
CSS3 Solutions
Essential Techniques for CSS3 Developers
MARCO CASARIO, NATHALIE WORMSER,
DAN SALTZMAN, ANSELM BRADFORD,
JONATHAN REID, FRANCESCO IMPROTA,
AND AARON CONGLETON
friendsof
an Apress company

For your convenience Apress has placed some of the front matter material after the index. Please use the Bookmarks and Contents at a Glance links to access them.

Apress®

Contents at a Glance

Introduction

CSS3 is the latest standard for CSS, the syntax to control the style and layout of web pages.

CSS3 is completely backward-compatible, so you will not have to change your existing designs. The CSS3 specification is still under development by the World Wide Web Consortium (W3C). However, many of the new CSS3 properties have been implemented in modern browsers and are available for you to experiment with today.

The CSS3 specification is made up of several “modules”, such as the following ones:

- Selectors
- Box Model
- Backgrounds and Borders
- Text Effects
- 2D/3D Transformations
- Animations
- Multiple Column Layout
- User Interface

In this pragmatic book, we have provided a series of solutions to common problems faced by developers approaching the new features in CSS3. You will therefore find a lot of ready-to-use code that you can build on in your own web applications.

Who is this book for?

This book is aimed at designers and developers who want to start using CSS3 right now.

CSS3 Solutions is, in fact, intended for readers who want to take their knowledge further with quick-fire solutions to common problems and best practice techniques to improve their CSS3 skills. The book is full of solutions with real-world examples and code to support you as you enter the world of CSS3 development.

What you need

To follow and create the examples shown in this book, you need a simple text editor. TextMate, UltraEdit, and Notepad++ are just some examples of powerful text editors with code support.

Conventions used in this book

This book uses several of conventions that are worth noting. The following terms are used throughout this book:

- *CSS* refers to the CSS 3 language.
- *Modern browsers* are considered to be the latest versions of Firefox, Safari, Chrome, and Opera, along with Internet Explorer 7 and newer (although Internet Explorer 10 is the most "modern" in terms of support for new features).

It is assumed that all the CSS examples in this book are contained in an external style sheet. Occasionally, HTML and CSS have been placed in the same code example for brevity.

Sometimes code won't fit on a single line in a book. Where this happens, we've used an arrow to break the line.

With these formalities out of the way, you're ready to get started.

Questions and Contacts

Please direct any technical questions or comments about the book to m.casario@comtaste.com.

For more information about other CSS Books, see our website: www.apress.com.

Chapter 1

CSS Basics

Cascading Style Sheets (which we'll refer to by their acronym, CSS) were created to separate the presentational layer from the logic of an application. Their purpose has always been to provide users with a simple language to define the styling aspects of web pages and their look and feel. A CSS style declares a series of properties for content, such as the font family, size of the font, color, and so on.

The World Wide Web Consortium (W3C) has released new versions of CSS over the years that add new functions. (You can see the W3C's more recent work at `www.w3.org/Style/CSS/current-work`.) One great step forward was the introduction of CSS statements to position the contents of a page.

With these new commands, web developers could finally abandon the approach of generating web page layouts by using HTML tables. Now developers can use the following three types of positioning:

- `static`: The default positioning of the browser. This refers to the traditional HTML positioning, in which each element is positioned on the basis of the data flow of the document.
- `absolute`: Allows you to use the content anywhere on the page, completely independent from the other elements, by specifying the position of each element on a Cartesian axis represented by the height and width of the browser window.
- `relative`: Allows you to declare an element in a position that is based on the previous element.

Over the past three years, on the other hand, we have witnessed significant acceleration in terms of new specifications. In fact, the W3C, which is responsible for most web standards, intends to insert an

approach that can be divided into modules as needed. This new approach features properties, techniques, and methods that are finally tuned to the real needs of those who make web sites.

> *The World Wide Web Consortium (W3C) is the main international standards organization for the World Wide Web (abbreviated as WWW or W3).*
>
> *Founded and headed by Tim Berners-Lee, the consortium is made up of member organizations that maintain full-time staff who work together to develop standards for the World Wide Web. As of July 2011, the W3C has 317 members. W3C was created to ensure compatibility and agreement among industry members in the adoption of new standards. Prior to its creation, incompatible versions of HTML were offered by different vendors, increasing the potential for inconsistency between web pages. The W3C works to get all those vendors to agree on a set of core principles and components that will be supported by everyone.*
>
> *Source: Wikipedia http://en.wikipedia.org/wiki/World_Wide_Web_Consortium*

CSS3 Modules

With CSS3, instead of writing only one specification divided into chapters, the W3C has changed the specification to be many separate modules, each of which is dedicated to a particular aspect of the CSS language. This modular approach relieves companies that make web browsers from having to implement the specifications in their entirety. Instead, a company can opt to support one module at a time by adding new CSS modules at every new release of its browser. And this is exactly what is happening now.

Let's take a closer look at the CSS3 modules that are currently available:

- **Selectors:** This part is the most stable and is implemented best by browsers. CSS3 selectors were conceived so that they will function even with complex XML documents. They can cross the hierarchy of a document and select elements based on the relationships between them (for example, being the *n*th child of one's parent). This module is currently at the stage of Candidate Recommendation.
- **CSS Template Layout:** This module, previously known as Advance Layout, specifies new ways to place elements based on the relationship between them to guarantee maximum flexibility. It is currently at the Working Draft phase.
- **Media Queries:** The CSS3 media queries are an addition to the normal @media rules that can assign styles on the basis of new parameters, such as the size of the screen and its proportions. This module is currently at the Candidate Recommendation stage.
- **CSS Backgrounds and Borders:** This module describes new functions for backgrounds and borders, including the possibility of extending background images and rounding border angles. It is currently at the Last Call stage.
- **CSS Basic User Interface:** New methods and properties have been introduced to this module to assign styles to the user interface of a web document (such as forms). It is currently at the Candidate Recommendation stage.

