Introduction to Web Applications

It is an application, which is used to publish the information through the web.

It is also called Distributed web application/ internet application

Here, the web application is hosted in the server (Web server) and the clients can access this application remotely through their client application (Web Browser).

Here, the client can make a request for a web site and the web server can respond back to the client by returning HTML content to the browser.

Example: Web Server & Web Browser

IIS, tomcat, apache (web server)

Internet Explorer, Chrome, Safari, Opera, Firefox (Web Browser)

Steps to deploy Web Application

- First create a web site (ASP.NET, JSP, Servlets, HTML, and PHP))
- Host it in the web server (IIS, Tomcat, Apache)
- Access the website from your machine through the browser(IE Or Chrome or any)

The client will use web browser for sending request to access any web application through URL (Uniform Resource Locator)

What is URL?

Uniform Resource Locator

It is the address of the web resource.

Here, the request (HTTP) is sent through URL and response is given back in HTTP format.

Example:

URL: http://www.marlabs.com/about.html

Http: Hypertext Transfer Protocol

www.marlabs.com (is the domain name Or Service name)

about.html: is the resource, which the client is sending request.

Types of Web Applications:

1) Static web application

2) Dynamic web application

In case of static web application, only the presentation layer will exist.

[What you see is what you get]

Here, content is published on UI will not modified during runtime.

It does not provide any user interactivity.

Note: -

For developing static web application, we don't need any development Environment.

We don’t need any programming language.

We can design by using HTML and host in the web server.

So that clients can access it by sending request through URL.

Note:

For static web application, we need the following tools:

1) Notepad or any text editor

2) Web server (for hosting the web page)

3) Web browser (for accessing web page from the web server)

Dynamic web application

It enables us to publish static and dynamic content on the UI.

Here, the content can be added to the page at runtime.

Here, the dynamic content is generated through certain script.

It provides user interactivity.

This script can be written on either on client side Or Server side.

Example for dynamic content

Display the current Date and Time.

Change back color of the page on button click.

Give some input and get output during runtime.

Perform form validations.

Animations

Dynamic Styles

Menu

PopUp windows

Calendar

And etc…

How to perform all these dynamic tasks?

For this purpose, we can use some script (client side or server side script)

Note:

We use Script for adding dynamic content to the web page.

The script can be either client script or server script.

What is script?

It is a kind of business logic (functionality)

What is client side script & Server side script?

That client script is executed on client machine (browser)

The server script is executed on server application (web server)

The client script is executed before we submit the form to the server.

The server script is executed after we submit the form.

In case of client side script, there will be no post back (similar to standalone).

In case of server side script, there will be a post back.

What is post back?

Sending data from client to server and getting dynamic response from server to client.

Note : - (dynamic web application)

Any Dynamic web application consists of the following sections

1) Presentation Layer (UI) (HTML & CSS)

2) Client Script (JavaScript)

3) Server Side Script (any Server side programming language: C#, VB.NET, F# and etc...)

HTML is for designing (display the content) the web page and CSS is for applying the styles to the html tags (color, font, positions and etc...)

HTML (hypertext markup language): we use for displaying static content on the web page.

CSS (Cascading style sheet): we use for applying the styles (look & feel) to the web page (backgroundcolor, color, font, margin, padding and etc...)

Note:

You can develop dynamic web application by using any web technology (asp.net, jsp, php or any server side web technology), but we use html and css for UI (presentation layer)

Note: CSS & HTML are static.

As soon as you request for any web page, HTML & CSS will be loaded.

For Client Side Script:

JavaScript (Netscape) (works in all browsers)

As per the request, the browser will run the script.

Note: -

Even for dynamic web application with presentation and client side script, we don't need to any runtime engine. The browser will take's care of the Parsing.
The browser includes inbuilt JavaScript Parser (Interpreter).

