HTML5

What is New in HTML5?

The DOCTYPE declaration for HTML5

<! DOCTYPE html>

<html></html>

New HTML5 Elements

The most interesting new elements are:

New semantic elements like <header>, <footer>, <article>, and <section>.

New form control attributes like number, date, time, calendar, and range.

New graphic elements: <svg> and <canvas>.

New multimedia elements: <audio> and <video>.

New HTML5 API's (Application Programming Interfaces)

The most interesting new API's are:

- HTML Geolocation
- HTML Drag and Drop
- HTML Local Storage
- HTML Session Storage
- HTML Application Cache
- HTML Web Workers
- HTML SSE(Server Sent Event) Notification

History:

Tim Berners-Lee invented the "World Wide Web" in 1989, and the Internet took off in the 1990s.

From 1991 to 1998, HTML developed from version 1 to version 4.

In 2000, the World Wide Web Consortium (W3C) recommended XHTML 1.0.

The XHTML syntax was strict, and the developers were forced to write valid and "well-formed" code.

In 2004, WHATWG (Web Hypertext Application Technology Working Group) was formed in response to slow W3C development, and W3C's decision to close down the development of HTML, in favor of XHTML.

WHATWG wanted to develop HTML, consistent with how the web was used, while being backward compatible with older versions of HTML.

In the period 2004-2006, the WHATWG initiative gained support by the major browser vendors.

In 2006, W3C announced that they would support WHATWG.

In 2008, the first HTML5 public draft was released.

In 2012, WHATWG and W3C decided on a separation:

WHATWG will develop HTML as a "Living Standard".

A living standard is never fully complete, but always updated and improved. New features can be added, but old functionality can not be removed.

The WHATWG Living Standard was published in 2012, and is continuously

updated.

W3C will develop a definitive HTML5 and XHTML5 standard, as a "snapshot" of WHATWG.

The W3C HTML5 recommendation was released 28 October 2014.

HTML5 Browser Support

HTML5 is supported in all modern browsers.

In addition, all browsers, old and new, automatically handle unrecognized elements as inline elements.

Because of this, you can "teach" older browsers to handle "unknown" HTML elements.

Define HTML5 Elements as Block Elements

HTML5 defines eight new semantic HTML elements. All these are block-level elements.

To secure correct behavior in older browsers, you can set the CSS display property to block:

```
header, section, footer, aside, nav, main, article, figure {
    display: block;
}
```

Adding New Elements to HTML

You can also add any new element to HTML with a browser trick.

This example adds a new element called <MyTag> to HTML, and defines a

display style for it:

```
<!DOCTYPE html>
<html>
<head>
  <title>Creating an HTML Element</title>
  <script>document.createElement("MyTag")</script>
  <style>
  MyTag {
      display: block;
      background-color: #ddd;
      padding: 50px;
      font-size: 30px;
  }
  </style>
</head>
<body>
<h1>My First Heading</h1>
<p>My first paragraph.</p>
<MyTag>My First Hero</MyTag>
</body>
</html>
```

Problem With Internet Explorer

You could use the solution described above, for all new HTML5 elements, but:

```
<!DOCTYPE html>
<html>
<head>
  <title>Styling HTML5</title>
  <!--[if lt IE 9]>
```

```
  <script src="http://html5shiv.googlecode.com/svn/trunk/html5.js"></script>
  <![endif]-->
</head>
<body>
<h1>My First Article</h1>
<article>
London is the capital city of England. It is the most populous city in the United Kingdom, with a metropolitan area of over 13 million inhabitants.
</article>
</body>
</html>
```

New Input Types in HTML5

HTML5 introduces several new input types for forms

HTML5 introduces several new <input> types like email, date, time, color, range, etc. to improve the user experience and to make the forms more interactive. However, if a browser failed to recognize these new input types, it will treat them like a normal text box.

In this section we're going to take a brief look at each of the following new input types:

- Color
- date
- datetime
- datetime-local
- email
- month
- number
- range
- search

- tel
- time
- yrs
- week

Input Type Color

The color input type allows the user to select a color from a drop-down color picker and returns the hex value for that color.

```
<!DOCTYPE html>
<html lang="en">
<head>
<title>HTML5 Color Input Type</title>
</head>
<body>
    <form>
        <label>
            Select Color: <input type="color" name="mycolor">
        </label>
    </form>
</body>
</html>
```

Warning:The input type="color" is not supported by Internet Explorer and Apple Safari. Currently supported by Firefox, Chrome and Opera browsers.

Input Type Date

The date input type allows the user to select a date from a drop-down calendar.

```
<!DOCTYPE html>
<html lang="en">
```

```
<head>
<title>HTML5 Date Input Type</title>
</head>
<body>
   <form>
      <label>
         Select Date: <input type="date" name="mydate">
      </label>
   </form>
</body>
</html>
```

Warning:The input type="date" is not supported by Internet Explorer and Firefox. Currently supported by Chrome, Safari and Opera browsers.

Input Type Datetime

The datetime input type allows the user to select a date and time along with time zone.

```
<!DOCTYPE html>
<html lang="en">
<head>
<title>HTML5 Datetime Input Type</title>
</head>
<body>
   <form>
      <label>
         Date & Time: <input type="datetime" name="mydatetime">
      </label>
   </form>
</body>
</html>
```

Warning:The input type="datetime" is not supported by Internet Explorer, Firefox, Chrome and Opera browsers. Currently supported by Apple Safari only.

Input Type Datetime-local

The datetime-local input type allows the user to select a local date and time. The local date and time doesn't provide timezone information.

```
<!DOCTYPE html>
<html lang="en">
<head>
<title>HTML5 Datetime-local Input Type</title>
</head>
<body>
    <form>
        <label>
            Local Date & Time: <input type="datetime-local"
name="mylocaldatetime">
        </label>
    </form>
</body>
</html>
```

Warning:The input type="datetime-local" is not supported by Internet Explorer and Firefox. Currently supported by Chrome, Safari and Opera browsers.

Input Type Email

The email input type allows the user to enter e-mail address. It is very similar to a standard text input type, but if it used in combination with the required attribute, the browser may look for patterns to ensure a valid e-mail address should be entered.

```
<!DOCTYPE html>
<html lang="en">
<head>
<title>HTML5 Email Input Type</title>
<style type="text/css">
   input[type="email"]:valid{
      outline: 2px solid green;
   }
   input[type="email"]:invalid{
      outline: 2px solid red;
   }
</style>
</head>
<body>
   <form>
      <label>
         Email Address: <input type="email" name="mycolor" required>
      </label>
   </form>
</body>
</html>
```

Note:You can style the email field for different validation states, when an value is entered using the :valid, :invalid or :required pseudo class.

Input Type Month

The month input type allows the user to select a month and year from a drop-down calendar.

```
<!DOCTYPE html>
<html lang="en">
<head>
```

```
<title>HTML5 Month Input Type</title>
</head>
<body>
   <form>
      <label>
         Select Month: <input type="month" name="mymonth">
      </label>
   </form>
</body>
</html>
```

Warning:The input type="month" is not supported by Internet Explorer and Firefox. Currently supported by Chrome, Safari and Opera browsers.