- **CSS Basic Box Model:** This module accounts for differences between horizontal and vertical writing by defining the box model of the elements. It is currently at the Working Draft stage.
- **CSS Marquee:** This module proposes a CSS solution to avoid the use of the marquee owner element. It is currently at the Proposed Recommendation stage.
- **CSS Cascading and Inheritance:** This defines the ways in which styles are assigned to elements via the cascade. It is at the working Draft stage.
- **CSS Color:** This module introduces new concepts and values to describe CSS colors. It's at the Last Call stage.
- **CSS Fonts:** This module includes new properties and values for CSS fonts, such as the use of fonts that can be downloaded with the `@font-face` directive. It's at the Working Draft stage.
- **CSS Generated Content for Paged Media:** This module extends the common CSS properties for printing with the introduction of footnotes and cross-references. It's at the Working Draft stage.
- **CSS Generated and Replaced Content:** This module introduces the concept of replacing the effective content of an element with the one generated by CSS. It's at the Working Draft stage.
- **CSS Hyperlink Presentation:** This module extends the normal way CSS processes hypertext links, thus providing greater control to the authors regarding their states. This module is at the Working Draft stage.
- **CSS Line Layout:** In this module, the layout of the inline elements is defined with more precision. It's at the Working Draft stage.
- **CSS Lists:** This module deals with list layouts with more detail and precision (ordered and unordered) than in earlier releases. It's at the Working Draft stage.
- **CSS Multicolumn Layout:** This module defines new properties and values to manage layout over several columns. It's at the Last Call stage.
- **CSS Namespaces:** This module defines the ways to select elements on the basis of the presence of a certain namespace. It's essential for formatting XML documents and is currently at the Candidate Recommendation stage.
- **CSSOM View Module:** This module allows authors to obtain information about elements without resorting to scripting. It's at the Working Draft stage.
- **CSS Paged Media:** This module extends the CSS properties for print to obtain headers, footers, and page numbers. It's at the Working Draft stage.
- **CSS Presentation Levels:** This module introduces the concept of multiple presentations of the same document. It's designed to facilitate particular layouts, such as those of presentation slides, and it's at the Working Draft stage.
- **Grid Positioning:** In the new CSS3 layout model, one positioned element forms a presentation grid. This module proposes a series of coordinates for the positioning of the floated elements that have an absolute position. The module is in the Working Draft stage.
- **CSS Text:** This module addresses the need for internationalization in defining new properties and values to control the text using CSS. It's at the Working Draft stage.

- **CSS 2D Transforms Module:** This module introduces concepts that are already featured in SVG (Scalable Vector Graphic) to CSS, such as transformation, rotation, and the scaling of elements. It's at the Working Draft stage.
- **CSS 3D Transformations Module:** This module extends the previous one with new specifications for transformations. It's at the Working Draft stage.
- **CSS Transitions Module:** This module introduces the concepts of transition and delay in transitions between states among the elements (for example, when an element receives focus and then loses it). It's at the Working Draft stage.
- **CSS Animations Module:** This module introduces new properties that can control the intermediate stages of the animation of the elements (for example, stages in a sequence). It's at the Working Draft stage.

Anatomy of a CSS3 declaration

CSS and HTML are inseparable friends. Therefore, to be able to fully take advantage of CSS statements, it is essential to understand the structure of an HTML document.

After a long period of silence, HTML recently has been brought back to life, thanks to the work of companies such as Apple, Google, Opera Software, and the Mozilla Foundation. They collaborated under the name of WHATWG (which stands for the Web Hypertext Application Technology Working Group, whose web site is at www.whatwg.org/) on the development of an updated and enhanced version of the old HTML.

Following this major interest, the W3C began to work on a new version of HTML, called HTML5. It's official name is Web Applications 1.0, and it introduces structural elements to HTML that have not been seen before.

These new elements bridge the gap between structure, defined by the markup; rendering characteristics, defined by styling directives; and the content of a web page, defined by the text itself. Furthermore, HTML5 introduced a native open standard to deliver multimedia content such as audio and video, collaboration APIs, local storage, geolocation APIs, and much more.

Each HTML5 document defines a tree structure for a document, known as the Document Object Model (DOM). The DOM is a programming API for HTML documents, as well as XML. It defines the logical structure of documents, and web developers can use it to create and build documents, access and modify their structure, or delete elements and content. You can learn more about the DOM at the W3C DOM page at www.w3.org/TR/WD-DOM/introduction.html.

CSS takes full advantage of this concept because its fundamental mechanism is based on heredity. This makes it possible for most properties set for an element to be inherited by its descendants, hence the term "Cascading."

Here is some simple HTML5 code:

```
<!DOCTYPE html>
 <head>
  <title>Page Tile</title>
 </head>
 <body>
  <h1>Title</h1>
```

```
  <div>
   <p>My first paragraph</p>
  </div>
  <section>
<h2>My Heading</h2>
 <p>This is a second paragraph</p>
</section>
 </body>
</html>
```

> *If you want to learn more about HTML5, Apress has published many books on the subject. One that I recommend is written by the same authors as this book:* HTML5 Solutions Essential Techniques for HTML5 Developers *(http://www.apress.com/9781430233862).*

This document, like any valid HTML document, is an ordered hierarchy of elements that are linked to one another by a parent-child relationship. This hierarchy forms what is defined as the Document Object Model.

The DOM is a cross-platform and language-independent convention for representing and interacting with objects in HTML (as well as XML). Web browsers usually use an internal model similar to the DOM to render a document.

The simple HTML5 code declared earlier can be represented like this:

```
|-> Document
  |-> Root Element (<html>)
   |-> Element (<head>)
    |-> Element (<body>)
     |-> Element (<div>)
      |-> paragraph
    |-> Section
      |-> header
```

This hierarchy defines the structure of the document. It's used by CSS to define the styles of the element via CSS rules.

A CSS rule is applied using selectors followed by one or more declarations, as shown in Figure 1-1.

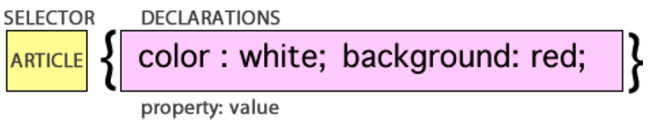

Figure 1-1. Declaration rule of a CSS.