HTML, CSS & JavaScript

1) Notepad or any text editor (dream weaver, front page, Visual Studio)

2) Web server (for hosting the web page)

3) Web browser (for accessing the web page)

If we write business logic on server side, that script is called Server Side Script.

C#

VB.NET

Java

VC++

F#

...

These are server side programming languages

Note:-

To write server side business logic to any web application, we need server side web technology.

(Web Architectures OR Server side web technologies)

ASP (DNA) (Distributed Internet application Architecture)

Asp.Net (.NET framework)

JSP & Servlets (Java Technology)

PHP

CGI

These all are server side web technologies.

We can use all these technologies for dynamic web application with Server Side Script.

For all these technologies, we need a runtime (compiler).

Web application development in ASP.NET:

Presentation: HTML & CSS

Client Side Script: JavaScript

Server Side Script: Any .NET Compatible language(C#.net, Vb.Net, VC++.net and etc...)

Server Side Web Technology: ASP.NET (framework)

Web Server: IIS

Web Browser: Internet Explorer or any...

IDE: Visual Studio 2012/Higher

Runtime: CLR (common language runtime)

Database: SQL Server 2008/Higher

Database connectivity: ADO.NET or LINQ to SQL or Entity Framework.

LINQ to SQL & Entity Framework:

These two are new database connectivity models.

These two are ORM Technologies (Object Relationship models)

LINQ: Language Integrated Query

IDE: Integrated Development Environment

It allows rapid application development

Web Application development in Java:

Presentation (UI): HTML & CSS

Client Side Script: JavaScript

Server Side Script: JSP& Servlets (Java syntaxes)

Web Server: Tomcat or Apache

Web Browser: IE, chrome, opera or firefox

IDE: Net beans, Eclipse, My Eclipse

Runtime: JVM

.NET Framework

FCL OR BCL: provides base classes for web Application

CLR: provides Runtime for server script

C#: provides syntaxes for new source code

ADO.NET: provides database Connectivity for Web application.

SQL Server: provides backend for web application.

Web Server (IIS): host for web application.

Web Browser (Internet Explorer): provides environment for request & response.

Asp.Net: provides web syntaxes (Server Side Web Technology)

HTML

Hypertext Markup Language

"Hypertext" is a term created by visionary Ted Nelson to describe non-linear writing in which you follow associative paths through a world of textual documents. The most common use of hypertext these days is found in the links on World Wide Web pages.

It is a markup language given by w3c (www consortium)

It is a client side web technology.

Its extension is .html

HTML includes inbuilt tags to display the content on the UI.

HTML tags are surrounded by the two characters < and >

The surrounding characters are called angle brackets

HTML tags normally come in pairs like and

The first tag in a pair is the start tag (opening tag), the second tag is the end tag (closing tag)

The text between the start and end tags is the element content

HTML tags are not case sensitive; means the same as <B>

HTML tags are not tag sensitive, <b> content. [No end tag, it works]

It allows us to present plain text as well as graphical text (images, audio, video, links, 2D and 3D text)

The current Version of HTML is: HTML 5

HTML5 is the next major revision of the HTML standard superseding HTML 4.01, XHTML 1.0, and XHTML 1.1.

HTML5 is a standard for structuring and presenting content on the World Wide Web.

The new standard incorporates features like video playback and drag-and-drop that have been previously dependent on third-party browser plug-ins such as Adobe Flash, Microsoft Silver light, and Google Gears.

HTML is the new standard, which includes:

* New HTML elements and attributes
* Full CSS3 support
* Video and Audio elements
* Local Storage
* And More

Why is HTML5 important?

You can create animation and playing music without plug-in, rich input control such as date picker, color picker, slider without JavaScript, and lastly offline data storage.

Browser Support:

The latest versions of Apple Safari, Google Chrome, Mozilla Firefox, and Opera all support many HTML5 features and Internet Explorer 9.0 will also have support for some HTML5 functionality.

