HTML5 Programming Cookbook

Contents

1 HTML5 Drag and Drop **1**

1.1 Setting up two divs . . . 1

1.2 Make the elements draggable . . . 2

1.3 Set up events to handle drag functionality . . . 2

1.4 Completing the drop and putting it all together . . . 3

1.5 Conclusion and further applications . . . 5

2 HTML5 File Upload Example **6**

2.1 Show File(s) information . . . 6

2.1.1 A single file . . . 6

2.1.2 Multiple files . . . 6

2.1.3 Previewing Files . . . 7

2.2 Upload The files . . . 8

2.2.1 The HTML . . . 9

2.2.2 The JavaScript . . . 9

2.2.3 The PHP script . . . 9

2.3 Download . . . 9

3 HTML5 Dropdown Menu With CSS3 **10**

3.1 Introduction . . . 10

3.2 HTML . . . 11

3.3 CSS . . . 11

3.4 Download the source code . . . 14

4 HTML5 Audio Player **15**

4.1 The minimal example . . . 15

4.2 Show the controls . . . 15

4.3 Tag Attributes . . . 17

4.3.1 Controls . . . 17

4.3.2 Autoplay . . . 18

4.3.3 Loop . . . 18

4.3.4 Preload . . . 18
4.4 Control the audio with Javascript . . . 18
4.4.1 Basic Play / Pause example . . . 18
4.4.2 The HTMLMediaElement . . . 19
4.4.2.1 Properties . . . 19
4.4.2.2 Methods . . . 19
4.5 Use Media Events . . . 20
4.5.1 Example . . . 20
4.6 Playlist Example . . . 21
4.6.1 The Specifications . . . 21
4.6.2 The code 21
4.7 Conclusion . . . 22

5 HTML5 Local Storage 23
5.1 Introduction . . . 23
5.2 Local Storage . . . 24
5.3 Session Storage . . . 26
5.4 Key points . . . 27
5.5 Download the Source Code . . . 27

6 HTML5 Graphics and Animation 28
6.1 Introduction . . . 28
6.2 Canvas element and context . . . 29
6.3 Draw a Graph . . . 29
6.4 Draw a Line . . . 30
6.5 Draw Arc . . . 30
6.6 Draw some more stuff . . . 31
6.7 Introducing requestAnimationFrame . . . 32
6.8 Download the source code . . . 34

7 HTML5 Offline Web Application 35
7.1 The Manifest . . . 35
7.2 Manifest Sections . . . 36
7.3 Application Cache API . . . 37
7.3.1 Events . . . 37
7.3.2 Properties . . . 37
7.3.3 Methods . . . 37
7.4 The online / offline events . . . 38
7.5 A Working Example . . . 38
7.5.1 Project structure . . . 38
7.5.2 The server.php file . . . 38
7.5.3 The main HTML file . . . 40
7.5.4 The JavaScript . . . 41
7.6 Download . . . 43

8 HTML5 Geolocation **44**

8.1 Introduction . . . 44
8.2 Security And Accuracy . . . 45
8.3 Weather Widget . . . 45
8.4 getCurrentPosition and watchPosition . . . 48
8.5 Position . . . 48
8.6 Handling JSON . . . 49
8.7 Download . . . 50
8.8 Reference . . . 50

9 HTML5 Form Validation **51**

9.1 Introduction . . . 51
9.1.1 min and max . . . 53
9.1.2 datalist . . . 54
9.1.3 placeholder . . . 55
9.1.4 autofocus . . . 55
9.1.5 pattern . . . 55
9.1.6 date, datetime-local,month,time,week . . . 56
9.1.7 email . . . 57
9.1.8 url . . . 58
9.1.9 color . . . 58

Copyright (c) Exelixis Media P.C., 2015

All rights reserved. Without limiting the rights under
copyright reserved above, no part of this publication
may be reproduced, stored or introduced into a retrieval system, or
transmitted, in any form or by any means (electronic, mechanical,
photocopying, recording or otherwise), without the prior written
permission of the copyright owner.

Preface

HTML5 is a core technology markup language of the Internet used for structuring and presenting content for the World Wide Web. As of October 2014 this is the final and complete fifth revision of the HTML standard of the World Wide Web Consortium (W3C). The previous version, HTML 4, was standardised in 1997.

Its core aims have been to improve the language with support for the latest multimedia while keeping it easily readable by humans and consistently understood by computers and devices (web browsers, parsers, etc.). HTML5 is intended to subsume not only HTML 4, but also XHTML 1 and DOM Level 2 HTML. (Source: http://en.wikipedia.org/wiki/HTML5)

In this ebook, we provide a compilation of HTML5 based examples that will help you kick-start your own web projects. We cover a wide range of topics, from graphics and animation, to geolocation and offline storage. With our straightforward tutorials, you will be able to get your own projects up and running in minimum time.

About the Author

WCGs (Web Code Geeks) is an independent online community focused on creating the ultimate Web developers resource center; targeted at the technical architect, technical team lead (senior developer), project manager and junior developers alike.

WCGs serve the Web designer, Web developer and Agile communities with daily news written by domain experts, articles, tutorials, reviews, announcements, code snippets and open source projects.

You can find them online at http://www.webcodegeeks.com/

Chapter 1

HTML5 Drag and Drop

Usability, an important part of web interface eases the way we communicate with web. Many new technologies and functionalities are invented to ease the development effort as well as improve the overall way in which users interact with web.

HTML5 has given many things as of today to improve the browser functionalities on client side with minimum amount of scripting. It provides a great way to implement drag and drop functionality in modern browsers. We are going to see how it is implemented with a basic example of dragging and dropping a image from one div to another.

To achieve drag and drop functionality with traditional HTML4, developers would have to use complex Javascript code. HTML 5 provides a Drag and Drop API that brings support to the browser making it much easier to code up. No extra plugins needed to be installed. It is supported by the following major browsers:

- Internet Explorer 9+
- Firefox
- Opera, Chrome
- Safari

Note: Drag and drop does not work in Safari 5.1.7 and earlier versions.

1.1 Setting up two divs

We will first code two create two div boxes. One div will contain the image to be dragged. The other div will be the destination where the image needs to be dragged.

```
<!DOCTYPE html>
<html>
<head>
    <style>
        #div1, #div2
        {float:left; width:280px; height:180px; margin:10px;padding:10px;border:1px solid # ↩
            aaaaaa;}
    </style>
</head>
<body>


 <img src="drag.png" width="250" height="150" id="drag1">
```

```
</body>
</html>
```

Output for above code is two div boxes with one div box containing our image

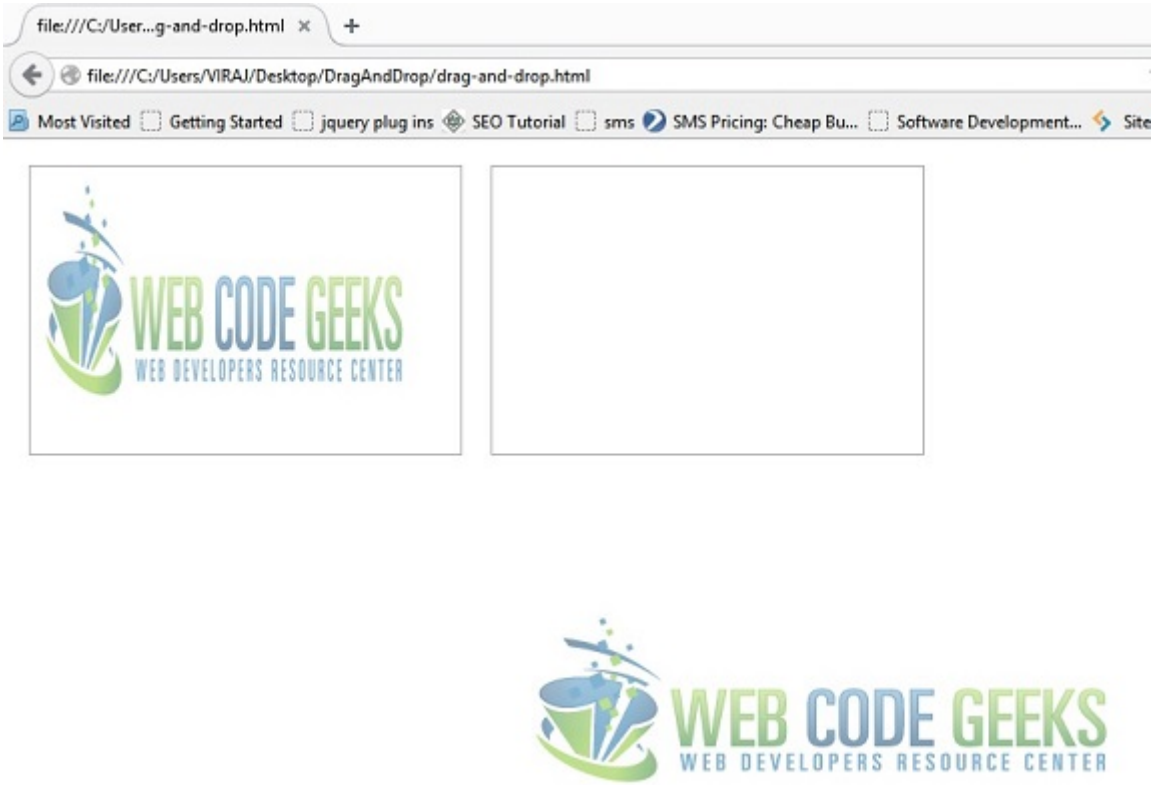

Figure 1.1: Initial divs

1.2 Make the elements draggable

Now, we need to first make the image draggable. Set the attribute `"draggable =true"`

```
<img src="drag.png" draggable="true" width="250" height="150" id="drag1">
```

1.3 Set up events to handle drag functionality

Set the `ondragstart` event in the `img` tag to call `dragInitiliaze()` function as follows :

```
<img src="drag.png" draggable="true" ondragstart="return dragInitialize(event)" width="250" ↩
    height="150" id="drag1">
```

The `ondragstart` event in `img` tag detects when the drag is initialized and then it calls the `dragInitiate()` function. The `dragInitiate()` function, then catches the event. It sets the `effectAllowed` value to "move" and has `dataTransfer.setData()` method which sets the data type and the value of the dragged data.

```
<script type="text/javascript">
```

```
function dragInitialize(ev) {
ev.dataTransfer.effectAllowed='move';
ev.dataTransfer.setData("Text", ev.target.getAttribute('id'));
return true;
}

</script>

<!-- In body add the following draggable attributes to first div -->


<img src="drag.png" draggable="true" ondragstart="return dragInitialize(event)" width="250" ↩
    height="150" id="drag1">
```

Figure 1.2: Dragging Image from one div to other

1.4 Completing the drop and putting it all together

By default, the elements that are set to be `draggable` cannot be dropped in any other elements. The drop functionality needs to be handled by events provided by Drag-and-Drop API. We have to take care of following things :

```
function allowDropStatus(ev) {
ev.preventDefault();
return false;
}

function dropComplete(ev) {
ev.preventDefault();
var src = ev.dataTransfer.getData("Text");
ev.target.appendChild(document.getElementById(src));
ev.stopPropagation();
return false;
}

<!-- Modify the 2nd div to accept the drop -->
```

- The div should listen to drop event so that it can accept the drop and place the image in its destination.
- `ondragover` listener event is fired whenever a dragged image is over the destination div
- `allowDropStatus()` prevents the default browser action so that we can code and handle the drop functionality.
- `dropComplete()` function does following three tasks :
- Prevents default browser action after the image has been dropped
- Fetches the data of image from `getData` which was stored while the image was selected for drag
- Appends the data to new div
- Stops the propagation of image

If you observe carefully, we can drag the image from first div to second. But, what if we want to drag the image back to first div. The image is set to `draggable`, so it will be dragged. But, our first div is not set to handle drop. Let's modify our first div so that it can handle the drop.

We put following two listeners in first div to accept drop :

- `ondragover` listener which calls `allowDropStatus` function
- `ondrop` listener which calls `dropComplete` function

```
<img src="drag.png" draggable="true" ondragstart="return dragInitialize(event)" width="250" ↩
     height="150" id="drag1">
```

Figure 1.3: Completing drop in second div

<a href="http://www.webcodegeeks.com/wp-content/uploads/2014/11/2.jpg"> </a>This completes our simple example for Drag and Drop. It is totally based on handling of events and listeners which are provided by native HTML5 API

1.5 Conclusion and further applications

So, now we can drag images back and forth efficiently. Drag and Drop functionality has numerous uses in improving the overall usability. Using the logic presented above and a glimpse of how various events of Drag-and-Drop API can be used, you can extend them to use and apply for any other functionality. As with any technology, HTML 5 Drag-and-Drop API has its own advantages and disadvantages. Its upto you too whether use it or not.

Download: You can download the full source code of this example here : **HTML 5 Drag and Drop**

Chapter 2

HTML5 File Upload Example

In this example we'll explain how to use HTML 5, to read information about file(s) selected by users, and to upload the file(s) on a server.

The FileApi is one of the most interesting features added with HTML 5. Now we can display file(s) information **before** they are uploaded on the server, and we can send them without "**posting**" a form.

We'll see how to access file(s) information when they are selected by users, and then upload them asynchronously using an Ajax Request.

2.1 Show File(s) information

2.1.1 A single file

Access information of a single file selected by the user.

Here is the HTML code :

```
<input type="file" id="fileinput" />
```

When the user choose a file, the "**change**" event is fired on the input element, so we can listen for it :

```
document.getElementById('fileinput').addEventListener('change', function(){
    var file = this.files[0];
    // This code is only for demo ...
    console.log("name : " + file.name);
    console.log("size : " + file.size);
    console.log("type : " + file.type);
    console.log("date : " + file.lastModified);
}, false);
```

As you can see, the FileApi is quite simple to use, it adds the **"files"** property on the HTMLInputElement.

Note: The "files" property is not writable, you can only read its content.

As you may have noticed, we retrieved the chosen file, by accessing the index 0 of the FileList collection : `this.files[0]`.

2.1.2 Multiple files

Now we'll display information about all the files selected by the user.

Here is the HTML code :

```
<input type="file" id="fileinput" multiple="multiple" />
```

We've just added the **`multiple="multiple"`** attribute to the HTML element, this will allow user to choose multiple files to upload.

```
document.getElementById('fileinput').addEventListener('change', function(){
    for(var i = 0; i<this.files.length; i++){
        var file =  this.files[i];
        // This code is only for demo ...
        console.group("File "+i);
        console.log("name : " + file.name);
        console.log("size : " + file.size);
        console.log("type : " + file.type);
        console.log("date : " + file.lastModified);
        console.groupEnd();
    }
}, false);
```

Note : you can filter elements by adding the **"accept"** attribute to the input element.

For example, if you only want your user to upload some images, you can filter on the "image/*" mime-type :

```
<input type="file" id="fileinput" multiple="multiple" accept="image/*" />
```

2.1.3 Previewing Files

As we can read the file(s) information, we can also read the content of the file, this, for example, can allow us to preview selected files before upload. Let's see an example with previewing images.

```
<!doctype html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Preview images</title>
    <style>
        #gallery .thumbnail{
            width:150px;
            height: 150px;
            float:left;
            margin:2px;
        }
        #gallery .thumbnail img{
            width:150px;
            height: 150px;
        }

    </style>
</head>
<body>

<h2>Upload images ...</h2>

<input type="file" id="fileinput" multiple="multiple" accept="image/*" />


<script src="gallery.js"></script>
</body>
</html>
```

The JavaScript code to manage the uploaded files:

gallery.js

```
var uploadfiles = document.querySelector('#fileinput');
uploadfiles.addEventListener('change', function () {
    var files = this.files;
    for(var i=0; i<files.length; i++){
        previewImage(this.files[i]);
    }

}, false);
```

And, the `previewImage` function that will display the image(s) selected by the user.

gallery.js

```
function previewImage(file) {
    var galleryId = "gallery";

    var gallery = document.getElementById(galleryId);
    var imageType = /image.*/;

    if (!file.type.match(imageType)) {
        throw "File Type must be an image";
    }

    var thumb = document.createElement("div");
    thumb.classList.add('thumbnail'); // Add the class thumbnail to the created div

    var img = document.createElement("img");
    img.file = file;
    thumb.appendChild(img);
    gallery.appendChild(thumb);

    // Using FileReader to display the image content
    var reader = new FileReader();
    reader.onload = (function(aImg) { return function(e) { aImg.src = e.target.result; }; ↩
        })(img);
    reader.readAsDataURL(file);
}
```

Here we introduced the `FileReader` object, that allow us to asynchronously read the contents of files.