Selectors can be any of the following:

- **Type Selectors:** These are represented by the name of a specific HTML element. They are used to select all specific types of an element in a document—for example:

```
body {color: red}
```

- **Class Selectors:** Each HTML can be assigned a class using the `class` attribute. You then assign a name to it that can be accessed by CSS—for example:

```
<h1 class="mytitle">This is a header</h1>
```

To apply one style to the class declared in HTML in the tag header, you precede the name of the class with a period (.):

```
.mytitle {font-family: Verdana}
```

- **ID Selectors:** Each HTML element can be assigned an ID, which is a unique reference for this element in the document—for example:

```
<section id="mystyle"></section>
```

To select an element that has been assigned a certain ID in CSS, you add the pound key (#) before the value of the ID:

```
#mystyle {color: black}
```

We've written this introduction to CSS selectors only to provide some essential concepts for readers who have never worked with CSS. It is not intended to be—and obviously doesn't provide—a full overview of selectors. In fact, there are other types of selectors: descendants, child, and so on. We have dedicated an entire chapter to this aspect of CSS3. You can find different solutions related to this vast subject in Chapter 2.

At this point, web developers have understood that CSS is an integral part of an HTML document. However, there are different ways to declare CSS for a document. You can have internal or external style sheets:

- **External CSS:** A style sheet defined in a separate file by the document with a .css extension.
- **Internal CSS:** The styles are included in the HTML document.

As far as external CSS files are concerned, you can link them by creating a `<link>` tag inside the head section:

```
<head>
<link rel="stylesheet" type="text/css" href="mystyle.css" />
</head>
```

The `<link>` element has a series of attributes that need to be specified:

- **rel: compulsory.** Describes the type of relation between the document and the linked file. It accepts the following values: `stylesheet` and `alternate stylesheet`.
- **href: compulsory.** Defines the absolute or relative URL of the style sheet.
- **type: compulsory.** Identifies the type of data to be connected. For CSS, the only possible value is `text/css`.
- **media: optional.** Declares the type of device to which the style sheet applies, such as the `handheld` property for handheld devices (typically, devices with a small screen and limited bandwidth).

Another way to load external CSS is to use the `@import` directive in the `<style>` element:

```
<style>
@import url(mystyle.css);
</style>
```

This approach of using the `@import` statement is one of the safest ways to solve compatibility issues between old and new browsers.

For internal style sheets, you define internal styles in the head section of an HTML page, by using the `<style>` tag, like this:

```
<head>
<style type="text/css">
body
{
 background: #FFFFCC;
}
</style>
</head>
```

There is another way to declare an internal style sheet: an inline style. The declaration can be made at the level of each tag in the page. To use inline styles, you declare the `style` attribute in the relevant tag:

```
<h1 style="color: red; font-size: 10px;">My Header</h1>
```

To conclude this section, we'll specify how to insert parts of a comment in a CSS. All you need to do is place the comment between these symbols:

```
/*
Multiline comment here
*/
```

Understanding the Box Model

In the previous section, we spoke about the structure of a document and how to apply a CSS rule to elements within a document. You can also use CSS to position elements within the page. This technique is called *CSS positioning*, or *CSS-P*. To use CSS-P rules, you need to understand how the browser physically draws the page on the screen based on the HTML code. The whole series of rules that manages the visual aspect of the elements is generally referred to as the *box model*.

Each box includes a certain number of basic components, and each can be modified with CSS properties. Instead of trying to explain with a thousand words what a box model represents, we'll use Figure 1-2 to provide a clearer illustration of the concept.

In the innermost part of the figure, you find the area of the page content, where you can see the background image and the sentence "content goes in here…". This is the area where the content of the HTML page is rendered: text, images, sections, paragraphs, media elements, and so on. The size assigned to this area is determined by the browser if you don't specify the width and height properties of the content.

On the outside of an element, you find the padding, which is empty space that can be created between the content and the border of the element to add some space between these elements.

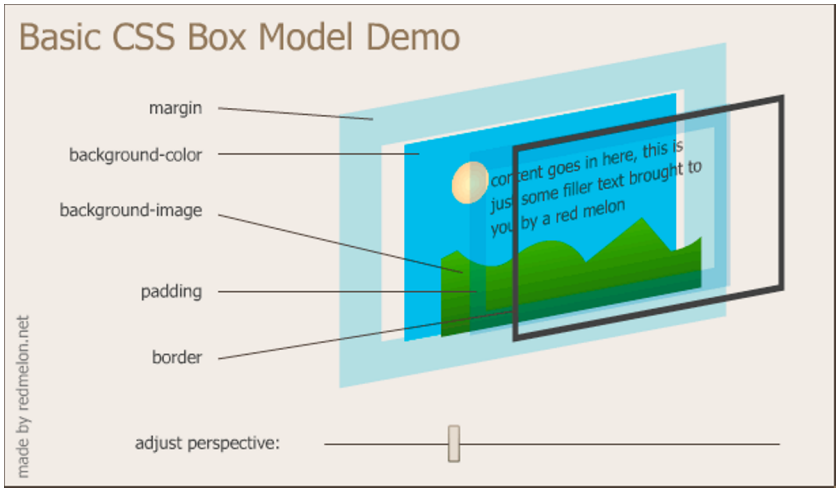

Figure 1-2. A CSS 3D box model that was created by redmelon.net and can be found at http://redmelon.net/tstme/box_model/.

Outside of that, you find the border, which is not an area but a variable line of dimension, style, and color that surrounds the padding zone and the content area. Finally, you get to the margin, which is a space that varies in size and separates a certain element from the adjacent areas.

The size of the box model, apart from the width of the content area, is obtained by this sum:

```
content width + left padding + left border width + right padding + right border width +
margin left + margin right
```

CSS3 doesn't introduce many new aspects here, but there are a couple of interesting ones. Later in this book, you'll see possible solutions for setting a border background and drawing rounded border corners, using the `border-image` and `border-radius` properties, respectively.