Input Type Number

The number input type can be used for entering a numerical value. You can also restrict the user to enter only acceptable values using the additional attributes min, max, and step.

```
<!DOCTYPE html>
<html lang="en">
<head>
<title>HTML5 Number Input Type</title>
<style type="text/css">
   input[type="number"]:valid{
      outline: 2px solid green;
   }
   input[type="number"]:invalid{
      outline: 2px solid red;
   }
</style>
</head>
```

```
<body>
    <form>
        <label>
            Select Number: <input type="number" value="1" min="1" max="10"
step="0.5" name="mynumber">
        </label>
    </form>
    <p><strong>Note</strong>: If you try to enter the number out of the
range (1-10) or text character it will show error.</p>
</body>
</html>
```

Input Type Range

The range input type can be used for entering a numerical value within a specified range. It works very similar to number input, but it offer a simpler control for entering a number.

```
<!DOCTYPE html>
<html lang="en">
<head>
<title>HTML5 Range Input Type</title>
</head>
<body>
    <form>
        <label>
            Select Number: <input type="range" value="1" min="1" max="10"
step="0.5" name="mynumber">
        </label>
    </form>
</body>
</html>
```

The input type="range" is supported by all the major browsers like Mozilla

Firefox, Google Chrome, Apple Safari, Internet Explorer 10+ and Opera.

Input Type Search

The search input type can be used for creating search fields.
A search field typically behaves like a regular text field, but in some browser like Google Chrome and Apple Safari as soon as you start typing in a search box a small cross appears on the right side of the field that lets you quickly clear the search field.

```
<!DOCTYPE html>
<html lang="en">
<head>
<title>HTML5 Search Input Type</title>
</head>
<body>
   <form>
      <label>
         Search Website: <input type="search" name="mysearch">
      </label>
   </form>
</body>
</html>
```

Warning:The input type="search" is not supported by Firefox, IE 9 and earlier versions. Currently supported by Chrome, Safari, IE 10+ and Opera browsers.

Input Type Tel

The tel input type can be used for entering a telephone number.

```
<!DOCTYPE html>
<html lang="en">
<head>
```

```
<title>HTML5 Tel Input Type</title>
</head>
<body>
   <form>
      <label>
         Telephone Number: <input type="tel" name="mytelephone" required>
      </label>
   </form>
</body>
</html>
```

Warning:The validation for the input type="tel" is currently not supported by any browser, but it is still useful. The iPhone and some Android devices using the inputtype="tel" to change the virtual keyboard with a numeric keypad.

Input Type Time

The time input type can be used for entering a time.

```
<!DOCTYPE html>
<html lang="en">
<head>
<title>HTML5 Time Input Type</title>
</head>
<body>
   <form>
      <label>
         Select Time: <input type="time" name="mytime">
      </label>
   </form>
</body>
</html>
```

Warning:The input type="time" is not supported by Internet Explorer and Firefox. Currently supported by Chrome, Safari and Opera browsers.

Input Type URL

The url input type can be used for entering web addresses i.e. URL's. You can use the multiple attribute to enter more than one URL. Like type="email", a browser may carry out simple validation on URL fields and present an error message on form submission.

```
<!DOCTYPE html>
<html lang="en">
<head>
<title>HTML5 URL Input Type</title>
<style type="text/css">
    input[type="url"]:valid{
        outline: 2px solid green;
    }
    input[type="url"]:invalid{
        outline: 2px solid red;
    }
</style>
</head>
<body>
    <form>
        <label>
            Website URL: <input type="url" name="mywebsite" required>
        </label>
    </form>
    <p><strong>Note</strong>: Enter URL in the form like http://
www.google.com</p>
</body>
</html>
```

Note:The validation for the type="url" is supported by all major browsers like Firefox, Chrome, Safari, Opera, Internet Explorer 10+.

Input Type Week

The week input type allows the user to select a week and year from a drop-down calendar.

```
<!DOCTYPE html>
<html lang="en">
<head>
<title>HTML5 Week Input Type</title>
</head>
<body>
    <form>
        <label>
            Select Week: <input type="week" name="myweek">
        </label>
    </form>
</body>
</html>
```

Warning:The input type="week" is not supported by Internet Explorer and Firefox. Currently supported by Chrome, Safari and Opera browsers.

HTML5 Canvas

The canvas element is used to draw graphics on a web page.

What is Canvas

The HTML5 canvas element can be used to draw graphics on the webpage via scripting (usually JavaScript). The canvas was originally introduced by Apple for the Mac OS Dashboard widgets and to power graphics in the Safari web browser.

Later it was adopted by the Firefox, Google Chrome and Opera. Now the canvas is a part of the new HTML5 specification for next generation web technologies.

The <canvas> element isn't supported in some older browsers, but is supported in recent versions of all major browsers such as IE 9+, Firefox, Chrome, Safari and Opera. By default the canvas element has 300px of width and 150px of height without any border and content. However the custom width and height can be defined using CSS height and width property whereas border can be applied using the CSS border property.

Understanding Canvas Coordinates

The canvas is a two-dimensional rectangular area. The coordinates of the top-left corner of the canvas are (0, 0) which is known as origin and the coordinates in the bottom-right corner are (canvas width, canvas height) as demonstrated below.

Tip:Place your mouse pointer within the canvas area demonstrated above and you will get its current coordinates relative to the canvas.

Drawing Path and Shapes on Canvas

In this section we're going to take a look at how to draw basic paths and shapes using the newly introduced HTML5 canvas element and JavaScript. Here is the base template for drawing paths and shapes onto the 2D HTML5 Canvas.

```
<!DOCTYPE html>
<html lang="en">
```

```
<head>
<meta charset="UTF-8">
<title>HTML5 Canvas</title>
<script type="text/javascript">
   window.onload = function(){
      var canvas = document.getElementById("myCanvas");
      var context = canvas.getContext("2d");
      // draw stuff here
   };
</script>
</head>
<body>
   <canvas id="myCanvas" width="300" height="200"></canvas>
</body>
</html>
```

All the lines except those from 7 to 11 are pretty simple. The window.onload function defines to access the canvas element we need to wait till the page load. Once the page loads, we can access the canvas element with document.getElementById(). Later we have defined a 2D canvas context by passing 2d into the getContext() method of the canvas object.

Drawing a Line

The most basic path you can draw on canvas is a straight line. The methods used for this purpose are moveTo(), lineTo() and stroke().
The moveTo() method defines the position of drawing cursor onto the canvas, whereas thelineTo() method used to define the coordinates of the line's end point, and finally thestroke() method is used to make the line visible.

```
<!DOCTYPE html>
<html lang="en">
<head>
```

```
<meta charset="UTF-8">
<title>Drawing a Line on Canvas</title>
<style type="text/css">
  canvas{
        border: 1px solid #000;
  }
</style>
<script type="text/javascript">
   window.onload = function(){
      var canvas = document.getElementById("myCanvas");
      var context = canvas.getContext("2d");
      context.moveTo(50, 150);
      context.lineTo(250, 50);
      context.stroke();
   };
</script>
</head>
<body>
   <canvas id="myCanvas" width="300" height="200"></canvas>
</body>
</html>
```

Drawing a Arc

You can create arc path like smiley faces using the arc() method. The basic syntax of arc()method can be given with:

```
context.arc(centerX, centerY, radius, startingAngle, endingAngle, counterclockwise);
```

The following example will draw a simple arc on the canvas.

```
<!DOCTYPE html>
<html lang="en">
<head>
```

```
<meta charset="UTF-8">
<title>Drawing an Arc on Canvas</title>
<style type="text/css">
  canvas{
        border: 1px solid #000;
  }
</style>
<script type="text/javascript">
   window.onload = function(){
      var canvas = document.getElementById("myCanvas");
      var context = canvas.getContext("2d");
      context.arc(150, 150, 80, 1.2 * Math.PI, 1.8 * Math.PI, false);
      context.stroke();
   };
</script>
</head>
<body>
   <canvas id="myCanvas" width="300" height="200"></canvas>
</body>
</html>
```

Drawing a Rectangle

You can create simple rectangle and square using the rect() method. The rect() method requires four parameter x, y position of the rectangle and its width and height. The basic syntax of the rect() method can be given with:

```
context.rect(x, y, width, height);
```

The following example will draw a simple rectangle centered on the canvas.