The mobile web browsers that come pre-installed on iPhones, iPads, and Android phones all have excellent support for HTML5.

New Features:

HTML5 introduces a number of new elements and attributes that helps in building a modern websites.

Following are great features introduced in HTML5:

New Semantic Elements: These are like <header>, <footer>, and <section>, <article>, <nav>.

Web Forms 2.0: Improvements to HTML web forms where new attributes have been introduced for <input> tag.

Persistent Local Storage: To achieve without resorting to third-party plugins.

Web Socket: A next-generation bidirectional communication technology for web applications.

Server-Sent Events: HTML5 introduces events which flow from web server to the web browsers and they are called Server-Sent Events (SSE).

Canvas: This supports a two-dimensional drawing surface that you can program with JavaScript.

Audio & Video: You can embed audio or video on your web pages without resorting to third-party plug-in.

Geolocation: Now visitors can choose to share their physical location with your web application.

Microdata: This lets you create your own vocabularies beyond HTML5 and extend your web pages with custom semantics.

Drag and drop: Drag and drop the items from one location to another location on a the same webpage.

Backward Compatibility

HTML5 is designed, as much as possible, to be backward compatible with existing web browsers. New features build on existing features and allow you to provide fallback content for older browsers.

It is suggested to detect support for individual HTML5 features using a few lines of JavaScript.

HTML5 Syntax:

DOCTYPEs in older versions of HTML were longer because the HTML language was SGML based and therefore required a reference to a DTD.

HTML 5 authors would use simple syntax to specify DOCTYPE as follows:

<!DOCTYPE html>

All the above syntax is case-insensitive.

Character Encoding:

HTML 5 authors can use simple syntax to specify Character Encoding as follows:

<meta charset="UTF-8">
All the above syntax is case-insensitive.

The <script> tag:

It's common practice to add a type attribute with a value of "text/javascript" to script elements as follows:

<script type="text/javascript" src="scriptfile.js"></script>
HTML 5 removes extra information required and you can use simply following syntax:

<script src="scriptfile.js"></script>

The <link> tag:

So far you were writing <link> as follows:

<link rel="stylesheet" type="text/css" href="stylefile.css">
HTML 5 removes extra information required and you can use simply following syntax:

<link rel="stylesheet" href="stylefile.css">

Example:

```
<!doctype html>
<html>
        <head>
            <title>demo page</title>
        </head>
        <body>
            <h1>Welcome Marlabs Software
            <h1> Server Side</h1>
            <b>HTML Demo</b>
        </body>
</html>
```

Note:

To create HTML Pages, open notepad or any html editor, write the html tags, save it in a specific folder with .html extension.

Once, .html page is created, host in a web server to publish it in the network.

Once, .html page is hosted in web server, we can access it through the network through the browser by sending the HTTP request (URL: Virtual path)

Note:

If we don't host the web page into the web server, we can't access it through the network.

In this case, we can access only in the local machine using file system.

For this purpose, we follow the below steps

Open the Browser and type the file system URL

file:///D:/demos/samplepage.html

[This is the physical path]

This is known to the current System.

We can't access in another system in same network.

Now, in order to publish this web page in a local network, the machine need to be installed a web server (IIS) and then the web page should be hosted into that web server.

Once the web page is hosted in the web server, all other people can access that web page through their browser by sending request [URL (virtual path)]

URL:

protocall://hostname/foldername/filename.html

URL: Uniform resource locator.

It shows the address of the web site in the web server.

It consists of protocol (channel), host name and folder and the page name.

syntax : -

http://localhost/foldername/pagename.html

http://www.marlabs.com/talentTransformation/feedback.html

Note:

You can host you web application in any web server.

For demonstration, I will show hosting in IIS.

Before hosting web page in the web server, the web server should be installed.