If you want to learn more about the important subject of the box model, you can refer to Chapter 6 or to the box model described by the W3C at `www.w3.org/TR/CSS2/box.html`.

Understanding CSS inheritance

Inheritance is one of the key concepts on which CSS techniques are based. In fact, CSS properties can inherit the display properties of dominant HTML tags, at least until you explicitly set a different value for a child element. This is why a property is applied to all child elements of the tag body, meaning the entire content of the page, if you set it to the body element.

Be careful, though, because not all properties are inherited. As a general rule, you can consider that the ones regarding box-model formatting (padding, margins, and borders) are properties that are not inherited by the child elements.

Earlier in the chapter, you saw that CSS can be declared and applied to the page as an external CSS, or it can be declared and applied as an internal or inline CSS. The type of declaration influences how a property will be inherited by an element, as well as how high or low the importance of each statement is.

To understand the way inheritance works, bear in mind that the CSS rule applied will be the one that is closest to the element in the code of the document. The order, therefore, is the following:

- **Inline style:** The ones that are applied last by the browser. They prevail over those that are declared in the page.
- **Internal CSS:** These prevail over the styles declared in the external CSS.

Consider the following example of an external style sheet declared as a CSS file named *styles.css*:

```
.myclass {
background-color:red;
}
```

This CSS file is then imported into the HTML page, where an internal and inline declaration has been added:

```
<link href="styles.css" type="text/css" rel="stylesheet" />
<style type="text/css">
. myclass {
background-color:white;
}
</style>
<article class="myclass" style="background-color:green; ">
CSS3 Rocks !
</article>
```

In the preceding example, which color will the web browser render for the article's background color? Read the code carefully.

Notice that the `article` element with the class name "`myclass`" gets its background color from an external style sheet, from styles defined on the page, and from an inline style. So the right reply to the question is that the article will get the `background-color: green` rule, inherited by the inline declaration, because it's the last rule applied by the browser.

Solution 1-1: Discovering CSS3 compatibilities across browsers

CSS3 provides a new set of tools to empower you to improve the look and feel of your web pages. However, like all new technology, it suffers from inconsistent cross-browser compatibility. In fact, there are new CSS3 features that work only on some browser versions, making web developers' lives more difficult. Therefore, it's essential to learn the compatibility matrix by heart for each version of each browser, or use a tool to help you out.

What's involved

There are many web sites that provide comparative tables to see at a glance which CSS3 features are supported by which browser. In this solution, you'll see some of the most popular ones:

- CSS3 Please (which you can check out at css3please.com) a Cross-Browser CSS3 Rule Generator
- CanIUse (which you can check out at www.caniuse.com) is a compatibility table for support of HTML5, CSS3, SVG and more in desktop and mobile browsers.
- FindmebyIP (which you can check out at www.findmebyip.com/litmus/) is a cute little app that presents your browsers' support for advanced HTML5 and CSS3 features in an easy to read manner.
- HTML5 Please, (which you can check out at html5please.com the new HTML5 and CSS3 features, knowing if they are ready for use.

How to build it

CSS3 Please is more than just a simple compatibility table to discover the browser support of CSS3. It allows you to edit CSS3 property values in real time that will be applied to the web page. By doing this, you can copy the all of the generated CSS values, or only some of them, and paste them into your own style sheet.

For example, you can interact with a `border-radius` property by changing the values contained in the following class:

```
.box_round {
 -webkit-border-radius: 12px; /* Saf3-4, iOS 1-3.2, Android ≤1.6 */
 -moz-border-radius: 12px; /* FF1-3.6 */
 border-radius: 12px; /* Opera 10.5, IE9, Saf5, Chrome, FF4, iOS 4, Android 2.1+ */
```

As you can see, the `.box_round` class provides the code for the various declarations to make the properties work on all types of browsers, including the following ones:

- Safari
- The iOS browser
- The Android browser
- Firefox
- Opera

The values assigned to each property can be changed on the fly, and the upper left box of the web page will change automatically to show the new values, as you can see in Figure 1-3.

It also comes with the following two interesting features:

`[to clipboard] [toggle rule off]`

Figure 1-3. The CSS3,Please box changes according to the CSS3 values.

You can use the first one to copy the CSS3 code in the clipboard. With the second one, you can decide whether or not to apply the CSS3 rule to the object.

This tool is essential both as a learning tool as well as a way to improve the productivity of your web development effortsbecause it provides cross-browser code.

HTML5 Please, on the other hand, is a traditional but well-built search engine that looks up the features of CSS3 and HTML5. It allows you to assess the HTML5 and CSS3 compatibility level for each property. For example, if you search for a CSS3 feature such as `border-radius`, you'll get the description shown in Figure 1-4.

On the lower left side, you can see a link labeled `View browser share %` that points to the Canluse.com table. Canluse.com is the reference comparison table for HTML5, CSS3, SVG support, and more in desktop and mobile browsers.

The Canluse.com service, like others, is completely free, and you can use it to quickly see HTM5 and CSS3 features and their compatibility, both as an indexed list and as a table. You also can use it to search for a particular property by using a search box, as shown in Figure 1-5.

By clicking an item on the list, you get more information and the view switches to Tables mode, as shown in Figure 1-6.

As a general rule, because CSS3 and HTML5 standards are evolving, you should pay attention to the features that you want to use in your web pages. In fact, you should apply elements from CSS3 gradually, as updates become necessary.

