**HTML5 DOC**

HTML5 is also cross-platform (it does not care whether you are using a tablet or a smartphone, a netbook, notebook or a Smart TV).

New Features in HTML5:

* The <canvas> element for 2D drawing
* The <video> and <audio> elements for media playback
* Support for local storage
* New content-specific elements, like <article>, <footer>, <header>, <nav>, <section>
* New form controls, like calendar, date, time, email, url, search

New Elements:

1. Canvas : To draw 2D graphics
2. Audio , video:
3. article, header, footer, section, nav (Sematic Elements)

**Canvas:**

The <canvas> element is only a container for graphics. You must use a script to actually draw the graphics.

*e.g. var c=document.getElementById("myCanvas");*

*var ctx=c.getContext("2d");*

*ctx.fillStyle="#FF0000";*

*ctx.fillRect(0,0,200,100); //* x,y,width,height

Draw Line:

*e.g. ctx.moveTo(0,0); // Start Point*

*ctx.lineTo(200,100); // End Point*

*ctx.stroke();*

Draw Circle:

*e.g. ctx.beginPath();  
 ctx.arc(95,50,40,0,2\*Math.PI); // x,y,r,start,stop  
 ctx.stroke();*

Text:

*e.g. ctx.font="20px Arial";*

*ctx.fillText("Hello World",10,20); // fillText(text,x,y)*

*ctx.strokeText("Hello World",10,50);*

Linear Gradients:

*e.g. var grd=ctx.createLinearGradient(0,0,200,0); // Create gradient*

*grd.addColorStop(0,"red");*

*grd.addColorStop(1,"white");*

*ctx.fillStyle=grd; // Fill with gradient*

*ctx.fillRect(10,10,150,80);*

Radial Gradients

*var grd=ctx.createRadialGradient(75,50,5,90,60,100); // Create gradient*

*grd.addColorStop(0,"red");*

*grd.addColorStop(1,"white");*

*ctx.fillStyle=grd; // Fill with gradient*

*ctx.fillRect(10,10,150,80);*

**SVG [Scalable Vector Graphics]**

* SVG is used to define vector-based graphics for the Web
* SVG defines the graphics in XML format
* SVG graphics do NOT lose any quality if they are zoomed or resized
* Every element and every attribute in SVG files can be animated
* SVG is a W3C recommendation

*<svg width="300" height="200">  
  <polygon points="100,10 40,180 190,60 10,60 160,180"  
  style="fill:lime;stroke:purple;stroke-width:5;fill-rule:evenodd;" />  
</svg>*

Difference between canvas and SVG:

|  |  |
| --- | --- |
| **Canvas** | **SVG** |
| * Canvas draws 2D graphics, on the fly (with a JavaScript). * Resolution dependent * No support for event handlers * Poor text rendering capabilities * You can save the resulting image as .png or .jpg * Well suited for graphic-intensive games | * SVG is XML based * Resolution independent * Support for event handlers * Best suited for applications with large rendering areas (Google Maps) * Slow rendering if complex (anything that uses the DOM a lot will be slow) * Not suited for game applications |

**HTML5 Video**

## e.g*. <video width="320" height="240" controls> <source src="movie.mp4" type="video/mp4">   <source src="movie.ogg" type="video/ogg"> Your browser does not support the video tag. </video>*

## The controls attribute adds video controls like play, pause and volume.

|  |  |  |  |
| --- | --- | --- | --- |
| **Browser** | **MP4** | **WebM** | **Ogg** |
| IE | YES | NO | NO |
| Chrome | YES | YES | YES |
| Firefox | NO Update: Firefox 21 running on Windows 7, Windows 8, Windows Vista, and Android now supports MP4 | YES | YES |
| Safari | YES | NO | NO |
| Opera | NO | YES | YES |

* MP4 = MPEG 4 files with H264 video codec and AAC audio codec
* WebM = WebM files with VP8 video codec and Vorbis audio codec
* Ogg = Ogg files with Theora video codec and Vorbis audio codec

**HTML5 Audio**

## e.g. *<audio controls>   <source src="horse.ogg" type="audio/ogg">   <source src="horse.mp3" type="audio/mpeg"> Your browser does not support the audio element. </audio>*

|  |  |  |  |
| --- | --- | --- | --- |
| **Browser** | **MP3** | **Wav** | **Ogg** |
| IE | YES | NO | NO |
| Chrome | YES | YES | YES |
| Firefox | NO **Update:** Firefox 21 running on Windows 7, Windows 8, Windows Vista, and Android now supports MP3 | YES | YES |
| Safari | YES | YES | NO |
| Opera | NO | YES | YES |