IIS: Internet Information services

To check whether IIS is installed or not:

Start =>run => inetmgr => check the status running or not (Default Web site)

Its default port no: 80

Hosting a web page to the web server (Web sharing) (IIS)

We have various ways to host web page into the IIS

First Method

Steps:

Copy all the documents (web documents) and paste in the physical directory of web server (IIS).

The default physical directory of IIS is:

C: \inetpub\wwwroot\

Note: -

Once the web page is hosted in web server, the end users can access those pages from their browsers by sending the virtual path (URL).

Syntax : -

http://localhost/pagename.html

http://localhost/SampleHTML.html

http://localhost/webdemos/demo.html

Note: You can replace ”localhost” with IP Address to access in remote machine.

In this case, every time we need to copy the web documents into the IIS Physical directory.

Instead, we can create a virtual directory in the IIS corresponding to physical directory.

What is virtual directory?

It is an alias name created for your physical directory in the web server (IIS).

So, in this case whatever we do in physical directory, all the changes are effected in the virtual directory.

The end users can automatically get the updated content.

How to create virtual directory

Web Sharing

Creating virtual directory

In this case, we need not to copy the web documents in the IIS physical directory.

We can create a virtual directory in IIS corresponding to physical directory of the local system.

Virtual Directory

Open IIS

Start =>run =>inetmgr

Local computer

Web sites

Default web site

Right click on default web site

=>Add Virtual Directory

=>Type alias name (Virtual Directory name)

=>Choose physical directory

=>Ok

Open the browser

Type the URL in the Address Bar.

http://localhost/VirtualDirectory/file.html

http://localhost/myweb/demo.html

Html syntaxes

Structure of HTML document:

```
<!doctype html >
<html>
    <head>
        <title>page title</title>
    </head>
    <body>
        <!--content of the page-->
    </body>
</html>
```

<! DOCTYPE>

A <! DOCTYPE> declaration helps the browser to display a web page correctly.

There are many different documents on the web. A browser can only display a document correctly, if it knows what kind of document it is.

You find many documents on web: html, .gif, .jpg,.xml, .css, .js and etc...

In case of HTML, we have many versions and many types.

There are also many different versions of HTML, and a browser can only display an HTML page 100% correctly if it knows the exact HTML version used in the page. This is what <!DOCTYPE> is used for.

<!DOCTYPE> is not an HTML tag. It is information (a declaration) to the browser about what version the HTML is written in.

We have many versions of HTML

HTML 1.0

HTML 2.0

HTML 3.0

HTML 4.01 (These are based on SGML)

HTML 5 (current version and it is not based on SGML)

HTML 5 works only in the new browsers like

IE9 and above

Firefox 4.0 and above

Google Chrome 6 and above

Apple Safari 5 and above

Opera 10.6 and above

The doctype contains the rules of the html document. We create html documents as per the specified document type. If we don't follow the rule, the page will not display content properly.

In case of HTML 4.01, if we don't specify the declaration it automatically takes.

For html page design, we have many tools in the market.

Some of the tools are bellow:

Dreamweaver

Front Page

Visual Studio

Eclipse

My eclipse

Netbeans

<!doctype> example for previous HTML versions

HTML 4.01 Strict

This DTD contains all HTML elements and attributes, but does NOT INCLUDE presentational or deprecated elements (like font). Framesets are not allowed.

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">

HTML 4.01 Transitional

This DTD contains all HTML elements and attributes, INCLUDING presentational and deprecated elements (like font). Framesets are not allowed.

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">

HTML 4.01 Frameset

This DTD is equal to HTML 4.01 Transitional, but allows the use of frameset content.

<! DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Frameset//EN"
"http://www.w3.org/TR/html4/frameset.dtd">

In case of HTML 5, the declaration is as below

<!DOCTYPE HTML> (not case sensitive)

<html>...</html>

It is the root element in html

All other tags should be nested in this tag.

Only one <html> tag exists in html document.