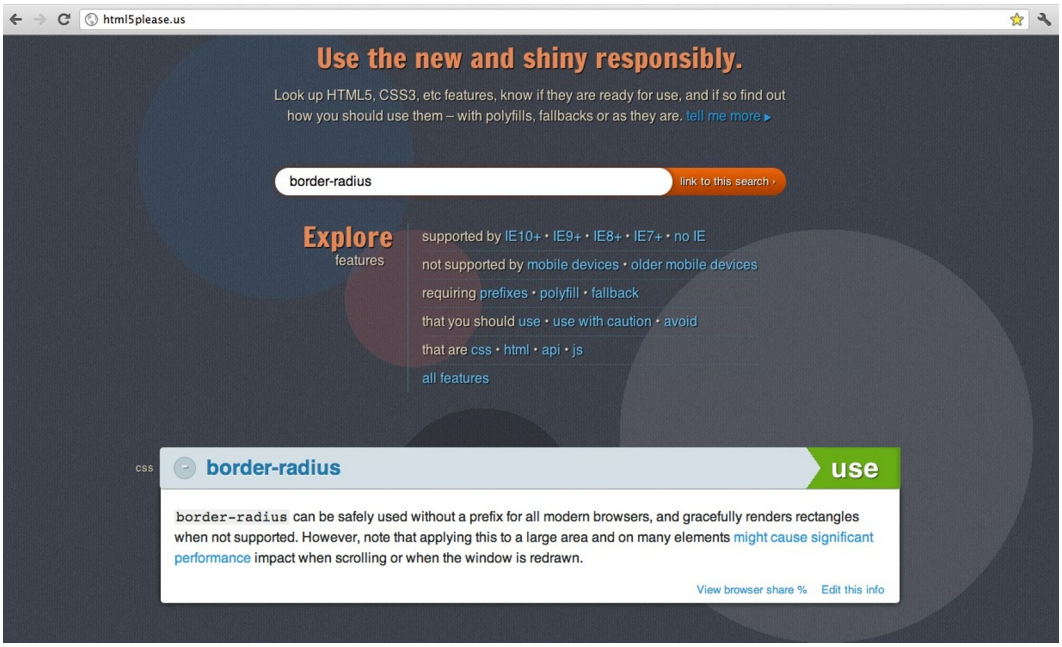

Figure 1-4. The HTML5 Please search box.

Figure 1-5. The table reference provided by CanIUse.com.

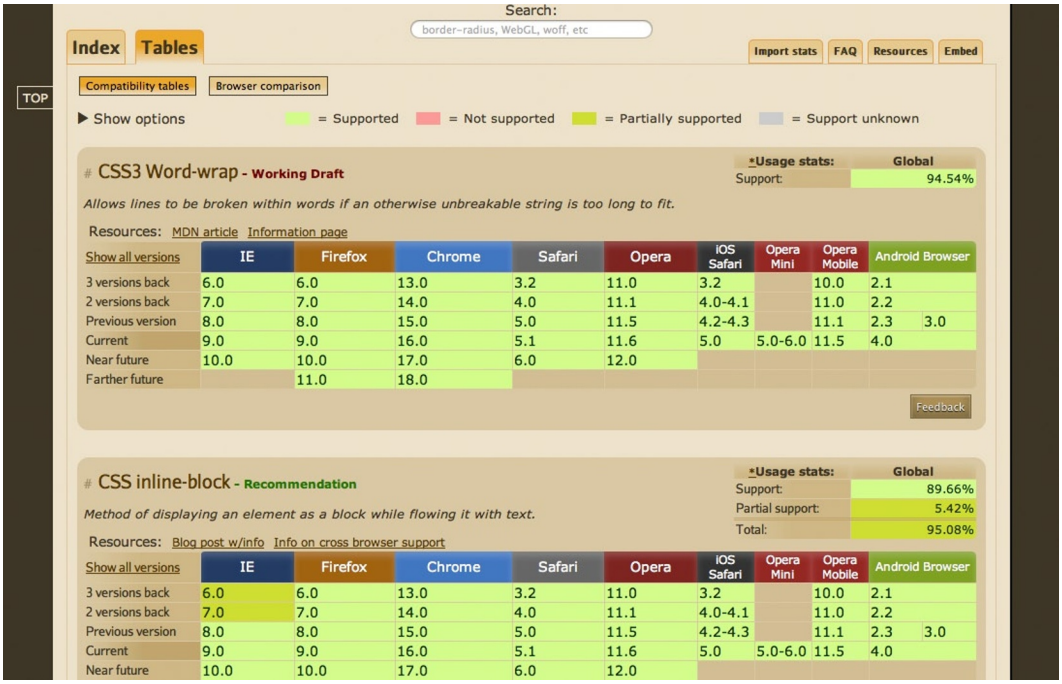

Figure 1-6. When you click on a property, you get extra information.

Expert tips

Canluse.com allows you to point directly at the feature you want to check by inserting `#feat equal` to the name of the feature in the URL, as follows:

`http://caniuse.com/#feat=css-boxshadow`

By inserting this URL in the browser, you can obtain the response from the site shown in Figure 1-7.

Solution 1-2: Adding a CSS3 file with JavaScript

Web developers can use JavaScript to load a CSS file dynamically, possibly on the basis of the type of page that is called by the user or the type of rights that a user has within an application. The following solution illustrates how to use the JavaScript language to load an external CSS file.

What's involved

The standard HTML5 procedure to load an external CSS file on a page is to point a reference to it in the HEAD section of your page with the `<script>` tag:

```
<head>
<link rel="stylesheet" type="text/css" href="myCSS.css" />
</head>
```