**HTML5 New Input Types**

* color : It will show color-picker e.g. <input type=”color” name=”clrName”/> [Work on Chrome and Opera]
* date : Show date picker [Ch, Sf, Op]
* datetime: [Sf, Op]
* datetime-local: [Ch, Sf, Op]
* email: [>Ie9, FF, Ch, Op]
* month: It shows Month and Year only [Ch, Sf, Op]
* number: e.g. <input type=”number” min=”1” max=”5” /> [>Ie9,Ch,Sf,Op]
* range: [> Ie9 and remaining]
* search: [Ch and Sf]
* tel: [No Support]
* time: [Ch, Sf, Op]
* url : [> Ie9, FF, Ch, Op]
* week: [Ch, Sf, Op]

**Content Editable:**

## e.g. *<ul contenteditable="false">*

## *<li> Break mechanical cab driver. </li>*

## *<li> Drive to abandoned factory*

## *<li> Watch video of self </li>*

## *</ul>*

**HTML5 New Form Elements**

HTML5 has the following new form elements:

<datalist>, <keygen> and <output>

The <**datalist**> element specifies a list of pre-defined options for an <input> element.

It provides **autocomplete** feature on input elements.

*Not supported in IE9*

*e.g. <input list="****browsers****">  
<datalist id="****browsers****">*

*<option value="Internet Explorer">*

*<option value="Firefox">*

*<option value="Chrome">*

*<option value="Opera">*

*<option value="Safari">  
</datalist>*

The <**keygen**> tag specifies a key-pair generator field in a form.

When the form is submitted, two keys are generated, one private and one public.

The <**output**> element represents the result of a calculation

**HTML5 New Form Attributes**

New attributes for <form>:

* autocomplete
* novalidate

New attributes for <input>:

* autocomplete
* autofocus : First name:<input type="text" name="fname" autofocus>
* form : We can use input as part of form even it is written outside the form
* formaction: Override the action attribute of form
* formenctype:
* formmethod
* formnovalidate
* formtarget
* height and width: only used with <input type="image">
* list: refers to a <datalist> element that contains pre-defined options
* min and max
* multiple: The multiple attribute works with email, and file input types.
* pattern (regexp)
* placeholder :
* required
* step

**What are Semantic Elements?**

A semantic element clearly describes its meaning to both the browser and the developer.

E.g. **non-semantic** elements: <div> and <span> - Tells nothing about its content.

E.g**. semantic** elements: <form>, <table>, and <img> - Clearly defines its content.

**New Semantic Elements in HTML5**

* <**header**>
* <**nav**> : Defines set of navigation links
* <**section**>: It is grouping of content typically with heading
* <**article**>: Used for blog, comments, news
* <**aside**>: It’s content should be related to the surrounding content like sidebar
* <**figcaption**>: Provides caption for figure element
* <**figure**>: Used for illustrations, diagrams, photos, code listings
* <**footer**>:

**Problem With Internet Explorer 8 And Earlier**

For IE8 and earlier we cannot apply style to new HTML5 elements.

Workaround : html5shiv.js file.

To enable the HTML5 Shiv (after downloading), insert the following code into the <head> element:

<!--[if lt IE 9]>  
<script src="html5shiv.js"></script>  
<![endif]-->

That code above is a comment that only IE reads, for versions earlier than IE9. It must be placed in the <head> element because Internet Explorer needs to know about the elements before it renders them.

**HTML5 Web-storage**

Earlier, With HTML5, web pages can store data locally within the user's browser. This was done by cookies. But web storage is more secure and faster.

It is also possible to store large amounts of data, without affecting the website's performance.

*Note: Internet Explorer 7 and earlier versions, do not support web storage.*

There are two new objects for storing data on the client:

* localStorage - stores data with no expiration date
* sessionStorage - stores data for one session

**Check browser support**: *if(typeof(Storage)!=="undefined")*

*Note: Key/value pairs are always stored as strings*

***localStorage****.clickcount=Number(localStorage.clickcount)+1;*

***sessionStorage****.clickcount=Number(sessionStorage.clickcount)+1;*

To store object in web storage:

*localStorage.doctors = JSON.stringify(doctors);*

And then retrieve

*var data = JSON.parse(localStorage.doctors)*

The content in localstorage is persistent as long as the user chooses to clear the storage

**What is Application Cache?**

It means that a web application is cached, and accessible without an internet connection.

