use this attribute to embed custom data within an HTML element. This attribute is specified with `data-` and followed by the attribute name. For example, an online gaming website can list the top players and use data attributes to store their scores.

```
<ul id="top-players">
  <li class="player-name" data-score="98.9">John Doe</li>
  <li class="player-name" data-score="80.5">Someone Else</li>
  <li class="player-name" data-score="70.2">Friend Someone Else</li>
</ul>
```

It is worth noting that the custom data attribute should only be used when we do not find any applicable or more appropriate attribute for that data. Storing the scores in the `class` attribute as `class="98.9"` is definitely not an applicable approach.

For further reference on data attributes, you can head over to the following pages:

- A documentation on custom data attributes available at `http://www.w3.org/html/wg/drafts/html/master/elements.html`
- *All You Need to Know About the HTML5 Data Attribute* (`http://webdesign.tutsplus.com/tutorials/htmlcss-tutorials/all-you-need-to-know-about-the-html5-data-attribute/`)
- An article on HTML5 data attributes by John Resig (`http://ejohn.org/blog/html-5-data-attributes/`)

Time for action – structuring the HTML document

Perform the following steps for structuring the HTML document:

1. Open the `index.html` file in your working directory.
2. Remove anything present between the <body> and </body> tags and replace it with the following code snippet to establish the header section. Our website's header is wrapped within the HTML5 <header> element and it contains the site logo that is wrapped within a element with a class of `logo`.

```
<header class="header">
  <div class="logo">
    <h1>Portfolio</h1>
  </div>
</header>
```

3. Then, put the following <form> element with a class of container and clearfix next to the <header> element that we just added. We use this to contain the website content.

```
<form class="container clearfix"> </form>
```

The <form> element is essentially an element like a element. We use <form> instead of as we will use the HTML form elements <input> and <label> to construct the website navigation.

You can head over to the article (http://reference.sitepoint.com/html/elements-form) from SitePoint to see the complete list of elements that are part of an HTML form.

4. Inside the <form> element for a container, we add the HTML structure for the website navigation. As we mentioned earlier, our website navigation is uncommon. We will use the radio button as an input type and each <input> element is assigned with a unique ID followed by their respective <label> element, as shown in the following code snippet:

```
<input class="nav-menu" id="all" type="radio" name="filter"
checked="checked"/>
<label for="all">All</label>

<input class="nav-menu" id="illustrations" type="radio"
name="filter"/>
<label for="illustrations">Illustration</label>

<input class="nav-menu" id="posters" type="radio" name="filter"/>
<label class="nav-menu" for="posters">Posters Design</label>

<input class="nav-menu" id="typography" type="radio"
name="filter"/>
<label for="typography">Typography</label>

<input class="nav-menu" id="packaging" type="radio"
name="filter"/>
<label for="packaging">Packaging</label>
```

5. Then add an HTML5 <section> element with a class of portfolio next to those <input> and <label> elements that we just added.

```
<section class="portfolio"></section>
```

This <section> element will be used to contain the portfolio, which includes the image thumbnails and the captions.

6. Inside this `<section>` element, we add the portfolio image thumbnails. Each image thumbnail is wrapped within the HTML5 `<figure>` element.

We have 12 image thumbnails and we will divide them into four columns. Skeleton has 16 columns of grid and 16 divided by four results in four columns. So, each `<figure>` element is assigned with classes of `four` and `columns` with two additional classes of `all` and its category name.

```
<figure class="four columns all poster">
  <img src="images/image-1.jpg" alt=
  "This is 1st portfolio thumbnail.">
</figure>
```

The classes of `four` and `columns` are assigned to apply the column styles from Skeleton, while the class of `all` will be used to select the `<figure>` element when we need to apply CSS rules to all `<figure>` elements. We will use the category name class to group the figures and also apply styles to the figures that share the same category.

We will also provide some text that describes the image with an `alt` attribute. This `alt` attribute is useful for the browser to show alternative information for the users, in case the image fails to load.

7. The image thumbnails are grouped into a category. We assign the category name with the `title` attribute in the `<figure>` element, as follows:

```
<figure class="four columns" title="poster">
  <img src="images/image-1.jpg">
</figure>
```

8. The image thumbnail will have a caption containing the portfolio's description. We will use the HTML5 `<figcaption>` element to contain the description text and place it inside the `<figure>` element, as follows:

```
<figure class="four columns all poster">
  <img src="images/image-1.jpg" alt=
  "This is 1st portfolio thumbnail.">
  <figcaption>
    <h4>Lorem Ipsum</h4>
    <p>Lorem ipsum dolor sit amet, consectetur adipiscing
    elit. Morbi molestie lobortis magna eget sagittis.</p>
  </figcaption>
</figure>
```

9. Then, we will add an HTML5 data attribute to `<figure>` to store the category name where the `<figure>` element is assigned and we simply name this attribute `data-category`.

```
<figure class="four columns all poster">
  <img src="images/image-1.jpg" alt=
  "This is 1st portfolio thumbnail.">
  <figcaption>
    <h4>Lorem Ipsum</h4>
    <p>Lorem ipsum dolor sit amet, consectetur adipiscing
    elit. Morbi molestie lobortis magna eget sagittis.</p>
  </figcaption>
</figure>
```