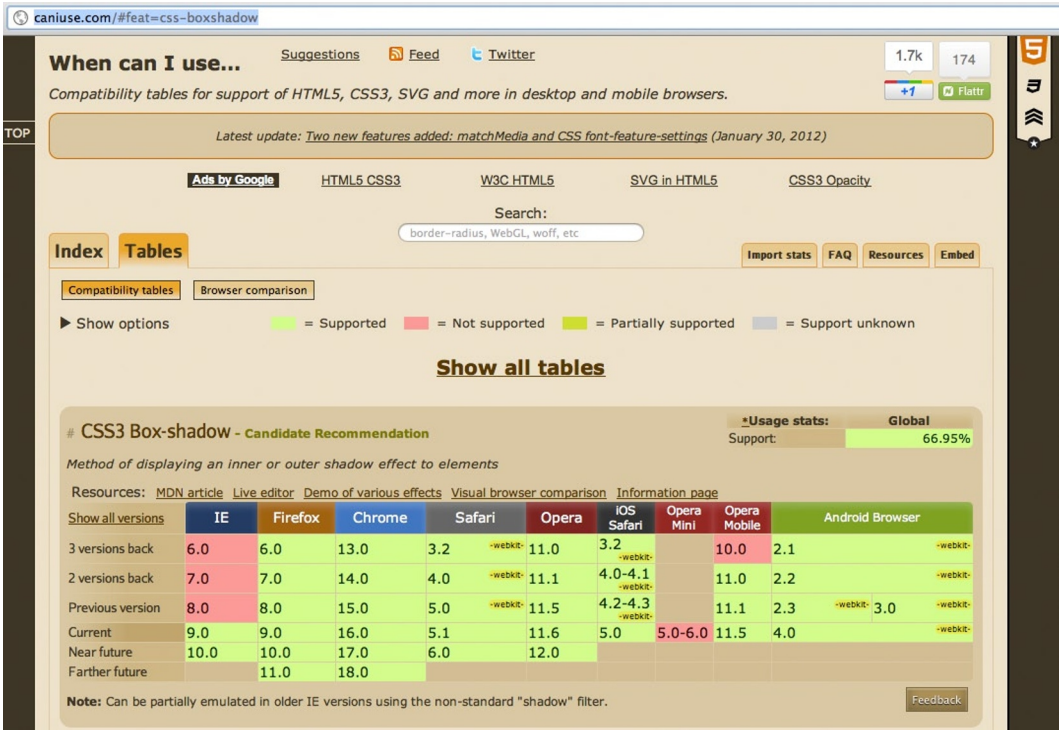

Figure 1-7. Calling a property directly from the browser address bar.

When the browser reads the content of the HTML page that is loaded, the CSS file is added to the page. Therefore, we could say that the external file is loaded *synchronously*.

However, with JavaScript, you can load the external file on demand using the `createElement()` method of the DOM object document to create the `<link>` tag that will then load the file:

```
document.createElement('link');
```

You can set the properties of the `link` object that has just been created to specify the type of content to load, `text/css`, and the pathway and name of the file to be loaded:

```
link.rel = 'stylesheet';
link.type = 'text/css';
link.href = 'myFileCSS.css';
link.media = 'all';
```

To be able to apply the `link` object to the page and load the CSS file, you have to call the DOM method `appendChild()`, which adds a node after the last child node of the specified element node:

```
document.getElementsByTagName('head')[0].appendChild(link)
```

This method returns the new child node.

Now let's see how to build the complete solution.

How to build it

In the previous section, we discussed methods that can be used to load a CSS file using JavaScript.

To do this, you create a JavaScript function that accepts two parameters: one is the name of the CSS file to be loaded, and the other is the ID assigned to the link tag. This second parameter allows you to check whether the file has already been loaded.

You start by writing the following function:

```
function loadCSSfile(filename, cssID)
{
var cssId = cssID
```

Insert an `if()` control that checks that the ID of the element isn't already in the page. This would mean that the CSS file has already been loaded:

```
if (!document.getElementById(cssId))
{
```

At this point, you can create the `link` object and set its properties:

```
  var head = document.getElementsByTagName('head');
  var link = document.createElement('link');
  link.id  = cssId;
  link.rel = 'stylesheet';
  link.type = 'text/css';
  link.href = filename;
  link.media = 'all';
  head[0].appendChild(link);
}
}
```

To use the JavaScript method you just created, all you have to do is recall it in a JavaScript block and associate it, for example, to the `onload` event of the `window` object:

```
<!doctype html>
 <html>
 <head>
  <title>onload test</title>
  <script>
function loadCSSfile(filename, cssID)
{

var cssId = cssID

if (!document .getElementById(cssId))
{
  var head = document.getElementsByTagName('head');
  var link = document.createElement('link');
```

```
  link.id  = cssId;
  link.rel = 'stylesheet';
  link.type = 'text/css';
  link.href = filename;
  link.media = 'all';
  head[0].appendChild(link);
}
}
  window.onload = load;
  </script>
 </head>
 <body>
  <p>The CSS file is loaded dynamically via JavaScript!</p>
 </body>
</html>
```

Or, if you want to insert the `loadCSSfile()` method in an external JavaScript file, first you have to load it and then call it like this:

```
<script type="text/javascript" src="externalJavascript.js"></script>
<script type="text/javascript">loadCSSfile('main.css', 'myCSSId')</script>
```

Expert tips

To set the properties of the link object, you use `setAttribute()`:

```
var link=document.createElement('link');
link.setAttribute("rel", "stylesheet")
link.setAttribute("type", "text/css")
link.setAttribute("href", 'myFileCSS')
```

The problem with this method is that Internet Explorer 6 doesn't support it consistently. So the script would not have worked under the aged Internet Explorer 6. If you're using JQuery or YUI Ajax frameworks, there are methods you can invoke from these libraries.

For JQuery, there is a plugin that loads CSS files and JavaScript files on demand and keeps track of what has already been loaded:

`code.google.com/p/rloader/`

For the Yahoo YUI library (shown in Figure 1-8), you can use the following method, which also supports cross-domain loading:

`yuilibrary.com/yui/docs/get/`

Solutions 1-3: Declaring multiple backgrounds for your web page

The background of a web page gives the finishing touch to a website. With CSS3, you can now use multiple background images.

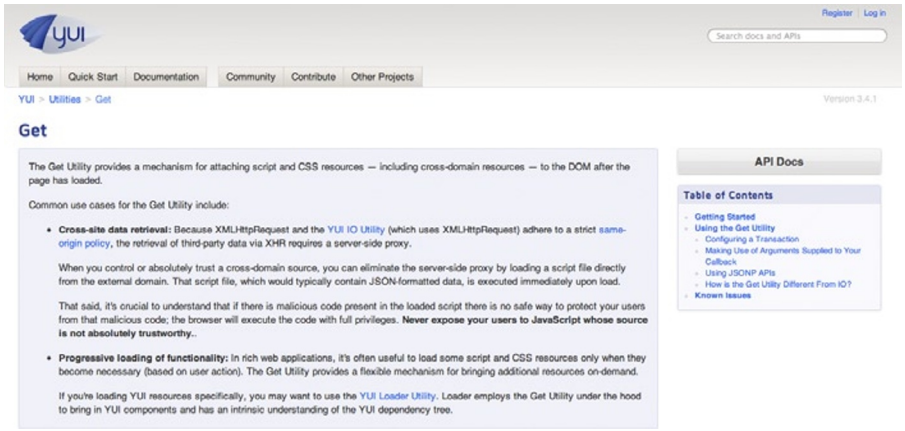

Figure 1-8. The YUI library page that documents the `Get` method.