<head>...</head>

It enables us to put header info of the page. (Page information)

This content is very useful for the Search engines. (SEO: Search Engine Optimization)

The <head> section is not for presenting. It is only for SEO.

It includes the following elements.

- Page title
- Page Meta data
- Script
- CSS code
- Link

Example:

```
<head>
        <title>sample html page</title>
        <meta charset="utf-8" >
    <meta name="description" content="Put here your meta description." >
    <meta name="keywords" content="Put here keyword sperated with comma."
    >
  <meta name="author" content="html5"/>
        <script type="text/JavaScript"></script>
        <style type="text/css"></style>
        <link href="stylesheet.css" rel="stylesheet" type="text/css">
</head>
```

<title>...</title>

This content is shown in the title bar of the browser window.

<meta ... />

This tag is used to add metadata to the web page.

This metadata is very use full for search engine optimization.

Always include <Meta /> tag to increase the web site rankings.

e.g.

<meta name="keywords" content="html,web,asp">

Here, we use these keywords in the search engine for searching this web page.

Example:

<meta name="description" content="static and dynamic web apps">

<meta name="keywords" content="jsp, asp, php">

<Meta name="Expires" content="15/08/2012">

<Meta http-equiv="Refresh" content="10">

[This will automatically refresh the page for every 10 seconds]

Note:

To increase the web site ranking, always put <Meta name="keywords" content="word1, word2,”> in your html page.

The <head> section is not for presenting on the UI.

It is used in the SEO.

Usage of <meta name="viewport" >

```
<meta name="viewport" content="width=device-width, initial-scale=1, maximum-scale=1">
```

Setting a viewport tells the browser how content should fit on the device's screen and informs the browser that the site is optimized for mobile

The above code tells the browser to set the viewport to the width of the device with an initial scale of 1. This example also allows zooming, something that may be desirable for a web site but not a web app. We could prevent zooming with user-scalable=no or cap the scaling to a certain level:

```
<meta name="viewport" content="width=device-width, initial-scale=1.0, minimum-scale=0.5 maximum-scale=1.0">
```

<body>...</body>

It is used to include the content, which will be published on the web page.

Only the content mentioned in the <body> section will be shown on the UI.

Example:

```
<body>
    <!--content goes here-->
</body>
```

Full structure of html document

```
<!doctype html>
<html>
    <head>
    </head>

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

Html document contains the following things:

- HTML Elements
- HTML Attributes
- HTML comments
- Entity

Note:

All the tags in HTML are predefined.

We can't create user defined tags in HTML.

HTML Elements

These are used to present the content on the web page as per the element.

presents the content in bold format

<h1>presents the content in header format</h1>

Note:

For every html element, specify opening and closing tag.

<h1>...</h1>

...

...

...

In between opening and closing tag, we specify the text.

HTML attributes

Html attributes describes the HTML Elements (Properties)

Attributes can provide additional Information about the HTML elements on your page.

Attributes always come in name/value pairs like this: name="value".

Attributes are always added to the start tag (opening tag) of an HTML element.

Basic tags in html

<html>....</html>

<head>...</head>

<title>...</title>

<script..> </script>

<meta ../>

<link .../>

<body>...</body>

Header tags

These tags are used to present Headers in the web page.

<h1> ...</h1>

<h2> ...</h2>

...

<h6> ...</h6>

Attributes of all header tags

ID

Style

CLASS

Tip: We can also apply styles using CSS (font-size, font-family, font-style and etc...)

NOTE:

ID is the common attribute for all the html tags.

It provides the uniqueness.

It is used in the JavaScript for manipulation of html elements.

<hr/>

It is used to present Horizontal Ruler with specified properties.

<hr width="100%"/>

<hr size="10" color="red"/> [size represents height of hr]

<hr style="height:10" color="green"/>

<hr color="blue" width="50%" align="center"/>

Here, size represents the height of the ruler.

: it breaks the line.

  in case of xhtml

Here, I am using self-closing (< />)

  in case of html

 and
 are same.

 is without close.

 with close tag (self close)

...

The tag defines a paragraph.

Browsers automatically add some space (margin) before and after each element.

It automatically applies line break.

The margins can be modified with CSS (with the margin properties).

 content

Working with Marquee:

<marquee LOOP="infinite" BGCOLOR="#fff" scrollamount="2" height="25" DIRECTION="left" onmouseover="this.stop()" onmouseout="this.start()">

Types of colors

Mono color: red, blue, green, cyan, pink, yellow...

RGB color :#ffffff [hexa decimal] [0,1,2,3... 9 ,A,B,C,D,E,F]

#b4b4b4

full white : #ffffff

full black : #000000