Instantiate the object with the `new FileReader()`, then tell the object to read the data from the file with the method `readAsUrl`.

The `onload` method is called after the content is read by the object like an event, then the content of the file is assigned to the image src attribute: `aImg.src =e.target.result;`

2.2 Upload The files

Now we will upload files using **XMLHttpRequest** (Ajax).

For each files selected by the user we will create an HTTP request to send the file to the server.

First create a function to upload a file using **XMLHttpRequest** :

```
function uploadFile(file){
    var url = 'server/index.php';
    var xhr = new XMLHttpRequest();
    var fd = new FormData();
```

```
    xhr.open("POST", url, true);
    xhr.onreadystatechange = function() {
        if (xhr.readyState == 4 && xhr.status == 200) {
            // Every thing ok, file uploaded
            console.log(xhr.responseText); // handle response.
        }
    };
    fd.append("upload_file", file);
    xhr.send(fd);
}
```

This function will create an ajax request (POST) on the **url** and send the file in the **"upload_file"** request parameter (we may access this parameter with the **`$_FILES['upload_file']`** variable.

Now we'll connect the `uploadFile` function to the javascript that manage the selected files :

2.2.1 The HTML

```
<input type="file" id="uploadfiles" multiple="multiple" />
```

2.2.2 The JavaScript

```
var uploadfiles = document.querySelector('#uploadfiles');
uploadfiles.addEventListener('change', function () {
    var files = this.files;
    for(var i=0; i<files.length; i++){
        uploadFile(this.files[i]); // call the function to upload the file
    }
}, false);
```

2.2.3 The PHP script

```
if (isset($_FILES['upload_file'])) {
    if(move_uploaded_file($_FILES['upload_file']['tmp_name'], "datas/" . $_FILES[' ↩
        upload_file']['name'])){
        echo $_FILES['upload_file']['name']. " OK";
    } else {
        echo $_FILES['upload_file']['name']. " KO";
    }
    exit;
} else {
    echo "No files uploaded ...";
}
```

2.3 Download

Download: You can download the full source code of this example here : **HTML5 File Upload Example**

Chapter 3

HTML5 Dropdown Menu With CSS3

3.1 Introduction

The web has evolved into something more than just linked documents; pages behave increasingly these days like apps. Even a few years ago we would be looking at using JavaScript to create interactive, attractive menus.

With HTML5 and CSS3 now standard in most modern browsers, web developers can easily and quickly create attractive and responsive menus. Today we will see how we can leverage HTML5 and CSS3 to create a simple navigation menu.

Here's a screenshot of what we'll be creating in this tutorial:

Figure 3.1: HTML5 Menu screenshot

This following browser versions or higher are supported by this example

- Google Chrome 6.0
- Internet Explorer 9.0
- Mozilla Firefox 4.0
- Safari 5.0
- Opera 11.1

3.2 HTML

HTML5 has many new elements/tags for creating navigation menu like <menu> & <menuitems>. Owing to lack of support in major browsers for these new elements <menu> & <menuitems> we will be using the <nav>.

An important part of creating a site navigation is understanding that the navigation is a list of links around your site which means that we will use an unordered list to store your links, not a table! We will also see how we can use the html character code (») to our advantage. Alternately, you can use a image to display as arrow.

The <head> html tag act as container to include the title,scripts , meta information and also link the relevant external resource. The <meta> tag is used to provide metadata about the HTML document. It will not be displayed on the page, but will be used by browsers , search engines , or other web services.We will use the <link> tag to link the external style sheet which is preferred approach as you can change the look of an entire site by changing one file and also helps in promoting reuse from architecture standpoint.

```
<head>
    <title>HTML5 / CSS3 Navigation Menu</title>
    <meta charset="UTF-8">
    <meta name="description" content="Free Web tutorials">
    <meta name="keywords" content="HTML,CSS">
    <meta name="author" content="WebCodeGeeks.com">
    <link rel="stylesheet" href="MenuStyle.css">
</head>
```

Although not mandatory, it's always a good practice to assign an id in your the and <nav> element so that you can refer them in in your scripting.

```
<body>
<div id="wrapper">
```

The tag defines a division or a section in an HTML document and group them together. The<nav> element is new in HTML5 and defines a set of navigation links and support the Global and Event Attributes in HTML. We will use the <ul> tag in conjunction with the <li> tag to create unordered lists of menu and submenu. For simplicity we have added only one level of submenu. In reality, you can add as many as you want by adding <li> items under the <ul> element. You would like to replace the "#" in the href attribute to a real document link in your implementation.

3.3 CSS

For this example we will be using the following body styling:

```
body {
line-height: 1;
font:12px/20px Arial;
background:#e0f3ff ;
}
```

We will style our nav & div element as follows. Depending on the target device (mobile browsers have less display area) and space requirement, you will alter the values.

```
#wrapper{
    width:620px;
    margin:0 auto;
    margin-top:100px;
}

#nav {
    position:relative;
    width:620px;
    margin:0 auto;
}
```

Then we will style each <ul> element as per their position in the navigation chain.

```
ul#navigation {
    margin:0px auto;
    position:relative;
    float:left;
    border-left:1px solid #c4dbe7;
    border-right:1px solid #c4dbe7;
}

ul#navigation li {
    display:inline;
    font-size:12px;
    font-weight:bold;
    margin:0;
    padding:0;
    float:left;
    position:relative;
    border-top:1px solid #c4dbe7;
    border-bottom:2px solid #c4dbe7;
}
```

Here is how to achieve transition using CSS3 only which are effects that let an element gradually change from one style to another In this example we have used same declaration for multiple selectors. You may want to have individual declaration for each selector. Depending on the screen rendering preference, modify the timing in transition declaration from 0.2s. Note , how there are different declaration for different browsers.

```
ul#navigation ul, ul#navigation ul li ul {
    list-style: none;
    margin: 0;
    padding: 0;
    visibility:hidden;
    position: absolute;
    z-index: 99999;
    width:180px;
    background:#f8f8f8;
    box-shadow:0 2px 2px -1px rgba(0, 0, 0, 0.055);
    opacity:0;
    -webkit-transition:opacity 0.2s linear, visibility 0.2s linear; ///* Chrome &amp;  ↩
        Safari*/
    -moz-transition:opacity 0.2s linear, visibility 0.2s linear; // /*Mozilla Firefox*/
    -o-transition:opacity 0.2s linear, visibility 0.2s linear; // /* Opera */
    transition:opacity 0.2s linear, visibility 0.2s linear;
}
```

In order to have the dropdown navigation we need to set the following property:

```
ul#navigation li a:hover {
    background:#f8f8f8;
```

```
    color:#282828;
}

ul#navigation li:hover > a {
    background:#fff;
}

/* Drop-Down Navigation */
ul#navigation li:hover > ul
{
    visibility:visible;
    opacity:1;
}

ul#navigation ul, ul#navigation ul li ul {
    list-style: none;
    margin: 0;
    padding: 0;
    visibility:hidden;
    position: absolute;
    z-index: 99999;
    width:180px;
    background:#f8f8f8;
    box-shadow:0 2px 2px -1px rgba(0, 0, 0, 0.055);
    opacity:0;
    -webkit-transition:opacity 0.2s linear, visibility 0.2s linear;
    -moz-transition:opacity 0.2s linear, visibility 0.2s linear;
    -o-transition:opacity 0.2s linear, visibility 0.2s linear;
    transition:opacity 0.2s linear, visibility 0.2s linear;
}

ul#navigation ul {
    top: 43px;
    left: 1px;
}

ul#navigation ul li ul {
    top: 0;
    left: 181px;
}

ul#navigation ul li {
    clear:both;
    width:100%;
    border:0 none;
    border-bottom:1px solid #c9c9c9;
}

ul#navigation ul li a {
    background:none;
    padding:7px 15px;
    color:#616161;
    text-shadow:1px 1px 0px #fff;
    text-decoration:none;
    display:inline-block;
    border:0 none;
    float:left;
    clear:both;
    width:150px;
}

ul#navigation li a.first {
```

```
    border-left: 0 none;
}

ul#navigation li a.last {
    border-right: 0 none;
}
```

Let's go through some of the CSS 3 important styles in there:

- Box-shadow: Add a nice shadow around our element. Syntax - box-shadow: h-shadow v-shadow blur spread color inset;
- RGBA: RGB simply means Red, Green, Blue, it's an alternative to using HEX colours. By having RGBA we've specified an "Alpha" or opacity value to our shadow
- Margin: Done using the web developers compass (top, right, bottom, left), it specifies the "gap" around the element
- Transition: The transition properties allow elements to change values over a specified duration, animating the property changes, rather than having them occur immediately. For compatibility in all supporting browsers, vendor prefixes are required, with the standard syntax declared last

3.4 Download the source code

This was an example of creating simple navigation menu using HTML5 and CSS3.

Download: You can download the full source code of this example here : **HTML5Menu**

Chapter 4

HTML5 Audio Player

In this example we will present you how to use the HTML5 `<audio />` element.

First, we'll present the `<audio />` tag and its attributes, for a quick audio integration in your HTML documents.

And, then we will continue with more advanced usage using JavaScript to interact with the `HTMLMediaElement`.

4.1 The minimal example

In order to allow your users to play music directly from the browser you simply have to add the following syntax (Assuming your audio file is located in the **files** folder).

```
<audio src="./files/audio.ogg">
    Your browser does not support the audio element.
</audio>
```

This will only enable audio in your document, but this will not show any player ...

Note: The paragraph inside the audio tag, will be displayed on old browsers that not support the audio tag. You can see the compatibility matrix on the CanIUse Web Site

4.2 Show the controls

The previous example did not diplay the player on the web page. To view the player, simply add the `controls` attribute in the audio tag:

```
<audio src="./files/audio.ogg" controls>
    Your browser does not support the audio element.
</audio>
```

This will display the browser's default player with the default controls.

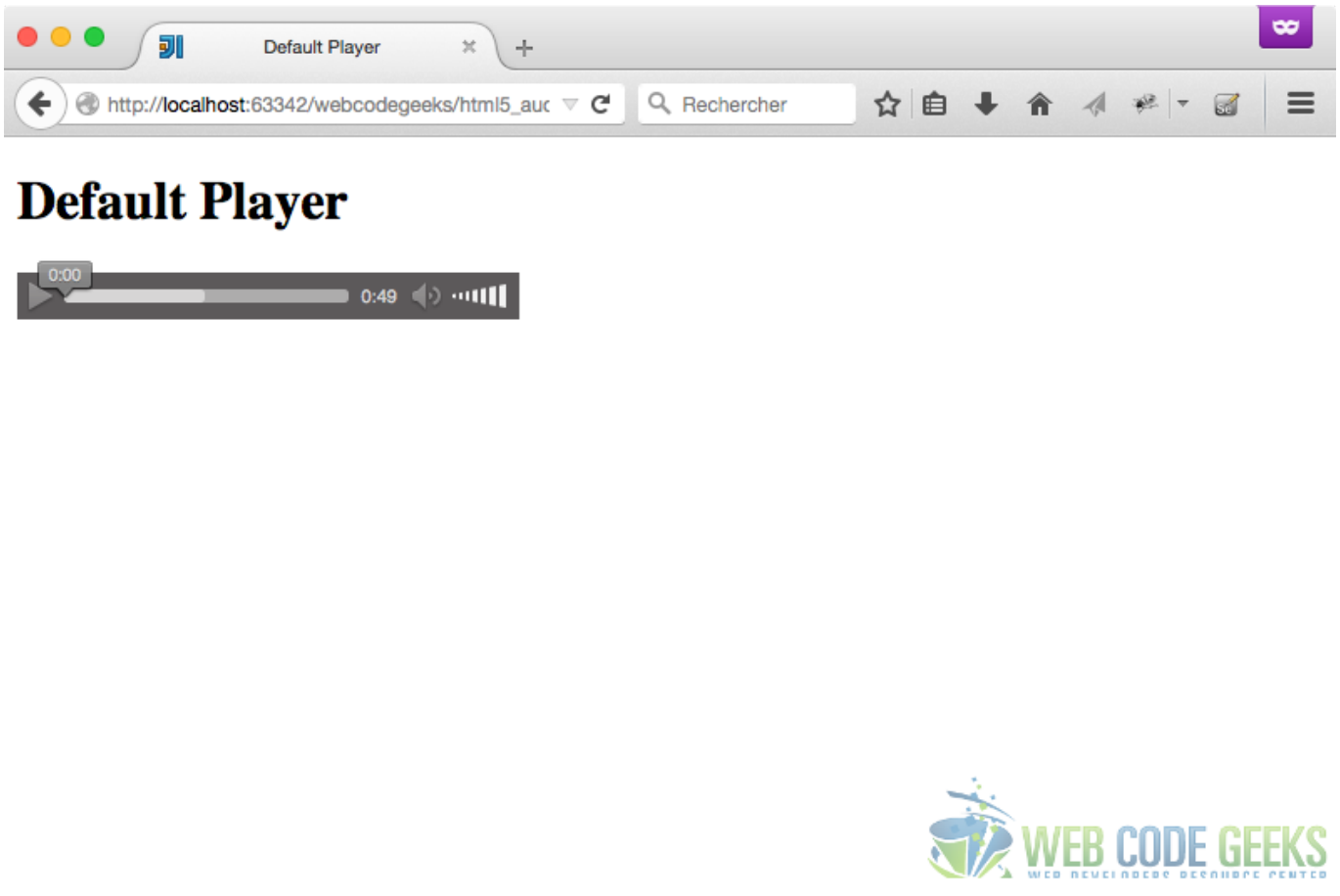

Figure 4.1: Default Audio Player on Firefox

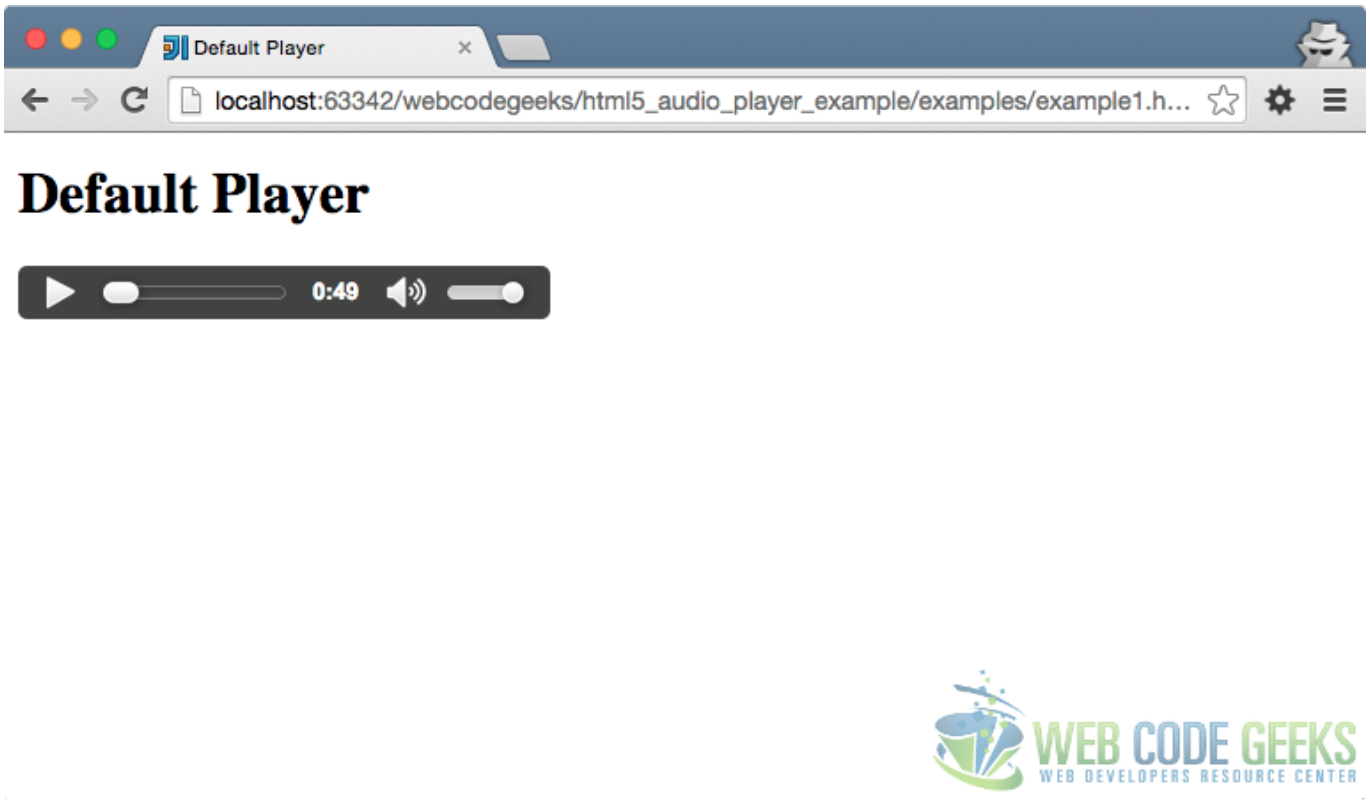

Figure 4.2: Default Audio Player on Chrome

Here is another way to define the media source:

```
<audio controls>
    Your browser does not support the audio element.
    <source src="./files/audio.ogg" />
</audio>
```

Note: All the audio format are not supported by all the browser, you can find a compatibility matrix on the Dedicated Wikipedia Web Page

4.3 Tag Attributes

The `<audio />` Tag accept attributes that will help you to manage the behaviour of the media on your page.