Now, let's add the rest of the image thumbnails, as follows.

```
<figure class="four columns all illustration" data-
category="illustration">
  <img src="images/image-2.jpg" alt=
  "This is 2nd portfolio thumbnail.">
  <figcaption>
    <h4>Lorem Ipsum</h4>
    <p>Lorem ipsum dolor sit amet, consectetur adipiscing
    elit. Morbi molestie lobortis magna eget sagittis.</p>
  </figcaption>
</figure>
<figure class="four columns all poster" data-category="poster">
  <img src="images/image-3.jpg" alt=
  "This is 3rd portfolio thumbnail.">
  <figcaption>
    <h4>Lorem Ipsum</h4>
    <p>Lorem ipsum dolor sit amet, consectetur adipiscing
    elit. Morbi molestie lobortis magna eget sagittis.</p>
  </figcaption>
</figure>
<figure class="four columns all typography" data-
category="typography">
  <img src="images/image-4.jpg" alt=
  "This is 4th portfolio thumbnail.">
  <figcaption>
    <h4>Lorem Ipsum</h4>
    <p>Lorem ipsum dolor sit amet, consectetur adipiscing
    elit. Morbi molestie lobortis magna eget sagittis.</p>
  </figcaption>
</figure>
<figure class="four columns all illustration" data-
category="illustration">
  <img src="images/image-5.jpg" alt=
  "This is 5th portfolio thumbnail.">
  <figcaption>
```

```
      <h4>Lorem Ipsum</h4>
      <p>Lorem ipsum dolor sit amet, consectetur adipiscing
      elit. Morbi molestie lobortis magna eget sagittis.</p>
    </figcaption>
  </figure>
  <figure class="four columns all poster" data-category="poster">
    <img src="images/image-6.jpg" alt=
    "This is 6th portfolio thumbnail.">
    <figcaption>
      <h4>Lorem Ipsum</h4>
      <p>Lorem ipsum dolor sit amet, consectetur adipiscing
      elit. Morbi molestie lobortis magna eget sagittis.</p>
    </figcaption>
  </figure>
  <figure class="four columns all illustration" data-
  category="illustration">
    <img src="images/image-7.jpg" alt=
    "This is 7th portfolio thumbnail.">
    <figcaption>
      <h4>Lorem Ipsum</h4>
      <p>Lorem ipsum dolor sit amet, consectetur adipiscing
      elit. Morbi molestie lobortis magna eget sagittis.</p>
    </figcaption>
  </figure>
  <figure class="four columns all typography " data-
  category="typography">
    <img src="images/image-8.jpg" alt=
    "This is 8th portfolio thumbnail.">
      <figcaption>
      <h4>Lorem Ipsum</h4>
      <p>Lorem ipsum dolor sit amet, consectetur adipiscing
      elit. Morbi molestie lobortis magna eget sagittis.</p>
    </figcaption>
  </figure>
  <figure class="four columns all package" data-category="package">
    <img src="images/image-9.jpg" alt=
    "This is 8th portfolio thumbnail.">
    <figcaption>
      <h4>Lorem Ipsum</h4>
      <p>Lorem ipsum dolor sit amet, consectetur adipiscing
      elit. Morbi molestie lobortis magna eget sagittis.</p>
    </figcaption>
  </figure>
  <figure class="four columns all poster" data-category="poster">
    <img src="images/image-10.jpg" alt=
    "This is 9th portfolio thumbnail.">
    <figcaption>
      <h4>Lorem Ipsum</h4>
      <p>Lorem ipsum dolor sit amet, consectetur adipiscing
      elit. Morbi molestie lobortis magna eget sagittis.</p>
```

```
  </figcaption>
</figure>
<figure class="four columns all package " data-category="package">
  <img src="images/image-11.jpg" alt=
  "This is 10th portfolio thumbnail.">
  <figcaption>
    <h4>Lorem Ipsum</h4>
    <p>Lorem ipsum dolor sit amet, consectetur adipiscing
    elit. Morbi molestie lobortis magna eget sagittis.</p>
  </figcaption>
</figure>
<figure class="four columns all illustration "
title="illustration">
  <img src="images/image-12.jpg" alt=
  "This is 10th portfolio thumbnail.">
  <figcaption>
    <h4>Lorem Ipsum</h4>
    <p>Lorem ipsum dolor sit amet, consectetur adipiscing
    elit. Morbi molestie lobortis magna eget sagittis.</p>
  </figcaption>
</figure>
```

10. Lastly, for the website footer area, add the following HTML5 `<footer>` element with the `container clearfix` class next to the `<div>` element defined for the container, which we just added in Step 3:

```
<footer class="container clearfix">
  <div class="contact">
    <ul>
      <li class="contact-name">John Doe</li>
      <li class="contact-phone">(+1) 1-234-5678-9</li>
      <li class="contact-email">me@johndoe.com</li>
    </ul>
  </div>
  <div class="social">
    <p class="copyright">Copyright John Doe 2012</p>
    <ul>
      <li class="social-dribbble">
      <a href="#">Dribbble</a></li>
      <li class="social-facebook">
      <a href="#">Facebook</a></li>
      <li class="social-twitter">
      <a href="#">Twitter</a></li>
    </ul>
  </div>
</footer>
```

What just happened?

We have just added the structure for the website to `index.html` using the HTML5 elements and establishing the website header, the image portfolio, and the website footer.

Summary

In this chapter, we started our first project and accomplished the following things:

- Unpacked the Skeleton package and walked through some of the components
- Learned how to use Skeleton responsive grid
- Set up a working directory as well as the project documents
- Prepared the project assets
- Structured the project document with HTML5 elements

Now that the project has been set up, we are going to make up and tweak the website's look with CSS3 in the next chapter.