```
/* RGB model css syntax */
p { color: #F00 } /* #rgb */
p { color: #FF0000 } /* #rrggbb */
p { color: rgb(255, 0, 0) } /* integer range 0 - 255 */
p { color: rgb(100%, 0%, 0%) } /* float range 0.0% - 100.0% */
/* RGB with alpha channel, added to CSS3 */  [ CSS 3 works only in new browsers ]
p { color: rgba(255, 0, 0, 0.5) } /* 0.5 opacity, semi-transparent */
```

Working with entities in html

We use entity to display a special character on UI.

In html, the elements and entities are pre-defined.

Syntax:

&entityname;

Example:

For **<** characeter, we use: <

For **>** character, se use: >

HTML Special Text Cases (HTML Entities)

Result	Description	Entity Name	Entity Number
	non-breaking space		
<	less than	<	<
>	greater than	>	>
&	Ampersand	&	&
¢	Cent	¢	¢
£	Pound	£	£

¥	Yen	¥	¥
€	Euro	€	€
§	Section	§	§
©	Copyright	©	©
®	registered trademark	®	®
™	Trademark	™	™

HTML5 Document Structure:

The following tags have been introduced for better structure:

section: This tag represents a generic document or application section. It can be used together with h1-h6 to indicate the document structure.

article: This tag represents an independent piece of content of a document, such as a blog entry or newspaper article.

aside: This tag represents a piece of content that is only slightly related to the rest of the page.

header: This tag represents the header of a section.

footer: This tag represents a footer for a section and can contain information about the author, copyright information, etc...

nav: This tag represents a section of the document intended for navigation.

dialog: This tag can be used to mark up a conversation.

figure: This tag can be used to associate a caption together with some embedded content, such as a graphic or video.

The markup for an HTM 5 document would look like the following:

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <title>sample page</title>
</head>
<body>
 <header><h1>Welcome Marlabs</header>
 <nav><a href="http://www.marlabs.com">click here for marlabs</a></nav>
 <article>
  <section>
        <p>Marlabs Training Bethlem</p>
  </section>
 </article>
 <footer><b>&copy;All rights Reserved.&reg;Marlabs-2013</footer>
</body>
```

<IMG/> tag

The <img> tag is supported in all major browsers.

The <img> tag defines an image in an HTML page.

The <img> tag has two required attributes: src and alt.

src: It is used to specify the source of the image.(.jpg or .gif or .bmp)

alt: it is used to specify the alternate text.

If the image source is not found, the alternate text will be shown on the ui.

Note: Images are not technically inserted into an HTML page; images are linked to HTML pages. The <img> tag creates a holding space for the referenced image.

Tip: To link an image to another document, simply nest the <img> tag inside <a>... tags.

Syntax: -

Note: -

We have many types of images

.jpg

.png

.gif

.bmp

Optional attributes:

align: specifies the alignment of an image according to souring elements.

top ,bottom ,middle,left,right

border: specifies the width of the border around the image. (Pixels)

height: height of the image .

hspace: whitespace on left and right of the image .

ismap: (ismap) specifies an image as a server side image-map

usemap: (#mapname) specifies an image as a client side image-map

vspace : specifies the white space on top and bottom of an image.

width : (pixels) specifies the width of the image.