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
```

```
<title>Drawing a Rectangle on Canvas</title>
<style type="text/css">
  canvas{
        border: 1px solid #000;
  }
</style>
<script type="text/javascript">
   window.onload = function(){
      var canvas = document.getElementById("myCanvas");
      var context = canvas.getContext("2d");
      context.rect(50, 50, 200, 100);
      context.stroke();
   };
</script>
</head>
<body>
   <canvas id="myCanvas" width="300" height="200"></canvas>
</body>
</html>
```

Drawing a Circle

There is no specific method for creating circle like rectangle's rect() method. However you can create a fully enclosed arc such as circle using the arc() method. The basic syntax for drawing a complete circle can be given with:

```
context.arc(centerX, centerY, radius, 0, 2 * Math.PI, false);
```

The following example will draw a complete circle centered on the canvas.

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<title>Drawing a Circle on Canvas</title>
```

```
<style type="text/css">
  canvas{
        border: 1px solid #000;
  }
</style>
<script type="text/javascript">
   window.onload = function(){
      var canvas = document.getElementById("myCanvas");
      var context = canvas.getContext("2d");
      context.arc(150, 100, 70, 0, 2 * Math.PI, false);
      context.stroke();
   };
</script>
</head>
<body>
   <canvas id="myCanvas" width="300" height="200"></canvas>
</body>
</html>
```

Applying Styles and Colors on Stroke

The default color of the stroke is black and its thickness is one pixel. But, you can set the color and width of the stoke using the strokeStyle and lineWidth property respectivley.

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<title>Example of Setting Stroke Color and Width</title>
<style type="text/css">
  canvas{
```

```
            border: 1px solid #000;
    }
</style>
<script type="text/javascript">
    window.onload = function(){
        var canvas = document.getElementById("myCanvas");
        var context = canvas.getContext("2d");
        context.lineWidth = 5;
        context.strokeStyle = "orange";
        context.moveTo(50, 150);
        context.lineTo(250, 50);
        context.stroke();
    };
</script>
</head>
<body>
    <canvas id="myCanvas" width="300" height="200"></canvas>
</body>
</html>
```

You can also set the cap style for the lines using the lineCap property. There are three styles available for the line caps — butt, round, and square.

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<title>Example of Setting Stroke Cap Style</title>
<style type="text/css">
    canvas{
            border: 1px solid #000;
    }
</style>
<script type="text/javascript">
```

```
    window.onload = function(){
        var canvas = document.getElementById("myCanvas");
        var context = canvas.getContext("2d");
        context.lineWidth = 10;
        context.strokeStyle = "orange";
        context.lineCap = "round";
        context.arc(150, 150, 80, 1.2 * Math.PI, 1.8 * Math.PI, false);
        context.stroke();
    };
</script>
</head>
<body>
    <canvas id="myCanvas" width="300" height="200"></canvas>
</body>
</html>
```

Filling Colors inside Canvas Shapes

You can also fill color inside the canvas shapes using the fillStyle() method. The following example will shows you how to fill a solid color inside a rectangle.

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<title>Example of Filling Color inside a Rectangle</title>
<style type="text/css">
   canvas{
            border: 1px solid #000;
   }
</style>
<script type="text/javascript">
    window.onload = function(){
```

```
        var canvas = document.getElementById("myCanvas");
        var context = canvas.getContext("2d");
        context.rect(50, 50, 200, 100);
        context.fillStyle = "#FB8B89";
        context.fill();
        context.lineWidth = 5;
        context.strokeStyle = "black";
        context.stroke();
    };
</script>
</head>
<body>
    <canvas id="myCanvas" width="300" height="200"></canvas>
</body>
</html>
```

Tip:It is recommended to use the fill() method before the stroke() method in order to render the stroke correctly.
Similarly, using the fillStyle() method you can fill color inside a circle.

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<title>Example of Filling Color inside a Circle</title>
<style type="text/css">
   canvas{
            border: 1px solid #000;
   }
</style>
<script type="text/javascript">
    window.onload = function(){
        var canvas = document.getElementById("myCanvas");
```

```
        var context = canvas.getContext("2d");
        context.arc(150, 100, 70, 0, 2 * Math.PI, false);
        context.fillStyle = "#FB8B89";
        context.fill();
        context.lineWidth = 5;
        context.strokeStyle = "black";
        context.stroke();
    };
</script>
</head>
<body>
    <canvas id="myCanvas" width="300" height="200"></canvas>
</body>
</html>
```

Filling Gradient Colors inside Canvas Shapes

You can also fill gradient color inside the canvas shapes instead of solid colors. There are two gradient style available in HTML5 — linear and radial.
The basic syntax of the linear gradient can be given with:
var grd = context.createLinearGradient(startX, startY, endX, endY);
The following example will show you how to fill linear gradient inside a rectangle using thecreateLinearGradient() method.

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<title>Example of Filling Linear Gradient inside Canvas Shapes</title>
<style type="text/css">
   canvas{
          border: 1px solid #000;
   }
```

```
</style>
<script type="text/javascript">
    window.onload = function(){
        var canvas = document.getElementById("myCanvas");
        var context = canvas.getContext("2d");
        context.rect(50, 50, 200, 100);
        var grd = context.createLinearGradient(0, 0, canvas.width,
canvas.height);
        grd.addColorStop(0, '#8ED6FF');
        grd.addColorStop(1, '#004CB3');
        context.fillStyle = grd;
        context.fill();
        context.stroke();
    };
</script>
</head>
<body>
    <canvas id="myCanvas" width="300" height="200"></canvas>
</body>
</html>
```

Similarly, you can fill canvas shapes with radial gradient using the createRadialGradient()method. The basic syntax of the radial gradient can be given with:

```
var grd = context.createRadialGradient(startX, startY, startRadius, endX, endY,
endRadius);
```

The following example will show you how to fill radial gradient inside a circle using thecreateRadialGradient() method.

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<title>Example of Filling Radial Gradient inside Canvas Shapes</title>
<style type="text/css">
   canvas{
```

```
            border: 1px solid #000;
    }
</style>
<script type="text/javascript">
    window.onload = function(){
        var canvas = document.getElementById("myCanvas");
        var context = canvas.getContext("2d");
        context.arc(150, 100, 70, 0, 2 * Math.PI, false);
        var grd = context.createRadialGradient(150, 100, 10, 160, 110, 100);
        grd.addColorStop(0, '#8ED6FF');
        grd.addColorStop(1, '#004CB3');
        context.fillStyle = grd;
        context.fill();
        context.stroke();
    };
</script>
</head>
<body>
    <canvas id="myCanvas" width="300" height="200"></canvas>
</body>
</html>
```

Drawing Text on Canvas

You can also draw text onto canvas. These texts can contain any Unicode characters. The following example will draw a simple greeting message "Hello World!" onto a canvas.