What's involved

Background image management has always been entrusted to the `background` property. To be valid, the declaration doesn't have to contain references to all its properties, but it has to at least contain the definition of the background color. For example, to create an image that is repeated horizontally as a background on the body, with a background color taken from an external image, you can write the following:

```
body{background: #7A515A url(gradient.jpg) fixed repeat-x bottom}
```

To use multiple background images for your pages, you can simply declare a simple comma-separated list under the `background` property.

Let's see how to obtain this result in this solution.

How to build it

To use multiple images as background, all you have to do is use the `background` property and specify two or more values in the URL:

```
background-image: url(myFirstBG.png), url(myFSecondBG.png);
```

Multiple backgrounds can also be specified using the background shorthand property:

```
background: url(myFirstBG.png) 0 0 no-repeat, url(mySecondBG.png) 0 0 repeat-x
```

Here is some detail from the CSS Backgrounds and Borders Level 3 specification (which is available at www.w3.org/TR/css3-background/#backgrounds):

> "The number of comma-separated items defines the number of background layers. Given a valid declaration, for each layer the shorthand first sets the corresponding layer of each of 'background-position', 'background-size', 'background-repeat', 'background-origin', 'background-clip' and 'background-attachment' to that property's initial value, then assigns any explicit values specified for this layer in the declaration. Finally 'background-color' is set to the specified color, if any, else set to its initial value.
>
> If one <box> value is present then it sets both 'background-origin' and 'background-clip' to that value. If two values are present, then the first sets 'background-origin' and the second 'background-clip'".

Here is a complete example (which works in all new browsers but does not display properly in Internet Explorer 8):

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8" />
<title>Solution 1-3: Declaring multiple backgrounds for your web page </title>
<style>
.boxBG{

background-image: url(firstImg.png), url(secondImg.png);
background-position: center bottom, left top;
background-repeat: no-repeat;
}
</style>
</head>
<body>
<section class="boxBG">
<p>
This has multiple backgrounds!
</p>
</section>
</body>
</html>
```

Expert tips

To make this style also work on old browsers that still don't support the loading of multiple images, all you have to do is add a line with the same `background` property but with only one image:

```
<style>
.boxBG{

  background: url(apple.jpg) no-repeat;
  background: url(firstImg.jpg) 0 0 no-repeat, url(secondImg.jpg) 100% 0 no-repeat;
  width: 500px;
  height :250px;
}
```

Solution 1-4: Controlling the image aspect ratio

Every HTML page works with media elements such as images. Most of the websites that allow users to upload images have one issue in common: the image's dimension. In fact, uploaded images might not be in the right size and can alter or even disrupt your page layout. You have to avoid this scenario.

With CSS3, you can control the image aspect ratio using JavaScript functions.

What's involved

To maintain the aspect ratio of the images on a page and get them to fit within a fixed area, you can use the `object-fit` CSS3 property.

As defined by the W3C, this property specifies how the contents of a replaced element should be fitted to the box established by its height and width. A *replaced element* is an element whose content is defined by an external resource such as an image.

These are the values that `object-fit` accepts:

- `fill`: The replaced content is sized to fill the element's content box. The object's concrete object size is the element's width and height.
- `contain`: The replaced content is sized to maintain its aspect ratio while fitting within the element's content box. Its concrete object size is resolved as a contain constraint against the element's width and height.
- `cover`: The replaced content is sized to maintain its aspect ratio while filling the element's entire content box. Its concrete object size is resolved as a cover constraint against the element's width and height.
- `none`: The replaced content is not resized to fit inside the element's content box. The object's concrete object size is determined using the default sizing algorithm with no specified size, and using a default object size equal to the replaced element's width and height.
- `scale-down`: Size the content as if `none` or `contain` was specified, whichever would result in a smaller concrete object size.

The `object-fit` property can be applied to images as well as to a video or SVG file.

Suppose that you want to control the aspect ratio of an image with the following CSS statement:

```
img {
 object-fit: contain;
}
```

Following is a complete example.

How to build it

In this solution, you create a simple image gallery that contains images of different sizes. Those images will use the `object-fit` property that's set to the value `contain`. By doing this, you're forcing all the images to fit inside the area and maintain the aspect ratio.

This is the complete code for this solution:

```
<!DOCTYPE html>
<html>
<head>
<title>Solution 1-4: Controlling the image aspect ratio</title>
<style>
div {
 margin-bottom: 20px;
 padding: 20px;
}
img {
 position: absolute;
 width: 100px;
 height: 100px;

 -ms-object-fit: contain;
 -moz-object-fit: contain;
 -o-object-fit: contain;
 -webkit-object-fit: contain;
 object-fit: contain;
}

div p {
  font-family:Arial, Helvetica, sans-serif;
       margin-left: 110px;

}

</style>
</head>
<body>

<h1>Wine Tasting</h1>

<div>
<img src="IMG_2407.jpg" width="480" height="640">
<p>Le Pergole Torte 2004</p>
</div>

<div>
<img src="IMG_2444.jpg" width="640" height="480">
<p>Tenuta San Guido Wines</p>
</div>

<div>
<img src="IMG_2538.jpg" width="480" height="640">
<p>Guidalberto 2004</p>
</div>

<div>
<img src="IMG_2535.jpg" width="480" height="640">
```

```
<p>Vernaccia di San Gimignano 200</p>

</div>
</body>

</html>
```

If you save and run the application in a web browser, you'll see that all the images fit inside the DIV and maintain the aspect ratio.

The CSS `object-fit` property performs the magic:

```
img {
 position: absolute;
 width: 100px;
 height: 100px;

 -ms-object-fit: contain;
 -moz-object-fit: contain;
 -o-object-fit: contain;
 -webkit-object-fit: contain;
 object-fit: contain;
}
```

You've declared the `object-fit` property using various suffixes to make it compatible with all the major browsers: `-ms`, `-moz`, `-o`, and `-webkit`.

Expert tips

There is another very useful property you can use with the images to specify their resolution: the `image-resolution` property. This property is defined by the W3C as a property that specifies the intrinsic resolution of all raster images (it cannot be used with vector images such as SVG) used in or on the element.

Reading the definition, you see that you can apply the property to both content images and decorative images (such as `background-image`). Its values have the following meanings:

- `<resolution>`: This specifies the intrinsic resolution explicitly. A "*dot*" in this case corresponds to a single image pixel.
- `from-image`: The image's intrinsic resolution is taken as that specified by the image format. If the image does not specify its own resolution, the explicitly specified resolution is used (if given). Otherwise, it defaults to `1ddpx`.
- `snap`: If the snap keyword is provided, the computed `<resolution>` (if any) is the specified resolution rounded to the nearest value that would map one image pixel to an integer number of device pixels. If the resolution is taken from the image, the intrinsic resolution being used is the image's native resolution similarly adjusted.

The following CSS declaration forces the image resolution to 300 dots per inch (dpi):

```
img { image-resolution: 300dpi }
```

The resolution in the image, if any, is ignored.

Solution 1-5: Resetting CSS3 default values

It's a sad reality that each browser uses its own rules to render HTML elements. This introduces a lot of inconsistency, and web designers and developers have to spend a lot of time making sure their web page renders the same across browsers.

A common solution to this problem is to use a CSS reset script that removes and neutralizes the inconsistent default styling of `elements`, `margin`, `padding`, `font`, and so on.

What's involved

Mainly, the differences across browsers are related to the `margin` and `padding` properties because each browser sets their values in a different way. So the simplest approach is to set a global selector that sets the `margin` and `padding` properties to zero:

```
* {
  margin: 0;
  padding: 0;
}
```

Dong this is not enough, however, because other important properties have to be considered, such as `outline` (that is, a line drawn around elements to make the element stand out), `border`, `font-size`, and many more. So you need to use a more sophisticated approach that takes into account all of these properties.

How to build it

The following CSS reset rules are the ones most frequently used by web designers and developers to reset the CSS properties:

```
* {
  vertical-align: baseline;
  font-weight: inherit;
  font-family: inherit;
  font-style: inherit;
  font-size: 100%;
  border: 0 none;
  outline: 0;
  padding: 0;
  margin: 0;
}
```

This approach uses the global selector and sets the properties to their default values. With these CSS rules, the browsers won't introduce inconsistencies in properties related to default margins and padding, line heights, font sizes, headings, and so on.

With the new HTML5 elements, you need to consider new HTML elements to add to your CSS3 reset rules, such as `video`, `footer`, `article`, `audio`, and so on.

So here's a more complete CSS3 reset script solution that uses these new HTML5 tags:

```
* { outline: 0; }
html, body, div, span, object, iframe,
h1, h2, h3, h4, h5, h6, p, blockquote, pre,
abbr, address, cite, code,
del, dfn, em, img, ins, kbd, q, samp,
small, strong, sub, sup, var,
b, i,
dl, dt, dd, ol, ul, li,
fieldset, form, label, legend,
table, caption, tbody, tfoot, thead, tr, th, td,
article, aside, canvas, details, figcaption, figure,
footer, header, hgroup, menu, nav, section, summary,
time, mark, audio, video {
  margin:0;
  padding:0;
  border:0;
  outline:0;
  font-size:100%;
  vertical-align:baseline;
  background:transparent;
}
input[type="submit"]::-moz-focus-inner, input[type="button"]::-moz-focus-inner { border : 0px; }
input[type="search"] { -webkit-appearance: textfield; }
input[type="submit"] { -webkit-appearance:none; }
```

If you use Google, you'll find several CSS reset scripts ready to use in your code. We'll briefly look at reset style sheets again in Chapter 7. Here are some of the ones most frequently used by the developer community:

- Eric Meyer's Reset CSS 2.0
- HTML5 Doctor CSS Reset
- Yahoo! CSS Reset (YUI 3)
- Vanilla CSS Un-Reset
- Universal Selector '*' Reset
- Tripoli CSS Reset by David Hellsing
- undohtml.css by Tantek Celik

Summary

In this first chapter, you learned that Cascading Style Sheets came about because of the need to separate the presentational layer from the logic of the application. Their aim has always been to provide the users with a simple language to define the styling aspects of web pages. This chapter showed you how

to declare a CSS style as a series of properties for content, such as styles for the font family, size of the font, color, and so on.

You used basic techniques to do the following:

- Discover CSS3 compatibilities across browsers
- Add a CSS3 file with JavaScript
- Declare multiple backgrounds for your web page

In the next chapter, we'll take a closer look at CSS selectors and address common issues that developers have with them.

Chapter 2

CSS Selectors

In the previous chapter, we talked about how to build a CSS rule. You saw how a rule in a style sheet is made up of one or more selectors and by a group of properties and relevant values, expressed in the following form:

```
selector {
  property: value;
  }
```

A *selector* represents a structure, meaning it specifies which elements of an HTML page the rule will apply to. The properties and the relevant values deal with the presentation of these elements.

Therefore, one can guess that selectors are a fundamental part of CSS. Indeed, they've been around since the very first CSS specifications. Selectors Level 1 and Selectors Level 2 are defined as the subsets of selector functionality; that is defined in the CSS1 and CSS2.1 specifications, respectively.

Some new elements have been introduced in the CSS3 version. In this chapter, we'll address the new features of CSS3 selectors.

Differences compared to CSS2 selectors

There aren't many differences between CSS2 selectors and the new CSS3 selectors. From a functional point of view, CSS3 selectors are now a part of the CSS3 Module, which has its own independent specification.