Events of the <IMG>

onabort,onclick,ondblclick,onmousedown,onmouseup,onmouseout,onmouseover,onkeypress,onkeyup,onkeydown

Example:

```
<img src="smiley.gif" alt="Smiley face" height="42" width="42" />
```

Note: We can make image as a link by nesting Image in the anchor tag.

Syntax: -

```
<a href="url"><img src="url" /> </a>
```

```
<a href="http://www.google.com"><img src="plain.jpg" alt="Sample Plain"
height="100px" width="300px" align="right"/></a>
```

Note:

In the above, we made only one click on image.

You click any where on the image; it will navigate to the specified URL.

Instead, we need to create multiple clickable areas. For this purpose, we use Image Map.

Image Map (<map> tag)

The <map> tag is used to define a client-side image-map. An image-map is an image with multiple clickable areas.

The name attribute of the <map> element is required and it is associated with the <img>'s usemap attribute and creates a relationship between the image and the map.

The <map> element contains a number of <area> elements that defines the clickable areas in the image map.

Example:

An image-map, with clickable areas:

```
<img src="planets.gif" width="145" height="126" alt="Planets" usemap="#planetmap"
/>
```

Note:

usemap attribute is used to associate an image with map (clickable areas)

```
<map name="planetmap">
```

```
 <area shape="rect" coords="0,0,82,126" href="sun.htm" alt="Sun" />
 <area shape="circle" coords="90,58,3" href="mercur.htm" alt="Mercury" />
 <area shape="circle" coords="124,58,8" href="venus.htm" alt="Venus" />
</map>
```

Map provides three shapes

1)rect

2)circle

3)polygon

In case of rect, we should specify, Top, Left and Right, Bottom coordinates.

In case of circle, we should specify, the center point and radius

In case of polygon, you can specify minimum 5 coordinates

Hyperlinks (navigation) <a>.. [anchor tag]

The <a> tag is supported in all major browsers.

The <a> tag defines an anchor. An anchor can be used in two ways:

To create a link to another document, by using the href attribute (External links)

To create a bookmark inside a document, by using the name attribute (Internal Link)

The <a> element is usually referred to as a link or a hyperlink.

The most important attribute of the <a> element is the href attribute, which indicates the link's destination.

Syntax: -

```
<a href="url">link text</a>
```

href : It indicates the destination of the link.

Hyperlink reference

By default, links will appear as follows in all browsers:

An unvisited link is underlined and blue (Link)

A visited link is underlined and purple (Vlink)

An active link is underlined and red (alink)

Syntax: - External link

 hyperlink text

 click for marlabs web site

Target:

It specifies where to open the requested page.

Target:

_blank: open new window or new tab

_self : in the same window

_top : same window, but at top

_parent: in the parent frame.

userdefinename: specify any name of the frame (will discuss in frames)

Internal links

We use internal links to create book marks.

This enables us to link to content in the same document (page).

It is used to move from one section to other section of the same page.

It is used to create index based content.

Syntax: -

... [for link]

... [for bookmark]

Example: For links

ADO.NET

ASP

C#

For creating books marks

ADO.NET

some content of ADO.NET

ASP

some content of J2EE

C#

some content of C#

Events of <a> tag

onblur,onclick,ondblclick,onfocus,onmousedown,onmouseup,onmouseover,onmouseout,onkeypress,onkeydown,onkeyup

Hyperlinks refer to:

1. Different HTML Pages

Link to Page 2

2. Different Places in the Same Page

Word¹

A word is a word.

3. An Email Address

Email Me

4. A File Server

```
<a href="ftp://ftp.iglu.org.il/">File Server</a>
```

<EMBED> tag

The <embed> tag is supported in all major browsers.

The <embed> tag defines a container for an external application or interactive content (a plug-in).