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<title>Example of Drawing Text on Canvas</title>
<style type="text/css">
    canvas{
            border: 1px solid #000;
    }
</style>
<script type="text/javascript">
    window.onload = function(){
        var canvas = document.getElementById("myCanvas");
```

```
        var context = canvas.getContext("2d");
        context.font = "bold 32px Arial";
        context.fillText("Hello World!", 50, 100);
    };
</script>
</head>
<body>
    <canvas id="myCanvas" width="300" height="200"></canvas>
</body>
</html>
```

You can also set the text color and its alignment on canvas.

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<title>Example of Setting Canvas Text Color and Alignment</title>
<style type="text/css">
   canvas{
            border: 1px solid #000;
   }
</style>
<script type="text/javascript">
    window.onload = function(){
        var canvas = document.getElementById("myCanvas");
        var context = canvas.getContext("2d");
        context.font = "bold 32px Arial";
        context.textAlign = "center";
        context.textBaseline = "middle";
        context.fillStyle = "orange";
        context.fillText("Hello World!", 150, 100);
    };
```

```
</script>
</head>
<body>
    <canvas id="myCanvas" width="300" height="200"></canvas>
</body>
</html>
```

You can also apply stroke on text using the strokeText() method. This method will color the perimeter of the text instead of filling it. However if you want to set both the fill and stroke on canvas text you can use both the fillText() and the strokeText() methods together.

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<title>Example of Adding Stroke to Canvas Text</title>
<style type="text/css">
   canvas{
            border: 1px solid #000;
   }
</style>
<script type="text/javascript">
    window.onload = function(){
        var canvas = document.getElementById("myCanvas");
        var context = canvas.getContext("2d");
        context.font = "bold 32px Arial";
        context.textAlign = "center";
        context.textBaseline = "middle";
        context.strokeStyle = "orange";
        context.strokeText("Hello World!", 150, 100);
    };
</script>
```

```
</head>
<body>
   <canvas id="myCanvas" width="300" height="200"></canvas>
</body>
</html>
```

Tip:It is recommended to use the fillText() method before the strokeText()method in order to render the stroke correctly.

HTML5 SVG

The SVG stands for Scalable Vector Graphics.

What is SVG

The Scalable Vector Graphics (SVG) is an XML-based image format that is used to define two-dimensional vector based graphics for the Web. A vector image can be scaled up or down to any extent without losing the image quality.
SVG images and their behaviors are defined in XML files — that means SVG images can be created and edited with any text editor. There are several other advantages of using SVG over other image formats like JPEG, PNG, GIF etc.

- SVG images can be searched, indexed, scripted, and compressed.
- SVG images can be created and modifed using JavaScript in real time.
- SVG images can be printed with high quality at any resolution.
- SVG content can be animated using the built-in animation elements.
- SVG images can contain hyperlinks to other documents.

Tip:The vector images are composed of a fixed set of shapes defined by math, while the bitmap or raster images are composed of a fixed set of dots called pixels.

Embedding SVG Into HTML Pages

You can embed SVG graphics directly into your document using the HTML5 <svg> element.

```
<!DOCTYPE html>
```

```
<html lang="en">
<head>
<meta charset="UTF-8">
<title>Embedding SVG Into HTML Pages</title>
</head>
<body>
   <svg width="300" height="200">
      <text x="10" y="20" style="font-size:14px;">
         Your browser support SVG.
      </text>
      Sorry, your browser does not support SVG.
   </svg>
</body>
</html>
```

Note:All the major modern web browsers like Firefox, Chrome, Safari, Opera, and Internet Explorer 9+ support inline SVG rendering.

Drawing Path and Shapes with SVG

The following section will explain you how to draw basic vector-based paths and shapes on the web pages using the HTML5 <svg> element.

Drawing a Line

Line is the most basic path you can draw with SVG. The following example will show you how to create a straight line using the SVG <line> element:

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<title>Create a Line with HTML5 SVG</title>
<style type="text/css">
   svg {
```

```
        border: 1px solid black;
    }
</style>
</head>
<body>
    <svg width="300" height="200">
        <line x1="50" y1="50" x2="250" y2="150" style="stroke:red; stroke-
width:3;" />
    </svg>
</body>
</html>
```

Drawing a Rectangle

You can create simple rectangle and square shapes using the SVG <rect> element. The following example will show you how to create and style a rectactangular shape with SVG:

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<title>Create a Rectangle with HTML5 SVG</title>
<style type="text/css">
    svg {
        border: 1px solid black;
    }
</style>
</head>
<body>
    <svg width="300" height="200">
        <rect x="50" y="50" width="200" height="100" style="fill:orange;
stroke:black; stroke-width:3;" />
    </svg>
</body>
</html>
```

Drawing a Circle

You can create circle shapes using the SVG <circle> element. The following example will show you how to create and style a circular shape with SVG:

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<title>Create a Circle with HTML5 SVG</title>
<style type="text/css">
   svg {
      border: 1px solid black;
   }
</style>
</head>
<body>
   <svg width="300" height="200">
      <circle cx="150" cy="100" r="70" style="fill:lime; stroke:black; stroke-
width:3;" />
   </svg>
</body>
</html>
```

Drawing Text with SVG

You can also draw text on web pages with SVG. The text in SVG is rendered as a graphic so you can apply all the graphic transformation to it but it is still acts like text — that means it can be selected and copied as text by the user.

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
```

```
<title>Render Text with HTML5 SVG</title>
<style type="text/css">
   svg {
      border: 1px solid black;
   }
</style>
</head>
<body>
   <svg width="300" height="200">
      <text x="20" y="30" style="fill:purple; font-size:22px;">
         Welcome to Our Website!
      </text>
      <text x="20" y="30" dx="0" dy="20" style="fill:navy; font-size:14px;">
         Here you will find lots of useful information.
      </text>
   </svg>
</body>
</html>
```

The attributes x and y of the <text> element defines the location of the top-left corner in absolute terms whereas the attributes dx and dy specifies the relative location.

You can also use the <tspan> element to reformat or reposition the span of text contained within a <text> element. Text contained in separate tspans, but inside the same text element can all be selected at once — when you click and drag to select the text. However the text in separate text elements can't be selected at the same time.

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<title>Rotate and Render Text with HTML5 SVG</title>
```

```
<style type="text/css">
    svg {
        border: 1px solid black;
    }
</style>
</head>
<body>
    <svg width="300" height="200">
        <text x="30" y="15" style="fill:purple; font-size:22px;
transform:rotate(30deg);">
            <tspan style="fill:purple; font-size:22px;">
                Welcome to Our Website!
            </tspan>
            <tspan dx="-230" dy="20" style="fill:navy; font-size:14px;">
                Here you will find lots of useful information.
            </tspan>
        </text>
    </svg>
</body>
</html>
```

Differences Between SVG and Canvas

The HTML5 introduced the two graphical elements Canvas and SVG for creating rich graphics on the web, but they are fundamentally different. The following table summarizes some of the basic differences between these two elements, which will help you to understand how to use the Canvas and SVG elements effectively and appropriately.

SVG	Canvas
Vector based (composed of shapes)	Raster based (composed of pixel)

Multiple graphical elements, which become the part of the DOM	Single HTML element similar to <img> in behavior
Modified through script and CSS	Modified through script only
Give better performance with smaller number of objects or larger surface, or both	Give better performance with smaller surface or larger number of objects, or both
Better scalability — can be printed with high quality at any resolution	Poor scalability — not suitable for printing on higher resolution

HTML5 Audio

There are different ways to play sounds on a Web Page.

Embedding Audio in HTML Document

Inserting sound onto a web page is not relatively easy, because browsers did not have a uniform standard for defining embedded media files.

In this chapter we'll demonstrates some of the many ways to embed sound in

your webpage, from the use of a simple link to the use of the latest HTML5 <audio> element.