Application cache gives an application three advantages:

* **Offline browsing** - users can use the application when they're offline
* **Speed** - cached resources load faster
* **Reduced server load** - the browser will only download updated/changed resources from the server

***Note****: Internet Explorer 10, Firefox, Chrome, Safari and Opera support Application cache.*

*A manifest file needs to be served with the correct MIME-type, which is "text/cache-manifest". Must be configured on the web server.*

**To enable application cache** *<html manifest="demo.appcache">*

**The manifest file:** File extension should be ‘ .appcache’.

It has three sections:

CACHE MANIFEST  
# 2012-02-21 v1.0.0  
/theme.css  
/logo.gif  
/main.js  
  
NETWORK:  
login.asp  
  
FALLBACK:  
/html/ /offline.html

1. **CACHE MANIFEST** - Files listed under this header will be cached after they are downloaded for the first time. The resource is available even no internet.
2. **NETWORK** - Files listed under this header require a connection to the server, and will never be cached
3. **FALLBACK** - Files listed under this header specifies fallback pages if a page is inaccessible

The first URI is the resource, the second is the fallback.

**Updating cache:** Application remains cached until one of the following happens

* The user clears the browser's cache
* The manifest file is modified
* The application cache is programmatically updated

**SSE (Server-Sent Event)**

A server-sent event is when a web page automatically gets updates from a server.

*var source=new EventSource("demo\_sse.php");  
source.onmessage=function(event)  
  {  
  document.getElementById("result").innerHTML+=event.data + "<br>";  
  };*

**Performance Best Practice:**

For Page load optimization:

1. Optimizing Cache:
2. Combine External JS

Partition the JavaScript into 2 files: one JS containing the minimal code needed to render the page at startup; and one JS file containing the code that isn't needed until the page load has completed.

Serve as few JavaScript files from the document <head> as possible, and keep the size of those files to a minimum.

Serve JavaScript of a rarely visited component in its own file. Serve the file only when that component is requested by a user.

1. Combine External CSS
2. Combine CSS images using CSS sprites

* Put external scripts after external stylesheets if possible.
* Put inline scripts after other resources if possible.

1. Avoid document.write :

Don’t use it to fetch external javascript as below

*<script>document.write(‘src/myjs.js’);</script>*

Instead we should use

*<script src=”src/myjs.js”></script>* in head tag.

1. Avoid CSS @import:

CSS **@import** allows stylesheets to import other stylesheets.

Instead of @import use *<link rel=”stylesheet” href=”css/myStyle.css”>*

1. Optimize Browser rendering:

* Use efficient CSS selectors :
  1. Avoid universal selectors,
  2. Use class or id selectors instead of element selectors
* Put CSS in document head
* Specify Image Dimensions

**HTML5 API:**

1. **Media** API [Video and Audio]: play(), pause(), load(), canPlayType()
2. **Text Track** API [Subtitle and traction for Audio and Video]: language, readyState, mode and label
3. **Drag and Drop**: Add draggable attribute set to true
4. **Canvas 2D context**:
5. **Fullscreen** API
6. **Page Visibility** API
7. **getUserMedia** API : This API provides access to device media. Using this API, the <video> tag, and canvas, you can take beautiful photos within your browser!
8. **Battery** API: The Battery API is obviously a mobile-targeted API providing insight into the device's battery level and status:

**Modenizr:**

In the past, the client-side application would use **navigator.userAgent** to determine which browser was in use and choose its code paths accordingly. Today, the accepted practice is to explicitly detect each feature the application intends to use.

[Modernizr](http://www.modernizr.com/) is an open-source JavaScript library that detects the support for browser features (geolocation, canvas, SVG, border-radius, etc.) and exposes its findings in CSS and JavaScript.

Once a script reference to Modernizr is included, Modernizr will add a CSS class to the html element for each feature it can detect. If the feature isn't supported, the CSS class will start with no-. For example, if a browser supports canvas and not webgl, its html element will look like this: <html class="canvas no-webgl ...">. Modernizr also exposes a Modernizr JavaScript object that has Boolean properties for each feature it can detect.

**Custom Build of Modenizr**

At the time of download we can specify which feature we required so that we can generate custom build of Modenizr.