4.3.1 Controls

This attribute will display the default browser player on the page, with the default controls, those controls are : **Play** and **Pause** buttons, **Volume** control, and the track duration.

This attribute is used to display the player (as we've seen in the previous section).

You can use it by simply adding the `controls` or `controls="controls"`. It does not matter.

```
<audio controls="controls">
    Your browser does not support the audio element.
    <source src="./files/audio.ogg" />
</audio>
```

4.3.2 Autoplay

This attribute will start playback once it's ready.

So, if you want to start playback on your page without displaying the player, you can do something like that:

```
<audio autoplay="autoplay" >
    Your browser does not support the audio element.
    <source src="./files/audio.ogg" />
</audio>
```

4.3.3 Loop

With this attribute, the audio will play again once it's finished.

```
<audio loop="loop" autoplay >
    Your browser does not support the audio element.
    <source src="./files/audio.ogg" />
</audio>
```

This will start playback automatically, and loop at the end of the media, it will start again.

4.3.4 Preload

This attributes will give a hint to the browser on how to treat the media file.

This attribute accept the following values:

- auto: The browser can download the whole file if it's needed by the user's needs.
- metadata: The user may not need the whole media, so the browser can only check for the metadatas (length) of the file.
- none: The browser won't download the file if the user does not need it. This can be use to minimize server trafic.

Note: The default value is auto

```
<audio preload="none" >
    Your browser does not support the audio element.
    <source src="./files/audio.ogg" />
</audio>
```

In this example the browser will request the server only when the user click on the **play** button.

4.4 Control the audio with Javascript

We used the `<audio />` Tag to display the player on the page, and to play the audio file.

Now we'll see how to manipulate audio with JavaScript.

4.4.1 Basic Play / Pause example

For the beginning, we will simply add an audio file in a page and manage the Play / Pause buttons in JavaScript.

Here is the HTML code:

```
<audio id="audio1">
    <source src="../files/2081-Coin_Drop-Willem_Hunt-569197907/mp3/Coin_Drop-Willem_Hunt ↩
        -569197907.mp3" />
</audio>
<button id="play">Play</button> <button id="stop">Stop</button>
```

The goal is to start play the sound when the user click on the play button, and pause the button when the user click on the Stop button.

Here is the JavaScript Code:

```
var audioElement = document.getElementById('audio1');

document.getElementById('play').addEventListener('click', function(){
audioElement.play();
}, false);

document.getElementById('stop').addEventListener('click', function(){
audioElement.pause();
}, false);
```

Quite simple, no ?

We get the `audioElement` by its ID in the document, and then, we add listener for clicks on the buttons. The `audioElement` variable is global in the scope so it can be accessed in the event callbacks functions.

4.4.2 The HTMLMediaElement

4.4.2.1 Properties

The type of the `audioElement` variable is `HTMLMediaElement` which has many properties, for example:

- `autoplay`: Reflects the value of the attributes (seen in previous section).
- `currentTime`: Contains the current playback time, in seconds. Setting this property will set the playback time at the value defined.
- `duration`: **(Read-Only)** The length, in seconds.
- `paused`: **(Read-Only)** Indicates if the playback is paused or not.
- `volume`: will get or set the volume of the media element : 0.0 is silent, and 1.0 is the loudest

You'll find the full properties list here : HTMLMediaElement.

4.4.2.2 Methods

The element has also some methods, we've seen `play()` and `pause()` in the basic example, here are the others:

- `canPlayType(mimetype)` : Determine if the browser can play the **mimetype** passed in argument. This function can return : `nothing` (empty string) if the browser is not able to play the type, `propably` if the browser seems to be able to play the type, `maybe` if it's impossible to tell if the type is playable or not.
- `fastSeek(time)` : this will seek directly to the given time.
- `load()` : This method will begin loading the media from the server

When actions are made with the `<audio />` tag, some events are fired, let see the events before using all together with a full example.

4.5 Use Media Events

Here are some of the events we can use with audio element.

- `progress`: The user agent is fetching media data.
- `error`: An error occurs while fetching the media data.
- `play`: Playback has begun. Fired after the `play()` method has returned, or when the autoplay attribute has caused playback to begin.
- `pause`: Playback has been paused. Fired after the pause() method has returned.
- `loadeddata`: The user agent can render the media data at the current playback position for the first time.
- `waiting`: Playback has stopped because the next frame is not available, but the user agent expects that frame to become available in due course.
- `playing`: Playback has started.
- `canplay`: The user agent can resume playback of the media data, but estimates that if playback were to be started now, the media resource could not be rendered at the current playback rate up to its end without having to stop for further buffering of content.
- `seeking`: The seeking IDL attribute changed to true and the seek operation is taking long enough that the user agent has time to fire the event.
- `seeked`: The seeking IDL attribute changed to false.
- `timeupdate`: The current playback position changed as part of normal playback or in an especially interesting way, for example discontinuously. **Note:** This event will be fired every second !
- `ended`: Playback has stopped because the end of the media resource was reached.
- `volumechange`: Either the volume attribute or the muted attribute has changed. Fired after the relevant attribute's setter has returned.

The full event list can be find here : http://www.w3.org/wiki/HTML/Elements/audio#Media_Events

4.5.1 Example

Let see an example on how to use those events.

In this example we will display information in the browser console ...

The HTML code ...

```
<audio id="audio2" controls>
    <source src="../files/2081-Coin_Drop-Willem_Hunt-569197907/mp3/Coin_Drop-Willem_Hunt ↩
        -569197907.mp3" />
</audio>
```

... and the JavaScript.

```
var audioElement = document.getElementById('audio2');

audioElement.addEventListener('playing', function(e){
  console.log('Audio playback has started ...');
  console.log('Playback started at : '+ e.target.currentTime +" seconds");
}, false);

audioElement.addEventListener('pause', function(e){
```

```
  console.log('Audio playback has been paused ...');
  console.log('Playback paused at : '+ e.target.currentTime +" seconds");
}, false);

audioElement.addEventListener('ended', function(e){
  console.log('Playback has ended');
}, false);

audioElement.addEventListener('volumechange', function(e){
  console.log("Volume has changed ...");
  console.log("Volume is now "+ e.target.volume);
}, false);
```

First, we get the audio element by its ID and store it in a variable: `audioElement`, then we add listeners on the audio element for the following events :

- `playing`, callback will log the string : Audio playback has started ..., and an info about the currentTime of the element, when the user click on the play button. Try to start, pause, then start again to see the currentTime info updated.
- `playing`, this is fired when the media has been paused ... When the user pause the playback, callback will be called.
- `ended`, this is fired when the media has ended.
- `volumechange`, this is fired the volume (of the element, not your computer) has changed, callback will log volume value.

For all the events we used the event target to get information about the audio element, we did not used the `audioElement` variable, but for this example we could do it ... I prefer to use the event target because it rely on the event, not the global scope.

4.6 Playlist Example

Let see an example on how to use the `<audio />` element in a real world ...

4.6.1 The Specifications

Imagine you are designing a web page for a Lounge Bar, you can add some smooth audio in background to improve user experience.

So we will create a little component that will play multiple files, it means that at the end of a file, the player will start the next audio file. We will also allow the user to stop the music, and to change the volume.

4.6.2 The code ...

Here is the HTML code i decided to use for the component:

```
    <!-- This will only display the player, and enable the audio on the document-->
    <audio controls autoplay></audio>
```

The encapsulation, will allow us to position the element more easily.

For example to position the player in the bottom left corner...:

```
#playlist {
    display: inline;
    position: fixed;
    bottom:5px;
    left:5px;
}
```

Then the JavaScript code:

```
// Playlist array
var files = [
"./files/2081-Coin_Drop-Willem_Hunt-569197907/mp3/Coin_Drop-Willem_Hunt-569197907.mp3",
"./files/2082-Hammering_Soung_6-Lisa_Redfern-411383436/mp3/Hammering_Soung_6-Lisa_Redfern ↩
    -411383436.mp3",
"./files/2083-Night_Sounds_-_Crickets-Lisa_Redfern-591005346/mp3/Night_Sounds_-_Crickets- ↩
    Lisa_Redfern-591005346.mp3"
];

// Current index of the files array
var current = 0;

// Get the audio element
var playlistPlayer = document.querySelector("#playlist audio");

// function that will be call at the end of the previous
function next() {
    // Check for last media in the playlist
    if (current === files.length - 1) {
        current = 0;
    } else {
        current++;
    }

    // Change the audio element source
    playlistPlayer.src = files[current];
}

// Check if the player is in the DOM
if (playlistPlayer === null) {
    throw "Playlist Player does not exists ...";
} else {
    // Start the player
    playlistPlayer.src = files[current];

    // Listen for the playback ended event, to play the next media
    playlistPlayer.addEventListener('ended', next, false)
}
```

This was very simple to do, and it will improve user experience.

Of course we could imagine many improvements such as getting the list of audio files from an Ajax Request, or even designing our own player, but the requirements are satisfied.

4.7 Conclusion

The HTML5 `<audio />` offers the ability to easily embed sound into your documents.

Download: You can download the full source code of this example here : **HTML5 Audio Player Example**

Chapter 5

HTML5 Local Storage

5.1 Introduction

HTML5 local storage allows web applications to store values locally in the browser and can survive the browser session, just like cookies. Unlike cookies that need to be sent with every HTTP request, thereby affecting website performance, local storage data as information is never transferred to the server.

Additionally, cookies allow you to store only 4K of data per domain, while the local storage allows you at least 5 MB per domain.Local storage offers a simple key - value store, like a hash table object and comes in two flavors:

- **Session Storage:** stores data for one browser session and is available till the browser/browser tab is closed. Popup windows opened from the same window can see session storage, and so can iframes inside the same window. However, multiple windows from the same origin (URL) cannot see each other's session storage.
- **Local Storage:** stores data with no expiration date. The data is available to all windows with the same origin (domain) even when the browser/browser tabs are reopened or closed.

Local storage is supported in the following browsers:

- Internet Explorer 8+
- Firefox
- Opera
- Chrome
- Safari

In both the cases, please note that the storage data will not be available between different browsers.

We will create a simple shopping list example using local storage as shown below:

Figure 5.1: HTML Storage

5.2 Local Storage

We will add the external javascript and stylesheet reference in the HTML header .

```
<head>
<title>HTML5 localStorage Example</title>
<meta charset="UTF-8">
<meta name="description" content="Free Web tutorials">
<meta name="keywords" content="HTML,CSS">
<meta name="author" content="WebCodeGeeks.com">
<script src="Storage.js"></script>
<link rel="stylesheet" href="StorageStyle.css">
</head>
```

We will call the java script function `doShowAll()` in `onload()` event to a check for browser compatibility and to dynamically draw the table that displays storage name/value pair.

```
<body onload="doShowAll()">
```

In the `CheckBrowser()` function, we would like to check if the browser supports HTML5 storage or not. There are few ways to do it. We will use the default API to check for the browser compatibility. Alternately, we could have used open source libraries, like Modernizr, that help us to detect the browser support for HTML5 and CSS features.

```
function CheckBrowser() {
    if ('localStorage' in window && window['localStorage'] !== null) {
        // we can use localStorage object to store data
```

```
        return true;
    } else {
            return false;
    }
}
```

Inside the `doShowAll()` if the `CheckBrowser()` function evaluates to true we will dynamically create the table for Shopping list during the page load . You can iterate the keys (property names) of the key-value pairs stored in the local storage, like this

```
for (i = 0; i <= localStorage.length - 1; i++) {
        key = localStorage.key(i);
        list += "|" + key + "\n|"
                        + localStorage.getItem(key) + "\n";
}
```

Based on the storage value, draw the table dynamically to display the key/value pair stored in local Storage.

```
function doShowAll() {
    if (CheckBrowser()) {
        var key = "";
        var list = "<th>Name</th><th>Value</th>\n";
        var i = 0;
        for (i = 0; i <= localStorage.length - 1; i++) {
            key = localStorage.key(i);
            list += "|" + key + "\n|"
                    + localStorage.getItem(key) + "\n";
        }
        if (list == "<th>Name</th><th>Value</th>\n") {
            list += "|_empty_\n|_empty_\n";
        }
        document.getElementById('list').innerHTML = list;
    } else {
        alert('Cannot store user preferences as your browser do not support local storage') ↩
            ;
    }
}
```

We create a separate `div` element to capture the user inputs and submission. We attach the corresponding JavaScript function in the `onClick` event for the buttons.

```
<input type=button value="Save"   onclick="SaveItem()">
<input type=button value="Modify" onclick="ModifyItem()">
<input type=button value="Remove" onclick="RemoveItem()">
```

Inside our style sheet we have formatted the `div` information as follows.

```
#PlayArea {
    border: 1px dotted blue;
    padding: 6px;
    background-color: #ccc;
    margin-right: 50%;
}
```

You can set properties on the `localStorage` object just like with a normal JavaScript object. Here is an example on how we can set the local storage property `myProperty` to the value `myValue`.

```
localStorage.myProperty = "myValue";
```

You can delete local storage property like this.

```
delete localStorage.myProperty;
```

Alternately use the following methods to access the `localStorage`

```
localStorage.setItem ('propertyName', 'value');
localStorage.getItem ('propertyName');
localStorage.removeItem ('propertyName');
```

For saving the key/value pair capture the value of the corresponding javascript object and call the `.setItem` method

```
function SaveItem() {
    var name = document.forms.ShoppingList.name.value;
    var data = document.forms.ShoppingList.data.value;
    localStorage.setItem(name, data);
    doShowAll();
}
```

To fetch an existing key/value pair we will use the `.getItem` method

```
function ModifyItem() {
    var name = document.forms.ShoppingList.name.value;
    document.forms.ShoppingList.data.value = localStorage.getItem(name);
    doShowAll();
}
```

We will use the `.removeItem` method to delete from the storage.

```
function RemoveItem() {
    var name = document.forms.ShoppingList.name.value;
    document.forms.ShoppingList.data.value = localStorage.removeItem(name);
    doShowAll();
}
```

There is another method for local storage that allows you to clear the entire local storage.We have called the `ClearAll()` function in the `onClick` event of the "Clear" button.

```
<input type=button value="Clear" onclick="ClearAll()">
```

We have used the `clear` method to clear the local storage as shown below.

```
function ClearAll() {
    localStorage.clear();
    doShowAll();
}
```

5.3 Session Storage

The `sessionStorage` object works in same way as the `localStorage`. You can substitute the above example with `sessionStorage` object to persist the data only for the session. Once the user closes the browser window, the storage will be cleared. To summarize, the API for `localStorage` and `sessionStorage` is exactly the same, and allows you to use the following methods:

- setItem(key, value);
- getItem(key)
- removeItem(key)
- clear()
- key(index)
- length

5.4 Key points

- The browser by default allows 5 MB of storage per domain . `QUOTA_EXCEEDED_ERR` is the exception that will get thrown if you exceed your storage quota of 5 megabytes. You might want to put a try, catch block to catch the specific error.
- The `localStorage` and `sessionStorage` object by default only support string name/value storage. If you want to store arrays or objects into local storage use `JSON.stringify` to store the value and `JSON.parse` to retrieve the value.
- In order to keep track programmatically of when the storage area changes, you can trap the `storage` event. The `storage` event is fired on the `window` object whenever you insert, update or delete a session or local storage property. The storage event is only emitted in other browser windows than the one that inserted, updated or deleted the session or local storage objects which means that for local storage, events are visible to all other windows open with the same origin, including pop up windows and iframes while for session storage events are only visible in pop up windows and iframes.
- The key name should be unique for each domain.

5.5 Download the Source Code

This was an example of HTML local storage.