Using the HTML5 audio Element

The newly introduced HTML5 <audio> element provides a standard way to embed audio in web pages. However, the audio element is relatively new, but it works in most of the modern web browsers. The following example simply inserts an audio into the HTML5 document, using the browser default set of controls, with one source.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>The HTML5 audio Element</title>
</head>
<body>
   <audio controls="controls" src="../audio/birds.mp3">
        Your browser does not support the HTML5 audio element.
    </audio>
</body>
</html>
```

An audio, using the browser default set of controls, with alternative sources.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>The HTML5 audio Element</title>
</head>
<body>
    <audio controls="controls">
```

```
        <source src="../audio/birds.mp3" type="audio/mpeg">
        <source src="../audio/birds.ogg" type="audio/ogg">
        Your browser does not support the HTML5 audio element.
    </audio>
</body>
</html>
```

The 'ogg' track in the above example works in Firefox, Opera and Chrome, while the same track in the 'mp3' format is added to make the audio work in Internet Explorer and Safari.

Linking Audio Files

You can make links to your audio files and pay it by ticking on them.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Example of Linking Audio Files</title>
</head>
<body>
    <p><a href="../audio/sea.mp3">Track 1</a></p>
    <p><a href="../audio/wind.mp3">Track 2</a></p>
</body>
</html>
```

Using the object Element

The <object> element is used to embed different kinds of media files into an HTML document. Initially, this element was used to insert ActiveX controls, but according to the specification, an object can be any media object such as video, audio, Java applets, ActiveX, document (HTML, PDF, Word, etc.), Flash animations or even images. Here's an example:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Example of Inserting Audio Using object Element</title>
</head>
<body>
    <object data="../audio/sea.mp3" width="200px" height="50px"></object>
    <object data="../audio/sea.ogg" width="200px" height="50px"></object>
</body>
</html>
```

Warning:The <object> element is not supported widely and very much depends on the type of the object that's being embedded. Other methods like HTML5 <audio>element or Google MP3 player could be a better choice in many cases.

Using the embed Element

The <embed> element is used to embed multimedia content into an HTML document.

The following code fragment embeds audio files into a web page.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Example of Inserting Audio Using embed Element</title>
</head>
<body>
    <embed src="../audio/wind.mp3" width="200px" height="50px">
    <embed src="../audio/wind.ogg" width="200px" height="50px">
</body>
</html>
```

Warning:However the <embed> element is very well supported in current browsers and defined as standard in HTML5, but your audio might not played

due to lack of browser support for that file format or unavailability of plugins.

HTML5 Video

There are different ways to play videos on a Web Page.

Embedding Video in HTML Document

Inserting video onto a web page is not relatively easy, because browsers did not have a uniform standard for defining embedded media files.
In this chapter we'll demonstrates some of the many ways of adding videos on web pages, from the latest HTML5 <video> element to the popular YouTube videos.

Using the HTML5 video Element

The newly introduced HTML5 <video> element provides a standard way to embed video in web pages. However, the video element is relatively new, but it works in most of the modern web browsers. The following example simply inserts a video into the HTML5 document, using the browser default set of controls, with one source.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>The HTML5 video element</title>
</head>
<body>
   <video controls="controls" src="../video/shuttle.mp4">
        Your browser does not support the HTML5 Video element.
    </video>
</body>
</html>
```

A video, using the browser default set of controls, with alternative sources.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>The HTML5 video element</title>
</head>
<body>
   <video controls="controls">
        <source src="../video/shuttle.mp4" type="video/mp4">
        <source src="../video/shuttle.ogv" type="video/ogg">
        Your browser does not support the HTML5 Video element.
    </video>
</body>
</html>
```

Using the object Element

The <object> element is used to embed different kinds of media files into an HTML document. Initially, this element was used to insert ActiveX controls, but according to the specification, an object can be any media object such as video, audio, Java applets, ActiveX, document (HTML, PDF, Word, etc.), Flash animations or even images.

The following code fragment embeds a Flash video into a web page.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Example of Inserting Video Using object Element</title>
</head>
<body>
   <object data="../video/blur.swf" width="400px" height="200px"></object>
</body>
```

```
</html>
```

Only browsers or applications that support Flash will play this video.

Warning:The <object> element is not supported widely and very much depends on the type of the object that's being embedded. Other methods could be a better choice in many cases. iPad and iPhone cannot display Flash videos.

Using the embed Element

The <embed> element is used to embed multimedia content into an HTML document.

The following code fragment embeds a Flash video into a web page.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Example of Inserting Video Using embed Element</title>
</head>
<body>
   <embed src="../video/blur.swf" width="400px" height="200px">
</body>
</html>
```

Embedding the YouTube Videos

This is the easiest and popular way to embed videos files in the web pages. Just upload the video on YouTube and insert HTML code to display that video in your web page.

Here's a live example followed by the explanation of whole process:

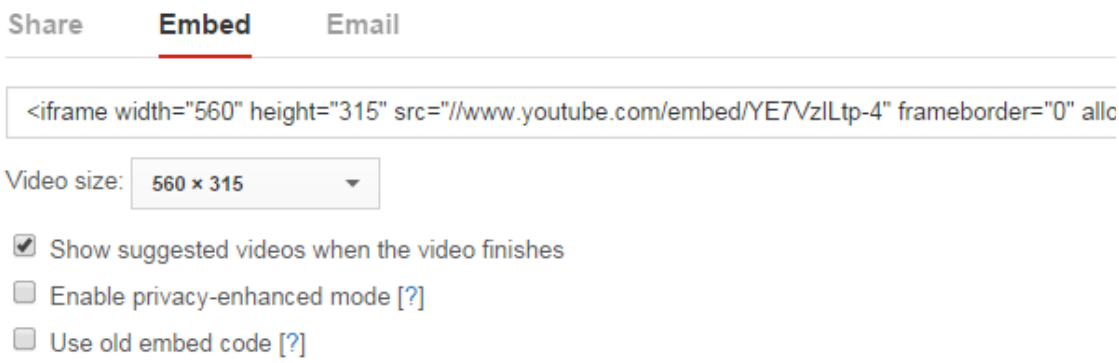
Step 1: Upload video

Go to YouTube Upload video page and follow the instructions to upload your video.

Step 2: Creating the HTML Code to embed the video

When you open your uploaded video in YouTube you will see something like this on the bottom of video. Browse and open your uploaded video in YouTube. Now look for the share button, located just below the video as shown in the figure.

When you click the Share button, a share panel will open displaying some more buttons. Now click on the Embed button, it will generate the HTML code to directly embed the video into the web pages. Just copy and paste that code into your HTML document where you want to display the video and you're all set. By default video embedded inside an iframe.



You can further customize this embed code such as changing the video size by selecting the customization option given just below the embed-code input box. Here's an example:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Example of Embedding a YouTube Video</title>
</head>
<body>
```

```
   <iframe width="560" height="315" src="//www.youtube.com/embed/
YE7VzlLtp-4" frameborder="0" allowfullscreen></iframe>
    <p>Source: <a href="http://www.youtube.com/watch?v=YE7VzlLtp-4"
target="_blank" rel="nofollow">http://www.youtube.com/watch?
v=YE7VzlLtp-4</a></p>
</body>
</html>
```

HTML5 Web Storage

The HTML5 web storage is used to store data on user's browser.

What is Web Storage

The HTML5's web storage feature lets you store some information locally on the user's computer, similar to cookies but it is more secure and faster. The information stored in the web storage isn't sent to the web server as opposed to the cookies where data sent to the server with every request. Also, where cookies lets you store a small amount of data (nearly 4KB), the web storage allows you to store up to 5MB of data.

There are two types of web storage, which differ in scope and lifetime:

Local storage — The local storage uses the localStorage object to store data for your entire website, permanently. That means the stored local data will be available on the next day, the next week, or the next year unless you remove it.

Session storage — The session storage uses the sessionStorage object to store data on a temporary basis, for a single window (or tab). The data disappears when session ends i.e. when the user closes that window (or tab).

