



Community Experience Distilled

HTML5 and CSS3 Transition, Transformation, and Animation

A handy guide to understanding Microdata, the new JavaScript APIs, and the new form elements in HTML5 and CSS3 along with transition, transformation, and animation using lucid code samples

Aravind Shenoy
Gianluca Guarini

[PACKT] open source*
PUBLISHING community experience distilled

www.allitebooks.com

HTML5 and CSS3 Transition, Transformation, and Animation

A handy guide to understanding Microdata, the new JavaScript APIs, and the new form elements in HTML5 and CSS3 along with transition, transformation, and animation using lucid code samples

Aravind Shenoy

Gianluca Guarini

[PACKT] open source 
PUBLISHING community experience distilled

BIRMINGHAM - MUMBAI

HTML5 and CSS3 Transition, Transformation, and Animation

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 authors, 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: November 2013

Production Reference: 1141113

Published by Packt Publishing Ltd.
Livery Place
35 Livery Street
Birmingham B3 2PB, UK.
ISBN 978-1-84951-994-6

www.packtpub.com

Cover Image by Neha Rajappan (neha.rajappan1@gmail.com)

Credits

Authors

Aravind Shenoy
Gianluca Guarini

Reviewers

Younes Baghor – w3bwizart
Rodrigo Encinas
Pavlo Iuriichuk
Paul Shipley
Yuxian, Eugene Liang

Acquisition Editor

Jonathan Titmus
Gregory Wild

Commissioning Editor

Sruthi Kutty

Technical Editors

Kapil Hemnani
Nikhil Potdukhe
Tarunveer Shetty

Project Coordinator

Amigya Khurana

Proofreader

Ting Baker

Indexer

Tejal Soni

Graphics

Disha Haria

Production Coordinator

Nilesh R. Mohite

Cover Work

Nilesh R. Mohite

About the Authors

Aravind Shenoy is an in-house author at Packt Publishing. An engineering graduate from the Manipal Institute of Technology, his core interests are technical writing, web designing, and software testing. He is a native of Mumbai, India, and currently resides there. He has written books such as *An overview on Apache Hadoop*, *JavaScript: Web Designing Fundamentals*, and *CSS Essentials in a Nutshell* and articles on various technologies.

I would like to thank my mom, Vatsala, my uncle Suresh Kamath, and sister Aruna for their continued patience and moral support. Thanks to the entire team at Packt Publishing who were involved in the entire process of publishing this book. Special thanks to the reviewers who helped me in this walk of life and to Azharuddin, Arun, Mayur, and Ankita (all my teammates at Packt Publishing) for motivating me in my journey.

Gianluca Guarini is a 25-year-old web developer, with strong design skills, working currently in Zurich. He was born in Avellino, a sunny city in the south of Italy, and he grew up designing things and playing video games from an early age on a Commodore 64. He graduated in Psychology of Communication in Milan and has worked collaborating with small web agencies as a freelancer, always searching for new projects to enhance his skills.

In 2011, he wrote an HTML5 guide for the biggest Italian web design e-learning portal HTML.it and in the same year he started a long collaboration with Radio DeeJay pushing the HTML5 technology into the mainstream in Italy thanks to an amazing working group. He won, with Radio DeeJay team, the first prize for the best Design/UX in Dev Unplugged (an HTML5 contest for the launch of Internet Explorer 9) realizing The Visual Player, an amazing project that combines the use of HTML5 Canvas, Video, Audio, and SVG features.

In 2012, he moved to Zurich to work as frontend developer for Gold Interactive, then a brand-new web agency that likes to start any new project always by exploiting the coolest HTML5 features needed to make them unique and great. He believes in open source and is always sharing his tricky codes on Twitter from his account @gianluca guarini. You can contact him at gianluca.guarini@gmail.com and <https://github.com/GianlucaGuarini>.

About the Reviewers

Younes Baghor - w3bwizart, born in Belgium, started his career as a welder/constructor building new trucks and later as a container repairer. At night and the weekends he worked in the food and beverage sector (restaurants and bars) where he started as a dishwasher and worked his way up to a maitre d'hotel. But it was time for a new challenge.

In 2007, he started his bachelor's degree and finished it in 2010. During this time he discovered the beauty of the Web. Although his education was strongly focused on server-side .NET, he spent most of his free time discovering the modern HTML5/CSS3/JavaScript standards and the surrounding APIs, to learn a better way to display the user interface in the browser to provide a better user experience.

His energy and drive come from curiosity and the desire for continuous learning. When working on a project he is very goal orientated, using a lot of communication, research, questions, and discussions to achieve the goal. He likes to look at the whole picture from development to design, costs, user experience, and branding and bringing a simple solution to improve the current way of working.

He specializes in HTML5, CSS3, JavaScript, Semantics, OOP, Mobile First, responsive design, progressive enhancement, and social media strategies.

I would like to thank Packt Publishing for giving me the opportunity to be a part of this book.

Rodrigo Encinas has worked for more than 12 years for companies of different means, from advertising and television to world-class fashion brands, or communication companies. Nowadays, he is a consultant for international companies helping with the development of web applications and improving the user experience with best practices and modern patterns such as HTML5, responsive web design, and single-page applications.

I would like to thank Packt Publishing for the good job done and I would like to thank the readers for your interest in this field and encourage you to learn how to build the Web of the future.

Pavlo Iuriichuk is a frontend lead developer working for GlobalLogic. He previously worked with HTML5 technologies stack in gaming and mobile web application projects in Ciklum. He came to HTML5 land from Flex and ActionScript. He graduated from Kyiv Polytechnic Institute, faculty of Applied Maths, about two years ago, so now he is connecting Maths with real software development and trying to make this life easier.

Paul Shipley had an extensive career spanning near 30 years in application development, mainly in the telecommunications, insurance, government, and manufacturing industries. He has worked on projects ranging from small desktop applications in Visual Basic through to large complex corporate mainframe applications using SAS and COBOL. He is currently freelancing creating websites and applications for small and medium businesses using HTML5/CSS3, GWT, and Responsive Web Design techniques.

He is also a blogger, published author, and conference presenter.

He is co-author of *Photoshop Elements 2: Zero to Hero* along with Tom Arah, Adam Juniper, Barry Beckham, and Todd Pierson (Wrox Press, 2002, ISBN/ISSN: 1904344232).

Yuxian, Eugene Liang is a researcher, author, web developer, and business developer. He enjoys solving difficult problems creatively in the form of implementing web applications using Python/Django/Tornado, JavaScript/jQuery/Node.js. He also enjoys researching areas of social network analysis, social computing, recommendation algorithms, link analysis, data visualization, data mining, information retrieval, business intelligence, and intelligent user interface. He previously authored *JavaScript Testing Beginner's Guide*. Find him at <http://www.liangeugene.com>.

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.

Table of Contents

Preface	1
Chapter 1: Introduction to HTML5	7
Standardizing HTML	7
Differences between HTML 4 and HTML5	8
Why we must start using HTML5	9
Myths about HTML5	10
Summary	11
Chapter 2: Using the New Markup to Build a Semantic Page	13
Easier and faster syntax	13
Header	14
Footer	15
Nav	16
Article	18
Aside	19
Microdata	20
Summary	22
Chapter 3: Flexible Box Model in CSS3	23
Understanding Flexbox	23
Flex Container properties	26
flex-direction	26
justify-content	28
align-items	29
flex-wrap	31
Properties of Flex Items	33
Order	33
Flex	35
Summary	36

Chapter 4: Web Forms in HTML5	37
New form attributes in HTML5	38
placeholder	38
autofocus	39
required	40
datalist	42
Understanding new input types in HTML5	43
search	43
email and url	45
date	46
week	47
month	48
color	50
Summary	51
Chapter 5: Advanced Features of HTML5	53
Audio and video	54
Drag-and-Drop	58
Geolocation	60
Webstorage	63
sessionStorage	63
localStorage	65
Offline web applications	67
Canvas	70
beginPath	72
closePath	72
moveTo	72
stroke and fill	72
arc	72
lineTo	73
Gradients	74
save and restore	75
Transformations	77
translate	77
rotate	78
scale	79
Animation	80
Summary	82
Chapter 6: CSS3 Animations	83
CSS3 transitions	84
The transition-duration property	86
The transition-timing-function property	89

The transition-delay property	90
CSS3 transforms	91
rotate	91
scale	92
translate	92
skew	93
translate (3D)	95
rotate (3D)	96
preserve-3d	96
CSS3 animation	97
@keyframes	98
animation-name	98
animation-duration	98
animation-delay	98
animation-timing-function	98
animation-iteration-count	98
animation-direction	99
animation-play-state	99
Summary	104
Chapter 7: Tools and Utilities in HTML5 and CSS3	105
Modernizr	105
Liveweave	107
HTML KickStart	109
HTML5 Boilerplate	110
The CSS3 Cheat sheet	112
Summary	113
Index	115

Preface

HTML5 and CSS3 technologies are changing the face of the web, they are making the way we build websites, add new features, and develop more immersive experiences much faster and accessible to the masses. Transitions, transformations, and animations have always required a specialized component, until now. Learn to harness the power of HTML5 and CSS3 to make your interactive and visually compelling designs a reality.

HTML5 and CSS3 Transition, Transformation, and Animation will introduce any developer or designer to this new, exciting, and world-changing technology. Using practical and easy-to-follow examples, create visually compelling and interactive websites without the overhead and previously time consuming external components.

This is your jumpstart in learning to develop and realize your vision with the power and flexibility of HTML5 and CSS3.

HTML5 and CSS 3 Transition, Transformation, and Animation is your kick start to developing beautifully elegant, interactive, and entertaining web pages. You will start with a gentle reminder of the evolution in HTML and CSS, and then jump straight in following along with this example-driven, fast-paced exploration to help you quickly develop these highly prized skills in HTML5 and CSS3. You will finish with multiple artifacts to twist and change to suit your wildest imagination.

What this book covers

Chapter 1, Introduction to HTML5, explains the evolution of HTML5 along with the myths and facts about HTML5.

Chapter 2, Using the New Markup to Build a Semantic Page, explains the semantic markup of HTML5 and how to use the new properties to build a semantic page. Microdata is also explained in detail.

Chapter 3, Flexible Box Model in CSS3, will explain the concept of the Flexible Box Model in CSS3. The properties of the Flex Container and Flex Items are explained in detail along with code examples for the same.

Chapter 4, Web Forms in HTML5, will explain the new web form elements of HTML5. You will learn about the new input types and the new input attributes used in HTML5.

Chapter 5, Advanced Features of HTML5, will explain a lot of modern concepts, such as offline web apps, Geolocation, drag-and-drop, Webstorage, and creating an audio and video player. HTML5 Canvas is explained in detail in this chapter.

Chapter 6, CSS3 Animations, will explain the transition, transformation, and animation features of CSS3. Code examples are used to describe all the prominent features used for the CSS3 animation purposes.

Chapter 7, Tools and Utilities in HTML5 and CSS3, will give examples of the various tools and utilities used in HTML5 and CSS3, which will make coding simpler.

What you need for this book

You just need to use an editor, such as Notepad or Notepad++ to practice the code examples in this book. You can also use advanced editors for these examples. However, we recommend that you use a notepad to practice it. You can also change the code to understand the difference in the output. Hence, to understand the concept well, you can modify the code and practice it to understand the subtle nuances of HTML5 and CSS3.

Who this book is for

Basic knowledge of HTML 4 and CSS is required to understand this book. If you are a web developer or designer and would love to learn and use the new technologies included within HTML5, this is the right book for you. Start at the beginning and learn some of awesome features around transitions, transformations, and animations. This book is for beginners with transitions, transformations, and animations that want a quick and simple kick-start using clear and reusable examples.

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, database table names, folder names, filenames, file extensions, pathnames, dummy URLs, user input, and Twitter handles are shown as follows: "We will use the `display` property of CSS to explain the functionality. The `display` property is set to `flex` or `inline-flex`."

A block of code is set as follows:

```
<html>
  <head>
    <style>
      #flex-container {
        display: -webkit-flex;
        display: flex;
        width: 500px;
        height: 500px;
        background-color: Navy;
      }


      #flex-item {
        background-color: Silver;
        width: 200px;
        height: 200px;
        margin: 20px;
      }
    </style>
  </head>
  <body>
    <div id="flex-container">
      <div id="flex-item">Alpha</div>
      <div id="flex-item">Beta</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:

```
#flex-item {
  background-color: lime;
  transition-property: background, border-radius;
-webkit-transition-property: background, border-radius;
  transition-duration: 2s, 6s;
-webkit-transition-duration: 2s, 6s;
  transition-timing-function: linear;
-webkit-transition-timing-function: linear;
```

```
width: 200px;  
height: 200px;  
margin: 20px;  
}
```

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: "If we see the output, the Flex Item **Dos** was defined after **Uno** in the code. However, we assigned an order value of -1 to **Dos**, therefore, **Dos** was displayed first."

 Warnings or important notes appear in a box like this.]

 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 through the subject of your message.

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 to our website, or added to any list of existing errata, under the Errata section of that title.

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.

1

Introduction to HTML5

HTML originated from a prototype created by Tim Berners-Lee in 1992. He felt that there was a possibility of linking documents together by the use of hypertext and the concept of HTML evolved. The drawback was that the commercial hypertext packages available at that time such as **ZOG** and **Intermedia** were customized to suit different types of computers and were too ambiguous in nature.

He developed HTML (HyperText Markup Language) and in conjunction, developed a protocol for accessing text from other documents via hyperlinks. The protocol was called HTTP, and this paved the way for the future. HTML itself was derived from a markup language called **SGML (Standard Generalized Markup Language)**.

Standardizing HTML

Standardization is an ongoing process. Modifications were made constantly and versions were released accordingly. The various versions of HTML that have been released are as follows:

- HTML 2.0 (November 1995)
- HTML 3.2 (January 1997)
- HTML 4.0 (December 1997)
- HTML 4.01 (December 1999)

A breakthrough in the field was the introduction of **CSS** along with HTML 4.0. Prior to the introduction of CSS, web designers and developers used HTML for formatting purposes. Formatting and styling a web page using HTML defeats the purpose of HTML, as HTML elements and attributes must only define the structure of the web page. The purpose of CSS was to break styling out from structural markup. With the introduction of CSS, we could separate presentation from content.

As a result, formatting could be separated from the HTML document and stored in a separate file, which could then be included in the document using a link tag. Hence, all the presentational HTML elements and attributes were replaced by CSS to provide versatility and better accessibility. Now, we can define a look or modify the look of a web page by making changes in the style sheet without actually altering the code.

As far as HTML is concerned, the latest version, which is HTML5, is still in the development stage. The Web Hypertext Application Technology Working Group (WHATWG) and the World Wide Web Consortium (W3C) are working together on HTML5. The proposed year for the release is around 2014.

Differences between HTML 4 and HTML5

HTML5 will soon be accepted as the benchmark. It differs from the previous versions of HTML in various ways. HTML5 works with modern browsers and also offers backward compatibility. HTML5 has a lot of new features that will change the approach in designing websites. Some of the features present in the older versions of HTML have been omitted from HTML5.

Let's take a look at the difference between HTML 4 and HTML5:

- The document type declaration in HTML5 is very simple. All we need to do is type `<!DOCTYPE html>` so that the browser can recognize that we will be working with HTML5.
- Character encoding in HTML5 is far simpler. Earlier it was written in the following manner:

```
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
```

In HTML5, it is written in the following way:

```
<meta charset="UTF-8">
```

- Elements such as `center`, `frame`, `frameset`, and `noframes` have been omitted from HTML5. Key elements such as `basefont`, `big`, `font`, and `blink` do not exist in HTML5. All the things that have been omitted indicate that CSS will be used for styling purposes.
- New elements have been introduced in HTML5. The new elements are as follows:
 - `<header>`
 - `<footer>`
 - `<nav>`

- `<section>`
- `<aside>`
- `<article>`

We will look at these new elements in detail in the following chapters.

- Attributes such as `border`, `cellpadding`, and `nowrap` to mention a few have been removed from HTML5 as their functionality can be handled better by CSS.
- Since HTML5 is used extensively for web-based applications, modifications have been made in the present APIs and new APIs have been introduced. APIs have been developed for media elements such as audio and video. Drag-and-drop APIs and elements such as canvas have been included in HTML5. Facilities such as offline data storage, are a feature of HTML5 and APIs have been developed for this purpose.
- Error handling is another feature of HTML5, which will make it easier to write valid HTML code. HTML5 has strict parsing rules to handle errors in the code.

Although it is still in the development stage, let's see why we should start using HTML5 right away.

Why we must start using HTML5

HTML5 is still in the development stage. However, the following are the reasons why we must start implementing it right away.

- The content becomes much more accessible with the use of tags such as `aside`, `article`, `header`, `footer`, and `section`. Earlier, there was no way to understand the `div` element in the code even if there was an ID assigned to it. That has changed after incorporating tags such as `header`, `footer`, and so on. HTML5 assists the developer in writing cleaner code. For example, copyright data can be indicated using the `footer` tag. It makes things much more systematic.
- We can embed audio and video, thereby eliminating the need for additional plugins. We can control the audio and video elements using HTML or JavaScript whereas the styling will be taken care of by CSS.
- Web storage is an impressive feature of HTML5. The data is stored in the user's browser. This feature is of great use when we surf through e-commerce sites as user preferences can be stored on the browser.

- We can develop offline applications with HTML5. In this case, the user can access the application offline and also sync the data back with the remote server once he is back online.
- HTML5 is extensively used to develop mobile web applications. Tablets and mobile web applications are in vogue. For example, HTML5, along with JQM (jQuery for Mobiles), is used for developing complex mobile applications. The look of a web-based application on a mobile phone or tablet is different than that on a desktop computer. Most of the latest browsers (such as Google Chrome, Safari, Opera, Mozilla Firefox, IE9) support HTML5. HTML5 offers cross-browser compatibility and hence, it is the markup language of the future. HTML5 is not compatible with earlier versions of IE but there is a workaround for that. We can add a JavaScript Shiv to the code, which will make the browser aware of HTML5. JavaScript Shiv, created by *Remy Sharp*, is found on Google codes and can be included in the HTML document header in the following manner:

```
<!-- [if IE] >  
  <script src="http://html5shiv.googlecode.com/svn/trunk/  
    html5.js"></script>  
<![endif] -->
```

- An advantage of HTML5 is that it includes form-related attributes that provide enhanced functionality. Earlier we had to use JavaScript for this purpose. HTML5 has new elements and attributes that support the input type and form elements. For example, we have date pickers and also input types such as e-mail which account for entry of e-mail addresses.

Myths about HTML5

There are some misconceptions about HTML5. Let's look at some of the myths and facts surrounding HTML5:

- We need to understand that HTML5 is not a replacement for Flash. We can embed audio and video in HTML5 but to think that it will replace Flash is a myth. Some things can be easily done in Flash than in HTML5. For example, live video streaming is not possible with HTML5.
- Another misunderstanding is that HTML5 is still in the nascent stage and cannot be used. One has to understand that standardization is an ongoing process. However, HTML5 has a lot of impressive features that would encourage any web designer to use it. Moreover, browsers are getting updated regularly and the latest versions of some of the browsers already support it.

- HTML5 works well with IE (Internet Explorer). That IE and HTML5 are not compatible at all is a myth. It is true that versions prior to IE9 are not fully compatible with HTML5, however, developers can always write a fallback code in such situations. Browsers are coming out with their latest versions quite frequently and IE9 already supports it.
- Most people assume that CSS comes along with HTML5. However, that is not true. CSS is used for styling and presentation whereas HTML5 deals with content. An HTML page without CSS will be good enough to use but might not look that good in terms of presentation.
- One more misconception is that HTML5 does not have a development environment. We can create an HTML file with just a simple text editor. However, as a developer, we can definitely use an integrated development environment (IDE) such as Eclipse, Visual Studio, or even Dreamweaver CS5 to work with HTML5.



Suppose we add `<!DOCTYPE html>` to the code, do you think that it would become an HTML5 code? My answer would be a firm "No". HTML5 is not just some kind of a document type. HTML5 has a lot of new elements, attributes, input types, and so on. It is a whole set of rules that enables the developer to define a web page in an impressive manner.


Summary

We had a look at the differences between HTML 4 and HTML5 along with the facts and myths of HTML5. In the coming chapters, we will be learning about HTML5 in detail. In the next chapter, we will learn how to use the new markup to build a semantic page.

2

Using the New Markup to Build a Semantic Page

The main benefit of HTML5 is to build a web page using cleaner code. HTML5 makes things more systematic. The code in HTML5 is written in such a way that even a novice can understand it. In a futuristic scenario, even machines would be able to understand HTML5 in such a way that they would be able to analyze data on the web.

 We are not talking about artificial intelligence here. The concept of HTML5 is quite different in the sense that machines will be able to understand the content of the code in a much better way, paving the way for the future, as the exchange of information becomes more systematic.

Easier and faster syntax

Sectioning is an important part of HTML5. The sectioning elements are used in HTML5 to build a semantic web page in a quicker way. The elements used in HTML5 for sectioning are as follows:

- `<header>`
- `<footer>`
- `<nav>`
- `<article>`
- `<aside>`

We will now look at each element in detail.

Header

A crucial reason for using semantics is to increase accessibility. Prior to HTML5, there was no specific way to tag content based on its definitions. HTML5 changed all that. The `<div>` tag was used extensively along with headings to create a header for the web page. However, now we have the header tag. A header tag may contain headings nested into it but that is not a necessity.

Searching stuff on the internet is possible due to the search engines such as Google Search and Bing. SEO (search engine optimization) is a concept that is crucial and imperative for a website. For example, search engines with the help of crawlers can access a web page content based on the elements that contain them.

Hence, the header element is very important as it provides more accessibility than other elements, like the footer. The search engine finds the content in the header tag quite easily. Moreover, using the header element accounts for an appropriate way of web designing. The header can be used to denote the heading of a blog, a web page, or an article.

Let's look at an example of a header element using the following code snippet:

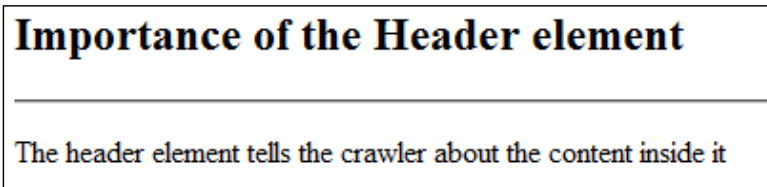
```
<!DOCTYPE html>
<html>
  <body>
    <header>
      <h2> Importance of the Header element </h2><hr>
    </header>
    <p> The header element tells the crawler about the content
      inside it </p>
  </body>
</html>
```



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.

The output of the code upon execution would look like this:



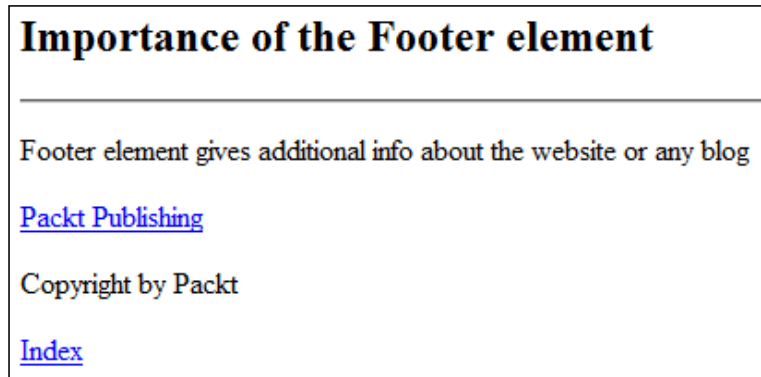
Footer

The footer element is used to denote things like the author's name, contact information, and copyright patent to mention a few. A footer allows the programmer to give a general idea about the website. However, footers can be used more than once in a document. An article can have its own independent footer in addition to another footer element in the source code for a specific web page. It completes the information in the document and makes it more meaningful.

Let's look at an example of a footer element using the following code snippet:

```
<!DOCTYPE html>
<html>
  <body>
    <header>
      <h2> Importance of the Footer element </h2><hr>
    </header>
    <p> Footer element gives additional info about the website or
      any blog </p>
    <footer>
      <a href="http://www.packtpub.com"> Packt Publishing </a>
      <br>
      <p> Copyright by Packt </p>
      <a href="/index.html">Index</a>
    </footer>
  </body>
</html>
```

The output of the code upon execution would look like this:



Nav

The nav element is an important part of HTML5 and is used extensively for navigation purposes. It doesn't mean that all links on a page have to be in the nav tag. For example, the footer element has links to varied content and the nav tag is not mandatory for these links.

The nav element is used when there is a group of navigation links contained in a specific section of the page. It is also used extensively where there are links to other pages or a part of the same page. Prior to HTML5, we used something like the following code snippet for navigation purposes:

```
<div id="nav">
<ul>
<li>...
```

However, with the nav tag, we can use more realistic code where we group the links in a nav element. Let's look at an example of a nav element using the following code snippet:

```
<!DOCTYPE html>
<html>
<body>
<header>
<h2> Importance of the Navigation element </h2>
</header>
<p> Use of the Nav element </p>
<hr>
<p> We can go to forums or refer to the articles on this page
<nav>
```

```
<ul>
<li> <a href="http://www.packtpub.com"> Packt Publishing
</li>
<li> <a href="http://www.packtpub.com/books/ajax"> Books on
AJAX </li>
<li> <a href="http://www.packtpub.com/latest_articles">
Latest Articles </li>
<li> <a href="http://packtlib.packtpub.com/"> Packt Online
Library </li>
<li> <a href="http://www.packtpub.com/news-center"> News
</li>
</nav>
<br>
<footer>
<p> <a href="http://www.packtpub.com/contact?r=1"> Contact
us </a> <p>
<a href="http://www.packtpub.com/about">About us</a>
</footer>
</body>
</html>
```

The output of the code upon execution would look like this:

Importance of the Navigation element

Use of the Nav element

We can go to forums or refer to the articles on this page

- [Packt Publishing](#)
- [Books on AJAX](#)
- [Latest Articles](#)
- [Packt Online Library](#)
- [News](#)

[Contact us](#)

[About us](#)

If you observe the code, you will find that there are links between the `<nav>` and `</nav>` tags as well as links in the footer tags. The content and links between the nav tags are for major navigation purposes, whereas the links in the footer tag are for some additional information.

Article

We often come across standalone content on a web page. There are blog entries, forum posts, and user comments to mention a few. The article element is used for this kind of data where the content is independent of its surroundings. It is a good practice to include header, footer, and headings within the article tags. We can also use the section element between the article tags.

Let's look at an example of an article element using the following code snippet:

```
<!DOCTYPE html>
<html>
  <body>
    <article>
      <h1> Fruit Facts </h1>
      <section>
        <h2> Citrus fruits </h2>
        Citrus fruits are good for your health.
        <br>
        Oranges and sweet limes are examples of citrus fruits.
        <br>
        They contain Vitamin C, which is good for colds and coughs.
      </section>
      <section>
        <h2> Dried fruit </h2>
        Dried fruit is very nutritious.
        <br>
        It is advisable to eat almonds to maintain good health.
        <br>
        Dried fruit helps boost the immune system.
      </section>
    </article>
  </body>
</html>
```

The output of the code upon execution would look like this:

```
Fruit Facts  
  
Citrus fruits  
  
Citrus fruits are good for your health.  
Oranges and sweet limes are examples of citrus fruits.  
They contains Vitamin C, which is good for colds and coughs.  
  
Dried fruit  
  
Dried fruit is very nutritious.  
It is advisable to eat almonds to maintain good health.  
Dried fruit helps boost the immune system.
```

In this code, we have included the section element within the article element. The article element may be standalone content and the section tag is used between the article tags as it is a generic content. Hence, the article element is used for independent content like RSS feeds and news articles whereas the section element is used to denote data of a generic nature.

Aside

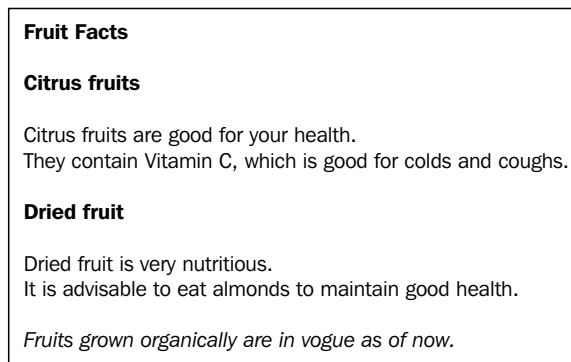
Content in an aside element is slightly related to the content outside it. It can be used within an article element in cases where the content is relevant to the content in the article element. Though the content in the aside element is standalone by nature, it can provide additional information about the content around it.

Let's look at an example of an aside element using the following code snippet:

```
<!DOCTYPE html>  
<html>  
  <body>  
    <article>  
      <h1> Fruit Facts </h1>  
      <section>  
        <h2> Citrus fruits </h2>  
        Citrus fruits are good for your health.  
        <br>  
        They contain Vitamin C, which is good for colds and coughs.
```

```
</section>
<section>
  <h2> Dried fruit </h2>
  Dried fruit is very nutritious.
  <br>
  It is advisable to eat almonds to maintain good health.
</section>
<br>
<aside>
  <p> <em> Fruits grown organically are in vogue as of now
    </em> <p>
</aside>
</article>
</body>
</html>
```

The output of the code upon execution would look like this:



We will now look at a new feature of HTML5 called **Microdata**.

Microdata

HTML 4.0 has a certain set of fixed elements, for example, we have the `<p>` and the `<h1>` elements for a specific purpose. However, in HTML5, we have the option of custom attributes. Suppose we have to define a person in HTML, we do not have an element for that. We cannot have elements for each and every thing. In HTML5, this can be resolved by using custom properties embedded in the document.

We can now use our own vocabulary in HTML5 and Microdata is used for that purpose. In Microdata, we use name-value pairs. We will now have a look at the global attributes used in HTML5 to define custom properties:

- **itemscope**: This attribute is used to notify that Microdata is being used in the web page. We create an item by using itemscope.
- **itemtype**: This attribute is a URL to define the item.
- **itemid**: This attribute is used to identify an item.
- **itemprop**: This attribute is used to define a property of an item. Microdata consists of name-value pairs and the value of the properties is defined here.
- **itemref**: This attribute is used to define the context of a web page.

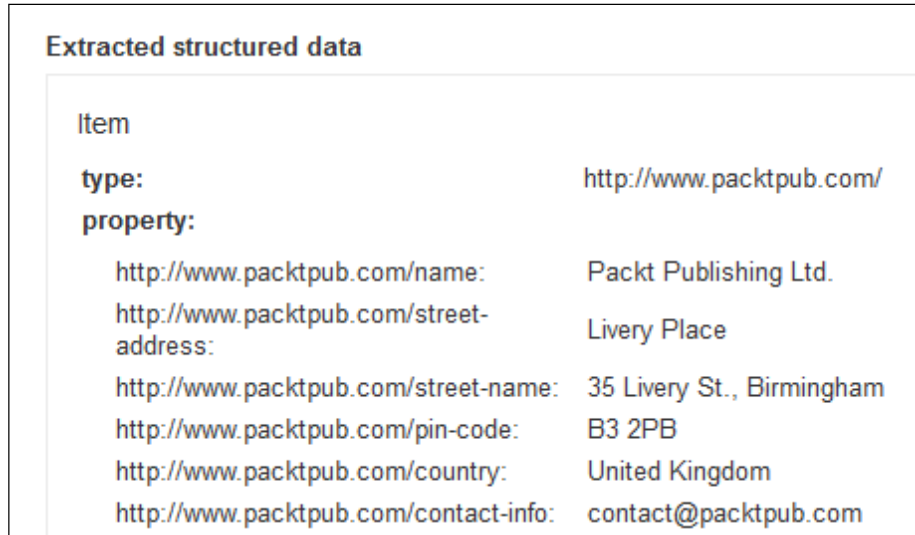
We will now look at a code snippet to see how these global attributes are used.

In the following code snippet, we will use all these global attributes so that we can understand its functionality. We are going to use these attributes within a section element.

```
<html>
  <head>
    <title> Info about Packt </title>
  </head>
  <body>
    <section itemscope itemtype="http://www.packtpub.com">
      <h1 itemprop="name"> Packt Publishing Ltd. </h1>
      <p itemprop="street-address"> Livery Place </p>
      <p itemprop="street-name"> 35 Livery St., Birmingham </p>
      <p itemprop="Pin-code"> B3 2PB </p>
      <p itemprop="country"> United Kingdom </p>
      <p itemprop="contact-info"> contact@packtpub.com </p>
    </section>
  </body>
</html>
```

Now, we will see how important Microdata is in today's world. There is a reason why Microdata must be included in HTML code. Microdata is not a standalone component. As a matter of fact, it enhances the semantic feature of HTML. If you observe the code, you will see that we have included custom properties.

SEO plays an essential role when searching for terms on the internet. Search engines like Google have bots and crawlers, which access specific information on the web page and it is because of this particular reason that some websites are displayed on the first page. In the earlier code, we defined name-value pairs (Microdata). Let's see how this code has an impact by using the Google Structured Data Testing Tool. When we enter the HTML code in the tool, we can see the extracted structured data as shown in the following figure:



The screenshot shows a box titled "Extracted structured data" containing a table of properties and their values for an item.

Extracted structured data	
Item	
type:	http://www.packtpub.com/
property:	
http://www.packtpub.com/name:	Packt Publishing Ltd.
http://www.packtpub.com/street-address:	Livery Place
http://www.packtpub.com/street-name:	35 Livery St., Birmingham
http://www.packtpub.com/pin-code:	B3 2PB
http://www.packtpub.com/country:	United Kingdom
http://www.packtpub.com/contact-info:	contact@packtpub.com

We can see how the Google Structured Data Testing Tool could read your HTML and find that specific data. Hence, Microdata is imperative as the SEO can easily indentify the contents of the web page and read it much more easily.

Summary

We had a look at various elements that are used to build a semantic web page. We have seen how semantics account for cleaner code and how Microdata makes it easy for the SEO to understand your HTML as well as enhance the semantics of our document. In the next chapter, we will look at the new Flexbox model in CSS3.

3

Flexible Box Model in CSS3

The web applications and the web pages that we see today are complex in nature. Prior to **CSS3**, web designers used floats and other CSS properties to design their web pages. However, it is not easy to style complicated web pages using CSS2. Hence, the concept of the **Flexible Box Model** was conceived to make it easier to design web pages keeping in sync with the times.



The Flexible Box Model is also known as **Flexbox**. In this chapter we will be referring to it as Flexbox.

Understanding Flexbox

Various versions of Flexbox have been released and the pattern has changed over the years. The latest revision of Flexbox was released in September 2012. In this chapter we will be working with the latest version of Flexbox. The latest version of Flexbox is supported by Google Chrome (Version 22 and higher) and Opera (Version 12.0 and higher) browsers. Browsers such as Firefox and Internet Explorer have not implemented it yet. We will be using the latest Google Chrome browser to demonstrate the functionality of Flexbox.

Flexbox consists of a Flex Container. It also consists of Flex Items, which are the children of the Flex Container. We will use the `display` property of CSS to explain the functionality. The `display` property is set to `flex` or `inline-flex`. If we use `display: flex;`, the Flex Container functions at the block level whereas if we use `display: inline-flex;`, then it is considered inline within the defined boundaries.

We will look at a simple way to use Flexbox by referring to the following code snippet:

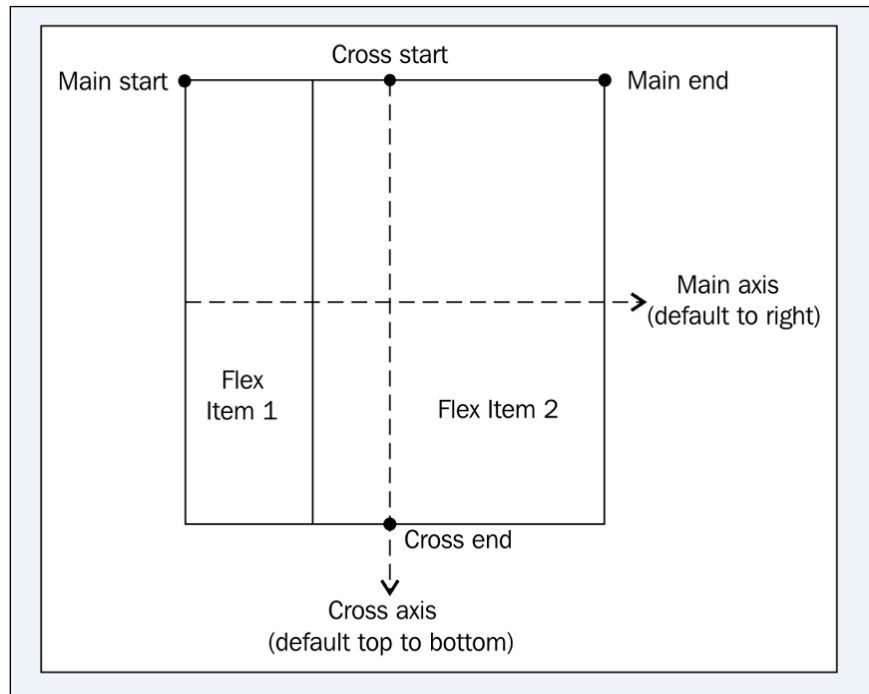
```
.flex-container
{
  display: flex;

  display: -webkit-flex;

}
```

If we look at the code, we can see that `flex` is defined twice. We have added the `-webkit` prefix to `flex` for a specific reason. The new Flexbox model is still not supported by all browsers. The `-webkit` prefix is used as we are using the Chrome browser. If the latest version of Firefox would support it, then a `-moz` prefix will be added to `flex`. We used to add a prefix to `flex` in the earlier versions of the Flexbox model. We will have to use the vendor specific prefix till it becomes a default standard.

Each Flex Container has its own Flex Line. There is a logical **Main axis** and **Cross axis** in Flexbox wherein the Flex Line follows the Main axis. The default direction is always left to right and top to bottom. The following figure depicts the concept so that you will have a better understanding of it:



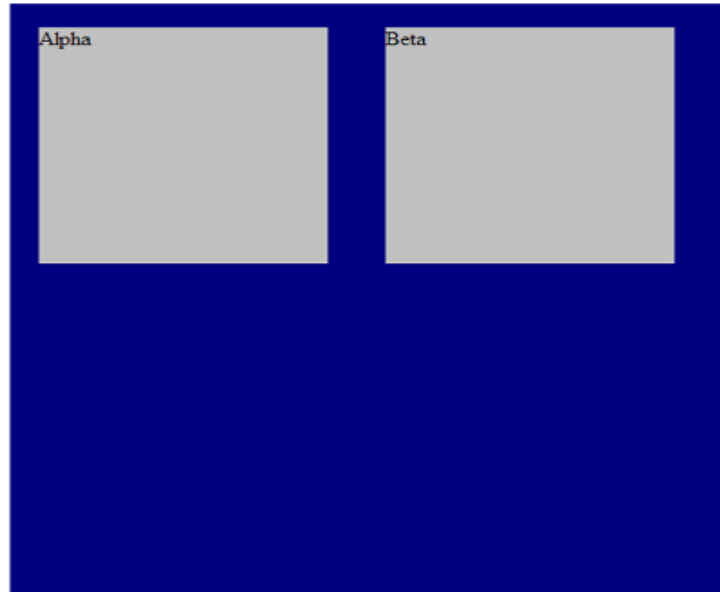
In this chapter, we will use code snippets to understand the functionality of Flexbox. Practically, we must always keep the HTML code and the CSS code separate. However, in the examples mentioned in this chapter, we will incorporate the CSS code into the HTML code using the `<style>` tag to keep it simple.

Let's look at a simple code to understand how Flexbox works.

```
<html>
  <head>
    <style>
      #flex-container {
        display: -webkit-flex;
        display: flex;
        width: 500px;
        height: 500px;
        background-color: Navy;
      }

      #flex-item {
        background-color: Silver;
        width: 200px;
        height: 200px;
        margin: 20px;
      }
    </style>
  </head>
  <body>
    <div id="flex-container">
      <div id="flex-item">Alpha</div>
      <div id="flex-item">Beta</div>
    </div>
  </body>
</html>
```


The output of the code would be as follows:



As we see, the default direction is from left to right. Since the direction is not mentioned, the Flex Items are aligned along the Main axis.

Flex Container properties

We will now look at the various properties of Flex Container.

flex-direction

As we had seen earlier, the default direction is from left to right and the line is along the Main axis. However, the writing mode can be changed using the following values:

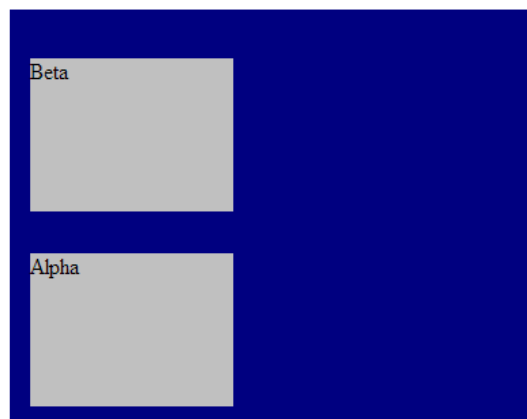
- `flex-direction: row-reverse;` This will swap the Main start and Main end. The direction will be along the Main axis but the direction will be from right to left.
- `flex-direction: column;` This will swap the Main axis and the Cross axis, following which the direction that the Flex Items are set will be vertical in nature.
- `flex-direction: column-reverse;` This will be the same as column but the items will be laid from bottom to top.

We will see an instance of this using the following code snippet:

```
<html>
  <head>
    <style>
      #flex-container {
        display: -webkit-flex;
        display: flex;
        -webkit-flex-direction: column-reverse;
        flex-direction: column-reverse;
        width: 390px;
        height: 300px;
        background-color: Navy;
      }

      #flex-item {
        background-color: Silver;
        width: 150px;
        height: 110px;
        margin: 15px;
      }
    </style>
  </head>
  <body>
    <div id="flex-container">
      <div id="flex-item">Alpha</div>
      <div id="flex-item">Beta</div>
    </div>
  </body>
</html>
```

The output of the code will be as follows:



We can see that the Main axis and the Cross axis have been swapped and the reverse value has changed the mode from bottom to top.

justify-content

We can change the position of Flex Items along the Main axis using this property.

The different values for this property are as follows:

- `justify-content: center;` This will position the Flex Items at the center along the Main axis.

The other values that can be assigned are:

- `justify-content: flex-start;`
- `justify-content: flex-end;`
- `justify-content: space-between;`
- `justify-content: space-around;`

`space-between` and `space-around` deal with the whitespace whereas the `flex-start` and `flex-end` will change the position from the start or the end depending on the direction.

Let's look at the following code snippet where we have used the `justify-content: flex-end;` value:

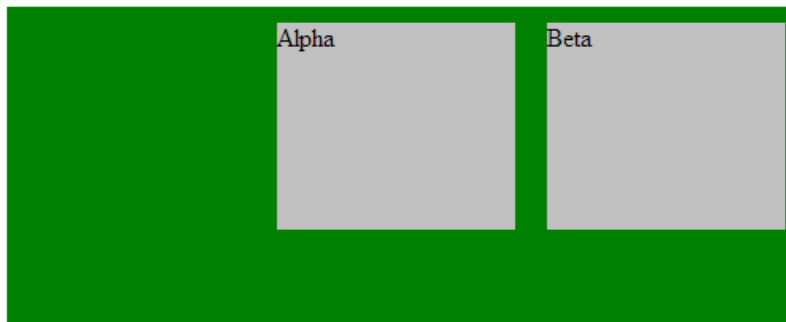
```
<html>
  <head>
    <style>
      #flex-container {
        display: -webkit-flex;
        display: flex;
        -webkit-justify-content: flex-end;
        justify-content: flex-end;

        width: 500px;
        height: 200px;
        background-color: Green;
      }

      #flex-item {
        background-color: Silver;
        width: 150px;
        height: 130px;
      }
    </style>
  </head>
</html>
```

```
        margin: 10px;
    }
</style>
</head>
<body>
  <div id="flex-container">
    <div id="flex-item">Alpha</div>
    <div id="flex-item">Beta</div>
  </div>
</body>
</html>
```

The output of the code will be as follows:



We can see that the position of the items has been inclined to the right as a result of the `flex-end` value.

align-items

While `justify-content` aligned the items along the Main axis, `align-items` will align the items along the Cross axis. This property has the following values:

- `align-items: center;`
- `align-items: flex-start;`
- `align-items: flex-end;`
- `align-items: baseline;`
- `align-items: stretch;`

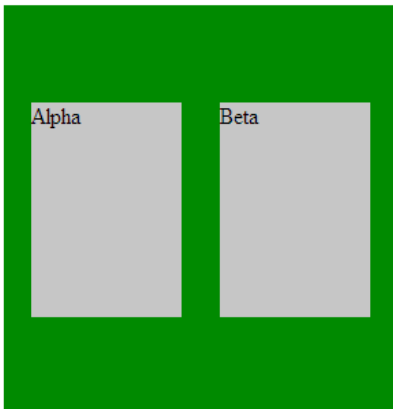
The `align-items` property will align the items along the center, the start, or the end of the Cross axis as defined whereas the `baseline` value will align it along the baseline while the `stretch` value as the name suggests will stretch the items from the start of the Cross axis to the end of it.

Let's look at an instance of `align-items` by referring to the following code snippet:

```
<html>
  <head>
    <style>
      #flex-container {
        display: -webkit-flex;
        display: flex;
        -webkit-align-items: center;
        align-items: center;
        width: 330px;
        height: 370px;
        background-color: Green;
      }

      #flex-item {
        background-color: Silver;
        width: 110px;
        height: 150px;
        margin: 25px;
      }
    </style>
  </head>
  <body>
    <div id="flex-container">
      <div id="flex-item">Alpha</div>
      <div id="flex-item">Beta</div>
    </div>
  </body>
</html>
```

The output of the code will be as follows:



As we can see, the items are positioned on the center of the Cross axis.

flex-wrap

At times, there are too many Flex Items and they will not fit in on a single Flex Line. Here the `wrap` feature comes into play. Using this property, additional Flex Lines are created on the Cross axis to accommodate these Flex Items.

Let's look at the following code sample to see how it works:

```
<html>
  <head>
    <style>
      #flex-container {
        display: -webkit-flex;
        display: flex;
        -webkit-flex-wrap: wrap;
        flex-wrap: wrap;
        width: 300px;
        height: 240px;
        background-color: Green;
      }

      #flex-item {
        background-color: Silver;
        width: 100px;
        height: 70px;
        margin: 10px;
      }
    </style>
  </head>
  <body>
    <div id="flex-container">
      <div id="flex-item">Alpha</div>
      <div id="flex-item">Beta</div>
      <div id="flex-item">Gamma</div>
    </div>
  </body>
</html>
```

The output of the code will be as follows:



As we can see, the **Gamma** item is accommodated on a different Flex Line due to the wrap feature.

The wrap-reverse property is a contrast of the wrap property. Let's look at the following code to understand the difference:

```
<html>
  <head>
    <style>
      #flex-container {
        display: -webkit-flex;
        display: flex;
        -webkit-flex-wrap: wrap-reverse;
        flex-wrap: wrap-reverse;
        width: 300px;
        height: 240px;
        background-color: Green;
      }

      #flex-item {
        background-color: Silver;
        width: 100px;
        height: 70px;
        margin: 10px;
      }
    </style>
  </head>
  <body>
    <div id="flex-container">
      <div id="flex-item">Alpha</div>
```

```
<div id="flex-item">Beta</div>
<div id="flex-item">Gamma</div>
</div>
</body>
</html>
```

The output of the code will be as follows:



As we can see from the output, it is the exact opposite of wrap.

So far, we have covered the different properties of the Flex Container. We will now discuss the properties of Flex Items that are commonly used in design.

Properties of Flex Items

We will now look at *order*, which is an excellent feature of CSS3 Flexbox.

Order

This feature displays the items in a particular hierarchy determined by the *order* value assigned to them. For example, if we assign a value of *-1* to a Flex Item, it will be displayed before any other Flex Item irrespective of the order in the document.

Let's look at the following code example to see how it works:

```
<html>
  <head>
    <style>
      .flex-container {
        display: -webkit-flex;
        display: flex;
      }
    </style>
  </head>
  <body>
    <div class="flex-container">
      <div class="flex-item">Beta</div>
      <div class="flex-item">Gamma</div>
    </div>
  </body>
</html>
```



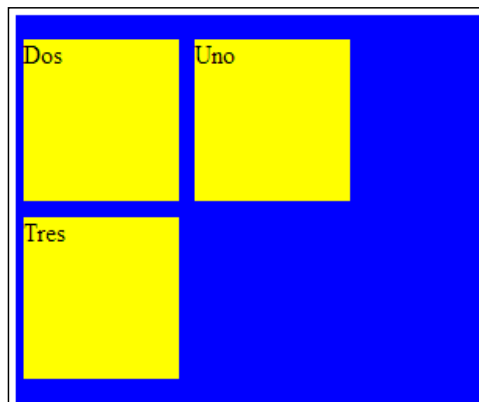
```
        -webkit-flex-wrap: wrap;
        flex-wrap: wrap;
        -webkit-align-content: center;
        align-content: center;
        width: 300px;
        height: 240px;
        background-color: Blue;
    }

    .flex-item {
        background-color: Yellow;
        width: 100px;
        height: 100px;
        margin: 5px;
    }

    .point{
        -webkit-order: -1;
        order: -1;
    }
</style>
</head>
<body>
    <div class="flex-container">
        <div class="flex-item"> Uno </div>

        <div class="flex-item point"> Dos </div>
        <div class="flex-item"> Tres </div>
    </div>
</body>
</html>
```

The output of the code will be as follows:



If we see the output, the Flex item **Dos** was defined after **Uno** in the code. However, we assigned an order value of `-1` to **Dos**, therefore **Dos** was displayed first.

We will now look at how flex is applied in Flexbox.

Flex

The `flex` property is an imperative part of the CSS3 Flexbox model.

Let's look at an example to understand the concept. Suppose we have three Flex Items: **Box One**, **Box Two**, and **Box Three**. The space on the Main axis has to be distributed among the three items. Suppose we want the items to fill up the space in the ratio 1:3:5. How do we do it? Here is where the `flex` property comes into play.

Let's look at the following code example to see how it works:

```
<html>
  <head>
    <title>
      The " flex " property
    </title>
    <style>

      .flex-container {
        display: -webkit-flex;
        display: flex;
        width: 700px;
        height: 300px;
        background-color: DeepSkyBlue;
      }

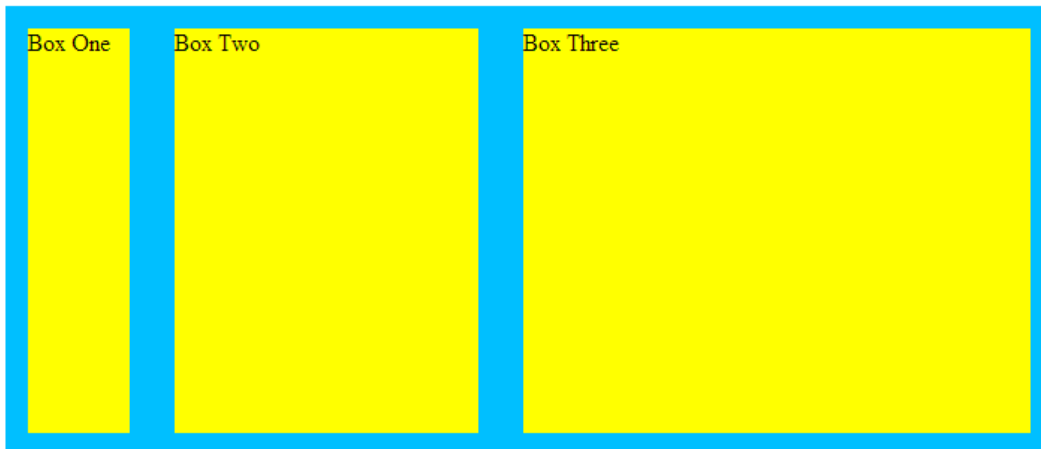
      .flex-item {
        background-color: Yellow;
        margin: 15px;
      }

      .first_item {
        -webkit-flex: 1;
        flex: 1;
      }

      .second_item {
        -webkit-flex: 3;
        flex: 3;
      }
    </style>
  </head>
  <body>
    <div class="flex-container">
      <div class="flex-item first_item">
        Uno
      </div>
      <div class="flex-item second_item">
        Dos
      </div>
      <div class="flex-item">
        Tres
      </div>
    </div>
  </body>
</html>
```

```
.third_item {
  -webkit-flex: 5;
  flex: 5;
}
</style>
</head>
<body>
  <div class="flex-container">
    <div class="flex-item first_item"> Box One </div>
    <div class="flex-item second_item"> Box Two </div>
    <div class="flex-item third_item"> Box Three </div>
  </div>
</body>
</html>
```

The output of the code will be as follows:



We can see that the space on the Main axis has been distributed in the ratio 1:3:5.

Summary

We have covered the imperative aspects of the CSS3 Flexbox model. However, browsers like Firefox and Internet Explorer have not implemented it as yet. It is now known that this is a standard model and will be implemented in the future releases.

In this Chapter we have explained the functionality and use of the Flexbox model. We have seen the Flexbox Container properties as well as properties of Flex Items like order and flex. CSS3 Flexbox has come a long way and thereby made designing easier. In the next chapter, we will look at the **Form** elements used in **HTML5**.

4

Web Forms in HTML5

Prior to HTML5, developers and web designers had to write a lot of code for basic things such as a datepicker. JavaScript was used extensively for basic things and it was difficult to write such code given the complexity of the web pages we come across nowadays.

HTML5 accounts for cleaner code and the introduction of new input types makes it easier for developers to design their web pages. In this chapter, we will discuss the new input types and attributes in HTML5.

If the input types and attributes are not compatible with earlier versions of the browsers, they will be ignored and the `<input type= text>` default will be considered.



Currently, some of the features of HTML5 are supported by the latest versions of Opera and Google Chrome. However, certain features of HTML5 are not supported by these browsers. Firefox support is still in the pipeline. After HTML5 is accepted as a norm, these features will be standardized and will be supported by all browsers.

The URL <http://www.wufoo.com/html5/> displays the browser support for the forms and attributes in HTML5. This page displays the new features and indicates which browsers are supporting which features and to what degree. As a whole, this represents the current state of HTML5 forms.

New form attributes in HTML5

We will first look at the new attributes in HTML5. The most commonly used attributes are described as follows:

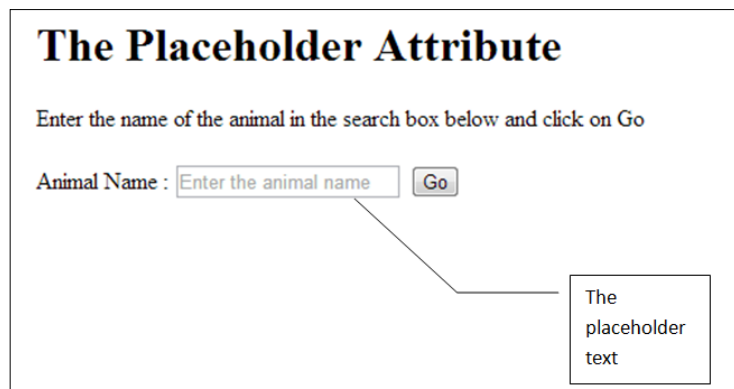
placeholder

A `placeholder` attribute tells the user what data needs to be entered in the form of text in a lighter shade. After the user enters characters in the text field, the text in the lighter shade disappears.

Let's look at the following code to understand it better:

```
<!DOCTYPE html>
<html>
  <head>
    <title>
      The Placeholder Attribute
    </title>
    <h1> The Placeholder Attribute </h1>
  </head>
  <div>
    Enter the name of the animal in the search box below and click
      on Go
    <br><br>
    <label for="animal">Animal Name : </label>
    <input id="animal" placeholder="Enter the animal name">
    <input type="submit" value="Go">
  </div>
</html>
```

In the following screenshot, the text box containing the lighter shade text demonstrates the `placeholder` effect:



We can see the text **Enter the animal name** in a lighter shade. After we click on the textbox, the text disappears. Hence, it is just a way to let the user know what input is expected.

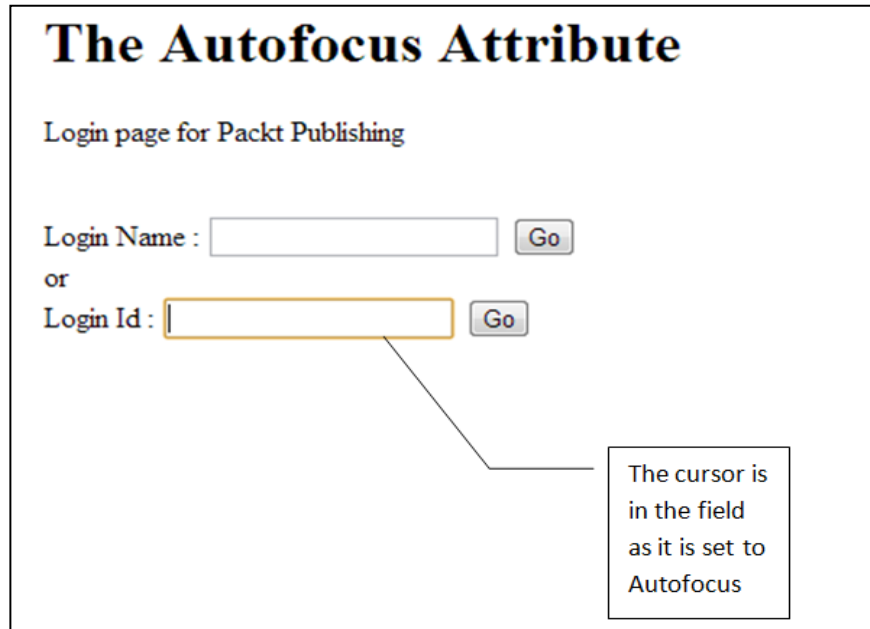
autofocus

With this attribute, the browser sets the focus on the field where autofocus is defined when the page is being loaded.

Let's look at the following code to understand it better:

```
<!DOCTYPE html>
<html>
  <head>
    <title> Autofocus attribute </title>
    <h1>The Autofocus Attribute</h1>
  </head>
  <p> Login page for Packt Publishing</p>
  <div>
    <br>
    <label for="name">Login Name :</label>
    <input id ="name" type="text">
    <input type="submit" value="Go">
    <br>
    or
    <br>
    <label for="loginid">Login Id :</label>
    <input id ="loginid" type="text" autofocus>
    <input type="submit" value="Go">
  </div>
</html>
```

In the following screenshot, the **Login Id** field is set to `autofocus`, as a result of which it has a cursor in it when the page loads:



Hence, the user doesn't have to click on a field when the page loads up as it is set to `autofocus`.

required

The `required` attribute makes it mandatory for the user to enter a value in the field. If the user doesn't enter any data in the field with the `required` attribute, then it will prompt the user accordingly.

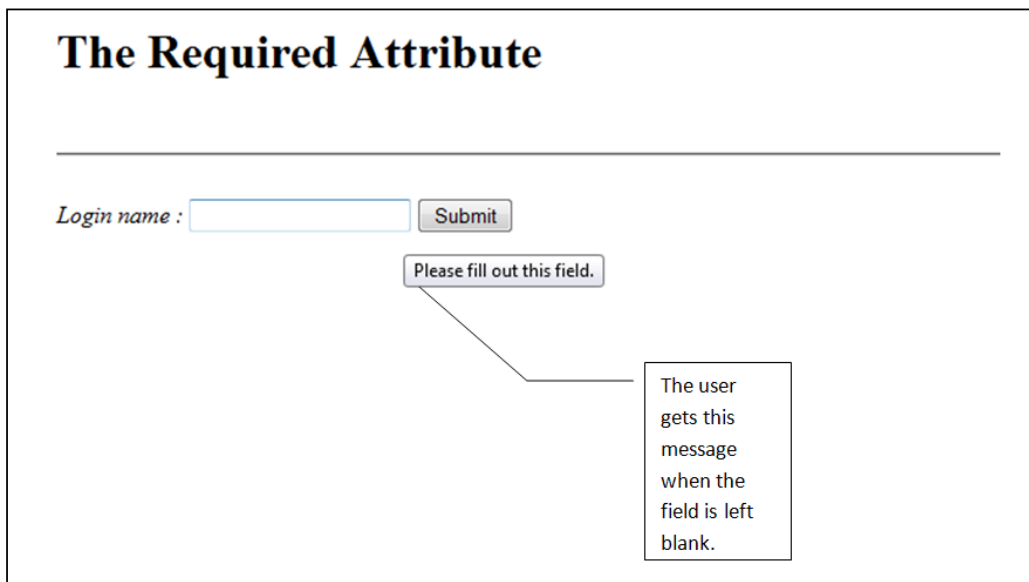
The output is different in Opera and Firefox, hence we will display screenshots for both the cases.

Let's look at the following code to understand it better:

```
<!DOCTYPE html>
<html>
  <head>
    <title> Required attribute </title>
    <h1> The Required Attribute </h1>
    <br>
  </head>
```

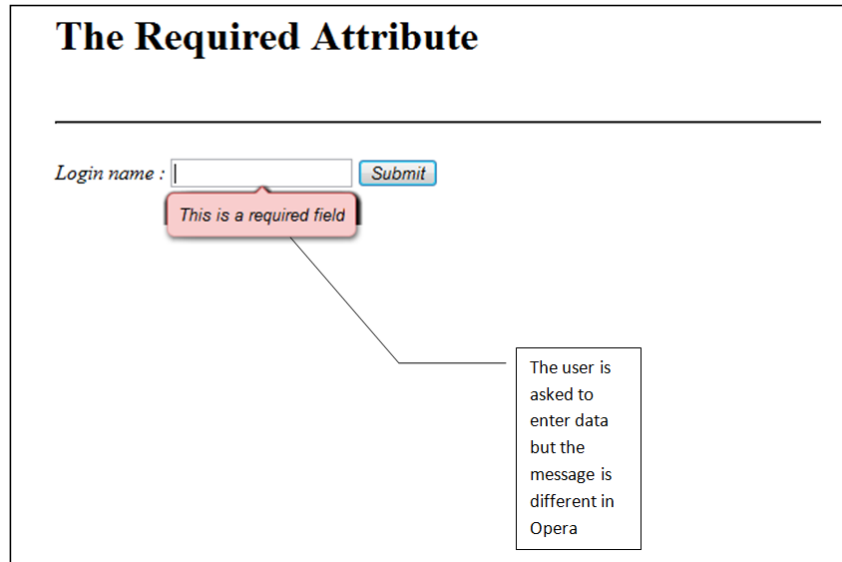
```
<hr>
<br>
<form>
  <label for="loginid"><em>Login name</em> : </label>
  <input name="loginid" type="text" required>
  <input type="submit" value="Submit">
</form>
</html>
```

Let's refer to the following screenshot displaying the code output where the user will be prompted to enter data as the specific field should not be left blank:



We have seen how the output looks in a Firefox browser.

We will now see the output when we execute the code using the Opera browser. Please refer to the following screenshot, which depicts the output in Opera:



Hence, it is mandatory to enter data in the field that has the `required` attribute assigned to it.

datalist

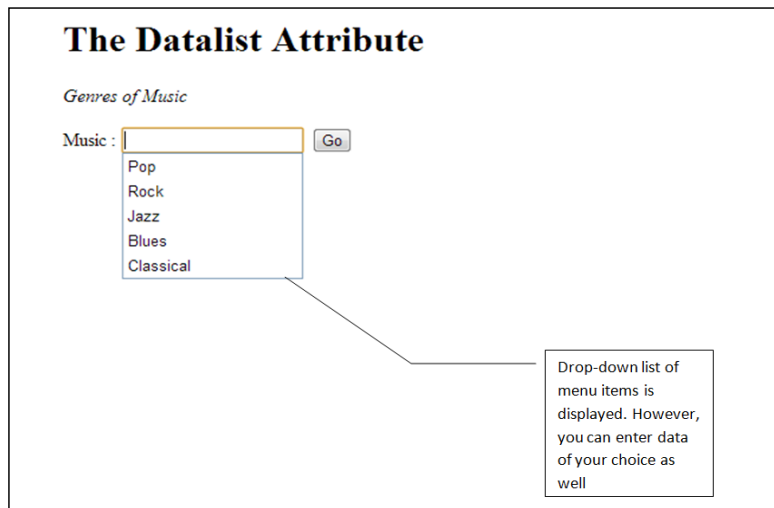
The `datalist` attribute is used to create a list of pre-defined items as a drop-down menu. On using this tag, the pre-defined menu items get displayed when we click on the textbox.

Let's look at the following code to understand it better:

```
<!DOCTYPE html>
<html>
  <head>
    <title>Datalist attribute</title>
    <h1> The Datalist Attribute </h1>
  </head>
  <p> <em> Genres of Music </em> </p>
  <div>
    <label for="packt"> Music : </label>
    <input id="packt" name="packt" type="text" list="music">
    <input type="submit" value ="Go">
    <datalist id="music">
      <option value="Pop">
```

```
<option value="Rock">
<option value="Jazz">
<option value="Blues">
<option value="Classical">
</datalist>
</div>
</html>
```

Let's look at the output of the code where we can see the drop-down list:



For example, the drop-down menu displays all the genres of music when we click in the textbox. However, we are not just limited to the menu items declared in the code. We can enter any data that is not a part of the list declared in the code.

Understanding new input types in HTML5

We will look at the new input types in HTML5. The following input types mentioned are explained along with their respective code.

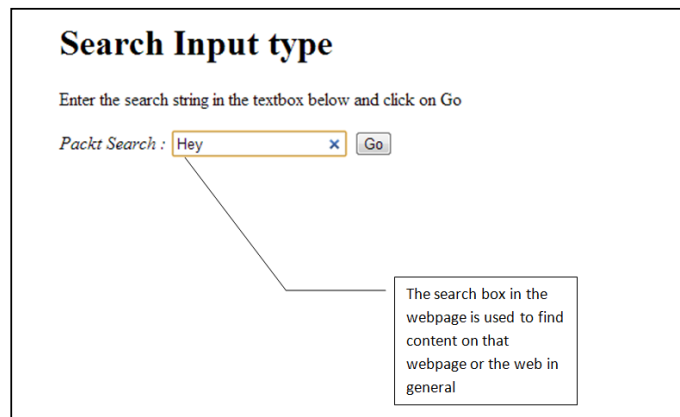
search

After the success of Google as a search engine, we can see a search box facility almost everywhere on the web. The search box is used on almost all the websites to find information on that site or the web in general. In HTML5, we can create a search box using the input type as `search`. The search box looks like a normal textbox but is used for search purposes.

Let's look at the following code to understand it better:

```
<!DOCTYPE html>
<html>
  <head>
    <title> Search Input type </title>
    <h1> Search Input type </h1>
  </head>
  <p> Enter the search string in the textbox below and click on Go
  </p>
  <div>
    <label for="searchid"> <em> Packt Search : </em> </label>
    <input id="searchid" type="search" placeholder="Enter Search
      Item here">
    <input type="submit" value="Go">
  </div>
</html>
```

The following screenshot displays the output of the code:



In the screenshot, we have entered **Hey** as the search string. If we click on the blue cross at the right hand side of the search box, the string will disappear and we can enter a new search string in its place.

The search box is a crucial aspect of any website as it assists the user in finding content on the web or that web page depending on the way it is developed.

email and url

The `email` and `url` input types are the latest additions made specifically for the purpose of e-mail and web addresses. The e-mail address has to be written with the domain name between an @ sign and a dot (.), for example, `xyz@abc.com`. It accepts the current standard format in which e-mail addresses are written.

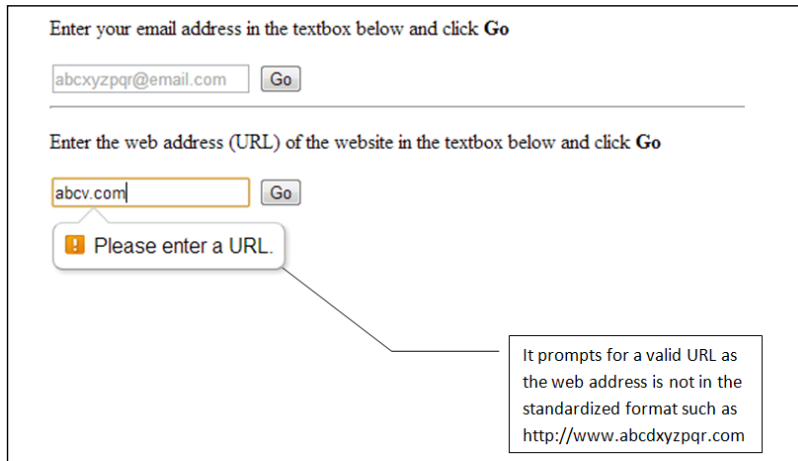
If we do not write the e-mail address in a proper format, then it prompts the user to enter a valid e-mail address.

Most of the URLs are in the format `http://www.xyzzzz.com`. The URL has to be entered in this standard format along with the `http` or `https` protocols. If we do not follow the standardized format, then it prompts the user to enter a valid URL.

Let's look at the following code to understand it better:

```
<!DOCTYPE html>
<html>
  <head>
    <title>Input types: Email and URL </title>
  </head>
  <body>
    <form>
      <p> Enter your email address in the textbox below and click
        <strong>Go</strong> </p>
      <input type="email" placeholder="abcxyzpqr@email.com">
      <input type="submit" value="Go">
      <br>
      <hr>
      <p> Enter the web address (URL) of the website in the
        textbox below and click <strong>Go </strong> </p>
      <input id="website" type="url">
      <input type="submit" value="Go">
    </form>
  </body>
</html>
```

The following screenshot displays the output of the code:



In the example, the URL entered is not in a standard format. Hence, it prompts for a standard URL. If we enter an invalid e-mail address, it would prompt for a valid e-mail address.

Hence, the structure of e-mail addresses and URLs have been defined to make it easy for the web designer to write cleaner code in a more systematic manner.

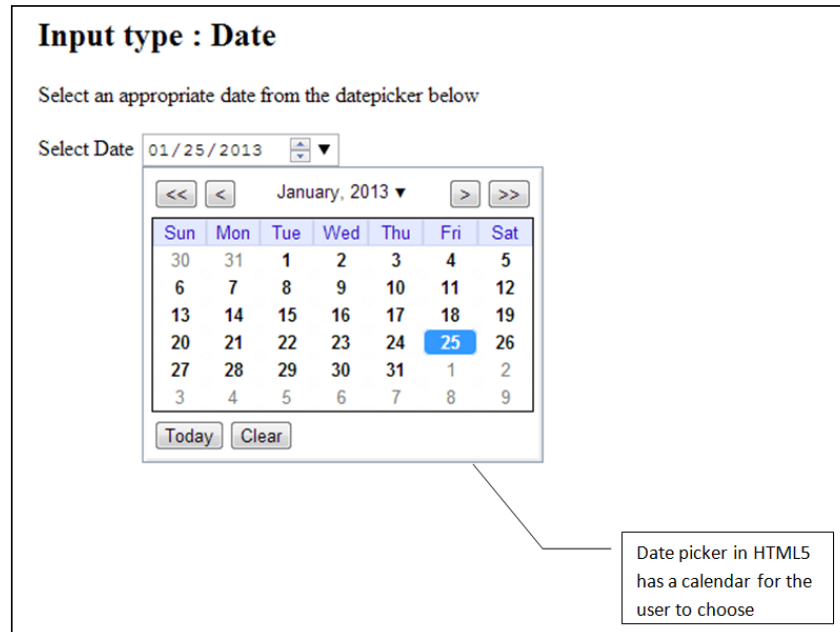
date

Prior to the date feature in HTML5, we had to use JavaScript to develop a datepicker. However, with the advent of HTML5, it is much easier to create one. HTML5 has a lot of date options like datetime, week, time, and month other than the date feature. We will learn more about it in this chapter.

Let's look at the following code to understand it better:

```
<!DOCTYPE html>
<html>
  <head>
    <title> Date Input type </title>
    <h2> Input type : Date </h2>
    <p> Select an appropriate date from the datepicker below </p>
  </head>
  <body>
    <label for="packt"> Select Date </label>
    <input id="packt" type="date" value="2013-01-25">
  </body>
</html>
```

The following screenshot displays the output of the code:



Hence, we do not have to invoke any JavaScript for creating a datepicker. We will now look at features like week, month, and time which are similar to the date feature.

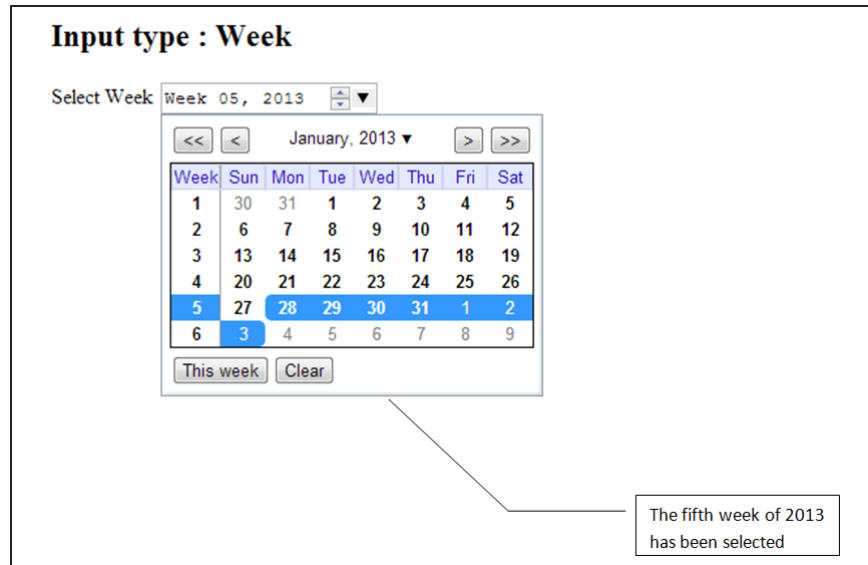
week

We can enter a week and click on the dropdown to see the dates in the week for a specific year.

Let's look at the following code to understand it better:

```
<!DOCTYPE html>
<html>
  <head>
    <title> Week </title>
    <h2> Input type : Week </h2>
  </head>
  <body>
    <label for="packt"> Select Week </label>
    <input id="packt" type="week">
  </body>
</html>
```

The following screenshot displays the output of the code:



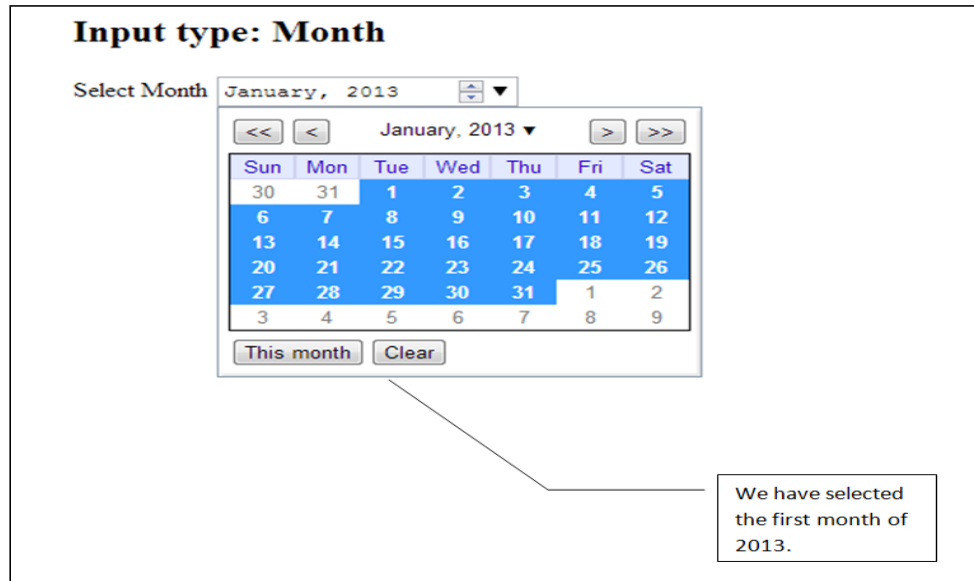
month

We can view and select a month for a particular year.

Let's look at the following code to understand it better:

```
<!DOCTYPE html>
<html>
  <head>
    <title> Month </title>
    <h2> Input type: Month </h2>
  </head>
  <body>
    <label for="packt"> Select Month </label>
    <input id="packt" type="month">
  </body>
</html>
```

The following screenshot displays the output of the code:

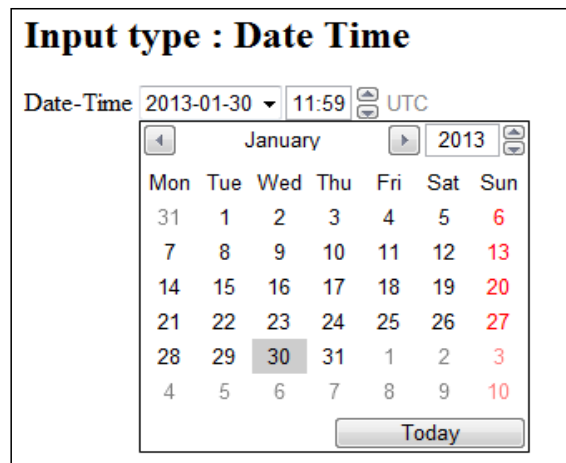


Similarly we can select the date and time using the `datetime` attribute.

We need to use the following code snippet for the same:

```
<label for="packt"> Date-Time </label>
<input id="packt" type="datetime">
```

The following screenshot displays the output of the code:



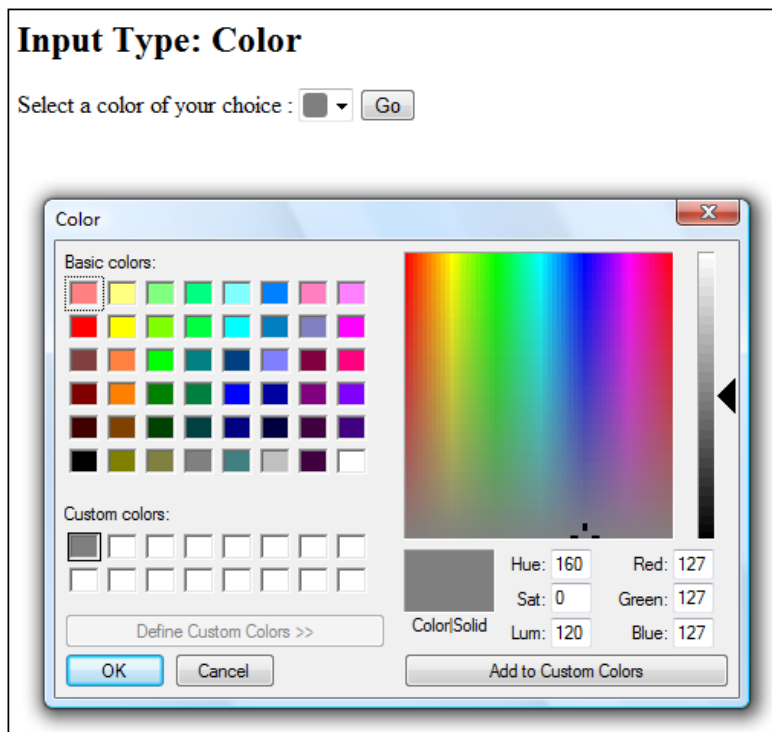
color

Earlier, we had to write a complex JavaScript code for creating a color palette. We can now create a color palette using HTML5.

Let's look at the following code to understand it better:

```
<!DOCTYPE html>
<html>
  <head>
    <title> Color input type in HTML5 </title>
    <h2> Input Type: Color </h2>
  </head>
  <div>
    <label for="color-ref"> Select a color of your choice :
    </label>
    <input id="color-ref" type="color">
    <input type="submit" value="Go">
  </div>
</html>
```

The following screenshot displays the output of the code:





Various browsers support different features. For example, the color attribute is supported by only the Opera browser at the time of writing. However, the standardization of HTML5 in the future will ensure that all other browsers support these various features.

Summary

In this chapter, we have covered the new form input types and form attributes of HTML5. In the next chapter, we will look at advanced features of HTML5 such as audio, video, and canvas to mention a few, which will provide an insight into the beauty of HTML5.

5

Advanced Features of HTML5

HTML5, along with the new JavaScript APIs, enables us to create web pages with a lot of features that were only possible with proprietary plugins, such as Flash in the past. With the introduction of HTML5, we can develop web pages with enhanced features and we will be discussing that in this chapter.

We will look at the following features of HTML5, which work with JavaScript APIs:

- Audio and video
- Drag-and-drop
- Geolocation
- Webstorage
- Offline web applications
- Canvas

Initially, we will look at the audio and video features.

Audio and video

HTML5 supports audio and video elements. Prior to HTML5, we used plugins, such as Flash player, extensively. The audio and video elements reduce the dependence on plugins to a large extent by allowing the developer to embed media elements into HTML documents.

Let's look at the following code to understand how it works:

```
<!DOCTYPE html>
<html>
  <title> HTML5 Audio player </title>
  <body>
    <audio autoplay = "autoplay" controls =
      "controls" id = "player">
      <source src = "aravind.ogg" />
      <source src = "aravind.mp3" />
      <p> Your browser doesn't support the audio tag </p>
    </audio>
    <div>
      <button onclick = "document.getElementById
        ('player').play()">Play</button>
      <button onclick = "document.getElementById('player')
        .pause()">Pause</button>
      <button onclick = "document.getElementById('player').
        volume+ = 0.1">Vol+ </button>
      <button onclick = "document.getElementById('player').
        volume- = 0.1">Vol- </button>
    </div>
  </body>
</html>
```



The output of the code will vary depending upon the browser in use.

The output of the code in Mozilla Firefox, Internet Explorer, Opera, and Google Chrome respectively is shown in the following screenshot:



If you observe the preceding code, we have included the OGG as well as the MP3 format for the audio track. The reason to include both formats is that the MP3 format is not open source and is patented. Hence, all the browsers do not support the MP3 format at the time of writing this chapter. However, some of the browsers do support the open source OGG file format.

The `controls` attribute allows the user to control the audio player. For example, we can play and pause the player, as and when required. The volume can be adjusted as per the discretion of the user.

The `autoplay` attribute will play the song automatically without the need to click on the Play icon.

The `source` `src` attribute is used for the location of the audio track.

As we see, it is easy to create an audio player with minimalistic code using HTML5.

In the audio code, we have used the following snippet:

```
<div>
  <button onclick =
    "document.getElementById('player').play()">Play</button>
  <button onclick =
    "document.getElementById('player').pause()">Pause</button>
  <button onclick = "document.getElementById('player').
    volume+ = 0.1">Vol+ </button>
  <button onclick = "document.getElementById('player').
    volume- = 0.1">Vol- </button>
</div>
```

We have used the preceding code in the `div` element to modify the player and add manual controls that are not inbuilt with the audio player and thereby, we can see the **Play**, **Pause**, **Vol+**, and **Vol-** buttons.



Remember while testing the code, the audio track must be in the same folder as the HTML file, or else you will have to give the complete URL or the complete location of the audio file.

Let's have a look at the procedure to develop a video player using HTML5.

The video player code is very similar to the audio player code. Let's look at the following code to understand how it works:

```
<!DOCTYPE html>
<html>
  <head>
    <title> Video Player in HTML5 </title>
    <style>
      video {
        box-shadow:0 0 15px #333;
        border-radius:10px;
      }
    </style>
  </head>
  <body>
    <video width = "700" height = "400" id = "packt"
      controls = "controls" autoplay = "autoplay">
      <source src = "Packt_Sample.ogv"/>
      <source src = "Packt_Sample.mp4"/>
```

```
</video>
<div>
  <button onclick =
    "document.getElementById('packt').play()">Play</button>
  <button onclick =
    "document.getElementById('packt').pause()">Pause</button>
  <button onclick = "document.getElementById('packt').
    volume+ = 0.1">Vol+ </button>
  <button onclick = "document.getElementById('packt').
    volume- = 0.1">Vol- </button>
</div>
</body>
</html>
```

We have some attributes that are different from the audio element. Apart from controls and autoplay, we have the poster attribute. The poster attribute helps us to link the video to a customized thumbnail image. In Firefox, the video player will be displayed as shown in the following screenshot:



The look may vary depending on the browser in use. The video player looks different in Chrome and Opera. Let's look at the other features of HTML5.

Drag-and-Drop

The ability to drag-and-drop elements; thereby, aiding in transfer of data, was done by complex JavaScript code prior to HTML5. However, with the advent of the drag-and-drop in HTML5, it has become much easier for the browsers to adopt this new feature.

Let's look at the following code to see how it works:

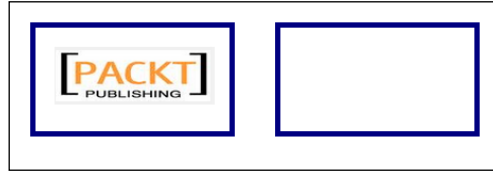
```
<!DOCTYPE HTML>
<html>
  <head>
    <style type = "text/css">
      #packt1, #packt2
      {float:left; width:85px; height:35px;

        margin:11px;padding:11px;border:3px solid navy;}
    </style>
    <script>
      function dropItem(ev) {
        ev.preventDefault();
      }

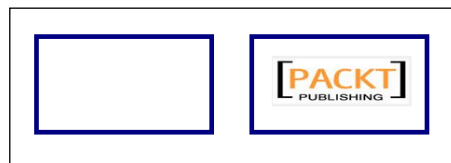
      function drag(ev) {
        ev.dataTransfer.setData("Text",ev.target.id);
      }

      function drop(ev) {
        ev.preventDefault();
        var abc = ev.dataTransfer.getData("Text");
        ev.target.appendChild(document.getElementById(abc));
      }
    </script>
  </head>
  <body>
    <div id = "packt1" ondrop = "drop(event)"
      ondragover = "dropItem(event)">
    <img src = "packt.png" draggable = "true"
      ondragstart = "drag(event)" id = "drag1" width = "75"
      height = "39"></div>
    <div id = "packt2" ondrop = "drop(event)"
      ondragover = "dropItem(event)"></div>
  </body>
</html>
```

The output of the code will be as shown in the following screenshot:



If we place the mouse on the **PACKT** logo and drag it to the adjoining box, the transfer of data will take place; thereby, displaying the following screenshot:



The **PACKT** logo can be moved back and forth between the two boxes by using the drag-and-drop feature.

Let's look at the code explanation to understand how it all works.

The `draggable` when set to the `true` feature enables the item to be dragged from its current state.

The value and type of the data to be dragged is set by the `dataTransfer.setData` method.

By default, the dragged item cannot be dropped elsewhere. Hence, we use the `event.preventDefault()` feature, as it allows the item to be dropped into another element.

The `dataTransfer.getData` method will enable us to return any data that was defined in the `dataTransfer.setData` method. Then, we append the dragged item to its destination.

Now, that you understand how the code works, let's look at the various attributes that can be used in the Drag and Drop API.

To understand the attributes better, we will look at the events being fired while the data is being dragged, and when the data is being dropped.

While the data item is being dragged, the following attributes come into picture:

- `dragstart`
- `drag`
- `dragend`

Let's look at these attributes and understand their functionality.

The `dragstart` event is fired when we place the mouse over a data item, and then try to drag the item. We have used the `ondragstart` event handler to demonstrate it. The `drag` event gets fired after that as the item is in the process of being dragged. Finally, when the item is to be dropped, the `dragend` event gets fired.

Let's look at the following events being fired when the item is to be dropped:

- `dragenter`
- `dragover`
- `dragleave`
- `drop`

When the dragged item is in the boundaries of the drop target, the `dragenter` event is fired. Immediately, the `dragover` event is fired as soon as the `dragenter` event is fired. We need to understand that these events happen when the data item is within the boundaries of the drop target. However, as soon as we drag the item outside the boundaries of the target, the `dragleave` event is fired. If we decide to drop the data item in the drop target, then the `drop` event is fired instead of the `dragleave` event.

Now that we have understood how the events are fired and the intricacies of the procedure, we will now look at why the drag-and-drop API is so useful.

The drag-and-drop feature not only assists in moving an object, but also assists in transfer of data within various applications and different frames. All the latest versions of Firefox, Google Chrome, and Opera support this feature. Internet Explorer 10 (IE10) supports this feature completely.

Let's now look at the Geolocation feature of HTML5.

Geolocation

Geolocation enables us to track down our physical presence at any place. The information assists us in letting people know about our location. The user can decide whether the information regarding the location may be shared or not as he will be prompted accordingly. All the latest versions of the browsers support this feature (including IE9).

Let's look at the following code to understand Geolocation better:

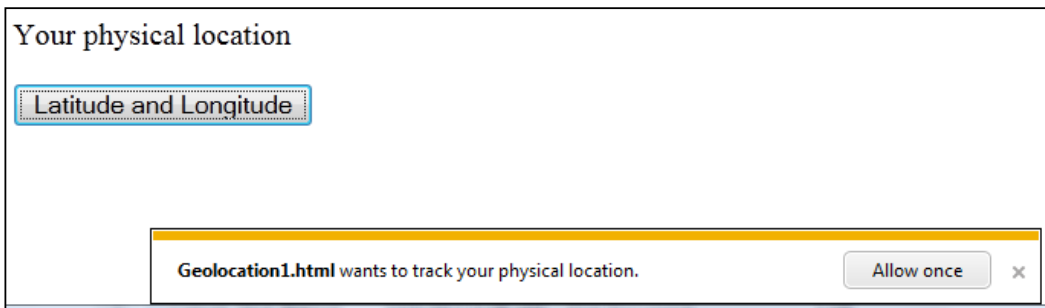
```
<!DOCTYPE html>
<html>
  <body>
    <p id = "packt">Your physical location</p>
    <button onclick = "getLocation()">Latitude and
      Longitude</button>
    <script>
      var x = document.getElementById("packt");
      function getLocation() {
        if (navigator.geolocation)
        {
          navigator.geolocation.getCurrentPosition(
            showLocation,displayError);
        }
        else {x.innerHTML = "Your browser does not support the
          Geolocation feature";}
      }
      function showLocation(position) {
        x.innerHTML = "Latitude: " + position.coords.latitude +
          "<br>Longitude: " + position.coords.longitude;
      }
      function displayError(error) {
        switch(error.code) {
          case error.PERMISSION_DENIED:
            x.innerHTML = "Permission issues, Access Denied."
            break;
          case error.POSITION_UNAVAILABLE:
            x.innerHTML = "As of now, Location
              info is not available."
            break;
          case error.TIMEOUT:
            x.innerHTML = "There seems to be a timeout issue,
              Please try later."
            break;
          case error.UNKNOWN_ERROR:
            x.innerHTML = "Error cause not found."
            break;
        }
      }
    </script>
  </body>
</html>
```

If we observe the preceding code, we can see that the `position.coords.latitude` and the `position.coords.longitude` help us in retrieving information about the coordinates of our physical location with respect to the latitude and longitude of that place.

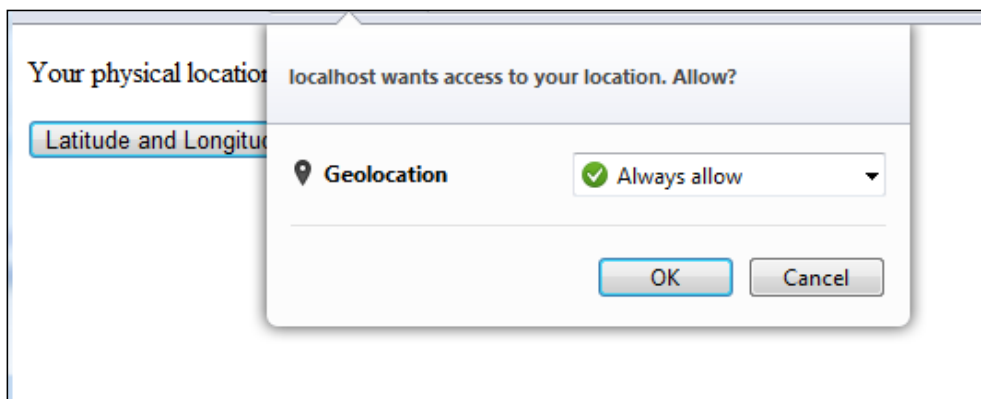
The `navigator.geolocation` function will help us determine whether the browser supports this feature.

We have also used the `switch` command to define the various errors that the user might come across due to some reason or the other.

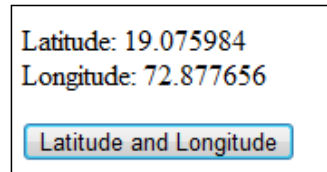
The output of the code will vary depending on the browser. Let's look at the way the user is prompted in IE with regards to the permission required to share his physical location, as shown in the following screenshot:



The message prompted to the user in Opera will be slightly different. In Opera, we will see the following screenshot:



Once the user grants the permission and clicks on the **Latitude and Longitude** button, the following screenshot would be displayed:



There is also a `watchPosition` method, where we can track the location whenever the user moves to a different location.

The geographical position of a user can be determined by this feature. Let's have a glance at the **Webstorage** feature of HTML5.

Webstorage

Storing information whenever we visit a website is known due to the concept of cookies. However, cookies are limited in size. The concept of cookies is that the browser sends these cookies to the web server every time we visit the web page.

However, HTML5 storage concept is more enhanced. The websites access this page with the aid of JavaScript whenever we visit the web page. The web page can only access the information stored by it and doesn't have access to any other data other than the stored information.

There are two types of storage methods used in HTML5 as follows:

- `sessionStorage`
- `localStorage`

sessionStorage

The `sessionStorage` property is used to store data for any specific session. Once we close the browser, the session expires, and data will not be stored outside of that session. The amount of data that can be stored can be large in size unlike cookies that work under limitations. The data is stored in key/value pairs.

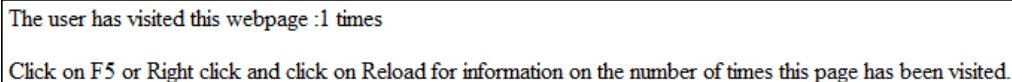
Let's look at the following code to understand how `sessionStorage` works:

```
<!DOCTYPE HTML>
<html>
  <head>
    <title> Session Storage: A part of HTML5 webstorage </title>
  </head>
  <body>

    <script type = "text/javascript">
      if(sessionStorage.visitWebsite) {
        sessionStorage.visitWebsite =
          Number(sessionStorage.visitWebsite) +1;
      }
      else {
        sessionStorage.visitWebsite = 1;
      }
      document.write("The user has visited this web page : " +
        sessionStorage.visitWebsite + " times");
    </script>
    <p> Click on F5 or Right click and click on Reload for
      information on the number of times this page has been
      visited.</p>
  </body>
</html>
```

In the preceding code, we can see the `sessionStorage` property is used, using which we can depict the number of times the user has visited that web page in that session. We also need to remember that once the browser is closed, the counter is reset.

Let's execute the preceding code and see the output in Opera, as shown in the following screenshot:



The user has visited this webpage :1 times

Click on F5 or Right click and click on Reload for information on the number of times this page has been visited.

Let's now reload the same page by right-clicking on the page or by pressing *F5* to reload it twice.

We will see the following output:

```
The user has visited this webpage :3 times
Click on F5 or Right click and click on Reload for information on the number of times this page has been visited.
```

We can see that the counter has been set to three as we reloaded the page **3 times**. Now, let's close the browser window, and execute the code again. On executing the code in a new browser, we can see the following page:

```
The user has visited this webpage :1 times
Click on F5 or Right click and click on Reload for information on the number of times this page has been visited.
```

We can see that the counter has been reset and the code output displays that the web page has been visited once. Hence, the data is stored for a specific session and will be lost once the session expires after closing the browser.

Let's look at the `localStorage` property.

localStorage

The `localStorage` property is not limited to a specific session. The `localStorage` property takes into account the total number of times that the website has been visited. Data is stored at the client side and can be accessed each time we access the website. When we implement the storage feature using HTML5, the data is stored in your local machine such as a laptop, a desktop, or a tablet (the client side). Let's assume that we close the browser window. When we access the website again, the data stored locally can be accessed without any time limit.

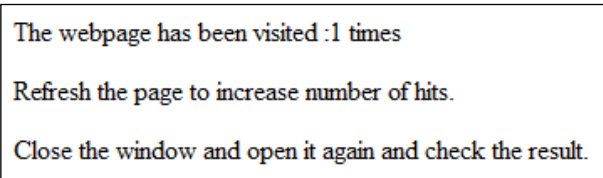
Let's look at the following code to see how `localStorage` works:

```
<!DOCTYPE HTML>
<html>
  <head>
    <title> Local Storage in HTML5 </title>
  </head>
  <body>
    <script type = "text/javascript">
      if(localStorage.visitWebsite) {
        localStorage.visitWebsite =
          Number(localStorage.visitWebsite) +1;
      }
      else {
```



```
        localStorage.visitWebsite = 1;
    }
    document.write("The web page has been visited : " +
        localStorage.visitWebsite + " times");
</script>
<p>Refresh the page to increase number of hits.</p>
<p>Close the window and open it again
    and check the result.</p>
</body>
</html>
```

Let's execute the preceding code in Opera. When we execute the code for the first time, we get the following output:

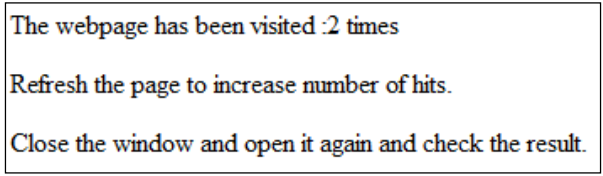


The webpage has been visited :1 times

Refresh the page to increase number of hits.

Close the window and open it again and check the result.

When we reload the browser, we get the following output:

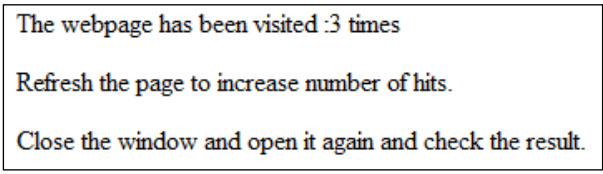


The webpage has been visited :2 times

Refresh the page to increase number of hits.

Close the window and open it again and check the result.

Now let's close the window and execute this code again in a new window to see the output, as shown in the following screenshot:



The webpage has been visited :3 times

Refresh the page to increase number of hits.

Close the window and open it again and check the result.

We can see that the counter is 3 even though we had closed the browser window and executed the code again in a new window. Hence, the stored data can be accessed at anytime with the aid of JavaScript in `localStorage`.

Let's now look at the **offline web applications** feature of HTML5.

Offline web applications

Offline web applications are a new feature of HTML5. When we visit a website that has this feature, the browser downloads all the files that are required for the user to access that specific website. Once the download is complete, the website can be accessed offline. This is particularly useful when there is intermittent internet connectivity. Once we are online, the web server can be updated in case changes have been made.

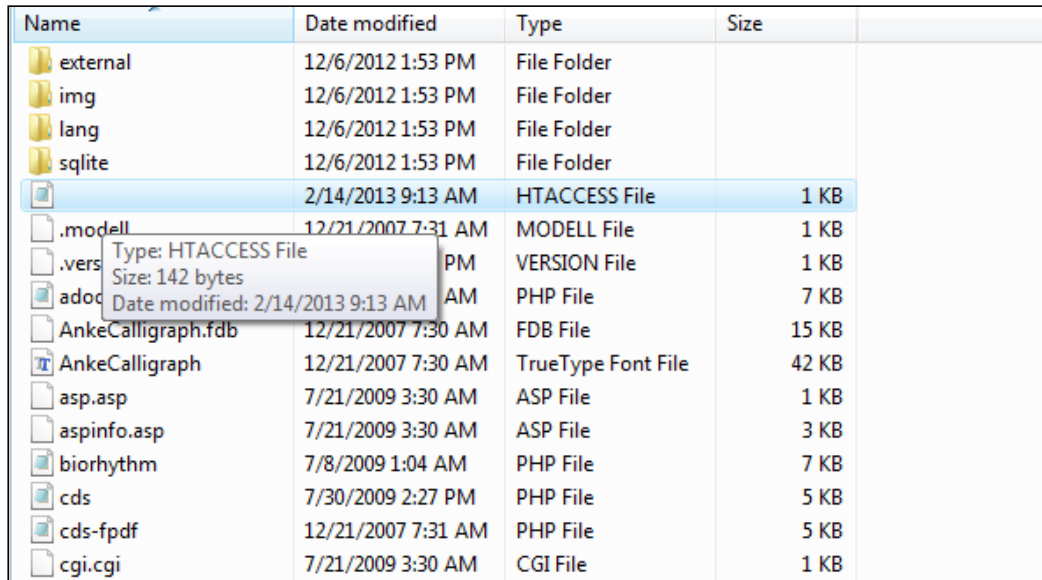
Let's look at the process of creating an offline web application.

We will now create an HTML page first. Have a look at the following code. Copy it to a notepad and save it as a HTML file so that we can use it later. I have saved this file as `aravind.html`. We will modify the `<html>` tag later, which will be explained in the course of the chapter.

```
<!DOCTYPE HTML>
<html>
  <head>
    <title> Offline web applications </title>
    <h1> Offline web applications </h1>
    <link rel = "stylesheet" type = "text/css"
      href = "aravind_style.css" />
  </head>
  <body>
    <p>
Offline web applications are a new feature of HTML5. When
we visit a website which has this feature, the browser
downloads all the files that are required for the user to
access that specific website. Once the download is
complete, the website can be accessed offline.
    </p>
    <div>
      This is particularly useful when there is intermittent
      internet connectivity. Once we are online, the web server
      can be updated in case changes have been made
    </div>
  </body>
</html>
```

Let's now add styling to it by using CSS. Type in the following code and save it as a CSS file with a name of your choice. I have saved this file as `aravind_style.css`.

We will be using a XAMPP server to demonstrate the procedure. Since it is a XAMPP server, the web server would be Apache. In the Apache folder, we can see a .HTACCESS file, as shown in the following screenshot:



Name	Date modified	Type	Size
external	12/6/2012 1:53 PM	File Folder	
img	12/6/2012 1:53 PM	File Folder	
lang	12/6/2012 1:53 PM	File Folder	
sqlite	12/6/2012 1:53 PM	File Folder	
.htaccess	2/14/2013 9:13 AM	HTACCESS File	1 KB
.modell	12/21/2007 7:31 AM	MODELL File	1 KB
.vers	12/21/2007 7:31 PM	VERSION File	1 KB
adoc	2/14/2013 9:13 AM	PHP File	7 KB
AnkeCalligraph.fdb	12/21/2007 7:30 AM	FDB File	15 KB
AnkeCalligraph	12/21/2007 7:30 AM	TrueType Font File	42 KB
asp.asp	7/21/2009 3:30 AM	ASP File	1 KB
aspinfo.asp	7/21/2009 3:30 AM	ASP File	3 KB
biorhythm	7/8/2009 1:04 AM	PHP File	7 KB
cds	7/30/2009 2:27 PM	PHP File	5 KB
cds-fpdf	12/21/2007 7:31 AM	PHP File	5 KB
cgi.cgi	7/21/2009 3:30 AM	CGI File	1 KB

Open the .HTACCESS file and add the following code:

```
AddType text/cache-manifest .manifest
```


We will now create a MANIFEST file, which will assist us in caching the web page for offline use.

Create a new file named MANIFEST. In this example, we will name it as aravind.manifest. Write the following code in the MANIFEST file:

```
CACHE MANIFEST
# A file called manifest

CACHE:
aravind.html
aravind_style.css
packt.png

NETWORK:
login.php
```


 I have used `aravind.html` and `aravind_style.css` as the HTML and CSS files. You can enter any name of your choice, such as `index.html`, `style.css` to mention a few. I have saved the MANIFEST file as `aravind.manifest`. You can create a MANIFEST file with the name of your choice. Remember that the extension must be `manifest`.

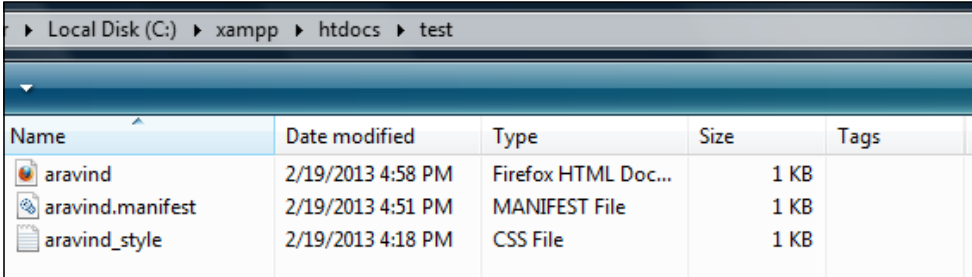
Anything that comes under `CACHE` would be cached except for the code mentioned under `NETWORK`.

We have added a `NETWORK` attribute to the code. We have mentioned `login.php` under it. This denotes that the `login.php` file must not be cached. Anything under `NETWORK` will not be cached. For example, a login page may have information, such as the username and password. Hence, these kind of pages that should not be available offline must be under the `NETWORK` declaration.

Now, we need to link the `manifest` property to the HTML code by modifying the HTML code, as shown in the following code snippet:

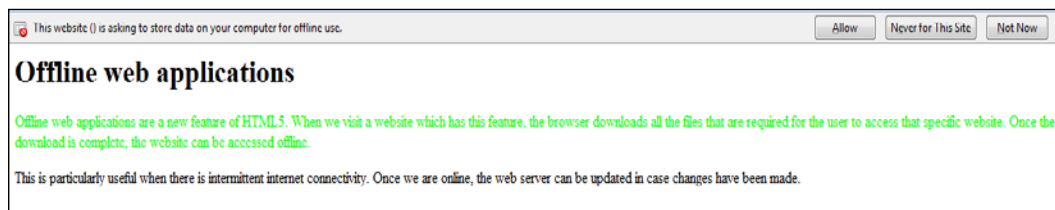
```
<html manifest = "/aravind.manifest">
```

Since we are working on XAMPP, we will save the `aravind.manifest`, `aravind.html`, and `aravind_style.css` file in the `htdocs` folder in XAMPP. Please refer to the following screenshot to see the saved files:



Name	Date modified	Type	Size	Tags
aravind	2/19/2013 4:58 PM	Firefox HTML Doc...	1 KB	
aravind.manifest	2/19/2013 4:51 PM	MANIFEST File	1 KB	
aravind_style	2/19/2013 4:18 PM	CSS File	1 KB	

Now, when we run the HTML file (`aravind.html`) in this case, we will see the following output:



We can see a prompt in the preceding screenshot that tells us whether we want to save the data for offline use.

This is pretty much it. At the time of writing this chapter, the latest versions of all the browsers except Internet explorer support this feature.

Let's now look at the `canvas` feature of HTML5.

Canvas

The `canvas` element is one of the awesome features of HTML5. It enables us to draw graphics within the defined boundaries. Though `canvas` is an HTML element, it works in tandem with the JavaScript API. Let's look at how the `canvas` element is defined in HTML5 as follows:

```
<canvas id = "demo" width = "500" height = "150"> </canvas>
```

The `id` attribute is optional, but very useful as it is extensively used in HTML5. The `width` and `height` attributes of the `canvas` element have to be defined to understand the boundaries of the `canvas` element. If the `height` and `width` attributes are not defined, then the default value is considered that is the `width` attribute would be 300 pixels and `height` would be 150 pixels by default. However, it is a good practice to define the `height` and `width` attributes in a `canvas` element.

We need to obtain a reference to the `canvas` element by using the following code snippet:

```
var canvas = document.getElementById('demo');
```

Once we obtain a reference, we need to obtain a reference to the 2d context in the `canvas` element; hence, we use the following code snippet in which we call the `getContext('2d')` on the defined `canvas` element:

```
var context = canvas.getContext('2d');
```

We will start coding right away to understand the functionality of `canvas` instead of wandering through loads of theory. Let's look at the following code:

```
<!DOCTYPE HTML>
<html>
  <head>
    <script type = "application/javascript">
      function drawRectangle() {
        var canvas = document.getElementById('packt');
        var context = canvas.getContext("2d");
        context.fillStyle = "rgb(200,0,0)";
```

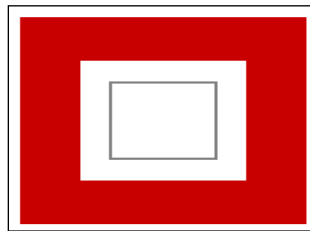
```
        context.fillRect (90, 90, 190, 190);
        context.clearRect (130,130,110,110);
        context.strokeRect (150,150,70,70);
    }
</script>
</head>
<body onload = "drawRectangle();" >
    <canvas id = "packt" width = "300" height = "300"></canvas>
</body>
</html>
```

The only primitive shape supported by the `canvas` element is the rectangle.



For other shapes, we use the **path** concept, which will be explained later in this chapter.

The output of the code is displayed in the following screenshot:



The `fillRect` property is used to draw a rectangle filled with color. Here we have used `fillStyle = "rgb(200,0,0)"`, which will fill the rectangle with red color.

The `clearRect` property will clear the defined rectangular space respective to the dimensions mentioned for clearing.

The `strokeRect` property draws an outline in a rectangular shape.

As we can see, all these properties have four parameters mentioned. The parameters can be defined as $(x, y, width, height)$. (x, y) are the coordinates along the x axis and y axis respectively. The origin of these coordinates is at the top-left corner of the web page at $(0, 0)$.

I guess that was pretty simple. Now, let's look at the procedure to create different shapes other than the rectangular primitive shape. We have to use the concept of path to understand the functionality.

beginPath

We start the path using the `beginPath()` method. If we call this method, the list of items that form a shape is reset, and as a result of that we can draw new shapes.

closePath

Using the `closePath()` method, we can close the shape. For example, if the pointer is at a point distant from the starting point, this method will close the shape by drawing a straight line to the start point. If the shape is closed already, then this method will not do anything as the purpose has already been served.

moveTo

We know that the x axis and y axis coordinates originate at the top-left corner of the canvas at (0,0) by default. If we want the coordinates to start from a different point on the screen, we will move the virtual pointer to that specific path using the `moveTo()` method. This method will take two coordinates (x, y), which will define the new starting point on the screen.

stroke and fill

The stroke function is used to draw the outline of any shape, whereas the fill function is used to fill color into a shape. If we are using the `fill()` method, we do not have to use `closePath()`, as the shape will be completely filled using the fill function.

arc

The `arc()` method is used to draw an arc on the canvas, and there are five parameters defined in the `arc()` method. Usually, an `arc()` method would be defined in the following format:

```
arc(x, y, radius, startAngle, endAngle, anticlockwise)
```

Let's look at the following parameters and understand what they mean:

- The `x` and `y` parameters are the coordinates on the x axis and y axis
- The `radius` parameter defines the radius of the arc
- The `startAngle` and `endAngle` define the angle in which the arc starts and ends respectively
- `anticlockwise` defines the direction of drawing the arc

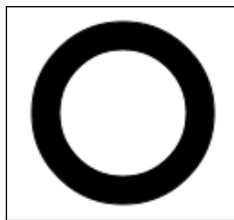
lineTo

The `lineTo()` method will draw a line from the starting point of the virtual pointer to the parameters defined in the function. For example, suppose we have `moveTo(10,20)` and `lineTo(50,70)` defined in the code, a line will be drawn from (10,20) to (50,70).

Let's now look at the following code to understand the procedure to draw concentric circles:

```
<html>
  <head>
    <script type = "application/javascript">
      function drawArc() {
        var canvas = document.getElementById('packt');
        var context = canvas.getContext('2d');
        context.beginPath();
        context.arc(75, 75, 50, 0, Math.PI*2, true);
        context.moveTo(110, 75);
        context.arc(75, 75, 35, 0, Math.PI*2, false);
        context.stroke();
        context.fill();
      }
    </script>
  </head>
  <body onload = "drawArc();" >
    <canvas id = "packt" width = "300" height = "300"></canvas>
  </body>
</html>
```

The output of the code will be as shown in the following screenshot:



The `Math` function in the code is a part of the JavaScript library. We have not used the `closePath` function here, as we have used the `fill` function at the end of the JavaScript code.

If we look at the preceding code, we have first obtained a reference to the `canvas` element, and then obtained a reference to the 2d context. We moved the virtual pointer to (110,75), and then drew the second circle.

We called the `drawArc` function when the page loaded up. Hence, we can see the circles. The `fill` attribute has filled the shape with the color black.

Now, we will discuss **gradients**, which are an imperative feature of the HTML5 canvas element.

Gradients

Gradients enable us to change from one color to another in a particular manner where the colors juxtapose with each other. We will discuss the concept of **linear gradients** in this chapter.

To start with, we have `createLinearGradient(x1, y1, x2, y2)`. The gradient starts from the point (x1, y1) and extends to (x2, y2). We can create a horizontal or vertical gradient by changing the x axis and y axis parameters.

Now, we will add colors to the gradients. We will declare a `linGrad` variable and assign it a gradient property in the following way:

```
var linGrad = context.createLinearGradient(20, 20, 50, 70);
```

In order to define a color to the gradient, we will use the `addColorStop` attribute in the following way:

```
linGrad.addColorStop(0, navy);
```

We can see that there are two parameters in the `addColorStop` property. The first parameter is to be defined from 0 to 1 to indicate the extent to which the gradient color has to be applied. The second parameter is the color which is to be used. We can add innumerable `colorStops` as per the requirement. While we add these various `colorStops`, we need to change the first parameter accordingly between the range of 0 and 1.

Let's look at the following code to understand the gradient concept better:

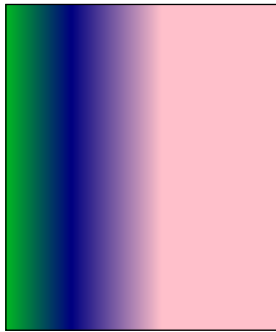
```
<html>
  <head>
    <script type = "application/javascript">

      function drawGrad () {
        var canvas = document.getElementById("packt");
        var context = canvas.getContext("2d");
```

```
var linGrad = context.createLinearGradient(0, 0, 100, 0);
linGrad.addColorStop(0, "lime");
linGrad.addColorStop(0.5, "navy");
linGrad.addColorStop(1, "pink");

context.fillStyle = linGrad;
context.fillRect(10, 10, 160, 200);
}
</script>
</head>
<body onload = "drawGrad();" >
  <canvas id = "packt" width = "200" height = "200"></canvas>
</body>
</html>
```

The output of the code would be as shown in the following screenshot:



We can see how the color varies and fades as well as the effect of the parameters on the `canvas` element. We have defined a `fillStyle` property and assigned it to the `linGrad` variable in the code. The `fillStyle` property fills up the rectangle according to the colors mentioned in the `addColorStop` attribute. We can also use `strokeStyle` property instead of the `fillStyle` property, in case we want to add colors to a rectangular outline.

Let's now look at the `save()` and `restore()` methods of the `canvas` element.

save and restore

The `save()` method saves the state of the `canvas` on a stack while the `restore()` method will return the last saved state from the stack.

Let's look at the following code to understand the `save()` and `restore()` methods better:

```
<!DOCTYPE HTML>
<html>
  <head>
    <script type = "text/javascript">
      function demoTranslation() {
        var canvas = document.getElementById('packt');

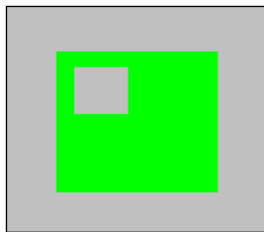
        var context = canvas.getContext('2d');

        context.fillStyle = "silver";
        context.fillRect(30, 30, 150, 100);

        context.translate(50, 25);

        context.fillStyle = "navy";
        context.fillRect(30, 30, 200, 100);
      }
    </script>
  </head>
  <body onload = "demoTranslation();" >
    <canvas id = "packt"></canvas>
  </body>
</html>
```

The output of the code will be displayed, as shown in the following screenshot:



Let's look at how the code works. The first rectangle of silver color is created and the settings are saved on the memory stack. Then, we define another rectangle of the color, lime, which is inside the silver rectangle. Then, we revoke the saved rectangle using the `restore()` method. We just change the dimensions of the restored rectangle and it is seen inside the lime rectangle. Hence, we can return a saved shape from the stack, which is saved by the `save()` method, using the `restore()` method.

Transformations

Transformations in HTML5 can be deployed in the following ways:

- translate
- rotate
- scale

translate

Translation means the ability to relocate the drawing on the canvas.

By using the `translate` property, we can relocate the drawn shape to a different location. We need to remember that we need to call the `translate` function, following which it would be implemented.

Let's look at the following code to understand the `translate` property better:

```
<!DOCTYPE HTML>
<html>
  <head>
    <script type = "text/javascript">
      function demoTranslation() {
        var canvas = document.getElementById('packt');

        var context = canvas.getContext('2d');

        context.fillStyle = "silver";
        context.fillRect(30, 30, 150, 100);

        context.translate(50, 25);

        context.fillStyle = "navy";
        context.fillRect(30, 30, 200, 100);
      }
    </script>
  </head>
  <body onload = "demoTranslation();" >
    <canvas id = "packt"></canvas>
  </body>
</html>
```

The output of this code would be as shown in the following screenshot:



If you read through the preceding code, you will find out the silver rectangle retains its original location. We have used the `translate` function after we defined the silver rectangle. The `translate` property worked with the blue rectangle. Hence, it is understood that the `translate` function must be called before the shape and its dimensions are defined.

rotate

We can rotate any shape within the boundaries of the canvas by using the `rotate()` method. The `rotate()` method is defined, as shown in the following code:

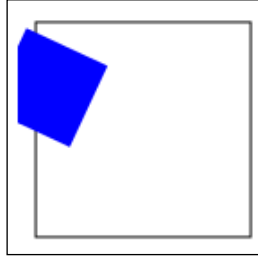
```
<!DOCTYPE HTML>
<html>
  <head>
    <script type = "text/javascript">
      function demoRotation() {
        var canvas = document.getElementById('packt');
        var xyz = canvas.getContext('2d');

        xyz.strokeRect(10, 10, 120, 120);

        xyz.rotate((Math.PI / 180) * 25); //rotate 25 degrees.

        xyz.fillStyle = "#0000ff";
        xyz.fillRect(10, 10, 50, 50);
      }
    </script>
  </head>
  <body onload = "demoRotation();">
    <canvas id = "packt"></canvas>
  </body>
</html>
```

The output of the code is displayed in the following screenshot:



We have used a variable, `xyz`, to obtain a reference to the 2d context of the canvas. Then, we draw a rectangular outline using the `strokeRect` property. Then, we call the `rotate` function. After we have called the `rotate` function, we define a rectangle using the `fillRect` property, and use `fillStyle` to fill it with blue color. Similar to the `translate` function, the `rotate` property works on the blue rectangle and not on the first rectangle, as the blue rectangle code was written after we called the `rotate` function. Hence, we need to call this method prior to defining any shape in order for it to rotate.

scale

Scaling of any shape on the canvas is possible by using the `scale` property.

Let's look at the following code to see how `scale` works:

```
<!DOCTYPE HTML>
<html>
  <head>
    <script type = "text/javascript">
      function demoScale() {
        var canvas = document.getElementById('packt');

        var context = canvas.getContext('2d');

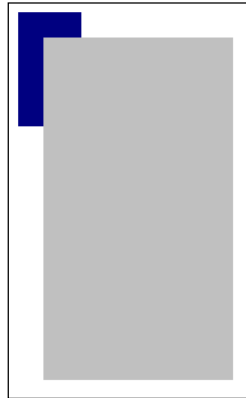
        context.fillStyle = "navy";
        context.fillRect(10, 10, 50, 90);

        context.scale(3, 3);

        context.fillStyle = "silver";
        context.fillRect(10, 10, 50, 90);
      }
    </script>
  </head>
</html>
```

```
</head>
<body onload = "demoScale();" >
  <canvas id = "packt" width = "400" height = "400"></canvas>
</body>
</html>
```

The output of the code will be, as shown in the following screenshot:



In the preceding code, we have used a scale of (3:3). Similar to `translate` and `rotate`, the `scale` property is applicable to only the second rectangle as the method was called before the silver rectangle. The navy-colored rectangle will remain in its original state as the method was not called before defining it.

Animation

If we observe the complex websites today, we can see that there is a lot of animation that goes on behind them. It is now easier to use animation with the advent of the canvas element in HTML5 in conjunction with JavaScript. In HTML5, we need to draw and redraw and clear the canvas so fast that it seems like an animation. We will be using the `window.requestAnimationFrame` property, which tells the browser that animation will be performed. It is a callback function that tells the browser to repaint the canvas for the next frame.

Let's look at the following code to understand animation better:

```
<html>
  <head>
    <style type = "text/css">
      #packt {
        border:lime 10px solid
      }
    </style>
  </head>
</html>
```

```
</style>

<title>Canvas tutorial</title>
<script type="text/javascript">
  var x = 0;
  var y = 15;
  var z = 5;

  function demoAnimation() {
    animationMethod = window.mozRequestAnimationFrame ||
                      window.webkitRequestAnimationFrame ||
                      window.msRequestAnimationFrame ||
                      window.oRequestAnimationFrame
    ;

    animationMethod(demoAnimation);

    x = (x + z);

    if(x <= 0 || x >= 370) {
      z = -z;
    }

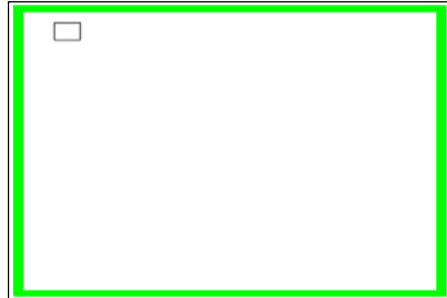
    draw();
  }

  function draw() {
    var canvas = document.getElementById("packt");
    var context = canvas.getContext("2d");

    context.clearRect(0, 0, 500, 170);
    context.strokeStyle = "black";
    context.strokeRect(x, y, 25, 25);
  }

  demoAnimation();
</script>
</head>
<body onload = "draw();">
  <canvas id = "packt" width = "400" height = "400"></canvas>
</body>
</html>
```


The output of the code is displayed in the following screenshot:



In the preceding screenshot, we see a rectangle moving back and forth within the defined boundaries of the canvas frame.

We have used CSS styling to depict the boundaries of the canvas in green. Then, we defined a `demoAnimation` function in JavaScript. Then, we invoked the callback function, `window.RequestAnimationFrame`, and assigned it to the `animationMethod` function. We have added a prefix of `moz`, `ms`, `webkit` before `RequestAnimationFrame` so that it is compatible with the latest versions of browsers such as IE, Mozilla Firefox, and Chrome. Then, we passed the `demoAnimation` function inside the `animationMethod`. By doing this, we have made sure that the browser will call the `demoAnimation` function when the next frame is ready to be drawn for animation purposes. The canvas has to be cleared for the next frame to be drawn. Hence, we use the `draw` function, which helps us do that. If we observe the preceding code, we can see that the `demoAnimation` function is called at the end. This is to get `window.RequestAnimationFrame` into action so that we can begin the animation process, and hence, this makes it mandatory.

Summary

In this chapter, we had a look at various features such as drag-and-drop, canvas, and Geolocation. We had a look at how audio and video can be embedded into an HTML document. We also understood the concepts of Webstorage and offline web applications.

In the next chapter, we will have a look at CSS3 Animations and understand the nuances of it. We will especially look at the Transformation, Transition, and Animation modules of CSS3 to understand the concepts better.

6

CSS3 Animations

As mentioned earlier, websites nowadays are complex and complicated. By complex and complicated, we are referring to the development of these websites and not the web pages themselves. We see animations and complex features. Prior to HTML5 and CSS3, JavaScript was used extensively for this purpose. HTML was incorrectly used for styling, when it was expected to design the structural markup of the page. However, with the advent of CSS, it is a good practice to use HTML for markup, and CSS for styling. CSS3 brings along transforms, transition elements, and animation features that make it easier to develop awesome features.

Moreover, the need for JavaScript has reduced considerably, as we can achieve the same with CSS3 in tandem with HTML5. In this chapter, we are going to discuss the following features:

- CSS3 transitions
- CSS3 transforms
- CSS3 animation

CSS3 transitions

Transitions enable us to determine the speed of animation as well as introduce delays, as and when required. They also enable the web designers to alter the state and the behavior of the elements in use. Transitions assist developers to achieve a smooth flow of animation and display the same accordingly, so that we can observe the change of state.



We have used the FlexBox code in *Chapter 3, Flexible Box Model in CSS3*, to illustrate transitions. The new FlexBox version is compatible with the latest versions of Google Chrome and Opera. Though CSS3 transitions, transformations, and animations are compatible with Firefox and IE, we will stick to Google Chrome to learn through these examples, as it will help us keep up with the new standards and in sync with the times.

The `transition-property` is used in conjunction with the properties that are defined to alter the state. When we define the `transition-property` along with the state altering properties, only those properties will be undergoing transition. We also need to remember that the vendor prefixes have to be used, as it is still in the development stage and not compatible with all the browser versions. For example, we use `-moz` for Mozilla, and so on.

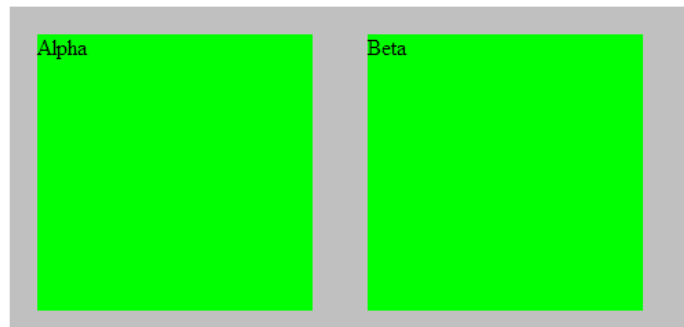
Let's look at the following code to understand the concept better:

```
<html>
  <head>
    <style>
      #flex-container {
        display: -webkit-flex;
        display: flex;
        width: 500px;
        height: 500px;
        background-color: Silver;
      }

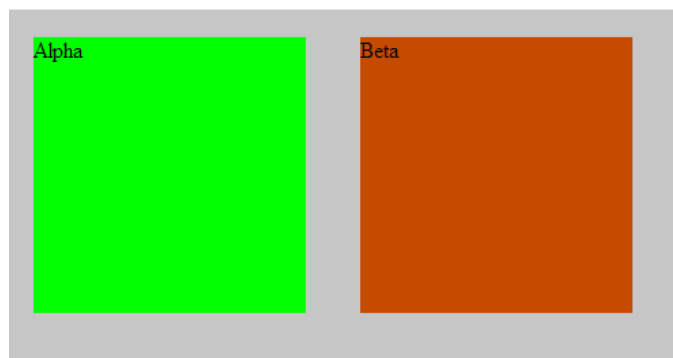
      #flex-item {
        background-color: lime;
        transition-property: background;
        -webkit-transition-property: background;
        transition-duration: 3s;
        -webkit-transition-duration: 3s;
        transition-timing-function: linear;
        -webkit-transition-timing-function: linear;
      }
    </style>
  </head>
</html>
```

```
        width: 200px;
        height: 200px;
        margin: 20px;
    }
    #flex-item:hover {
        background: red;
    }
</style>
</head>
<body>
  <div id = "flex-container">
    <div id = "flex-item">Alpha</div>
    <div id = "flex-item">Beta</div>
  </div>
</body>
</html>
```

We have defined the background property to undergo transition (`transition-property: background ;`), as seen in the following screenshot:



Once we hover over the **Beta** rectangular item, the color will change to red eventually, as displayed in the following screenshot:



While we hover over **Beta**, we can observe the transition from green to red. We have introduced a transition duration of three seconds, and the change in color can be seen gradually spanning over three seconds.

In the preceding code, we have applied transition to the background. However, there are lots of properties to which we can apply the `transition-property`. We have listed some of the properties as follows:

- `background-color`
- `border-spacing`
- `font-size`
- `border-radius`
- `color`
- `margin`
- `max-width`
- `max-height`
- `right`
- `top`
- `vertical-align`
- `padding`

Let's look at the other properties that can be applied along with the `transition-property` as follows:

- `transition-duration`
- `transition-timing-function`
- `transition-delay`

The transition-duration property

The `transition-duration` tells us the time period for the transition to occur. The duration can be defined in seconds or milliseconds. We can enter multiple values if we are defining two or more properties.

Let's look at the following code to understand it better:

```
<html>
  <head>
    <style>
      #flex-container {
        display: -webkit-flex;
```

```
        display: flex;
        width: 500px;
        height: 500px;
        background-color: Silver;
    }

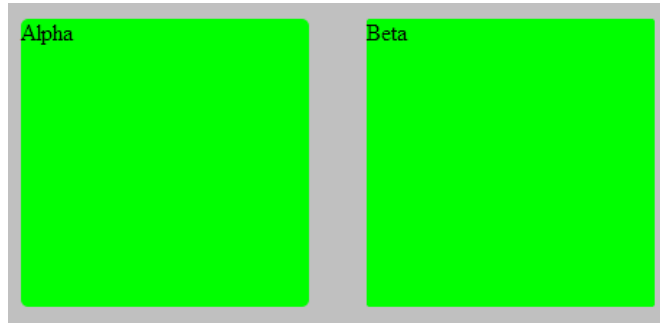
    #flex-item {
        background-color: lime;
        transition-property: background, border-radius;
        -webkit-transition-property: background, border-radius;
        transition-duration: 2s, 6s;
        -webkit-transition-duration: 2s, 6s;
        transition-timing-function: linear;
        -webkit-transition-timing-function: linear;

        width: 200px;
        height: 200px;
        margin: 20px;
    }
    #flex-item:hover {
        background: red;
        border-radius: 70%
    }

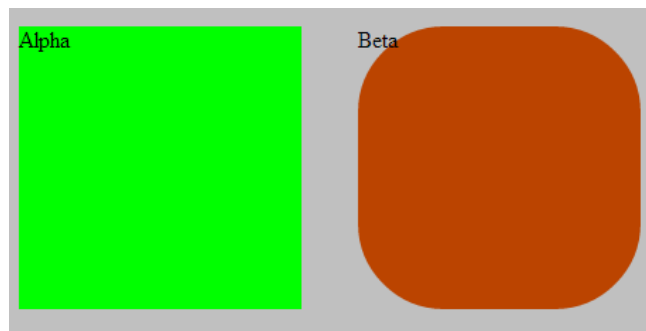
</style>
</head>
<body>
    <div id = "flex-container">
        <div id = "flex-item">Alpha</div>
        <div id = "flex-item">Beta</div>
    </div>
</body>
</html>
```

We have defined the background, following which, we have defined `border-radius` as the second property with the `transition-property`. We have also defined the transition duration as `2s` and `6s` respectively.

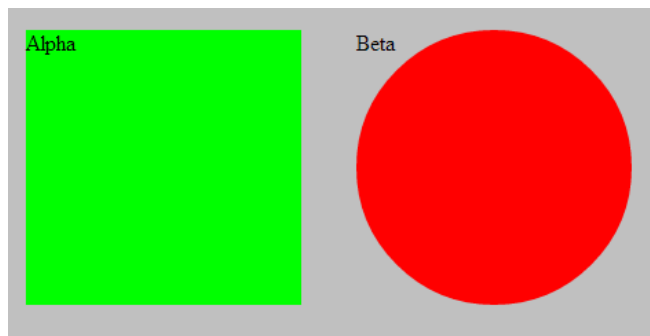
The output of the code would initially be as shown in the following screenshot:



If we hover the cursor over the **Beta** rectangle for two seconds, the color would change from lime to red in two seconds. The border radius would still be in a transition mode. After two seconds, the output would be as shown in the following screenshot:



After six seconds, the `border-radius` property transition would be complete, and we can see the change in the shape, as shown in the following screenshot:



The transition-timing-function property

The `transition-timing-function` property is used to define the speed of transition. If we set it to `linear`, it will be gradual at a constant speed. If we change the value from `linear` to `ease-in`, the transition is slow at first, and then it picks up speed during the process. However, if we choose `ease-out`, the speed initially is fast, and it slows down from then on.

Let's look at the following code to understand the concept better:

```
<html>
  <head>
    <style>
      #flex-container {
        display: -webkit-flex;
        display: flex;
        width: 500px;
        height: 500px;
        background-color: Silver;
      }

      #flex-item {
        background-color: lime;
        transition-property: background, border-radius;
        transition-duration: 3s, 3s;
        -webkit-transition-duration: 3s, 3s;

        -webkit-transition-property: background, border-radius;
        transition-timing-function: ease-in;
        -webkit-transition-timing-function: ease-in;

        width: 200px;
        height: 200px;
        margin: 20px;
      }
      #flex-item:hover {
        background: red;
        border-radius: 70%
      }
    </style>
  </head>
  <body>
    <div id = "flex-container">
      <div id = "flex-item">Alpha</div>
      <div id = "flex-item">Beta</div>
    </div>
  </body>
</html>
```


When we execute the preceding code, we can observe that the speed of transition is slow at first, but the speed increases as the transition takes place. In place of `ease-in`, if we use `ease-out`, it starts quickly, but the speed decreases towards the end.

The transition-delay property

The `transition-delay` property is used to postpone the transition. If we set a `transition-delay` of two seconds, then the transition would start after two seconds. We can delay more than one property, by specifying multiple `transition-delay` values separated by a comma.

Let's execute the following code and you will see how it works:

```
<html>
  <head>
    <style>
      #flex-container {
        display: -webkit-flex;
        display: flex;
        width: 500px;
        height: 500px;
        background-color: Silver;
      }
      #flex-item {
        background-color: lime;
        transition-property: background, border-radius;
        transition-duration: 1s, 1s;
        -webkit-transition-duration: 1s, 1s;
        -webkit-transition-property: background, border-radius;
        transition-timing-function: linear;
        -webkit-transition-timing-function: linear;
        transition-delay: 3s, 3s;
        -webkit-transition-delay: 3s, 3s;
        width: 200px;
        height: 200px;
        margin: 20px;
      }
      #flex-item:hover {
        background: red;
        border-radius: 70%
      }
    </style>
  </head>
```

```
<body>
  <div id = "flex-container">
    <div id = "flex-item">Alpha</div>
    <div id = "flex-item">Beta</div>
  </div>
</body>
</html>
```

On executing the preceding code, we can see that there is a delay of three seconds before the transition begins. Hence, that explains the `transition-delay` property. Now, we will look at our next section, *CSS3 transforms*. Instead of defining each property individually, we can use `transition:`, which is a shortcut to define various values.

It can be defined as follows:

```
transition: background 3s linear, border-radius 2s ease-in 1s;
```

The `transition-property`, `transition-duration`, `transition-timing-function`, and lastly, `transition-delay` properties are defined at once in the order respectively.

Let's now look at the CSS3 transformation module.

CSS3 transforms

In CSS3, we can change the position of the elements without disrupting the normal flow. Apart from that, we can define the two dimensional as well as the three dimensional outlook of the elements. In CSS 2D transforms, we can rotate, skew, translate, and scale the elements. We will first look at the 2D transforms in CSS3, in which we will learn about the rotate, skew, translate, and scale properties of CSS3.

rotate

Using the `rotate` feature, we can rotate any element clockwise or counterclockwise. If we use a positive value as the parameter value in degrees, then the element will be rotated clockwise. A negative value will rotate the element counterclockwise. We can rotate an element to an extent of 360 degrees.

We use the following syntax to use the 2D `rotate` feature:

```
transform: rotate(45deg);
```

If we pass 45 degrees as the parameter value, then the element will rotate by 45 degrees.

scale

Using the `scale` feature, we can increase the apparent size of the element. If the scale is less than 1, then it will decrease the apparent size of the element. If the scale is greater than 1, it increases the apparent size of the element.

We use the following syntax to use the 2D `scale` feature:

```
transform: scale(2);
```

If we pass 2 as the parameter value, then the apparent size will be twice the real size.

Let's look at what the following syntax would do:

```
transform: scaleX(value);
```

```
transform: scaleY(value);
```

```
transform: scale(value);
```

`scaleX` will change the apparent width of the element along the x axis.

`scaleY` will change the apparent height of the element along the y axis.

`scale(value 1, value 2)` will change the apparent width and height along the x axis and y axis respectively.

translate

Using the `translate` feature, we can change the apparent position of the element along the x axis and the y axis.

Whenever we draw an element, the default coordinates along the x axis and y axis are (0,0) respectively. To change the x axis and y axis coordinates, we use the `translate` feature.

We use the following syntax to use the 2D `translate` feature:

```
transform: translateX(value);
```

```
transform: translateY(value);
```

```
transform: translate(x-axis value, y-axis value);
```

`translateX` will change the initial position to the x axis parameter value.

`translateY` will change the initial position to the y axis parameter value.

`translate(value1, value2)` will change the position with regards to the parameter values along the x axis and y axis respectively.

Values can be defined as percentages or in pixels.

skew

Using the *skew* feature, we can change the angle of the element along the x axis and y axis.

We use the following syntax to use the 2D *skew* feature:

```
transform: skewX(value)
```

```
transform: skewY(value)
```

skewX will distort the element along the x axis, and *skewY* will change the angle along the y axis.

However, to skew the element along the x axis and y axis, we need to use the following syntax:

```
transform: skew(xdeg, ydeg)
```

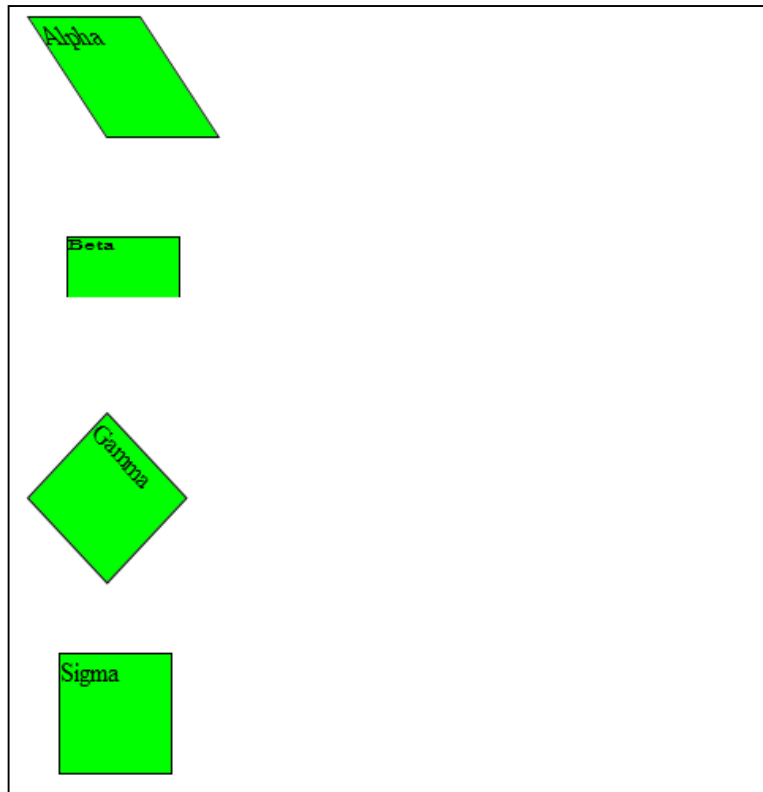
Suppose we have to use multiple properties. In that case, we have to use a single *transform*: property and assign various properties to it on the same line. We cannot define multiple transform values, as the latest value will override the previous values.

Let's look at the following code to understand the concept better:

```
<html>
  <head>
    <title> 2D CSS3 Transforms </title>
    <style>
      div {
        width:70px;
        height:70px;
        background-color: lime;
        border:1px solid black;
      }
      div#div1 {
        transform:skew(35deg) translateX(35px);
        -webkit-transform:skew(35deg) translateX(35px);
      }
      div#div2 {
        transform:scale(1,0.5) translateX(35px);
        -webkit-transform:scale(1,0.5) translateX(35px);
      }
      div#div3 {
        transform:rotate(45deg) translateX(35px);
        -webkit-transform:rotate(45deg) translateX(35px);
      }
    </style>
  </head>
</html>
```

```
div#div4 {
  transform:translate(30px, 40px);
  -webkit-transform:translate(30px, 40px);
}
</style>
</head>
<body>
  <div id = "div1">Alpha</div>
  <br><br>
  <div id = "div2">Beta</div>
  <br><br>
  <div id = "div3">Gamma</div>
  <br><br>
  <div id = "div4">Sigma</div>
</body>
</html>
```

The output of the code is displayed in the following screenshot:



When we check the output, we can see that the same rectangle as defined in the CSS is displayed in four different ways. In `div#div1`, we have used the `skew` property, as a result of which we see the distortion. In `div#div2`, we have used the `scale` property, due to which we can see the change in the size. In `div#div3`, we have used the `rotate` property, due to which we can see the rectangle tilted at an angle. And in `div#div4`, we have used the `translate` property, due to which the positioning of the co-ordinates along the x axis and y axis has changed respectively. If you observe the preceding code properly, we can see that in `div#div1`, `div#div2`, and `div#div3`, we have used the `translateX(35px)` in conjunction with `skew`, `rotate`, and `scale`, as we had to position the element at some distance from the x axis.

Hence, we have observed how 2D transforms are used in CSS3. Let's now look at the 3D transforms used in CSS3. Before we understand 3D transforms, we need to have a glance at the `perspective` property. The `perspective` property is used as displayed in the following syntax:

```
transform: perspective(value in pixels);
```

The value in pixels determines the proximity of the perspective. A higher value will make the element apparently distant, whereas a low value will make the perspective appear closer and will indicate a real life image of the element. Let's understand this concept in a better way.

Imagine we are standing near a sculpture. The sculpture will look clearer if you are standing near it. However, if we observe the same sculpture from a distance of 100 meters, we will have a different view of it. Hence, in the concept of 3D transforms, when we use a higher perspective value, the object will appear to be distant whereas when we use a lower perspective value, the object will appear to be its actual size. Hence, keeping in mind the third dimensional aspect, we need to make sure that the perspective value is defined appropriately.

In 2D transforms, we came across the x axis and y axis, where we can decide the height and width. However, we have a third axis in 3D transforms, which will assist us in deciding the depth along with the width and height. The third angle axis is called the z axis. Let's look at the various 3D properties of CSS3 transforms. We will then club the examples together to understand it in a better way.

translate (3D)

The `translate` property in 3D is different from the 2D `translate` property because the z axis comes into picture. We use the following syntax to use the 3D `translate` feature:

```
transform: translateZ(value);
```

The value entered for the `translate` property is decisive, as it will decide the position of the element on the z axis. A positive value will bring the element closer to the z axis, whereas a negative value will push the element away from the z axis. Hence, the apparent size of the element in the third dimension can be manipulated using the `translate` property.

rotate (3D)

The `rotate` property in 3D transforms adds the z axis to the prevalent x axis and y axis. We will use the following syntax to use the 3D `rotate` property:

```
transform: rotateX(value);  
transform: rotateY(value);  
transform: rotateZ(value);
```

The `rotateZ` property will rotate the element along the z axis. The `rotateX` and `rotateY` property will bend the element horizontally and vertically along the x axis and y axis respectively. The value is decisive as a negative value will rotate the element counter-clockwise, whereas a positive value will rotate the element clockwise.

There is a limitation to the 3D transforms. The `scale` property can be used, but since we define a perspective in it, it is not used widely. The `skew` property also exists but it is applicable only to the x axis and y axis. The `skew` property cannot be implemented on the z axis. We will now look at the `preserve-3d` feature.

preserve-3d

Suppose we talk about a parent element under which there are several child elements. Let's assume that the transforms are applied on the parent element. At the same time, let's assume that a different transform is applied on the nested element. Do you think it is going to work? It will not. Hence, we have a property so that child elements can retain their individuality. The `preserve-3d` feature is to be implemented with the `transform-style` feature on the parent element so that the nested elements can be transformed uniquely. We will use the following syntax to use the `preserve-3d` property:

```
.parent class {transform-style: preserve-3d;}
```

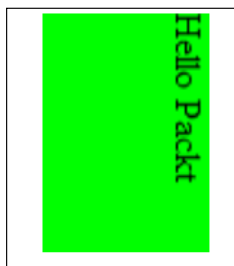
We have now discussed the various 3D transforms and their properties. Let's have a look at the following code to understand it better:

```
<!DOCTYPE html>  
<html>  
  <head>
```

```
<style>
  div {
    width:100px;
    height:75px;
    background-color:lime;
    transform:perspective(350px) rotateZ(90deg)
      translateX(30px)
      translateY(-35px) translateZ(150px);
    -webkit-transform: perspective(350px) rotateZ(90deg)
      translateX(30px)
      translateY(-35px) translateZ(150px);
  }
</style>
</head>
<body>
  <div>Hello Packt</div>
</body>
</html>
```

If we observe the preceding code, we have used the `rotateZ`, `translate`, and `perspective` properties.

As per the parameter values passed, the element is rotated around the z axis by 90 degrees. The positioning of the element on the x axis is shifted by 30 pixels, and we are looking at it through a perspective set at 350 pixels. Hence, the output of the code is displayed as shown in the following screenshot:



Now that we have understood the transition and transformation features, let's look at animation in CSS3.

CSS3 animation

In transition, we have seen the change from one state to another. However, it doesn't fit the bill when it comes to multiple states. Hence, the animation feature is used for this purpose.

Let's discuss the various properties of CSS3 animations, and then we will incorporate all of that in a code to understand it better.

@keyframes

The points at which the transition should take place can be defined using the `@keyframes` property. As of now, we need to add a vendor prefix to the `@keyframes` property as it is still in its development state. In future, when it is accepted as a standard, then we do not have to use a vendor prefix. We can use `percentage` or `from` and `to` keywords to implement the change in state from one CSS style to another.

animation-name

We need to apply animation to an element. The `animation-name` property enables us to do so, by applying it to the animation name defined in the keyframes rule. However, it cannot be a standalone property and has to be used in conjunction with other animation properties.

animation-duration

Using the `animation-duration` feature, we can define the duration of the animation. If we specify the animation duration as 5 seconds, changes in the CSS defined states will need to be completed within five seconds.

animation-delay

Similar to the `delay` property in transition, the `animation-delay` feature will delay the animation by the time period specified.

animation-timing-function

Similar to the `timing function`, the `animation-timing-function` property decides the speed of transition. It behaves the same way as the transition timing function that we have seen earlier.

animation-iteration-count

We can decide the number of iteration carried out in the animation phase using the `animation-iteration-count` property. Setting this property to infinite will mean that the animation will never stop.

animation-direction

We can decide the direction of the animation using the `animation-direction` property. We can use values, such as `reverse`, and `alternate` to define the direction of the element to be animated.

animation-play-state

Using the `animation-play-state` feature, we can determine whether the animation would be running or paused accordingly.

Now that we had a look at these properties, we will now incorporate some of these properties in a code and understand the functionality in a better way. Hence, to gain a practical insight, let's look at the following code:

```
<!DOCTYPE html>
<html>
  <head>
    <style>
      div:hover {
        width:200px;
        height:100px;
        border:2px dotted;
        border-radius:5px;
        border-color: navy;
        background:red;
        position:relative;
        animation:packt 5s;
        -webkit-animation:packt 5s; /* Safari and Chrome */
        animation-iteration-count:3;
        animation-direction:alternate;
        animation-play-state:running;
        -webkit-animation-iteration-count:3;
        -webkit-animation-direction:alternate;
        -webkit-animation-play-state:running;
      }

      @keyframes packt {
        0% {background:lime; left:0px; top:0px;}
        25% {background:pink; left:300px; top:0px;}
        50% {background:yellow; left:300px; top:300px;}
        75% {background:silver; left:0px; top:300px;}
        100% {background:lime; left:0px; top:0px;}
      }
    </style>
  </head>
</html>
```

```
@-webkit-keyframes packt {
  0%   {background:lime; left:0px; top:0px;}
  25%  {background:pink; left:300px; top:0px;}
  50%  {background:yellow; left:300px; top:300px;}
  75%  {background:silver; left:0px; top:300px;}
  100% {background:lime; left:0px; top:0px;}
}

</style>
</head>
<body>
  <br>
  <div> PACKT : Always finding a way </div>
</body>
</html>
```

We have used `-webkit` as the prefix in the preceding example, as we are executing the code in Google Chrome. Please use the `-moz` prefix for Firefox and `-o-` for Opera. At the time of writing, Internet Explorer 10 supports this feature whereas the previous versions of IE do not support it.

This code when executed will have three iterations as defined in the code. After three iterations, the animation will stop automatically. The direction is `alternate`, as a result of which the animation would be in a different direction after the first iteration. The play state doesn't include any pauses and hence the element will be moving constantly. We have used the `hover` command and the animation would work once we hover over the `div` element. We have also defined the percentage in keyframes. Hence, the transition will take place as per the colors mentioned with respect to the position set in terms of percentage.

Let's now look at another code example to understand animations better:

```
<!DOCTYPE html>
<html>
  <head>
    <style>
      body {
        background:#000;
        color:#fff;
      }
      #trigger {
        width:100px;
        height:100px;
        position:absolute;
        top:50%;
      }
    </style>
  </head>
  <body>
    <div id="trigger">
    </div>
  </body>
</html>
```

```
margin:-50px 0 0 -50px;
left:50%;
background: black;
border-radius:50px;

/*set the animation*/
/*[animation name] [animation duration] [animation timing
function] [animation delay] [animation iterations count] [animation
direction]*/
animation: glowness 5s linear 0s 5 alternate;
-moz-animation: glowness 5s linear 0s 5 alternate;
/* Firefox */
-webkit-animation: glowness 5s linear 0s 5 alternate;
/* Safari and Chrome */
-o-animation: glowness 5s linear 0s 5 alternate;
/* Opera */
-ms-animation: glowness 5s linear 0s 5 alternate;
/* IE10 */
}
#trigger:hover {
animation-play-state: paused;
-moz-animation-play-state: paused;
-webkit-animation-play-state: paused;
-o-animation-play-state: paused;
-ms-animation-play-state: paused;
}
/*animation keyframes*/
@keyframes glowness {
0% {box-shadow: 0 0 80px orange;}
25% {box-shadow: 0 0 150px red;}
50% {box-shadow: 0 0 70px pink;}
75% {box-shadow: 0 0 50px violet;}
100% {box-shadow: 0 0 100px yellow;}
}

@-moz-keyframes glowness /* Firefox */ {
0% {box-shadow: 0 0 80px orange;}
25% {box-shadow: 0 0 150px red;}
50% {box-shadow: 0 0 70px pink;}
75% {box-shadow: 0 0 50px violet;}
100% {box-shadow: 0 0 100px yellow;}
}

@-webkit-keyframes glowness /* Safari and Chrome */ {
0% {box-shadow: 0 0 80px orange;}
```

```

    25% {box-shadow: 0 0 150px red;}
    50% {box-shadow: 0 0 70px pink;}
    75% {box-shadow: 0 0 50px violet;}
    100% {box-shadow: 0 0 100px yellow;}
}

@-o-keyframes glowness /* Opera */ {
    0% {box-shadow: 0 0 80px orange;}
    25% {box-shadow: 0 0 150px red;}
    50% {box-shadow: 0 0 70px pink;}
    75% {box-shadow: 0 0 50px violet;}
    100% {box-shadow: 0 0 100px yellow;}
}

@-ms-keyframes glowness /* IE10 */ {
    0% {box-shadow: 0 0 20px green;}
    25% {box-shadow: 0 0 150px red;}
    50% {box-shadow: 0 0 70px pink;}
    75% {box-shadow: 0 0 50px violet;}
    100% {box-shadow: 0 0 100px yellow;}
}
</style>
<script>
    // animation started (buggy on firefox)
    $('#trigger').on('animationstart mozanimationstart
    webkitAnimationStart oAnimationStart
    msanimationstart',function() {
    $('p').html('animation started');
    })
    // animation paused
    $('#trigger').on('mouseover',function() {
    $('p').html('animation paused');
    })
    // animation re-started
    $('#trigger').on('mouseout',function() {
    $('p').html('animation re-started');
    })
    // animation ended
    $('#trigger').on('animationend mozanimationend
    webkitAnimationEnd oAnimationEnd
    msanimationend',function() {
    $('p').html('animation ended');
    })
    //iteration count
    var i = 0;

```

```
    $('#trigger').on('animationiteration mozanimationiteration
    webkitAnimationIteration oAnimationIteration
    msanimationiteration',function() {
        i++;
        $('p').html('animation iteration='+i);
    })
</script>
</head>
<body>
    <div id = "trigger"></div>
</body>
</html>
```

The output of the code on execution would be as shown in the following screenshot:



We have used `-webkit` as the prefix in the preceding example, as we are executing the code in Google Chrome. Please use the `-moz` prefix for Firefox and `-o-` for Opera. Comments are added in the code so that we can understand it easily.

Apart from HTML5 and CSS3, we have used a bit of jQuery. Let's go through the animation part of the code to understand it better. In the CSS3 styles, we have mentioned the animation direction as `alternate`, as a result of which the animation would be in a different direction after the first iteration.

We have used the `hover` property. In this code, whenever we hover over the object, the animation is paused. We have also defined the `glowness` of the object in keyframes. We have also mentioned how the color change and defined a `box-shadow` attribute for the animation in keyframes.

We have defined the `script` tag, in which we have included the JavaScript and jQuery code.

We have used the `trigger` attribute. The `trigger()` method triggers a particular event and the default behavior of an event with regards to the chosen elements. We have used the `mouseover` and `mouseout` properties. The `mouseover` and `mouseout` event fires when the user moves the mouse pointer over an element and out of an element respectively. We have used those events in conjunction with the start, end, and pausing of the animation. Therefore, we see how we can create complex animations using CSS3.

We can work wonders with animation and it will get better once it is accepted as a standard. Till then, we have to do with the vendor prefix.

Summary

In this chapter, we discussed CSS3 Transition, Transformation, and Animation in detail. We also looked at the various properties and variations that come along with them. In the next chapter, we will be looking at the tools and utilities that can make web designing quicker and easier.

7

Tools and Utilities in HTML5 and CSS3

There are a lot of tools and utilities available on the web that assist web designers with building HTML5-based websites. Web designers can use these tools and utilities to build robust and complicated websites. These tools have been tested comprehensively and provide immense help to developers designing web pages.

We will cover the following tools and utilities, which are popular and commonly used in the industry:

- Modernizr
- Liveweave
- HTML KickStart
- HTML5 Boilerplate
- The CSS3 Cheat sheet

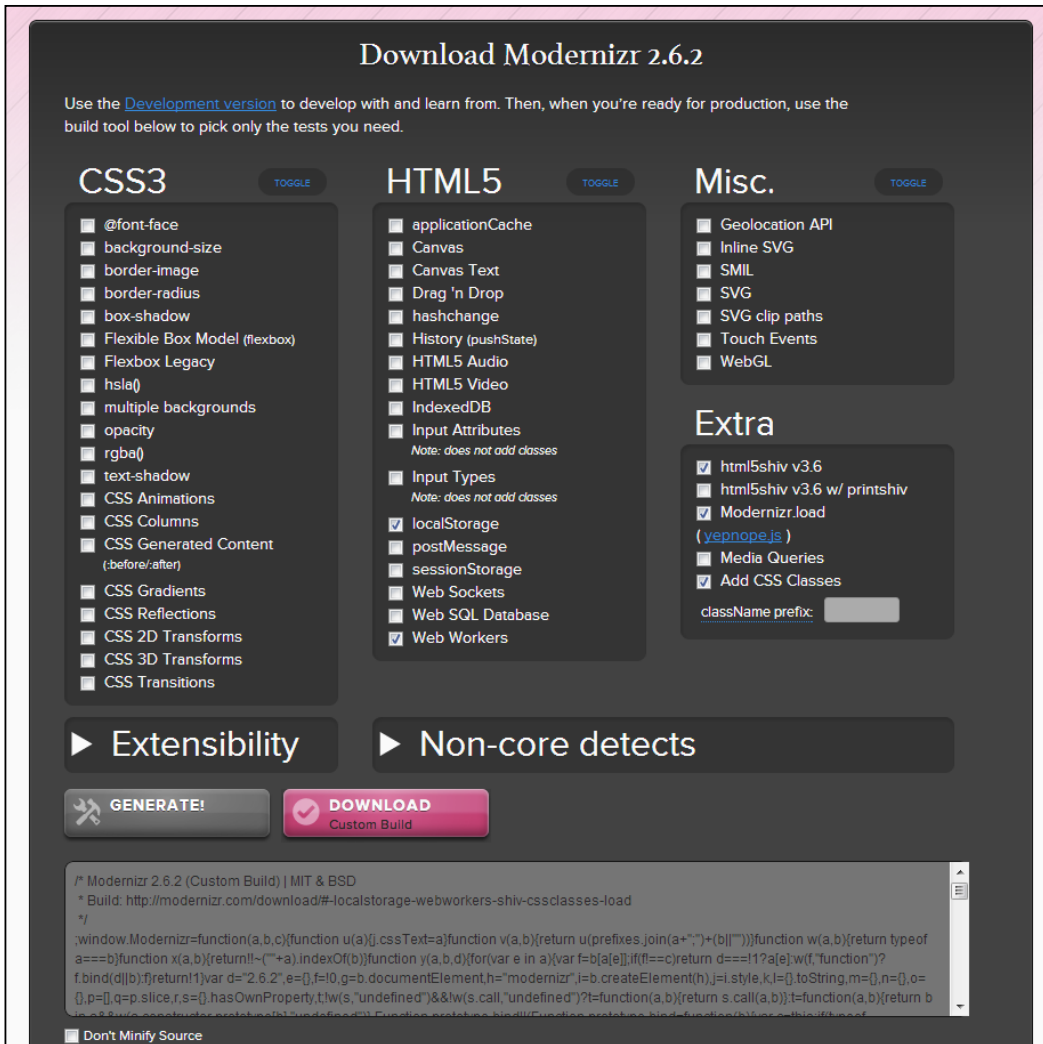
Modernizr

HTML5 native elements and features are not supported completely by all browsers. Browsers such as Google Chrome and Opera somewhat support many features, but not all of them. **Modernizr** helps us in feature detection and gives us information regarding features supported and not supported by these browsers.

A JavaScript object named Modernizr is created as a product of the tests conducted. Classes are added to the HTML elements, denoting the features are supported by the browser in use. Hence, we can write code in a systematic manner, as we know the compatibility metrics.

We can download Modernizr from the following URL <http://modernizr.com/download/>

The following screenshot depicts the customized manner in which we can implement Modernizr:



A custom build can be generated with regards to the features that need to be tested for compatibility with the specific browser.

It also contains the **html5 shiv** element that allows HTML5 features to be incorporated in Internet Explorer, eliminating the need to write a complicated JavaScript code.

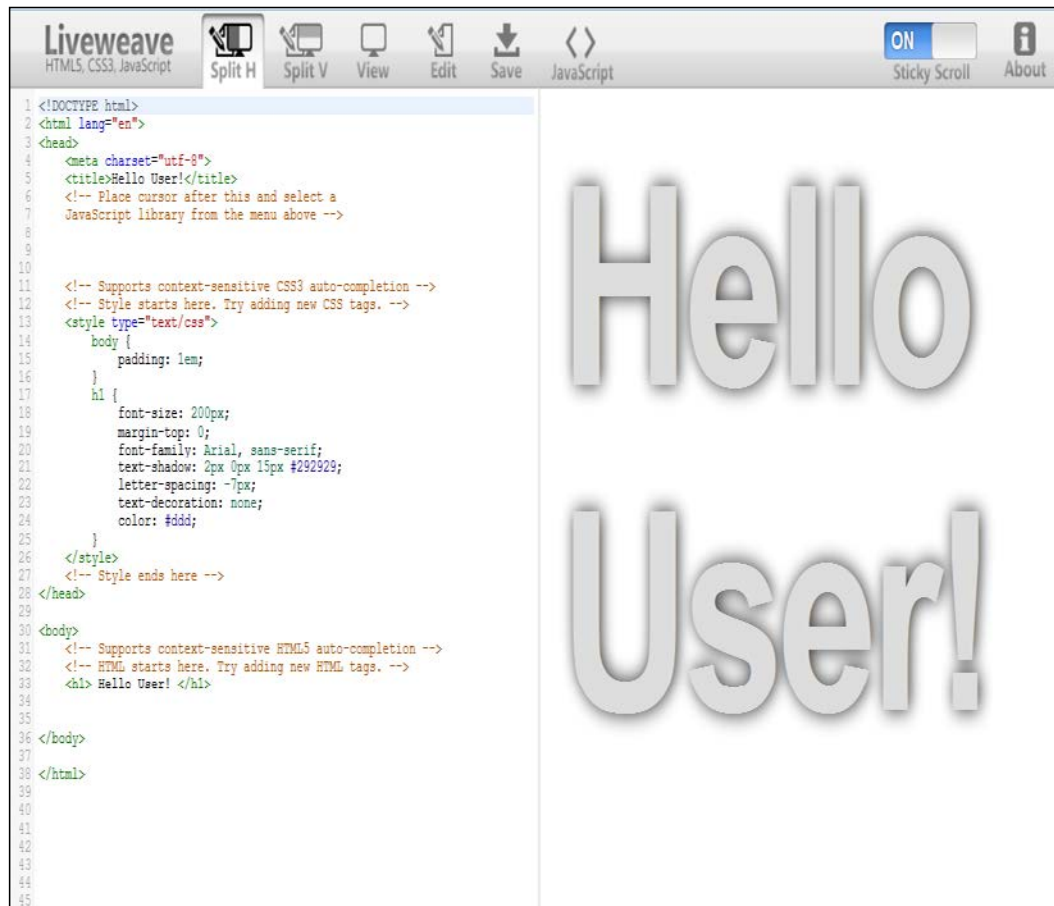
Documentation for Modernizr can be found at <http://modernizr.com/docs/>.

Modernizr is very useful as we foray into evolving web technologies.

Liveweave

Liveweave is an ideal platform to practice the HTML5 and CSS3 code.

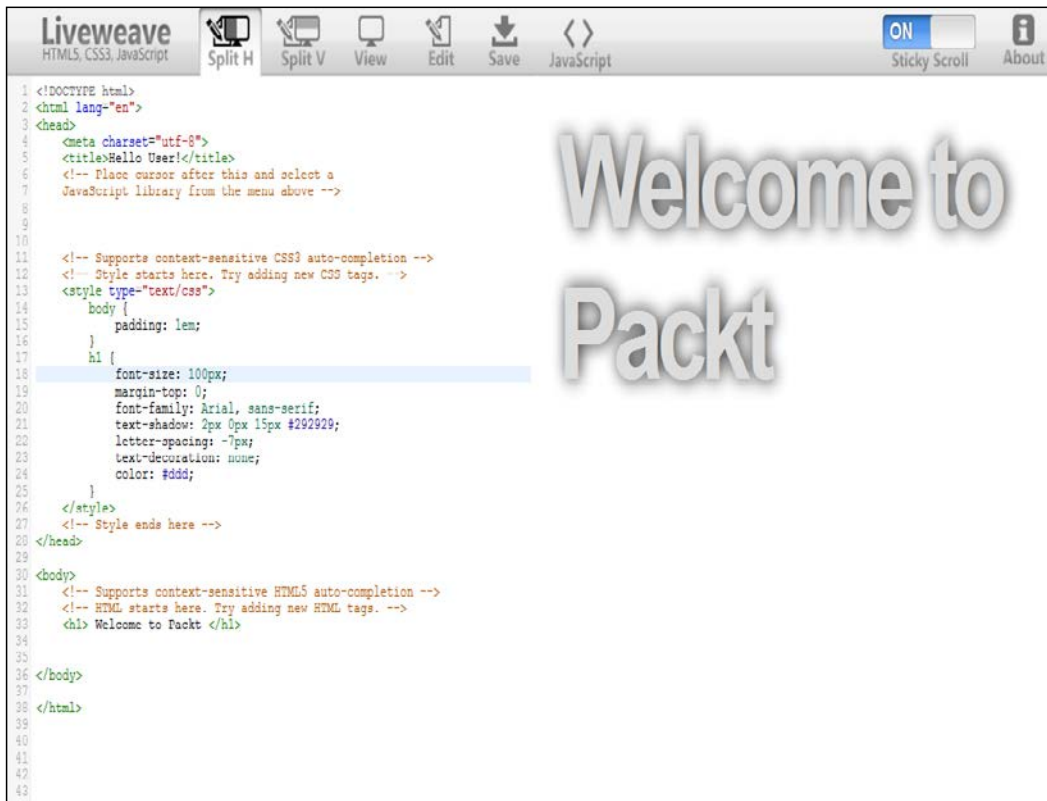
The platform enables us to write the code and execute it as well. The output is generated in milliseconds, and we can modify the code accordingly to suit our purpose. The following screenshot depicts the homepage of Liveweave:



Let's change the code a bit now, by performing the following steps:

1. Click on **Edit** on the screen.
2. Make changes to the code. Let's replace the code within the `body` tag. We will remove **Hello User!** and replace it with **Welcome to Packt**.
3. We will also change the **font-size** value to **100px**, and then save the code.
4. Click on the **Split H** option, which will render the output of the code to the right-hand side. **Split V** is another option where the output will be rendered below the code. As of now, we will click on **Split H**.

Please refer to the following screenshot to view the output:



Referring to the screenshot, **Hello User!** has been replaced with **Welcome to Packt** in the output. We can also notice the change in the **font-size** value of the text. Hence, this platform is ideal to practice HTML5 and CSS3. The main advantage is that developers do not have to execute the code every time, as the output is generated as we make modifications on Liveweave.

HTML KickStart

HTML KickStart helps us in developing websites faster, by providing ready-made layouts and predefined functions. We will have a look at how HTML KickStart works. At the time of writing, it is compatible with Google Chrome, and it is yet to be implemented in Firefox.

Let's look at the following code snippet to see how HTML KickStart works:

```
<!DOCTYPE html>
<html>
  <head>
    <title> HTML KickStart is cool </title>
    <link rel = "stylesheet" type = "text/css" href =
      "C:\Users\Aravind Shenoy\Desktop\HTML5 Tools\
        kickstart\css\kickstart.css" />
    <script type = "text/javascript" src = "C:\Users\Aravind
      Shenoy\Desktop\HTML5 Tools\kickstart\js\
        kickstart.js"></script>
  </head>
  <!-- Menu Horizontal -->
  <ul class = "menu">
    <li class = "current"><a href = "">Login</a></li>
    <li><a href = "">Version of the Application</a></li>
    <li><a href = "">Downloads</a></li>
    <li><a href = "">About us</a></li>
    <li><a href = "">Contact us</a></li>
    <li><a href = "">Customer Care info</a></li>
  </ul>
</html>
```

I have stored the KickStart .css and .js files in the following folders respectively:



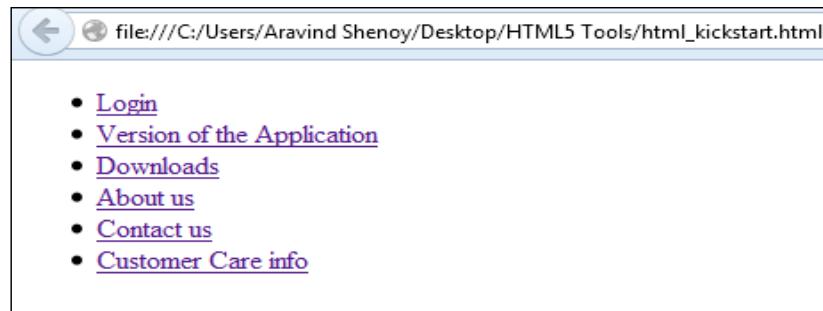
C:\Users\Aravind Shenoy\Desktop\HTML5 Tools\
kickstart\css\kickstart.css

C:\Users\Aravind Shenoy\Desktop\HTML5 Tools\
kickstart\js\kickstart.js

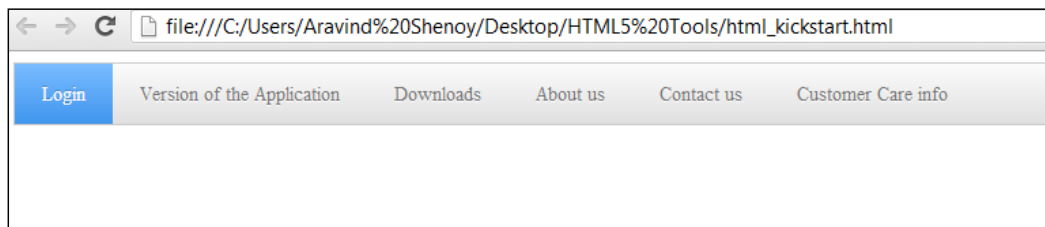
The path may be changed with regards to the location where you store the KickStart files.

The path of the source, where the KickStart file is stored, must be mentioned in the `link` and `script` tags.

Suppose we do not use the preceding code snippet related to KickStart, then the output would be displayed as shown in the following screenshot:



However, if we use the HTML KickStart code snippet in the code, the output will be as displayed in the following screenshot:



As we can see in the preceding screenshot, the output is on a horizontal axis and looks dapper due to preconfigured styles. Hence, we can use KickStart to build a website faster and in an efficient manner.

The HTML KickStart files can be downloaded from <http://www.99lime.com/>.

HTML5 Boilerplate

HTML5 Boilerplate is a package which contains most of the essential items that a web designer would need to build a web app or website in HTML5.

It can be downloaded from <http://html5boilerplate.com/>. You can also download a custom build, as per your requirement. After you download and extract the ZIP file, you can see the content, as shown in the following screenshot:

Name	Date modified	Type	Size
css	2/4/2013 1:24 PM	File Folder	
doc	2/4/2013 1:24 PM	File Folder	
img	2/4/2013 1:24 PM	File Folder	
js	2/4/2013 1:24 PM	File Folder	
.gitattributes	2/4/2013 1:24 PM	GITATTRIBUTES File	1 KB
.gitignore	2/4/2013 1:24 PM	GITIGNORE File	1 KB
.htaccess	2/4/2013 1:24 PM	HTACCESS File	20 KB
404	2/4/2013 1:24 PM	Firefox HTML Doc...	5 KB
apple-touch-icon	2/4/2013 1:24 PM	PNG Image	1 KB
apple-touch-icon-57x5...	2/4/2013 1:24 PM	PNG Image	1 KB
apple-touch-icon-72x7...	2/4/2013 1:24 PM	PNG Image	1 KB
apple-touch-icon-114x...	2/4/2013 1:24 PM	PNG Image	2 KB
apple-touch-icon-144x...	2/4/2013 1:24 PM	PNG Image	2 KB
apple-touch-icon-prec...	2/4/2013 1:24 PM	PNG Image	1 KB
CHANGELOG.md	2/4/2013 1:24 PM	MD File	6 KB
CONTRIBUTING.md	2/4/2013 1:24 PM	MD File	4 KB
crossdomain	2/4/2013 1:24 PM	XML Document	1 KB
favicon	2/4/2013 1:24 PM	Icon	1 KB
humans	2/4/2013 1:24 PM	Text Document	1 KB
index	2/4/2013 1:24 PM	Firefox HTML Doc...	2 KB
LICENSE.md	2/4/2013 1:24 PM	MD File	2 KB
README.md	2/4/2013 1:24 PM	MD File	3 KB
robots	2/4/2013 1:24 PM	Text Document	1 KB

The preceding screenshot displays all the files in Boilerplate.

CSS files include basic `css` that assist you in defining styles, and our project's CSS files will be stored here.

The `doc` folder is used to store all the project documentation in addition to the HTML5 Boilerplate documentation.

The `js` folder contains jQuery and JavaScript libraries, as well as the code that is a part of the project.

The **apple-touch-icon** is customized for Apple iOS.

The .htaccess file aids in web server configuration related to Apache.

We even have a crossdomain facility, which allows us to handle data across various domains.

In addition to this, Modernizr and html shiv are also an integral part of HTML5 Boilerplate.

The entire documentation is available at

<https://github.com/h5bp/html5-boilerplate/blob/v4.1.0/doc/TOC.md>

The CSS3 Cheat sheet

The CSS3 Cheat sheet will surely assist the user, as it defines a list of all the styles in CSS3. It can be used as a reference, as it will be handy while we are writing the CSS3 code.

The Cheat sheet can be found at

<http://media.smashingmagazine.com/wp-content/uploads/images/css3-cheat-sheet/css3-cheat-sheet.pdf>

The following screenshot is a preview of how the Cheat sheet looks:

Cascading Style Sheets (CSS 3)		
BACKGROUND	BORDER	BOX MODEL
background	border-top	float
background-image background-position background-size background-repeat background-attachment background-origin background-clip background-color	border-top-width border-style border-color	left right none
background-attachment	border-top-color	height
scroll fixed	border-style	auto length %
background-break	border-top-width	max-height
bounding-box each-box continuous	thin medium thick length	none length %
background-clip	border-width	max-width
length % border-box padding-box content-box no-clip	thin medium thick length	none length %
background-color	border-radius	min-height
color transparent	border-top-right-radius border-bottom-right-radius border-bottom-left-radius border-top-left-radius	none inherit length %
background-image	border-top-right-radius	min-width
url none	length	none inherit length %
background-origin	border-bottom-right-radius	width
border-box padding-box content-box	length	auto % length
background-position	border-bottom-left-radius	margin
top left top center top right center left center center center right bottom left bottom center bottom right x-% y-% x-pos y-pos	border-top-left-radius	margin-top margin-right margin-bottom margin-left
	box-shadow	margin-bottom
	inset [length length length length <color>] none	auto length %
	border-style	margin-left
	none hidden dotted dashed solid double groove ridge inset outset	auto

Designers can view the CSS3 properties and the values that come along with it, thereby applying it as per the requirement.

Summary

We had a look at some of the tools and utilities in HTML5 and CSS3, which make web designing easier. Personally speaking, I suggest that you copy the code into an editor, such as Notepad or Notepad++, and alter it to see the varied output.

Coding is an art, which gets better with practice. Hence, we need to implement it practically, in order to know the subtle nuances of HTML5 and CSS3. However, we can achieve that after a considerable amount of practice. We have covered a lot of HTML5 and CSS3 features in this book.

However, we are just on the shore; the sea of knowledge is far beyond.

You can check out the Packt website at www.packtpub.com, which has a lot of books on HTML5 and CSS3, which are customized to suit your needs.

The following are some of the books we have at Packt Pub on HTML5 and CSS3 to mention a few:

- *HTML5 Boilerplate Web Development*
- *HTML5 Canvas Cookbook*
- *Responsive Web Design with HTML5 and CSS3*

Please check out our website for further details.

You can alternatively visit the Packt online library at <http://packtlib.packtpub.com/>, where you will gain access to various books and articles.

Index

Symbols

`<div>` tag 14
`@keyframes` property 98

A

advanced features, HTML5
about 53
audio 54, 55
canvas element 70, 71
drag-and-drop 58-60
geolocation 60-63
offline web applications 67-69
video 54, 55
webstorage 63

align-items property 29, 30
animation 80, 82
animation-delay feature 98
animation-direction property 99
animation-duration feature 98
animation-iteration-count property 98
animation-name property 98
animation-play-state feature 99-103
animation-timing-function property 98
arc() method 72
article element 18, 19
aside element 19
audio 54
autofocus attribute 39, 40

B

beginPath() method 72
Bing 14

C

canvas element
about 70, 71
arc() method 72
beginPath() method 72
closePath() method 72
fill() method 72
gradients 74, 75
lineTo() method 73, 74
moveTo() method 72
restore() method 76
save() method 75
stroke function 72

clearRect property 71
closePath() method 72
color input type 50
CSS 7
CSS3 23
CSS3 animation
`@keyframes` property 98
about 97
animation-delay feature 98
animation-direction property 99
animation-duration feature 98
animation-iteration-count property 98
animation-name property 98
animation-play-state feature 99-103
animation-timing-function 98

CSS3 Cheat sheet
about 112
structure 112

CSS3 transforms
about 91
preserve-3d feature 96, 97
rotate (3D) feature 96

- rotate feature 91
- scale feature 92
- skew feature 93, 95
- translate (3D) feature 95
- translate feature 92

CSS3 transitions

- about 84-86
- transition-delay property 90, 91
- transition-duration property 86-88
- transition-timing-function property 89, 90

D

- datalist attribute** 42, 43
- date input type** 46, 47
- drag-and-drop feature** 58-60
- dragenter event** 60
- dragleave event** 60
- dragstart event** 60
- drawArc function** 74

E

- email input type** 45

F

- features, HTML5** 9, 10
- fill() method** 72
- fillRect property** 71
- Flash player** 54
- Flexbox**
 - about 23, 24
 - working 25
- Flex Container**
 - about 24
 - properties 26
- flex-direction property** 26-28
- Flexible Box Model** 23
- Flex Items**
 - properties 33
- Flex Line** 24
- flex property** 35
- flex-wrap property** 31-33
- footer element** 15
- form attributes, HTML5**
 - autofocus 39, 40
 - datalist 42, 43

- placeholder 38
- required 40, 42

G

- geolocation** 60-63

global attributes, HTML5

- itemid 21
- itemprop 21
- itemref 21
- itemscope 21
- itemtype 21

- Google Chrome** 105

- Google Search** 14

- Google Structured Data Testing Tool** 22

- gradients** 74, 75

H

- header element** 14

HTML5

- advanced features 53
- features 9, 10
- form attributes 38-42
- input types 43-50
- misconceptions 10, 11
- sectioning elements 13
- used, for developing video player 56, 57
- versus HTML 4 8, 9

HTML

- about 7
- standardizing 7, 8
- versions 7

HTML 4

- versus HTML5 8, 9

HTML5 Boilerplate

- about 110, 111
- URL, for documentation 112
- URL, for downloading 111

- html5 shiv element** 107

HTML KickStart

- about 109
- URL, for downloading files 110
- working 109, 110

HTTP 7

- HyperText Markup Language.** *See* **HTML**

I

input types, HTML5

- color 50
- date 46, 47
- email 45
- month 48, 49
- search 43, 44
- url 45
- week 47

Intermedia 7

itemid attribute 21

itemprop attribute 21

itemref attribute 21

itemscope attribute 21

itemtype attribute 21

J

justify-content property 28, 29

L

linear gradients 74

lineTo() method 73, 74

Liveweave 107, 108

localStorage property 65, 66

M

Math function 73

Microdata 20-22

Modernizr

- about 105
- URL, for documentation 107
- URL, for downloading 106

month input type 48, 49

moveTo() method 72

N

nav element 16, 17

navigator.geolocation function 62

O

offline web applications 67-69

Opera 105

order property 33, 34

P

placeholder attribute 38

preserve-3d feature 96, 97

properties, Flex Container

- align-items 29, 30
- flex-direction 26-28
- flex-wrap 31-33
- justify-content 28, 29

properties, Flex Items

- flex 35
- order 33, 34

R

required attribute 40, 42

restore() method 76

rotate (3D) feature 96

rotate feature 91

rotate() method 78, 79

S

save() method 75

scale feature 92

scale property 79, 80

search engine optimization. *See* SEO

search input type 43

sectioning 13

sectioning elements, HTML5

- article 18, 19
- aside 19
- footer 15
- header 14
- nav 16-18

SEO 14

sessionStorage property 63, 65

SGML 7

skew feature 93, 95

Standard Generalized Markup

Language. *See* SGML

standardization process, HTML 7, 8

stroke function 72

strokeRect property 71

switch command 62

T

transformations

- about 77
- rotate() method 78, 79
- scale property 79, 80
- translate property 77, 78

transition-delay property 90, 91

transition-duration property 86-88

transition-property 84

transition-timing-function property 89, 90

translate (3D) feature 95

translate feature 92

translate property 77, 78

trigger() method 104

U

url input type 45

V

video 54

video player

- developing, HTML5 used 56, 57

W

watchPosition method 63

Web Hypertext Application Technology Working Group. *See* WHATWG

Webstorage

- about 63
- localStorage property 65, 66
- sessionStorage property 63, 65

week input type 47

WHATWG 8

World Wide Web Consortium (W3C) 8

wrap-reverse property 32

Z

ZOG 7



Thank you for buying HTML5 and CSS3 Transition, Transformation, and Animation

About Packt Publishing

Packt, pronounced 'packed', published its first book "*Mastering phpMyAdmin for Effective MySQL Management*" in April 2004 and subsequently continued to specialize in publishing highly focused books on specific technologies and solutions.

Our books and publications share the experiences of your fellow IT professionals in adapting and customizing today's systems, applications, and frameworks. Our solution based books give you the knowledge and power to customize the software and technologies you're using to get the job done. Packt books are more specific and less general than the IT books you have seen in the past. Our unique business model allows us to bring you more focused information, giving you more of what you need to know, and less of what you don't.

Packt is a modern, yet unique publishing company, which focuses on producing quality, cutting-edge books for communities of developers, administrators, and newbies alike. For more information, please visit our website: www.packtpub.com.

About Packt Open Source

In 2010, Packt launched two new brands, Packt Open Source and Packt Enterprise, in order to continue its focus on specialization. This book is part of the Packt Open Source brand, home to books published on software built around Open Source licences, and offering information to anybody from advanced developers to budding web designers. The Open Source brand also runs Packt's Open Source Royalty Scheme, by which Packt gives a royalty to each Open Source project about whose software a book is sold.

Writing for Packt

We welcome all inquiries from people who are interested in authoring. Book proposals should be sent to author@packtpub.com. If your book idea is still at an early stage and you would like to discuss it first before writing a formal book proposal, contact us; one of our commissioning editors will get in touch with you.

We're not just looking for published authors; if you have strong technical skills but no writing experience, our experienced editors can help you develop a writing career, or simply get some additional reward for your expertise.



Responsive Web Design with HTML5 and CSS3

ISBN: 978-1-84969-318-9 Paperback: 324 pages

Learn responsive design using HTML5 and CSS3 to adapt websites to any browser or screen size

1. Everything needed to code websites in HTML5 and CSS3 that are responsive to every device or screen size
2. Learn the main new features of HTML5 and use CSS3's stunning new capabilities including animations, transitions and transformations
3. Real world examples show how to progressively enhance a responsive design while providing fall backs for older browsers



Developing Windows Store Apps with HTML5 and JavaScript

ISBN: 978-1-84968-710-2 Paperback: 184 pages

Learn the key concepts of developing Windows Store apps using HTML5 and JavaScript

1. Learn about the powerful new features in HTML5 and CSS3
2. Quick start a JavaScript app from scratch
3. Get your app into the store and learn how to add authentication



Mobile First Design with HTML5 and CSS3

ISBN: 978-1-84969-646-3 Paperback: 122 pages

Roll out rock-solid, responsive, mobile first designs quickly and reliably

1. Make websites that will look great and be usable on almost any device that displays web pages
2. Learn best practices for responsive design
3. Discover how to make designs that will be lean and fast on small screens without sacrificing a tablet or desktop experience



Dreamweaver CS6 Mobile and Web Development with HTML5, CSS3, and jQuery Mobile

ISBN: 978-1-84969-474-2 Paperback: 268 pages

Harness the cutting-edge features of Dreamweaver for mobile and web development

1. A basic, compressed, updated introduction to building advanced web sites with Dreamweaver
2. A focused exploration of employing cutting edge HTML5 techniques such as native media
3. An in-depth explanation of how to build inviting, accessible mobile sites with Dreamweaver CS6, responsive design, and jQuery Mobile

Please check www.PacktPub.com for information on our titles