Download: You can download the full source code of this example here: **HTMLStorage**

Chapter 6

HTML5 Graphics and Animation

6.1 Introduction

Animation is the visualization of change, having reference points for the start and end states for comparison. Before HTML 5, it was limited to usage of Adobe Flash, CSS and JavaScript mostly. In this tutorial we will introduce the HTML `canvas` element and the `requestAnimationFrame` function in JavaScript to help us create simple animations.

The HTML `canvas` element (introduced in HTML5) is a **container** for canvas graphics and defined as a resolution-dependent bitmap canvas which can be used for rendering graphs, game graphics, or other visual images on the fly. It is a rectangular area inside a HTML page with no content or border of its own.

JavaScript can be used to draw anything you want on the canvas. It has several methods for drawing paths, boxes, circles, text, and graphic images.You can have more than one `canvas` element on the same page. Each canvas will show up in the DOM, and each canvas maintains its own state. If you give each canvas an id attribute, you can access them just like any other element.

```
<canvas id ="a" width="300" height="225"></canvas>
```

Now you can easily find that `canvas` element in the DOM with JavaScript, create a context, and then utilize the Canvas API in HTML 5 to draw visualizations

The `requestAnimationFrame` function requests the browser to call the animation loop at the time of redrawing the screen. Previously, developing animation with Javascript was a bit clunky as compared to ones created with CSS as you had to work with `setInterval` and `setTimeOut` to power your animation loop. There were problems in repainting the screen. With this new function, those problems are all gone and we are able to produce polished animation using the HTML 5 and javascript alone.

The canvas element is supported in the following browser versions or above.

- Google Chrome 4.0
- IE 9.0
- Firefox 2.0
- Safari 3.1
- Opera 9.0

The `requestAnimationFrame` is supported by the following browsers versions or above.

- Google Chrome 10.0
- IE 10
- Firefox 4.0
- Safari 6.0
- Opera 15.0

6.2 Canvas element and context

We need to understand the difference between Canvas element and Canvas context. While the element is a node in DOM, the context is the object with properties and methods that you can use to represent graphics inside canvas element. The context can be `2d` or `webgl` (3d). You need to use the `getContext` method to return a reference to a Context object

In the following sections we will use various Canvas properties and methods. Here is a screenshot of what we will be developing today.

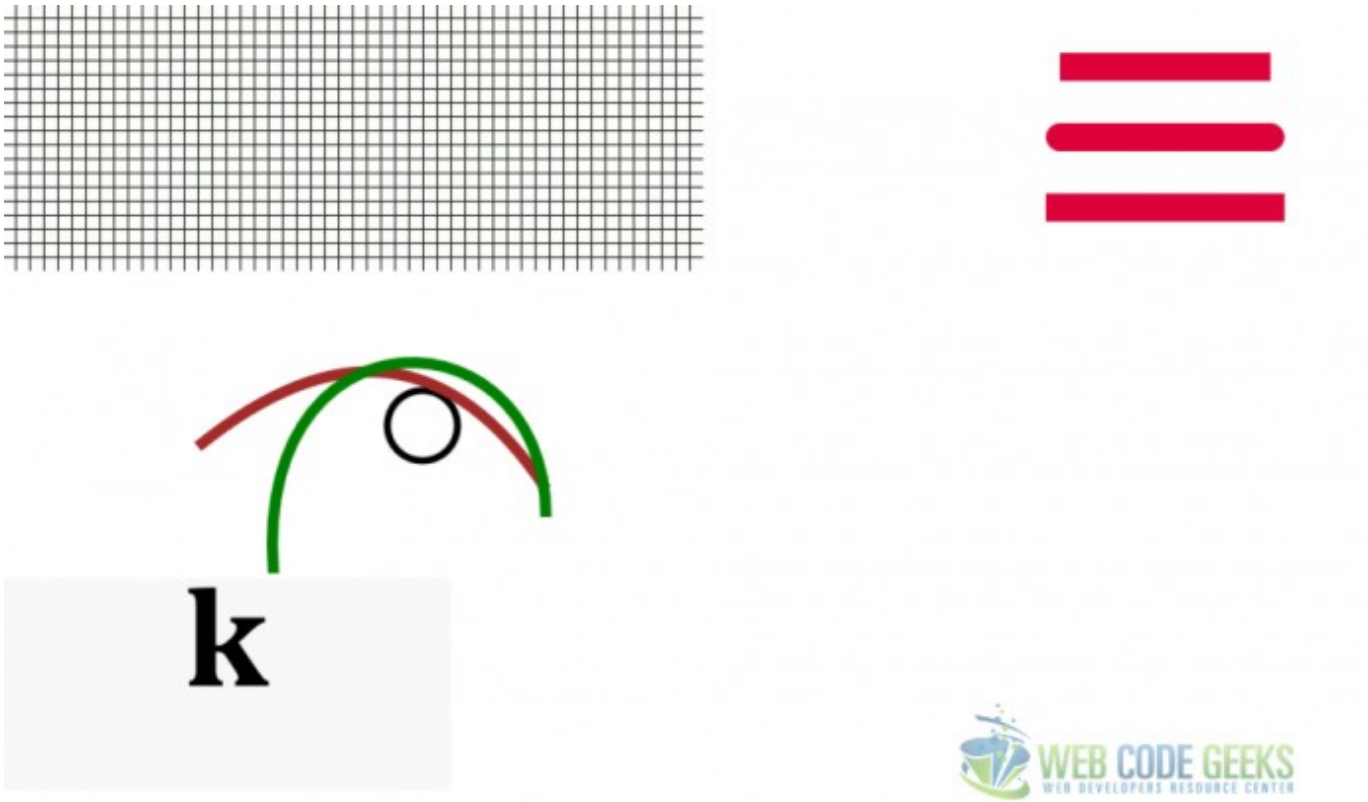

Figure 6.1: HTML Graphics

6.3 Draw a Graph

We will draw a graph paper use the combination of horizontal and vertical lines. First define a HTML canvas element with a id as shown below

```
<canvas id="GraphCanvas" width="550" height="190"></canvas>
```

Now in your javascript, grab a handle to reference the canvas, and get a reference to the context object, before we draw the lines.

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

Inside a loop draw the lines as shown below. The `moveTo()` and `lineTo()` method moves and draws line to the specific start-point and endpoint respectively.

Draw the horizontal line

```
for (var x = 0.5; x < 500; x += 10) {
          context.moveTo(x, 0);
          context.lineTo(x, 375);
        }
```

Draw the vertical line.

```
for (var y = 0.5; y < 375; y += 10) {
          context.moveTo(0, y);
          context.lineTo(500, y);
        }
```

The reason we set the x and y set at 0.5 is because we want to draw a line that is only one pixel wide; therefore we need to shift the coordinates by 0.5 perpendicular to the line's direction.

Now we need to actually draw it on the canvas using the `stroke()` method. The `strokestyle` property allows us to specify the color code.

```
context.strokeStyle = "#0B1907";
context.stroke();
```

6.4 Draw a Line

To draw a line, you use the following compulsory methods in sequence:

- Use the `beginPath` method to declare that we are about to draw a new path
- Use the `moveTo(x, y)` moves the drawing cursor to the specified starting point.
- `lineTo(x, y)` draws a line from the starting position to a new position.
- In order to make the line visible, we apply a stroke to the line using `stroke` function

By default the stroke color is defaulted to black. You can use the the `strokeStyle` property of the canvas context to set a different color of an HTML5 Canvas line

```
context.beginPath();
context.moveTo(200, canvas.height / 2 - 50);
context.lineTo(canvas.width - 200, canvas.height / 2 - 50);
context.lineWidth = 20;
context.strokeStyle = '#DF013A';
context.lineCap = 'butt';
context.stroke();
```

The `lineCap` property defines the three possible cap style butt, round or square. In order to define the width of an Canvas line, we can use the `lineWidth` property of the canvas context. Both these properties must be set before calling the `stroke()` method.

```
context.lineWidth = 15;
```

6.5 Draw Arc

The `arc()` method creates an arc/curve (used to create circles, or parts of circles). It is defined by a center point, a radius, a starting angle, an ending angle, and the drawing direction (either clockwise or anticlockwise). Arcs can be styled with the `lineWidth`, `strokeStyle`, and `lineCap` properties. The arc method as defined below accepts the following parameters.

x and y coordinates of the center of the circle, r the raidus of the circle, sAngle is the start angle expressed in radians , eAngle is the end angle and last one is weather it would clockwise or counterclockwise.

```
context.arc(x,y,r,sAngle,eAngle,clockwise);
```

The 0 for the starting angle is at the 3 o'clock position of the arc's circle. Arcs can be styled with the `lineWidth`, `strokeStyle`, and `lineCap` properties as highlighted below.

```
var canvas = document.getElementById('myCanvas');
var context = canvas.getContext('2d');
var x = canvas.width / 2;
var y = canvas.height / 2;
var radius = 75;
var sAngle = 1.1 * Math.PI;
var eAngle = 1.9 * Math.PI;
var counterClockwise = false;
context.beginPath();
context.arc(x, y, radius, sAngle, eAngle, counterClockwise);
context.lineWidth = 15;
context.strokeStyle = 'black';
context.stroke();
```

6.6 Draw some more stuff

Similar to the arc, we can draw 2 types of curves namely quadratic and Bezier curve. If a path does not exist, use the `beginPath()` and `moveTo()` methods to define a context point. Both the curves can be styled with the lineWidth, strokeStyle, and lineCap properties.

To draw a quadratic curve, define a control point that dictates the curvature of your quadratic curve by creating two imaginary tangential lines which are connected to the context point and the ending point. The distance of control point from the context point and end point determines the how broad your curve is. Longer the distance, sharper is your curve. Play around with the co-ordinates as shown below to form a better idea on the curve. Here is the syntax

```
context.quadraticCurveTo(Cp1X,Cp1Y,EpX,EpY);
```

where Cp1X and Cp1Y indicates the x and y co-ordinates of Control point while EpX and EpY are the x and y co-ordinates of the end point.

```
context.beginPath();
context.moveTo(140, 120);//define the context point
context.quadraticCurveTo(288, 0, 388, 150);
context.lineWidth = 8;
context.strokeStyle = 'brown';
context.stroke();
```

To create a Bezier curve with HTML5 Canvas, we use the `bezierCurveTo()` method.Unlike quadratic curves, Bezier curves are defined with two control points instead of one, allowing us to create more complex curvatures than quadraticCurve . You begin by using the `moveTo()` to create the context point

```
context.moveTo(200, 250);//create context point
```

Now you can call the `bezierCurveTo(Cp1X,Cp1Y, Cp2X,Cp2yY,EpX, EpY)`. Cp1X,Cp1Y, Cp2X and Cp2Y represent the x and y co-ordinates of the 2 control points. EpX and EpY are the x and y co-ordinates of the endpoints. Conceptually, the first part of the curve is tangential to the imaginary line that is defined by the context point and the first control point. The imaginary line that is defined by the second control point and the ending point defines the tangential curve of the second part.

```
context.beginPath();
context.moveTo(200, 250);//create context point
context.bezierCurveTo(150, 15, 388, 10, 388, 170);
context.lineWidth = 8;
context.strokeStyle = 'green';
context.stroke();
```

Drawing a shape like rectangle on a canvas is quite easy as well. Call the `context.rect()` method which accepts x and y position co-ordinates along with the width and height.

```
context.beginPath();
context.rect(180, 50, 200, 100);
context.fillStyle = 'green';
context.fill();
context.lineWidth = 7;
context.strokeStyle = 'black';
context.stroke();
```

6.7 Introducing requestAnimationFrame

Since the advent of `requestAnimationFrame` (sometimes referred as rAF method), developing animations using JavaScript has never been easier. This rAF method allows the browser to handle some of the complicated animation tasks for you, such as managing the frame rate. The inherent problem of using `setTimeout` and `setInterval` was the browser painting the frames quicker than the screen can display then (most computer screens have a refresh rate of 60 frames per second or FPS) resulting unnecessary computation.

Due to the fact that `setTimeout` and `setInterval` based animation will continue to run even if the page is not visible to user, which means that it's not battery friendly animation especially for mobile devices . Using rAF gives the browser the ability to optimize your animations to make them smoother and more resource efficient by eliminating the possibility of unnecessary draws and combining multiple animations into a single re-flow and repaint cycle.

Let's build our first animation with rAF. We start off by defining a `div` element inside html for container.

```
<h1 id="symbol">*</h1>
```

As rAF is fairly new, we need to check the vendor specific variant of the function in your javaScript. There are four vendor prefixes that you have to deal with:

- **moz** for Mozilla Firefox
- **webkit** for Google Chrome, Apple Safari and other webkit-based browsers
- **o** for Opera
- **ms** for Microsoft Internet Explorer

```
var requestAnimationFrame= window.requestAnimationFrame||
window.webkitRequestAnimationFrame||
window.mozRequestAnimationFrame||
window.oRequestAnimationFrame||
window.msRequestAnimationFrame;
```

As long as the variable evaluates to true to any one of the expressions, we will call `requestAnimationFrame` with the callback function that is responsible to redraw the screen. In our example we first evaluate the `div` element for display.

```
var symbols = document.querySelector("#symbol");
```

Then we assign a variable the characters we want to display an animate. Note how we used the `Math` DOM object for random selection and assignment to the `div`

```
var l = displayCHARS[Math.floor(Math.random()*displayCHARS.length)];
symbols.textContent = l;
requestAnimationFrame(animate);
```

Typically the frame rate is 60 frames per second(FPS). You can consider using `setTimeout` to delay when the next `request AnimationFrame` call gets made.

```
setTimeout(function(){
        requestAnimationFrame(animate);
    },1000/10);
```

If you need to stop your animation loop,you can consider using `cancelAnimationFrame`.Just like rAF, obtain a vendor specific handle for the function to start with.

```
var cancelRAF = window.cancelAnimationFrame||
        window.webkitCancelAnimationFrame||
        window.mozCancelAnimationFrame||
        window.msCancelAnimationFrame ;
```

Store the request specific rAF id to be passed to your cancelRAF function.

```
var requestID;      //store the requestAnimationFrame requestID value
requestID = requestAnimationFrame(animate);
cancelRAF(requestID);
```

Finally let's use the `requestAnimationFrame` on a canvas element to have a little animation of our own. Here is what we will develop today.

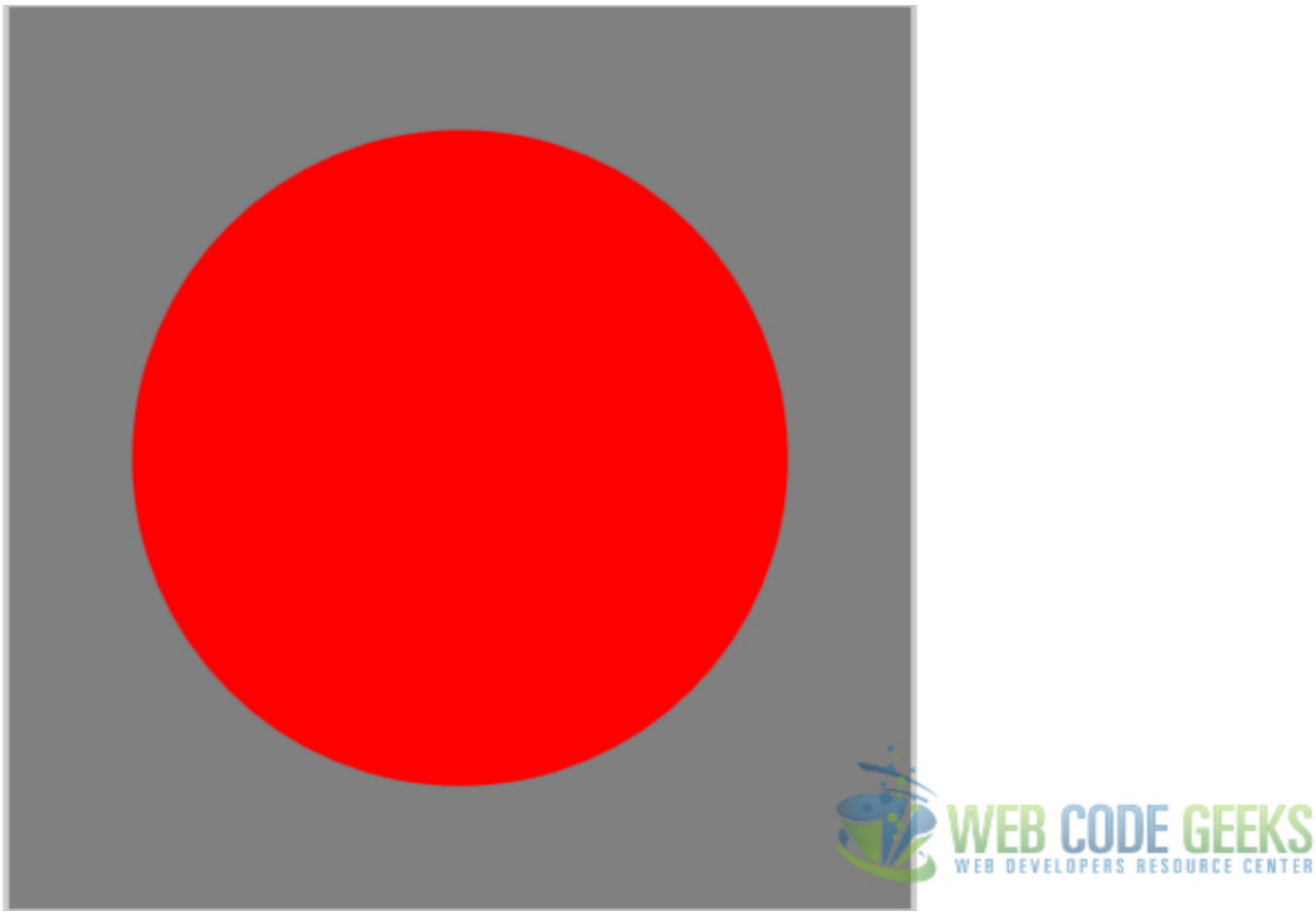

Figure 6.2: HTML Animation

We will use `clearRect` function of the context to clear the specified pixels within a given rectangle.Then use the `fillStyle` to color the background and `fillRect()` to draws a "filled" rectangle.

```
            mainContext.clearRect(0, 0, canvasWidth, canvasHeight);
            // color in the background
            mainContext.fillStyle = "grey";
            mainContext.fillRect(0, 0, canvasWidth, canvasHeight);
```

Now draw the circle and color it.

```
mainContext.beginPath();
var radius = 25 + 150 * Math.abs(Math.cos(angle));
mainContext.arc(225, 225, radius, 0, Math.PI * 2, false);
mainContext.closePath();
// color in the circle
mainContext.fillStyle = "red";
mainContext.fill();
```

6.8 Download the source code

Download: You can download the full source code of this example here: **HTMLCanvas**

Chapter 7

HTML5 Offline Web Application

In this example we will talk about writing an "Offline" Web Application. Offline ? Sounds weird, doesn't it ?