Tip:The HTML5's web storage feature is supported in all major modern browsers like Firefox, Chrome, Opera, Safari and Internet Explorer 8+.

The localStorage Object

As stated earlier, the localStorage object stores the data with no expiration date. Each piece of data is stored in a key/value pair. The key identifies the name of the information (like 'first_name'), and the value is the value associated

with that key (say 'Peter').

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<title>Example of HTML5 Local Storage</title>
<script src="http://ajax.googleapis.com/ajax/libs/jquery/1.11.1/
jquery.min.js"></script>
<script type="text/javascript">
// Check if the localStorage object exists
if(localStorage){
   $(document).ready(function(){
          $(".save").click(function(){
                 // Get input name
                 var firstName = $("#firstName").val();

                 // Store data
                 localStorage.setItem("first_name", firstName);
                 alert("Your first name is saved.");
          });
          $(".access").click(function(){
                 // Retrieve data
                 alert("Hi, " + localStorage.getItem("first_name"));
          });
   });
} else{
    alert("Sorry, your browser do not support local storage.");
}
</script>
</head>
<body>
```

```
    <form>
        <label>First Name: <input type="text" id="firstName"></label>
      <button type="button" class="save">Save Name</button>
      <button type="button" class="access">Get Name</button>
    </form>
</body>
</html>
```

Example explained:

The above JavaScript code has the following meaning:

localStorage.setItem(key, value): Stores a value associated with a key.

localStorage.getItem(key): Retrieves the value associated with the key.

You can also remove a particular item from the storage by passing the key value to theremoveItem() method, like localStorage.removeItem(key). However, if you want to remove the complete storage use the clear() method, likelocalStorage.clear(). The clear() method clears all key/value pairs from localStorage at once, so think carefully before you using it.

Note:The web storage data (both local and session) will not be available between different browsers, for example the data stored in Firefox browser will not available in Google Chrome, Internet Explorer or other browsers.

The sessionStorage Object

The sessionStorage object work in the same way as localStorage, except that it stores the data only for one session i.e. the data remains until the user closes that window or tab.

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<title>Example of HTML5 Session Storage</title>
<script src="http://ajax.googleapis.com/ajax/libs/jquery/1.11.1/
```

```
jquery.min.js"></script>
<script type="text/javascript">
// Check if the localStorage object exists
if(localStorage){
   $(document).ready(function(){
          $(".save").click(function(){
                // Get input name
                var lastName = $("#lastName").val();

                // Store data
                sessionStorage.setItem("last_name", lastName);
                alert("Your last name is saved.");
          });
          $(".access").click(function(){
                // Retrieve data
                alert("Hi, " + localStorage.getItem("first_name") + " " +
sessionStorage.getItem("last_name"));
          });
   });
} else{
    alert("Sorry, your browser do not support local storage.");
}
</script>
</head>
<body>
    <form>
          <label>Last Name: <input type="text" id="lastName"></label>
       <button type="button" class="save">Save Name</button>
       <button type="button" class="access">Get Name</button>
    </form>
</body>
</html>
```

HTML5 Application Cache

Your can create offline applications with HTML5 caching feature.

What is Application Cache

Typically most web-based applications will work only if you're online.

But HTML5 introduces an application cache mechanism that allows the browser to automatically save the HTML file and all the other resources that needs to display it properly on the local machine, so that the browser can still access the web page and its resources without an internet connection.

Here are some advantages of using HTML5 application cache feature:

Offline browsing — Users can use the application even when they're offline or there are unexpected disruptions in network connection.

Improve performance — Cached resources load directly from the user's machine rather than the remote server hence web pages load faster and performing better.

Reduce HTTP request and server load — The browser will only have to download the updated/changed resources from the remote server that minimize the HTTP request and reduce the server load.

Tip:The HTML5's application cache feature is supported in all major modern browsers like Firefox, Chrome, Opera, Safari and Internet Explorer 10+.

Caching Files with a Manifest

To cache the files for offline uses, you need to complete the following steps:

Step 1: Create a Cache Manifest File

A manifest is a special text file that tells browsers what files to store, what files not to store, and what files to replace with something else. The manifest file always starts with the wordsCACHE MANIFEST (in uppercase). Here is an

example of a simple manifest file:

Explanation of code

You might think what that code was all about. OK, let's get straight into it. A manifest file can have three distinct sections: CACHE, NETWORK, and FALLBACK.

Files listed under the CACHE: section header (or immediately after the CACHE MANIFESTline) are explicitly cached after they're downloaded for the first time.

Files listed under the NETWORK: section header are white-listed resources that are never cached and are not available offline.

The FALLBACK: section specifies fallback pages the browser should use in case the connection to the server cannot be established. Each entry in this section lists two URIs — the first is the resource, the second is the fallback.

Lines starting with a '#' are comment lines.

Warning:Do not specify the manifest file itself in the cache manifest file, otherwise it will be nearly impossible to inform the browser a new manifest is available.

Step 2: Using Your Cache Manifest File

After creating, upload your cache manifest file on the web server — make sure the web server is configured to serve the manifest files with the MIME type text/cache-manifest.

Now to put your cache manifest into effect, you need enable it in your web pages, by adding the manifest attribute to the root <html> element, like this:

```
<!DOCTYPE html>
<html lang="en" manifest="example.appcache">
<head>
    <title>HTML5 Application Cache</title>
</head>
<body>
```

```
    <!--The document content will be inserted here-->
</body>
</html>
```

Note:On the Apache web servers, the MIME type for the manifest (.appcache) files can be set by adding AddType text/cache-manifest .appcache to a .htaccess file within the root directory of the application.

HTML5 Web Workers

A web worker is a JavaScript code that runs in the background of the web page.

What is Web Workers

If you try to do intensive task with JavaScript that is time-consuming and require hefty calculations it will freeze up the web page and interrupt the user until the job is completed. It happens because JavaScript code always runs in the foreground.

HTML5 introduces a new technology called Web worker that is specifically designed to do background work independently of other user-interface scripts, without affecting the performance of the page. Unlike JavaScript the Web worker doesn't interrupt the user and the web page remain responsive because they are running tasks in the background.

Tip:The HTML5's web worker feature is supported in all major modern browsers like Firefox, Chrome, Opera, Safari and Internet Explorer 10+.

Create a Web Worker File

The simplest use of web workers is for performing a time-consuming task. So here we are going to create a simple JavaScript task that counts from zero to 100,000. Let's create an external JavaScript file named "worker.js" and type the following code.

Note:The postMessage() method is used to send a message (like the numbers in the example above) back to the web page from the web worker file.

Doing Work in the Background with Web Worker

Now that we have created our web worker file. In this section we are going to initiate the web worker from an HTML document that runs the code inside the file named "worker.js" in the background and progressively displays the result on the web page.

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<title>Example of HTML5 Web Worker</title>
<script type="text/javascript">
    if(window.Worker){
        // Set up global variable
        var worker;

        // Create a new web worker
        worker = new Worker("../js/worker.js");

        // Fire onMessage event handler
        worker.onmessage = function(event){
            document.getElementById("result").innerHTML = event.data;
        };
    } else{
        alert("Sorry, your browser do not support web worker.");
    }
</script>
</head>
<body>
    <div id="result">
        <!--Received messages will be inserted here-->
    </div>
</body>
```

```
</html>
```

Example explained:

The JavaScript code in the above example has the following meaning:

The statement var worker = new Worker("worker.js"); creates a new web worker object, which is used to communicate with the web worker.

When the worker posts a message, it fires the onmessage event handler that allows the code to receive messages from the web worker.

The event.data element contains the message sent from the web worker.

Note:The code that a worker runs is always stored in a separate JavaScript file. This is to prevent web developer from writing the web worker code that attempts to use global variables or directly access elements on the page.

Terminate a Web Worker

So far you have learnt how to create worker and start receiving messages. However, you can also terminate a running worker in the middle of a calculation.

The following example will show you how to start and stop worker from a web page through clicking the HTML buttons. It utilizes the same JavaScript file 'worker.js' what we have used in the previous example to count the numbers from zero to 100000.

```
<!DOCTYPE html>
<html lang="en">
<head>
<title>HTML5 Web Worker</title>
<script type="text/javascript">
    // Set up global variable
    var worker;

    function startWorker(){
```

```
        // Initialize web worker
        worker = new Worker("../js/worker.js");

        // Run update function, when we get a message from worker
        worker.onmessage = update;

        // Tell worker to get started
        worker.postMessage("start");
    }

    function update(event){
        // Update the page with current message from worker
        document.getElementById("result").innerHTML = event.data;
    }

    function stopWorker(){
        // Stop the worker
        worker.terminate();
    }
</script>
</head>
<body>
    <h1>Web Worker Demo</h1>
    <button onclick="startWorker();" type="button">Start web worker</
button>
    <button type="button" onclick="stopWorker();">Stop web worker</button>
    <div id="result">
        <!--Received messages will be inserted here-->
    </div>
</body>
</html>
```

Tip:Use web worker for performing only heavy-weight JavaScript tasks that do

not interrupt the user-interface scripts (i.e. scripts that respond to clicks or other user interactions). It's not recommended to use web workers for short tasks.

HTML5 Server-Sent Events

The HTML5 server-sent events allows a web page to get information from the web server automatically.

What is Server-Sent Events

HTML5 server-sent events is a new way for the web pages to communicating with the web server. It is also possible with the XMLHttpRequest object that lets your JavaScript code make a request to the web server. But its a one-for-one exchange — that means once the web server provides its response, the communication is over.

However, there are some situations where web pages require a longer-term connection to the web server. A typical example is stock quotes on finance websites where price updated automatically. Another example is a news ticker running on various media websites.

You can create such effect with the HTML5 server-sent events feature. It allows a web page to hold an open connection to the web server so that the web server can send a new response automatically at any time, and there's no need to reconnect, and run the same server script from scratch over and over again.

Tip:The HTML5's server-sent events feature is supported in all major modern browsers like Firefox, Chrome, Safari and Opera except Internet Explorer.

Sending Messages with a Server Script

Let's create a PHP file named 'server_time.php' and type the following script into it. It will simply reports the current time of the web server's built-in clock in regular intervals. Later we will receive this time and update the web page accordingly.

```
<?php
header("Content-Type: text/event-stream");
```

```
header("Cache-Control: no-cache");

// Get the current time on server
$currentTime = date("h:i:s", time());

// Send it in a message
echo "data: " . $currentTime . "\n\n";
flush();
?>
```

The first two line of the PHP script sets two important headers. First, it sets the MIME type totext/event-stream, which is required by the server-side event standard. The second line tells the web server to turn off caching otherwise the output of your script may be cached.

Every message send through HTML5 server-sent events must start with the text "data:" followed by the actual message text and the new line character sequence ("\n\n").

Finally, we have used the PHP flush() function to make sure that the data is sent right away, rather than buffered until the PHP code is complete.

Processing Messages in a Web Page

The EventSource object is used to receive server-sent event messages. Let's create an HTML document named 'demo_sse.html' and place it in the same project directory where the 'server_time.php' file is located. This HTML document simply receives the current time reported by the web server and display it to the user.

```
<!DOCTYPE html>
<html lang="en">
<head>
<title>HTML5 Server-Sent Events</title>
<script type="text/javascript">
   window.onload = function(){
      var source = new EventSource("server_time.php");
      source.onmessage = function(event){
         document.getElementById("result").innerHTML += "New time received
from web server: " + event.data + "<br>";
      };
```

```
    };
</script>
</head>
<body>
    <div id="result">
        <!--Server response will be inserted here-->
    </div>
</body>
</html>
```

HTML5 Geolocation

The HTML5 geolocation feature is used to find out the location of the site visitor.

What is Geolocation

The HTML5 geolocation feature lets you find out the geographic coordinates (latitude and longitude numbers) of the current location of your website's visitor. This feature is helpful for providing better browsing experience to the site visitor. For example, you can return the search results that are physically close to the user's location.

Finding a Visitor's Coordinates

Getting the position information of the site visitor using the HTML5 geolocation API is fairly simple. It utilizes the three methods that are packed into the navigator.geolocation object —getCurrentPosition(), watchPosition() and clearWatch().

The following is a simple example of geolocation that displays your current position. But, first you need to agree to let the browser tell the web server about your position.

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<title>Example of HTML5 Geolocation</title>
```

```
<script type="text/javascript">
    function showPosition(){
        if(navigator.geolocation){
            navigator.geolocation.getCurrentPosition(function(position){
                var positionInfo = "Your current position is (" + "Latitude: " +
position.coords.latitude + ", " + "Longitude: " + position.coords.longitude + ")";
                document.getElementById("result").innerHTML = positionInfo;
            });
        } else{
            alert("Sorry, your browser does not support HTML5 geolocation.");
        }
    }
</script>
</head>
<body>
    <div id="result">
        <!--Position information will be inserted here-->
    </div>
    <button type="button" onclick="showPosition();">Show Position</button>
</body>
</html>
```

Note:The browser won't share the visitor location with a web page unless the visitor gives it explicit permission. The geolocation standard makes it an official rule to get user permission for every website that wants location data.

Dealing with Errors and Rejections

There may be a situation when a user does not want to share his location data with you. To deal with such sort of situation, you can supply two functions when you call thegetCurrentLocation(). The first function is called if your geolocation attempt is successful, while the second is called if your geolocation attempt ends in failure.

```
<!DOCTYPE html>
```

```
<html lang="en">
<head>
<meta charset="UTF-8">
<title>Example of Handling Geolocation Errors and Rejections</title>
<script type="text/javascript">
    // Set up global variable
    var result;

    function showPosition(){
        // Store the element where the page displays the result
        result = document.getElementById("result");

        // If geolocation is available, try to get the visitor's position
        if(navigator.geolocation){
            navigator.geolocation.getCurrentPosition(successCallback,
errorCallback);
            result.innerHTML = "Getting the position information...";
        } else{
            alert("Sorry, your browser does not support HTML5 geolocation.");
        }
    };

    // Define callback function for successful attempt
    function successCallback(position){
        result.innerHTML = "Your current position is (" + "Latitude: " +
position.coords.latitude + ", " + "Longitude: " + position.coords.longitude + ")";
    }

    // Define callback function for failed attempt
    function errorCallback(error){
        if(error.code == 1){
            result.innerHTML = "You've decided not to share your position, but it's
OK. We won't ask you again.";
```

```
        } else if(error.code == 2){
            result.innerHTML = "The network is down or the positioning service
can't be reached.";
        } else if(error.code == 3){
            result.innerHTML = "The attempt timed out before it could get the
location data.";
        } else{
            result.innerHTML = "Geolocation failed due to unknown error.";
        }
    }
</script>
</head>
<body>
    <div id="result">
        <!--Position information will be inserted here-->
    </div>
    <button type="button" onclick="showPosition();">Show Position</button>
</body>
</html>
```

Showing Location on Google Map

You can do very interesting things with geolocation data, like showing the user location on Google map. The following example will show your current location on Google map based the latitude and longitude data retrieved through the HTML5 geolocation feature.