HTML5 allows us to create a "list" of files accessible after the browser is disconnected.

The browser will always have to download the main HTML file and all the files in this "list" at least once.

Let's see how this works.

7.1 The Manifest

The "list" is called the manifest file. This manifest is declared with an attribute of the `<html>` Tag like this:

```
<!DOCTYPE html>
<html manifest="cache.manifest">
<body>

</body>
</html>
```

The filename does not matter, but as a convention you may use names like : manifest.appcache, myapplicaiton.manifest, ...

The important thing is the Content-Type return by the server, it must be: `text/cache-manifest`.

In Apache you will have to add a directive: `AddType text/cache-manifest .manifest` to force the server to serve all .manifest file with the `text/cache-manifest` Content-Type.

In NGINX, you will have to edit the `mime.types` file and add the following line: `text/cache.manifestmanifest;`

For IIS, yous will have to add the mime type. Have a look at the answer on this StackOverflow.

And its first line must be: `CACHE MANIFEST`. For example:

```
CACHE MANIFEST
# This is a comment
index.html
css/style.css
js/script.js
```

This file will tell the browser to keep three files in cache: index.html, style.css and script.js.

So now, if you are disconnected and you refresh your browser, you will always be able to use your web page.

7.2 Manifest Sections

We did not see it in the previous section, but the manifest file is structured with sections:

- `CACHE`: Files listed in this section are explicitly cached after they are downloaded.
- `NETWORK`: Files listed in this section are explicitly **NOT** cached. "All requests to such resources bypass the cache, even if the user is offline"
- `FALLBACK`: This section will specifies fallback pages the browser will use when a resource is not accessible. Entries in this section lines of mapping urls: URL_REQUESTED URL_FALLBACK.

Note: All the files listed outside any sections are considered listed in the explicit `CACHE` section.

So let see a more complex `Manifest file`:

```
CACHE MANIFEST
# This is a comment
CACHE:
index.html
css/style.css
js/script.js

NETWORK:
# All requests to the Api need a network connection
/api

FALLBACK:
add.html offline.html
```

This manifest file will caches 3 files, tell the browser to always try to download resources inside /api, and if the add.html file is requested, the display the offline.html file.

One side effect using application cache is that the browser won't try to download updates cached files until the manifest file itself is updated ...

This can be very disturbing during development, or when files are frequently updated.

To solve this problem we can add a version number in a comment line at the top of the `manifest` file to force the browser to update its cache.

For example:

```
CACHE MANIFEST
# V1.0.1 <-- update the version to force the browser to update its cache.

# This is a comment
CACHE:
index.html
css/style.css
js/script.js

NETWORK:
# All requests to the Api need a network connection
/api

FALLBACK:
add.html offline.html
```

7.3 Application Cache API

The HTML5 Application Cache comes with a JavaScript Api, that will allow us (developers) to interact with the cache.

We can access the Application cache by the `window.applicationCache` object.

7.3.1 Events

Here are the events introduced by the Application Cache:

- `checking`: Fired when the user agent is looking for updates, or is trying to download the manifest file for the first time. (This is always the first event in the sequence.)
- `error`: The manifest file is not accessible (404), or any other errors when trying to downlod the manifest.
- `noupdate`: The manifest did not change ... nothing to update.
- `downloading`: The manifest was found or updated, so the browser starts to download resources listed in it.
- `progress`: Fired during download progress.
- `updateready`: Fired when resources in the manifest have been redownloaded.
- `cached`: Fired when resources have been cached. (Last event in sequence.)

7.3.2 Properties

The `window.applicationCache` object has only one property: `status` This property returns the current status of the `applicationCache`. The value of the property is one of this constant:

- `UNCACHED`: **0** A special value that indicates that an application cache object is not fully initialized.
- `IDLE`: **1** The application cache is not currently in the process of being updated.
- `CHECKING`: **2** The manifest is being fetched and checked for updates.
- `DOWNLOADING`: **3** Resources are being downloaded to be added to the cache, due to a changed resource manifest.
- `UPDATEREADY`: **4** There is a new version of the application cache available. There is a corresponding updateready event, which is fired instead of the cached event when a new update has been downloaded but not yet activated using the swapCache() method.
- `OBSOLETE`: **5** The application cache group is now obsolete.

7.3.3 Methods

Here are the methods of the `window.applicationCache` object.

- `update()`: Invoke the cache download process, not really necessary, the browser, will generally take care of updating the cache.
- `abort()`: Cancel the cache download process.
- `swapCache()`: Switches to the most recent application cache, if there is a newer one. If there isn't, throws an InvalidStateError exception.

7.4 The online / offline events

The API also add two events for Online status detection. These events are: `online` and `offline`.

Let see a simple example:

```
window.addEvenListener('online', function(){
    console.log('online');
}, false)
window.addEvenListener('offline', function(){
    console.log('offline');
}, false)
```

Now we know the necessary for a "real world" example ...

7.5 A Working Example

As an example we can imagine a simple web application, to manage your todo list. You must not be able to add a task when you're offline, we will do that by disabling submit button.

NOTE: The example will use the Zepto.js library to facilitate, the ajax calls and DOM manipulation.

7.5.1 Project structure

So the project structure will be:

```
/
--/css/
--/css/style.css
--/js/
--/js/script.js
--/server/server.php
--/server/tasks.json
--/index.html
```

The goal is to write an application that will allow you to add / remove / edit tasks, tasks will be saved in a static json file (`tasks.json`) on the server (simulated here by the `server.php` file).

7.5.2 The server.php file

As I said this will simulate a server, so **do not use this file in production**, this is only for testing purpose.

```
// We we only manage POST Requests
if ($_SERVER['REQUEST_METHOD'] === 'POST') {
    $message = array();
    $file  = dirname(__FILE__)."/tasks.json";
    $content = file_get_contents($file);
    $tasks = json_decode($content, true);
    switch($_POST['action']){
        case "list":
            // create the response
            $message["type"] = "OK";
            $message["data"] = $tasks;
            break;
        case "add":
            // create the task
            $task = array('id'=> uniqid(), "task"=>$_POST["task"]);
```

```
            // push it to the array pf tasks
            array_push($tasks, $task);

            // save the file :
            file_put_contents($file, json_encode($tasks));

            // create the response
            $message["type"] = "OK";
            $message["data"] = $task;

            break;
        case "remove":
            // Get the task id
            $id = $_POST["id"];
            $tmpTasks = array();
            $found = false;
            foreach($tasks as $task){
                if( $task["id"] !== $id ){
                    array_push($tmpTasks, $task);
                } else {
                    $found = true;
                }
            }
            if( $found ){
                // Ok task was deleted
                $message["type"] = "OK";
                $message["data"] = $id;
                file_put_contents($file, json_encode($tmpTasks));
            } else {
                $message["type"] = "KO";
                $message['message'] = "Task No Found";
            }
            break;
        case "update":
        case "edit":
            // Get the task id
            $id = $_POST["id"];
            $taskPosted = $_POST["task"];
            $tmpTasks = array();
            $found = false;
            foreach($tasks as $task){
                if( $task["id"] === $id ){
                    $task["id"] = $id;
                    $task["task"] = $taskPosted;
                    $found = true;
                }
                array_push($tmpTasks, $task);
            }
            if( $found ){
                // Ok task was deleted
                $message["type"] = "OK";
                $message["data"] = array('id'=>$id, 'task'=>$taskPosted);
                file_put_contents($file, json_encode($tmpTasks));
            } else {
                $message["type"] = "KO";
                $message['message'] = "Task No Found";
            }
            break;
        case 'reset':
            // save the file :

            file_put_contents($file, json_encode(array()));
```

```
            $message["type"] = "OK";
            $message["data"] = "";
            break;
        default:
            $message = array(
                "type"=>"ERR",
                "message" =>"Action unknown : ".$_POST["action"]."."
            );
            break;
    }
    // Print the result
    echo json_encode($message);
} else {
    $message = array(
        "type"=>"ERR",
        "message" =>"Request should be post, nothing to do ..."
    );
    echo json_encode($message);
}
```

This little php script allow you to use some actions to manage the task list, some of the actions needs some extra post datas, here is the API definition:

action	parameters	description
`list`	none	Get the full list of tasks.
`add`	task : STRING	Add the task, will generate a unique id, and return a task : { "type":"OK", "data":{ "id":"54e390fbc8823", "task":"rere" } }
`remove`	id : INT	Remove the task identified by the ID, will return the id as message, or an error message : { "type":"OK", "data":"54e390fbc8823" } or { "type":"KO", "message":"Task No Found" }
`update` or `edit`	id : INT task : STRING	Update the task identified by the ID with the Task sent, will return the id as message, or an error message : { "type":"OK", "data":"54e390fbc8823" } or { "type":"KO", "message":"Task No Found" }
`reset`	none	Reset the task list (empty list), return an empty message : { "type":"OK", "message":"" }
`default`	none	Should not append, return an error message : { "type":"KO", "message":"Task No Found" }

7.5.3 The main HTML file

Here is the `index.html` source file:

```
<!DOCTYPE html>
<html manifest="app.manifest">
<head lang="en">
    <meta charset="UTF-8">
    <title>Task List Example</title>
    <link href="css/style.css" rel="stylesheet" />
```

```
</head>
<body>

    <form id="taskform">
        <input type="hidden" id="task_id" value="-1" />
        <input type="text" id="task" placeholder="Add a Task ..." />
        <input type="submit" value="Add" />
    </form>
    <ul id="tasklist">

<button id="reset">Clear All Tasks</button>


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

Here is the `app.manifest` file:

```
CACHE MANIFEST
# version 1
# This is a comment
CACHE:
index.html
css/style.css
js/script.js
js/zepto.min.js

NETWORK:
server/server.php
```

7.5.4 The JavaScript

I won't show the full JS file as you can download it, I will only explain some parts of the code.

First of all we need to detect if the browser is **onLine** or **offLine**, as the whole code is cached (via the app.manifest file), the application can be opened without being onLine (don't need to do the first request if it was done once before).

```
if(navigator.onLine){
    // browser is online so I can load the list from the server
    server.list(init);
    // set the connection status to true
    setConnected(true);
} else {
    // Not online, set the connection status to false
    setConnected(false);
}
```

Here is the function `setConnected`

```
function setConnected(isConnected){
    var buttons = 'input[type="submit"], button';
    if( isConnected ){
        $(buttons).removeAttr('disabled', 'disabled');
    } else {
        $(buttons).attr('disabled', 'disabled');
    }
}
```

This function only enable or disable the buttons.

Now we need to listen to the `online` and `offline` events to manage the connection status:

```
$(window).bind('online', function(){
    // If the list was never loaded
    if( ! loaded ){
        // we load the list from the server
        server.list(init);
    }
    // set the connection status to true
    setConnected(true);
});
$(window).bind('offline', function(){
    // set the connection status to false
    setConnected(false);
});
```

I think the above code, is documented enough to understand it.

Here is what the application looks like in "online" mode:

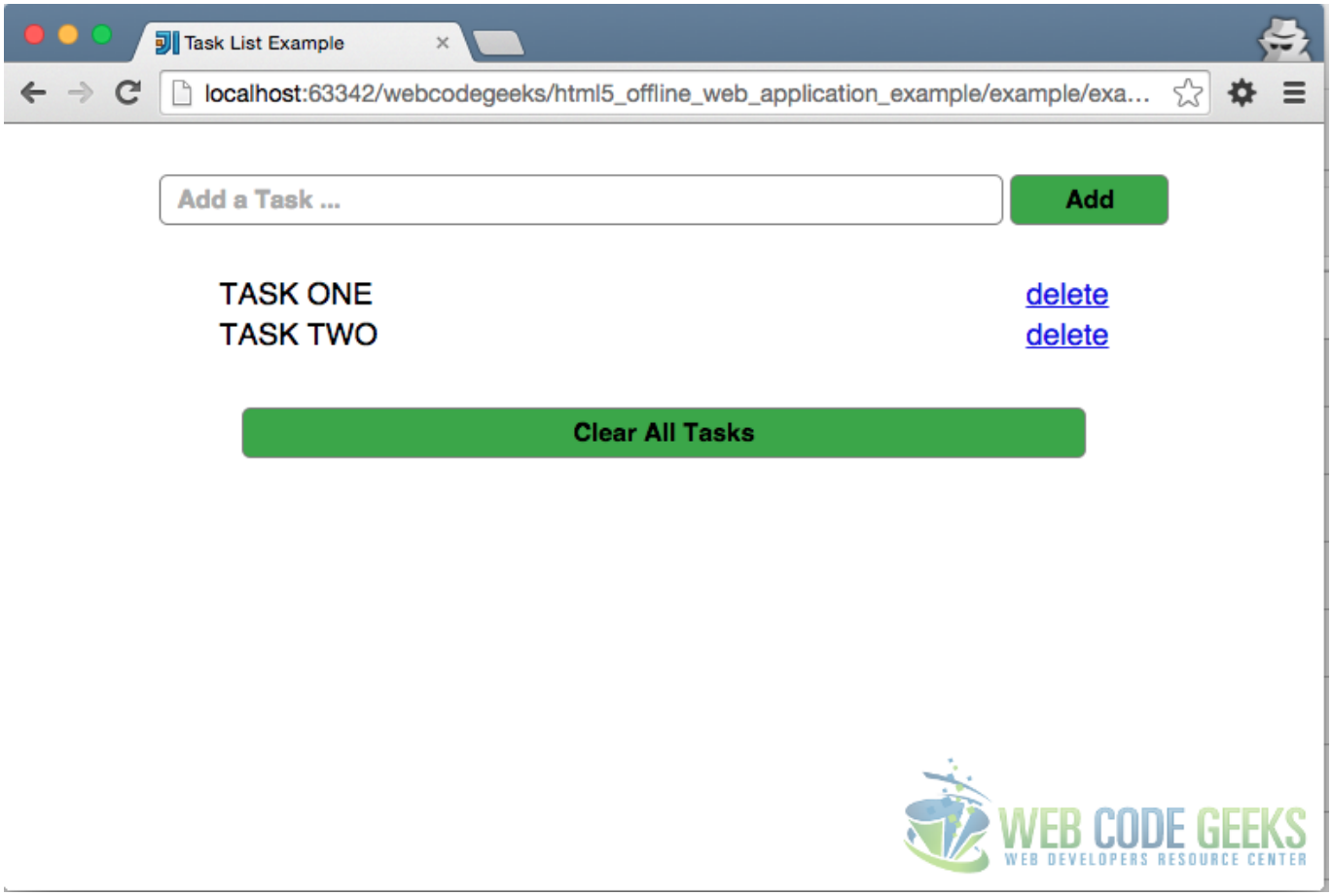

Figure 7.1: HTML5 Offline Web Application Example OnLine

Here is what the application looks like in "offline" mode:

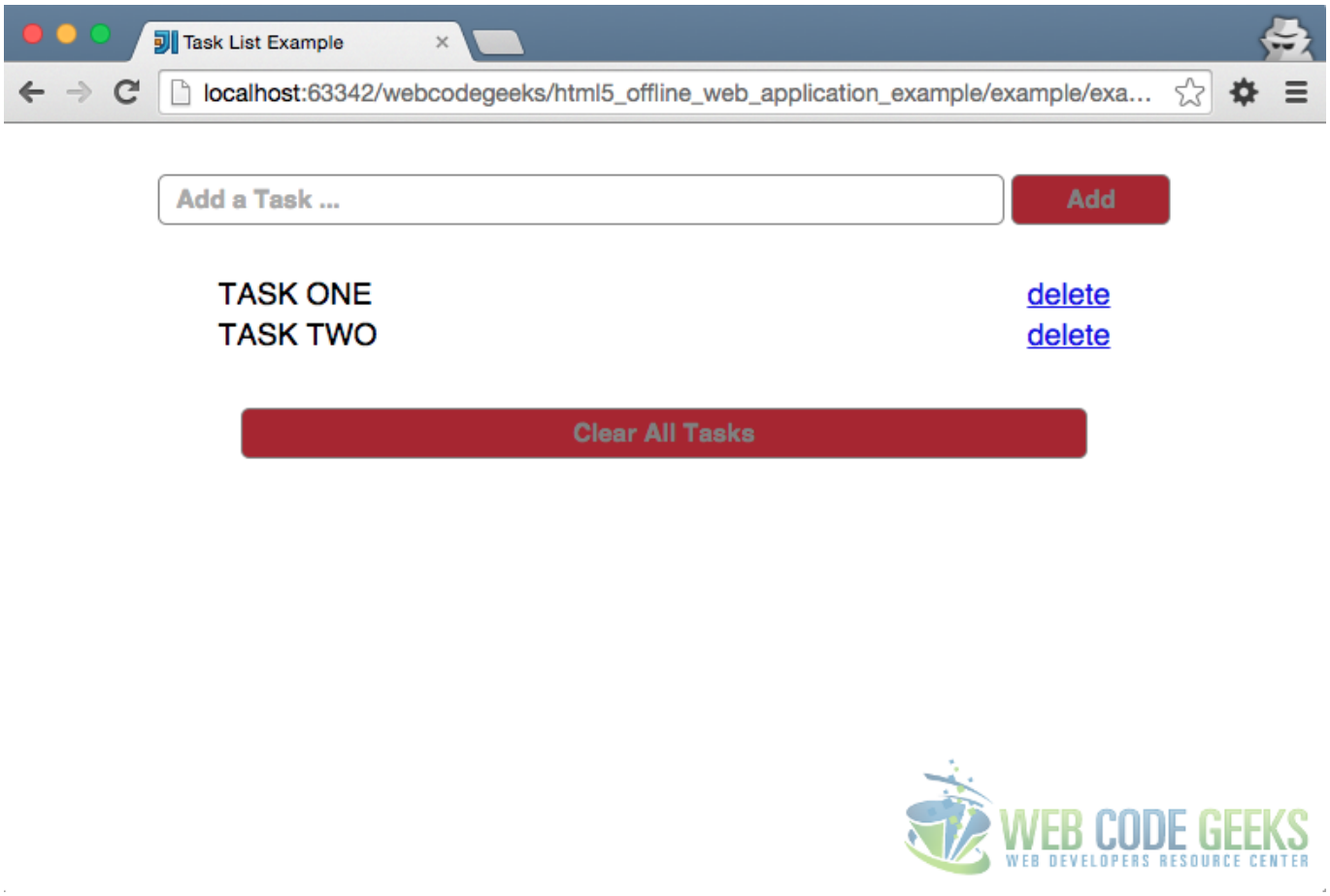

Figure 7.2: HTML5 Offline Web Application Example OffLine

7.6 Download

Download: You can download the full source code of this example here: **HTML5 Offline Web Application Example**

Chapter 8

HTML5 Geolocation

8.1 Introduction

At the heart of every location-based application is positioning and Geolocation. Geolocation is similar to the use of positioning systems but is more focused on determining a meaningful location (e.g. a street address) rather than just a set of geographic coordinates.

In this tutorial you will learn the Geolocation capabilities of HTML5. The API provides a method to locate the user's more or less exact position. This is useful in a number of ways ranging from providing a user with location specific information to providing route navigation.

There is more than one way to figure out where you are - your IP address, your wireless network connection, which cell tower your phone is talking to, or dedicated GPS hardware that calculates latitude and longitude from information sent by satellites. Gone are the days when we would inspect the client IP address and make a reasonable guess as to the where that device was located.

In HTML5, we have a set of APIs to effectively allow the client-side device (i.e. your iPhone 3G+, Android 2.0+ phones, or even your conventional desktop browsers) to retrieve geographic positioning information with JavaScript.

The Geolocation API is supported by the following browsers and smartphones. Minimum version requirements are mentioned as well.

- Google Chrome 5.0
- Internet Explorer 9.0
- Firefox 3.5
- Safari 5.0
- Opera 16.0
- Iphone 3.0
- Android 2.0
- Opera Mobile 10
- Blackberry OS 6.0

8.2 Security And Accuracy

When you're talking about sharing your physical location with a remote web server, note that this can compromise user privacy, the position is not available unless the user approves it. The browser will take care of this, and a message will either appear as a popup box (as shown below), or at the top of the browser (implementation is browser specific) requesting the user's permission. In case of smartphones, the relevant apps that uses GeoLocation may request the permission during installation itself.

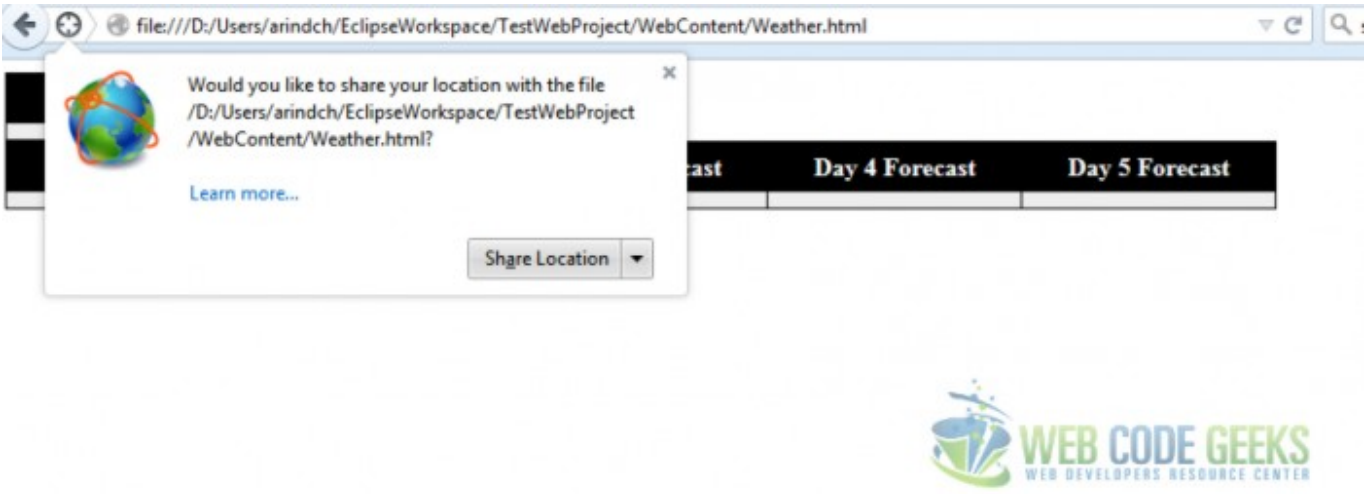

Figure 8.1: Firefox InfoBar

The infobar that pops up in the browser, depicts a lot of information. It is non-modal and tab-specific so it will disappear while switching to another browser window or tab. There is no way for a website to bypass the permission infobar. The blocking nature of the infobar ensures that the website cannot determine your location before you answer. As the end user,

- are told which website wants to know your location
- can choose to share or not to share your location
- can opt to remember your choice (share or don't share) to make this infobar disappear for ever for this website

Different techniques are used to obtain the user's location, and each technique has its own varying degree of accuracy. A desktop browser is likely to use WiFi (accurate to 20m) or IP Geolocation which is only accurate to the city level and can provide false positives. Mobile devices tend to use triangulation techniques such as GPS (accurate to 10m and only works outside), WiFi and GSM (Global System for Mobile) / CDMA (Code division multiple access) cell IDs (accurate to 1000m). Please refer to the reference section link that talks in details about wireless geolocation techniques.

8.3 Weather Widget

In this example, we will develop a simple weather widget using the HTML geolocation API and OpenWeather APIs. I must admit that I did not pay much emphasis on the display aspect of the example. Neither have I fully exploited the API offerings from OpenWeather API. Here is a screenshot of what we are going to develop today.

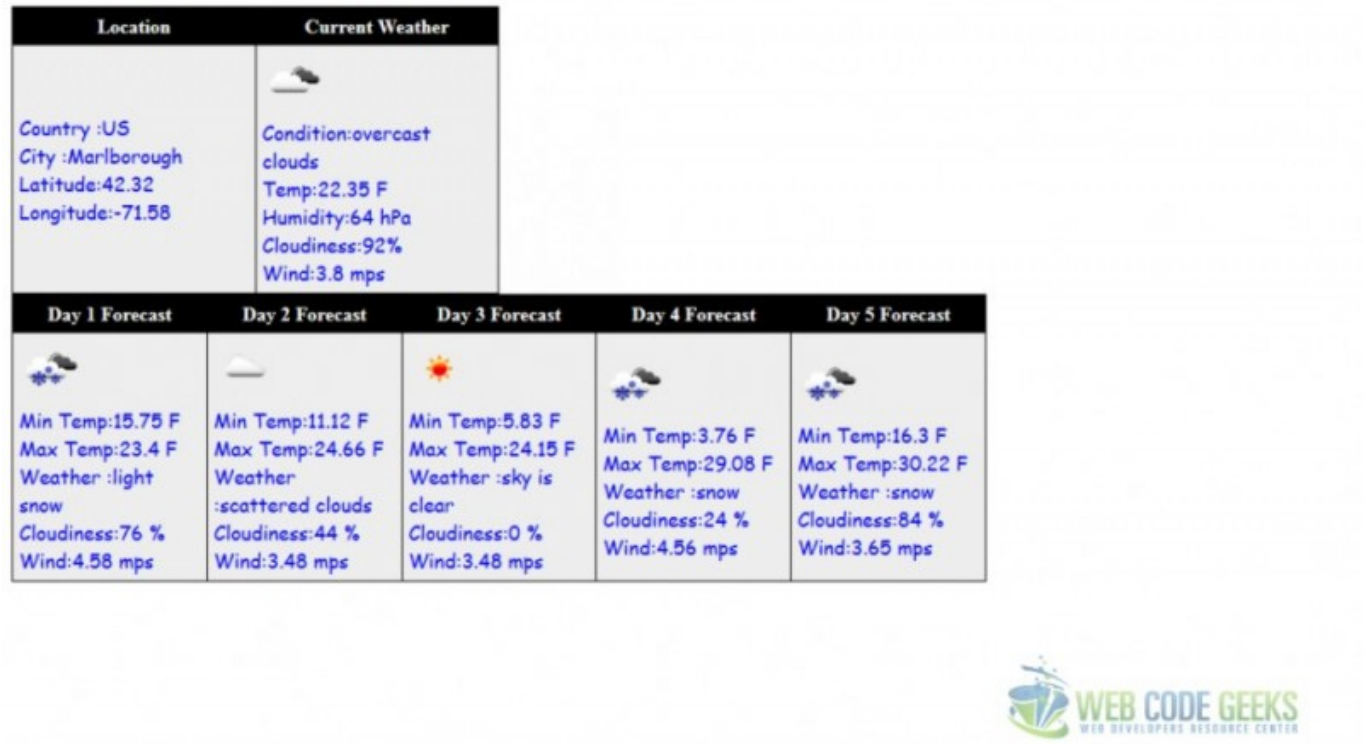

Figure 8.2: HTML Geolocation

We will start of with the html. We refer to the external JavaScript and CSS and call the inital JavaScript function in body `onload()`

```
<link rel="stylesheet" href="WeatherStyle.css">
<script src="WeatherWidget.js"></script>
</head>
<body onload="getLocation()">
```

2 different html `table` elements are defined and assigned a CSS style

```
<table class="myTable">
<table class="ForeCastTable">
```

We have also defined `div` element inside the table and assigned a specific id so that we can overwrite the content using JavaScript.

We will be using the free OpenWeather API to access the weather info in JSON format. We will parse the inputs and display them For more information and additional capabilities please refer to OpenWeather API link in the Reference section.

Let's get started by defining the global variables. The reason we have broken the URLs is that we have to build the URLs at the runtime Notice the separate static URLs for current weather, 5 day forecast and Image URL

```
//define the global variables
//current weather URL
var BASE_URL = "http://api.openweathermap.org/data/2.5/weather?";
var UrlParams = "&units=imperial&type=accurate&mode=json";
// forecast URL
var Forecast_URL = "http://api.openweathermap.org/data/2.5/forecast/daily?";
var ForeCast_Params = "&cnt=5&units=imperial&type=accurate&mode=json";
// Image base URL
var IMG_URL = "http://openweathermap.org/img/w/";
```

In the `getLocation()` function we will determine if Geolocation is supported

```
if (navigator.geolocation) {
```

If the geolocation is not supprted by the browser, display the error message.

```
} else {
        alert("Geolocation is not supported by this browser");
```

In the API there are two functions available to obtain a user's location:`getCurrentPosition` and `watchPosition`. We will discuss the difference between the functions shortly. For now let's see how they work. Both of these methods return immediately, and then asynchronously attempt to obtain the current location. They take the same number of arguments.

1. successCallback - called if the method returns successfully. For example , getCurrentWeatherData is called.

2. errorCallback - In case we encounter error for getCurrentPosition the errorCallback (displayError) is invoked.

3. [options] - a number of options are available:

a. enableHighAccuracy - Defaults setting is false. Setting this property to true may cause a slower response time and in the case of a mobile device, greater power consumption as it may use GPS.

b. timeout - Default is 0 which indicates infinite. depicts the maximum length of time in milliseconds to wait for a response. We set the timeout value to our computed value

```
var timeoutVal = 10 * 1000 * 1000;
```

a. maximumAge - in Milliseconds and the default value is 0 which means get new position object immediately. It denotes the maximum age of a cached position that the application will be willing to accept.

```
navigator.geolocation.getCurrentPosition(getCurrentWeatherData, // successCallback
displayError, { //errorCallback
enableHighAccuracy : true, // high accuracy
timeout : timeoutVal, //  in milliseconds
maximumAge : 0 // in milliseconds
});
```

Inside our error callback function, we handle the error graciously by converting the return code to a simple error message

```
function displayError(error) {
    var errors = {
        1 : 'Permission denied',
        2 : 'Position unavailable',
        3 : 'Request timeout'
    };
    alert("Error: " + errors[error.code]);
}
```

Let's summarize the steps :

- Determine if Geolocation is supported
- If supported, run the `getCurrentPosition()` method. If not, call the `errorcallback(displayError)`
- If the `getCurrentPosition()` method is successful, it returns a coordinates object to the function specified in the parameter (getCurrentWeatherData)
- The getCurrentWeatherData() function uses the `position` object for further processing.

8.4 getCurrentPosition and watchPosition

The main difference between `getCurrentPosition` and `watchPosition` is that `watchPosition` keeps informing your code when the position change, thereby updating the user's position. This is very useful if you want to track the movement of your end user like developing a driving direction apps. On the other hand the `getCurrentPosition` is a once off.

The `watchPosition` method return `watchID` which can be used to call the `clearWatch` when you want to stop the position constantly being updated

8.5 Position

When the user's position is returned, it is contained within a `Position` object which contains a number of properties

Position Object	
Property	**Notes**
coords.latitude	decimal degrees in double
coords.longitude	decimal degrees in double
coords.altitude	Height in metres of the position above the reference ellipsoid
coords.accuracy	The accuracy in metres of the returned result can allow the application in determining if the returned result is accurate enough for the application purpose
coords.altitudeAccuracy	The accuracy in metres of the returned altitude
coords.heading	Direction of travel of the hosting device, clockwise from true north
coords.speed	The current ground speed of the hosting device in metres per second
timestamp	Like a Date() object indicates the timestamp of when the position was acquired

Figure 8.3: Position

If we are able to obtain a `Position` object we will use the relevant properties to form the URL as shown below.