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<title>Example of Showing Geolocation on Google Map</title>
<script type="text/javascript">
    function showPosition(){
```

```
        navigator.geolocation.getCurrentPosition(showMap);
    }

    function showMap(position){
        // Get location data
        var latlong = position.coords.latitude + "," + position.coords.longitude;

        // Set Google map source url
        var mapLink = "http://maps.googleapis.com/maps/api/staticmap?
center="+latlong+"&zoom=16&size=400x300&output=embed";

        // Create and insert Google map
        document.getElementById("embedMap").innerHTML = "<img alt='Map
Holder' src='"+ mapLink +"'>";
    }
</script>
</head>
<body>
    <button type="button" onclick="showPosition();">Show My Position on
Google Map</button>
    <div id="embedMap">
        <!--Google map will be embedded here-->
    </div>
</body>
</html>
```

The above example will simply show the location on the Google map using a static image. However, you can also create interactive Google maps with dragging, zoom in/out and other features that you have come across in your real life.

```
<!DOCTYPE html>
<html lang="en">
```

```
<head>
<meta charset="UTF-8">
<title>Example of Showing Location on Google Map</title>
<script src="http://maps.google.com/maps/api/js?sensor=false"></script>
<script type="text/javascript">
function showPosition(){
    if(navigator.geolocation){
        navigator.geolocation.getCurrentPosition(showMap, showError);
    } else{
        alert("Sorry, your browser does not support HTML5 geolocation.");
    }
}

// Define callback function for successful attempt
function showMap(position){
    // Get location data
    lat = position.coords.latitude;
    long = position.coords.longitude;
    var latlong = new google.maps.LatLng(lat, long);

    var myOptions = {
        center: latlong,
        zoom: 16,
        mapTypeControl: true,
        navigationControlOptions:
{style:google.maps.NavigationControlStyle.SMALL}
    }

    var map = new google.maps.Map(document.getElementById("embedMap"),
myOptions);
    var marker = new google.maps.Marker({position:latlong, map:map,
title:"You are here!"});
}
```

```
// Define callback function for failed attempt
function showError(error){
    if(error.code == 1){
        result.innerHTML = "You've decided not to share your position, but it's
OK. We won't ask you again.";
    } else if(error.code == 2){
        result.innerHTML = "The network is down or the positioning service can't
be reached.";
    } else if(error.code == 3){
        result.innerHTML = "The attempt timed out before it could get the
location data.";
    } else{
        result.innerHTML = "Geolocation failed due to unknown error.";
    }
}
</script>
</head>
<body>
    <button type="button" onclick="showPosition();">Show My Position on
Google Map</button>
    <div id="embedMap" style="width: 400px; height: 300px;">
        <!--Google map will be embedded here-->
    </div>
</body>
</html>
```

Check out the following URL to learn more about the Google Maps Javascript API:https://developers.google.com/maps/documentation/javascript/reference.

Monitoring the Visitor's Movement

All the examples we've used so far have relied on the getCurrentPosition() method. However, the geolocation object has another method watchPosition() that allow you to track the visitor's movement by returning the updated position as the location changes.

The watchPosition() has the same input parameters as getCurrentPosition(). However,watchPosition() may trigger the success function multiple times — when it gets the location for the first time, and again, whenever it detects a new position.

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<title>Example of Getting Current Position</title>
<script type="text/javascript">
    // Set global variable
    var watchID;
    function showPosition(){
        if(navigator.geolocation){
            watchID = navigator.geolocation.watchPosition(successCallback);
        } else{
            alert("Sorry, your browser does not support HTML5 geolocation.");
        }
    }
    function successCallback(position){
        toggleWatchBtn.innerHTML = "Stop Watching";

        // Check position has been changed or not before doing anything
        if(prevLat != position.coords.latitude || prevLong !=
position.coords.longitude){

            // Set previous location
            var prevLat = position.coords.latitude;
            var prevLong = position.coords.longitude;

            // Get current position
            var positionInfo = "Your current position is (" + "Latitude: " +
```

```
position.coords.latitude + ", " + "Longitude: " + position.coords.longitude + ")";
            document.getElementById("result").innerHTML = positionInfo;

        }

    }
    function startWatch(){
        var result = document.getElementById("result");

        var toggleWatchBtn = document.getElementById("toggleWatchBtn");

        toggleWatchBtn.onclick = function(){
            if(watchID){
                toggleWatchBtn.innerHTML = "Start Watching";
                navigator.geolocation.clearWatch(watchID);
                watchID = false;
            }
            else{
                toggleWatchBtn.innerHTML = "Aquiring Geo Location...";
                showPosition();
            }
        }
    }

    // Initialise the whole system (above)
    window.onload = startWatch;
</script>
</head>
<body>
    <button type="button" id="toggleWatchBtn">Start Watching</button>
    <div id="result">
        <!--Position information will be inserted here-->
    </div>
```

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

HTML5 Drag and Drop

The HTML5 has native support for the drag and drop feature.

Drag and Drop an Element

The HTML5 drag and drop feature allows the user to drag and drop an element to another location. The drop location may be a different application. While dragging an element a translucent representation of the element is follow the mouse pointer.

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<title>Example of HTML5 Drag and Drop</title>
<script type="text/javascript">
    function dragStart(e){
        // Sets the operation allowed for a drag source
        e.dataTransfer.effectAllowed = "move";

        // Sets the value and type of the dragged data
        e.dataTransfer.setData("Text", e.target.getAttribute("id"));
    }
    function dragOver(e){
        // Prevent the browser default handling of the data
        e.preventDefault();
        e.stopPropagation();
    }
    function drop(e){
        // Cancel this event for everyone else
        e.stopPropagation();
```

```
        e.preventDefault();

        // Retrieve the dragged data by type
        var data = e.dataTransfer.getData("Text");

        // Append image to the drop box
        e.target.appendChild(document.getElementById(data));
    }
</script>
<style type="text/css">
    #dropBox{
        width: 200px;
        height: 200px;
        border: 5px dashed gray;
        background: lightyellow;
        text-align: center;
        margin: 20px 0;
        color: orange;
    }
    #dropBox img{
        margin: 10px;
    }
</style>
</head>
<body>
    <h2>Drag & Drop Demo</h2>
    <p>Drag and drop the image into the drop box:</p>
    <div id="dropBox" ondragover="dragOver(event);" ondrop="drop(event);">
        <!--Dropped image will be inserted here-->
    </div>
    <img src="../images/kites.jpg" id="dragA" draggable="true"
ondragstart="dragStart(event);" width="180" height="180" alt="Flying Kites">
</body>
```

</html>

Tip:You can make an element draggable by setting the draggable attribute to true, like draggable="true". However, in most browsers, text selections, images, and anchor elements with an href attribute are draggable by default.

Drag and Drop Events

A number of events are fired during the various stages of the drag and drop operation. But mouse events such as mousemove are not fired during a drag operation. The following table provides you a brief overview of all the drag and drop events.

Event	Description
ondragstart	Fires when the user starts dragging an element
ondragenter	Fires when a draggable element is first moved into a drop listener

ondragover	Fires when the user drags an element over a drop listener
ondragleave	Fires when the user drags an element out of drop listener
ondrag	Fires when the user drags an element anywhere; fires constantly but can give X and Y coordinates of the mouse cursor
ondrop	Fires when the user drops an element into a drop listener successfully
ondragend	Fires when the drag action is complete, whether it was successful or not. This event is not fired when dragging a file to the browser from the desktop.

Note:The HTML5's drag and drop feature is supported in all major modern browsers like Firefox, Chrome, Opera, Safari and Internet Explorer 9+.