```
var WeatherNowAPIurl = BASE_URL + "lat=" + position.coords.latitude
            + "&lon=" + position.coords.longitude + UrlParams;
    var WeatherForecast_url = Forecast_URL + "lat=" + position.coords.latitude
            + "&lon=" + position.coords.longitude + ForeCast_Params;
```

In order to call the OpenWeather API we will use the `XMLHttpRequest` object is used to exchange data with a server behind the scenes. When a request to a server is sent, we want to perform some actions based on the response.The `onreadystatechange` event is triggered every time the `readyState` changes. It stores a function (or the name of a function) to be called automatically each time the `readyState` property changes The `readyState` property holds the status of the `XMLHttpRequest`.

Following are the possible values for `readystate`:

- 0: request not initialized

- 1: server connection established
- 2: request received
- 3: processing request
- 4: request finished and response is ready

The `status` property returns 200: "OK" or 404: Page not found. When `readyState` is 4 and status is 200, the response is ready. We are taking the JSON(Java Script Object Notation) `responseText` and parsing it before calling the Parse() function that takes cares of displaying the content from the JSON object.

```
var xmlhttp = new XMLHttpRequest();
xmlhttp.onreadystatechange = function() {
if (xmlhttp.readyState == 4 &amp;amp;&amp;amp; xmlhttp.status == 200) {
var JSONobj = JSON.parse(xmlhttp.responseText);
Parse(JSONobj);
}
}
```

Now you can use the open annd send properties of XMLHttpRequest object to call the API as shown below.

```
xmlhttp.open("GET", WeatherNowAPIurl, true);
xmlhttp.send();
```

8.6 Handling JSON

To access the JSON object in JavaScript, parse it with `JSON.parse()`, and access it via "." or "[]". To refresh our memories . JSON is similar to XML as it is "self describing" , hierarchical and can be fetched using `XMLHttpRequest` and parsed with many programming languages. However, JSON doesn't use a end tag, shorter in size and permits usage of arrays.

Unlike XML, JSON can use JavaScript and doesn't need a XML parser to be parsed. The data in JSON is name/value pairs separated by commas.Curly braces hold objects while Square brackets hold arrays.Here is an example of JSON response we got back from OpenWeather API call.

```
{
"cod": "200",
"message": 0.3399,
"city": {
"id": 4945840,
"name": "Northborough",
"coord": {
"lon": -71.641182,
"lat": 42.319538
},
"country": "US",
"population": 0
},
"cnt": 5,
"list": [{
"dt": 1424970000,
"temp": {
"day": 24.73,
"min": 18.86,
"max": 25.9,
"night": 18.86,
"eve": 21.69,
"morn": 20.59
},
"pressure": 1009.86,
```

```
"humidity": 51,
"weather": [{
"id": 803,
"main": "Clouds",
"description": "broken clouds",
"icon": "04d"
}],
"speed": 4.02,
"deg": 24,
"clouds": 76
},
```

Here is an example of how we have parsed the JSON response

```
// current Location
document.getElementById("location").innerHTML = "Country :"
+ obj.sys.country + "<br>" + "City :" + obj.name + "<br>"
+ "Latitude:" + obj.coord.lat + "<br>" + "Longitude:"
+ obj.coord.lon + "<br>";
// current weather
document.getElementById("weatherNow").innerHTML = "<img src='" + IMG_URL
+ obj.weather[0].icon + ".png'> " + "<br> Condition:"
+ obj.weather[0].description + "<br>" + "Temp:" + obj.main.temp
+ " F<br>" + "Humidity:" + obj.main.humidity + " hPa <br>"
+ "Cloudiness:" + obj.clouds.all + "% <br>" + "Wind:"
+ obj.wind.speed + " mps <br>";
```

As mentioned earlier, I have only touched the tip of the iceberg here. Please refer to the API documentation from OpenWeather and feel free to experiment with the various API functions.

8.7 Download

Download: You can download the full source code of this example here: **Geolocation Example**

8.8 Reference

- Wireless Geolocation Techniques A Survey
- OpenWeather API

Chapter 9

HTML5 Form Validation

9.1 Introduction

HTML5 has introduced a lot of changes welcomed by developers, including adding additional types of form input elements that helps in form processing activities like validation. Validating web forms has always been a painful task for many developers. Performing client side validation in a user-friendly as well as developer-friendly way is really hard. Furthermore informing users about the validation error in a pleasant way is a tedious task.

Before HTML5 there was no means of implementing validation natively; therefore, developers resorted to a variety of JavaScript based solutions. Let's get started with some coding using the new and exciting features of HTML5. Lot of these features are dependent on a specific browser versions. The version supports are getting updated every day. We have tried to present the features (that we intend to use) versus the browser version in the example in a table.

HTML5 Tags	Internet Explorer	Google Chrome	Mozilla Firefox	Opera	Safari
min and max	10	5	16	10.6	5.1
datalist	10	20	4	9	
placeholder	10	4	4	11	4
autofocus	10	4	4	10	4
pattern	10	5	4	9.6	
date		20		9	5
color		20	29	11	8
datetime-local		20		11	
email	10	10	4	10.6	5
month		31		9	5
number	10	7	29	9	5
range	10	6	23	11	4
search	10	6	3.6	10.6	5
time		20		9	5
url	10	10	4	10.6	5
week		31		9	5

WEB CODE GEEKS
WEB DEVELOPERS RESOURCE CENTER

Figure 9.1: Comparison

Here is a screenshot of what we we will develop today.

Figure 9.2: HTML Validation

Note how the elements are color coded depending on whether they are required or not. Here is how we did it. First mark the element in html as required.

```
<input type="email" id="email" name="email" value="" required placeholder="arinchat@gmail. ↩
    com">
```

Then use a style to color code the required fields.

```
input:required {
 background: hsl(180, 50%, 90%);
 border: 1px solid #999;
}
```

If you don't fill out a required field and try to submit, you will get a prompt with the error as shown below.

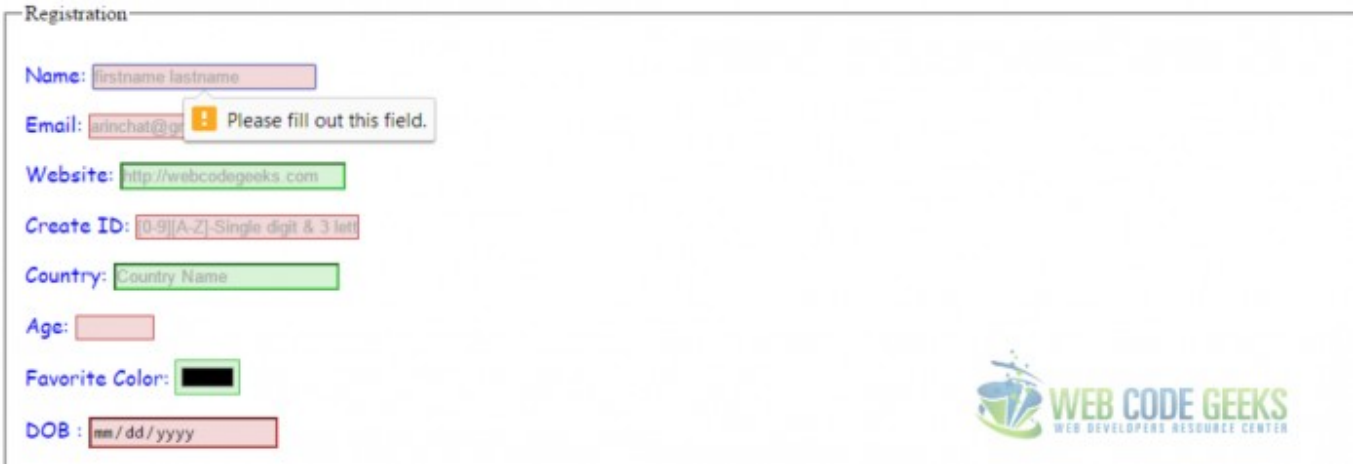

Figure 9.3: HTMLRequired

9.1.1 min and max

In HTML5 the min attribute together with the max attribute to create a range of legal values. The attributes works with the following input types: number, range, date, datetime, datetime-local, month, time and week. Let's look at few interesting things before we look at the attributes.

```
<label for="Age">Age:</label>
<input type="number" id="Age"name="Age" value="" required min="18" max="75">
```

The associated `label` tag has `for` attribute for the label tag matches up with the `id` attribute of the associated input tag. On top of the fact that the label is helping to give meaning to the input controls, it also means that if you click the label, the associated input tag receives the focus.

They also help with accessibility, as the text in the label will be read out to screen reader users: This is old trick, thought that I though is wort mentioning. Another thing that we did here was to have , providing context-sensitive help when field it describes receives focus. Again this trick is in addition to the feature `placeholder` feature that we will look at a little bit later.

```
<div>
<label for="Age">Age:</label>
<input type="number" id="Age"name="Age" value="" required min="18" max="75">
<span id="age-format" class="help">Format: Number between 18 and 75</span>
```

We have a little css style element that defines the way we view the span

```
.help {
display: none;
font-size: 90%;
}

input:focus+.help {
display: inline-block;
}
```

Here is the screenshot of validation error for min and max attributes:

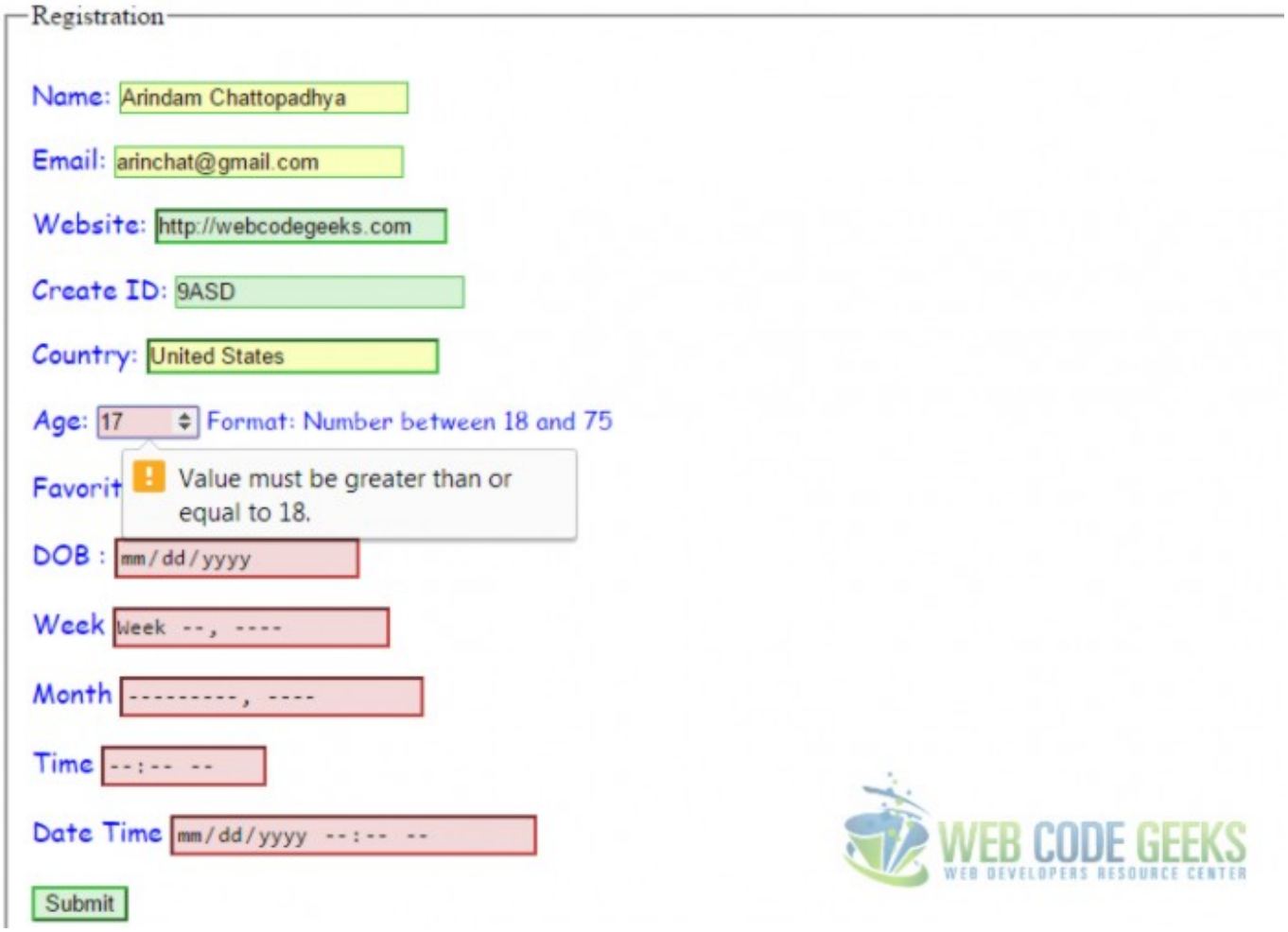

Figure 9.4: HTML min and max

9.1.2 datalist

A `datalist` element gets an ID attribute and contains numerous `OPTION` elements, just as a `<select>` element would: Once the `datalist` element is in place, a list attribute gets added to an `<input>` element which refers to the `list` id. In a nutshell, the `datalist` tag is used to provide an "autocomplete" feature on `<input>` element.

```
<input list="Countries" placeholder="Country Name">
 <datalist id="Countries">
 <option value="USA">
 <option value="Greece">
 <option value="UK">
 <option value="India">
 <option value="France">
 </datalist>
```

Please see the screenshot of `datalist` usage below:

Figure 9.5: DataList

9.1.3 placeholder

The ability to set placeholder text in an input field is new and welcome addition to Html5 webforms. The placeholder attribute can only contain text, not HTML markup. Placeholder text is displayed inside the input field as long as the field is empty. When you click on (or tab to) the input field and start typing, the placeholder text disappears.

```
<input type=text autofocus id="name" name="name" value="" placeholder="firstname lastname"  ↩
    required>
```

9.1.4 autofocus

HTML 5 has introduced a new Boolean attribute called `autofocus` attribute on all webform controls. As soon as the page loads, it moves the input focus to a particular input field. See the example below:

```
<input type=text autofocus>
```

9.1.5 pattern

This attributes helps the developer to specify a JavaScript regular expression for the field's value to be checked against. Hence we can use pattern to implement specific validation like phone number, product code, zip code etc. The pattern attribute works with the following input types: text, search, url, tel, email, and password. See the example code below.

```
<input type="text" id="ID"required name="ID" pattern="[0-9][A-Z]{3}"
 placeholder="[0-9][A-Z]-Single digit & 3 letters"
 title="Single digit followed by three uppercase letters">
```

Here is screenshot of validation error if you don't enter the field in the expected pattern:

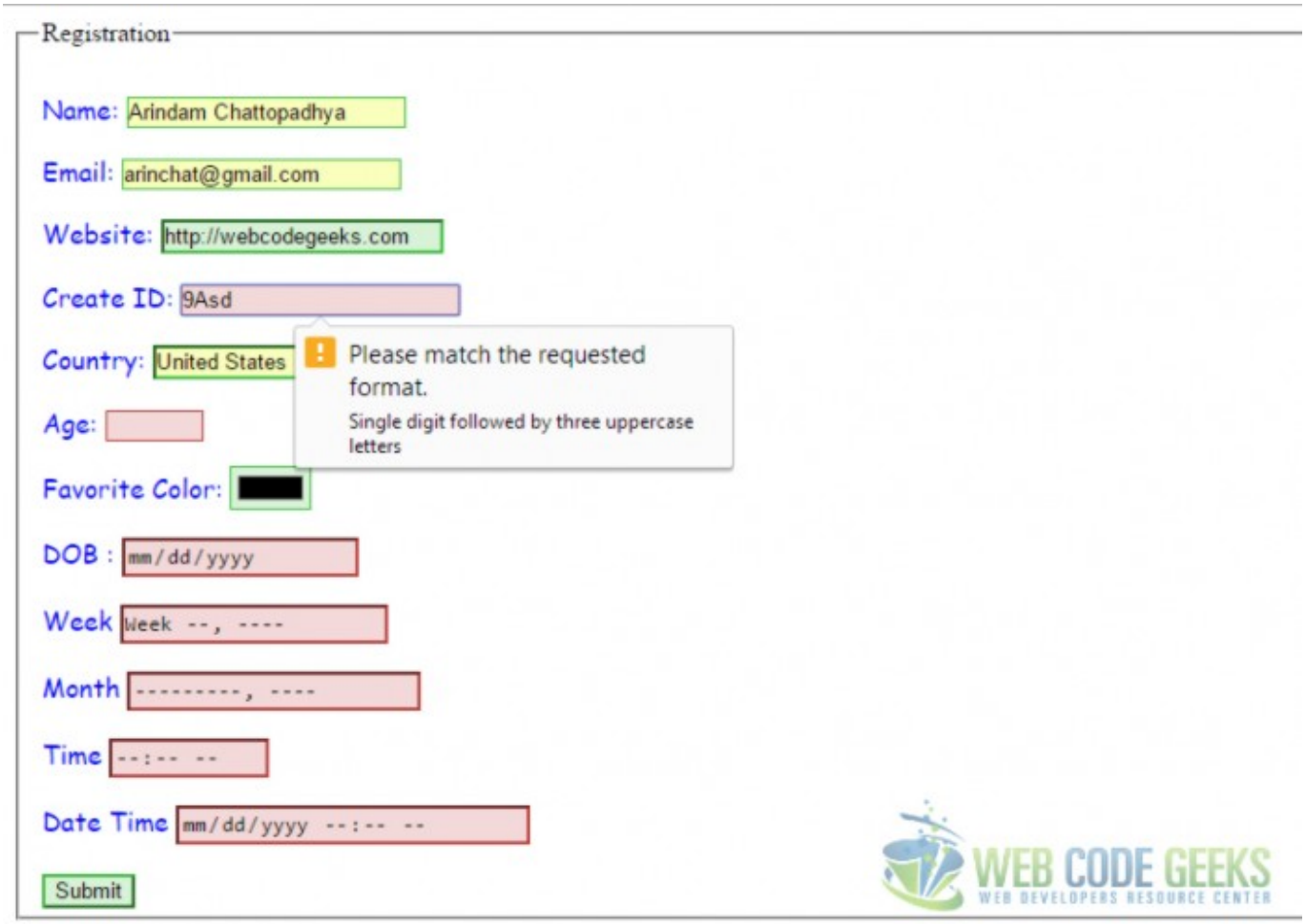

Figure 9.6: HTML Pattern

9.1.6 date, datetime-local,month,time,week

HTML5 finally defines a way to include a native date picker control without having to script it yourself. In fact, it defines 5 kinds of them: date, month, week, time, datetime-local.

```
<input type="date" id="Date" name="Date">

<input type="week" id="Week" name="Week">

<input type="month" id="Month" name="Month">

<input type="time" id="Time" name="Time">

<input type="datetime-local" id="DateTime" name="DateTime">
```

Below is screenshot of datepicker included in HTML5:

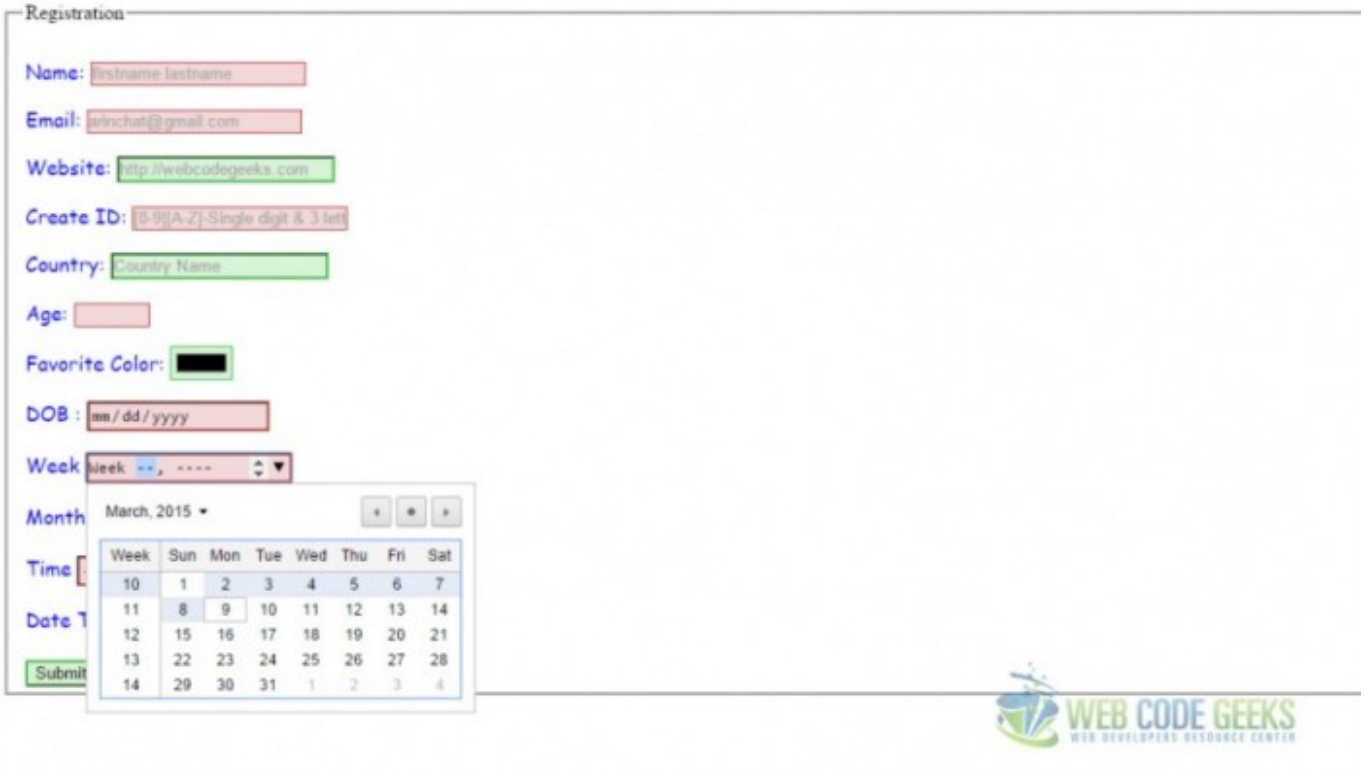

Figure 9.7: Date Picker

9.1.7 email

The input type as `email` is used for input fields that should contain an e-mail address. Depending on browser support, the e-mail address can be automatically validated when submitted. Here is a screenshot of an error shown if you do not fill the email in the correct way. Some smartphones recognize the email type, altering their virtual keyboard to the keyboard to match email input.

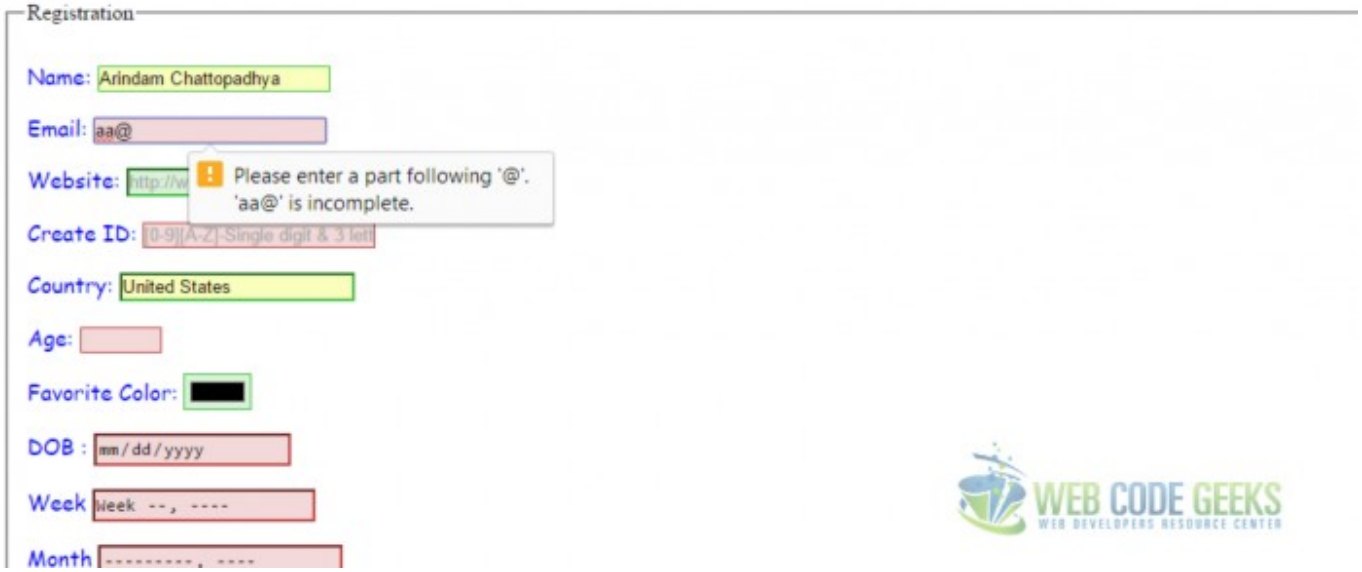

Figure 9.8: HTML Email Validation

9.1.8 url

The `<input type="url">` is used for input fields that should contain a URL address. Most modern desktop browsers simply render `type="url"` like a regular text box, so users won't even notice until they submit the form. Depending on browser support, the url field can be automatically validated when submitted. Some smartphones like iPhone recognize the url type, alters its virtual keyboard and adds ".com" a period, a forward slash, to the keyboard to match url input. Here is a screenshot of validation error for url.

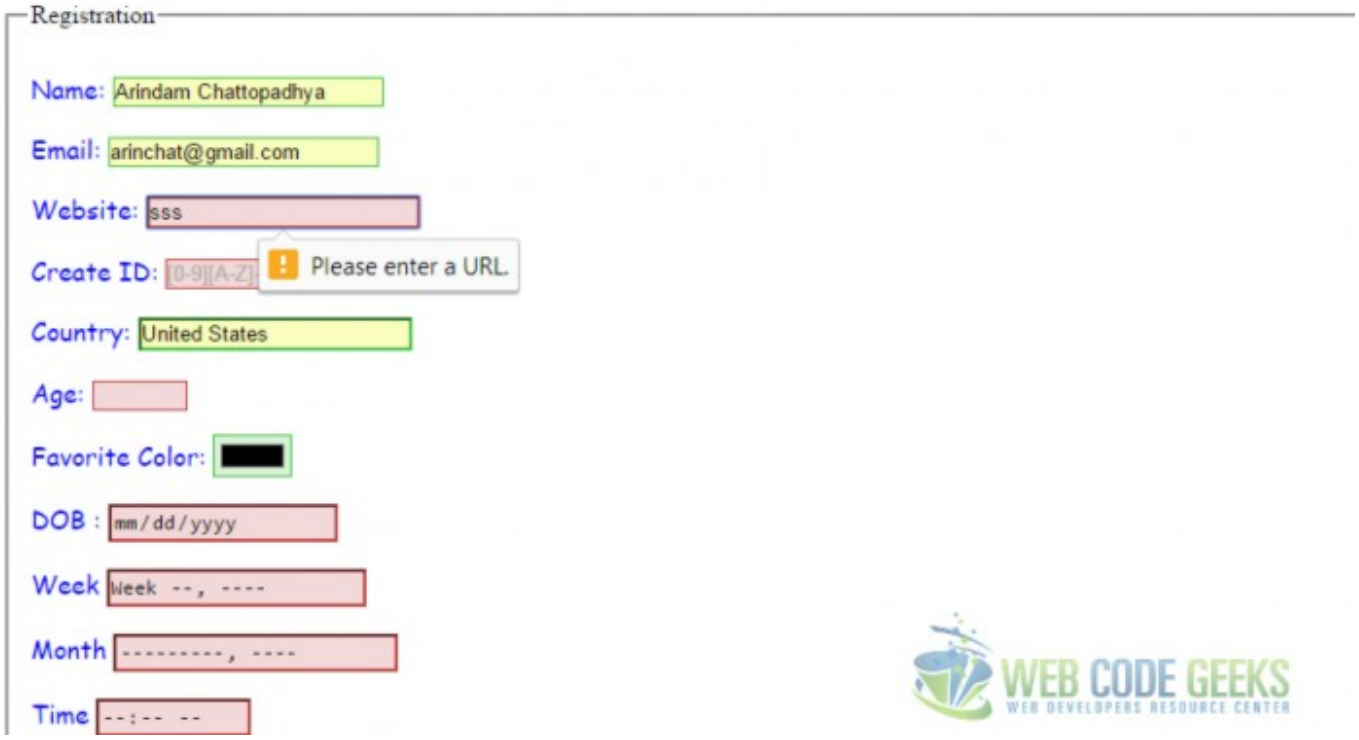

Figure 9.9: HTML URL Validation

9.1.9 color

The `<input type="color">` is used for input fields that should contain a color. You will get a color picker if your browser supports. This has not much of a usage for validation.I thought of mentioning it as the feature is pretty cool .

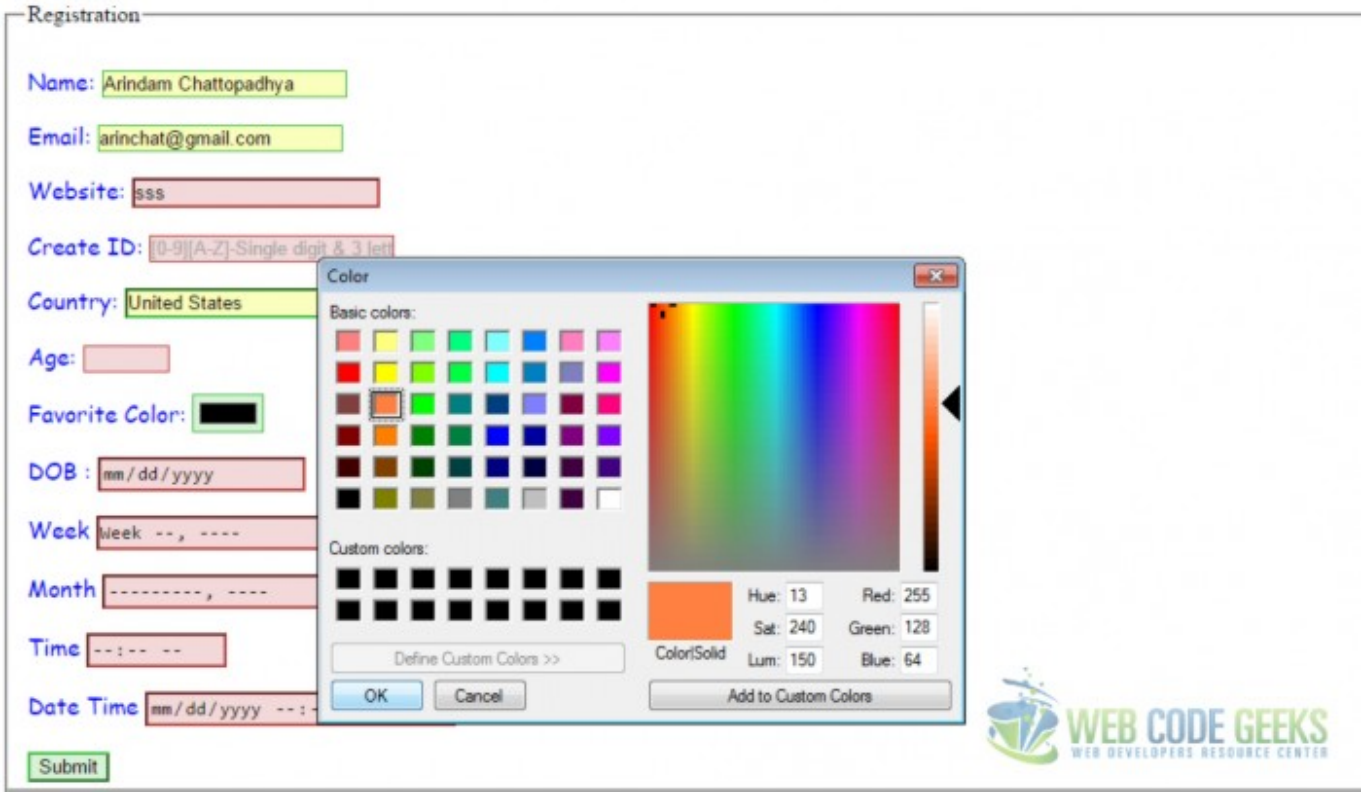

Figure 9.10: HTML Color Picker

Download: You can download the full source code of this example here: **HTML5Validation**