We use <embed> tag for connecting to flex application or silver light application.

Attributes of <embed> tag

Attributes

New: New in HTML5.

Attribute	Value	Description
height	Pixels	Specifies the height of the embedded content
src	URL	Specifies the address of the external file to embed
type	MIME_type	Specifies the MIME type of the embedded content
width	pixels	Specifies the width of the embedded content

Example:

```
<EMBED SRC="sound.swf" AUTOSTART="TRUE" LOOP="TRUE" WIDTH="100"
HEIGHT="100"></EMBED>
```

List tags

These are used to present the content in certain list format.

Types of lists

- Ordered list
- Unordered list
- definition list

Ordered list

This enables us to list the content in some order

1,2,3 ...

a,b,c,...

A,B,C,...

i,ii,iii,...

I,II,III,...

Syntax: -

```
<OL type="" start="">
        <li>content for list1  </li>
        <li>content for list2 </li>
        ..
        ..
</OL>
```

Type: it is used to specify the type of order list.

type="A" type="a" type="1" type="i" type="I"

default : 1

Note:

You can also nest the lists.

You can put lists inside another list.

The list item might be simple text or hyperlink or image or any thing.

Un-orderd list

It presents content in bulleted format.

syntax : -

```
<UL type="">
        <li>..... </li>
        <li>..... </li>
</UL>
```

type="circle"

type="disc"

type="square"

Note:

The default is disc.

We can use CSS3 to specify custom image as bulleted style.

Definition list

It enables us to list the content custom format.

Syntax: -

```
<DL>
        <DT>....</DT>
                <DD>...</DD>

        <DT>...</DT>
                <DD>...</DD>
        ...
```

</DL>

DL: definition list

DT: definition type

DD: definition data.

Note: -

The content of the list might be any thing.

e.g

<img> tag

<a>

May be again list. (Nested list)

Example:

```
<!doctype html>
<html>
    <head>
        <title>demo page</title>
    </head>
    <body>
    <h1>list demo</h1>
    <h3>types of lists</h3>

    <h3>Ordered List</h3>
    <ol type="1">
        <li>Ordered List</li>
        <li>Un-ordered list</li>
        <li>definition list</li>
    </ol>
```

```
        <h3>Un-ordered list:client side web tech</h3>
        <ul type="square">
                <li>html</li>
                <li>CSS</li>
                <li>Javascriipt</li>
                <li>ajax</li>
        </ul>

        <h3>.NET Technologoy </h3>
        <h3>definition list</h3>
        <dl>
                <dt>CLR
                        <dd>Common Language runtime
                <dt>CTS
                        <dd>Common Type system
                <dt>CLS
                        <dd>common language specification
        </dl>
        </body>
</html>
```

Working with <table> tag

The <table> tag is supported in all major browsers.

The <table> tag defines an HTML table.

An HTML table consists of the <table> element and one or more <tr>, <th>, and <td> elements.

<tr> : table row

<th> : table head (cell with bold text)

<td> : table data (cell with normal text)

The <tr> element defines a table row, the <th> element defines a table header, and the <td> element defines a table cell.

<tr> Table Row

<th> table head (bolder)

<td> table data

A more complex HTML table may also include <caption>, <col>, <colgroup>, <thead>, <tfoot>, and <tbody> elements.

Basic syntax:

```
<table >
    <tr>
        <th>head1</th>
        <th>head2</th>
    </tr>
    <tr>
        <td>data1 in row1</td>
        <td>data2 in row1</td>
    </tr>
</table>
```

Attributes of the <table> tag (optional)

align, bgcolor, border, cellpadding, cellspacing, summary, width, frame, rules

Frame attribute:

It specifies which parts of the outside borders that should be visible.

<table frame="box">...</table>

frame="box" ,"above" "below" , "hsides" , "vsides"

Syntax: -

```
<table>
    <caption>...</caption>
    <tr>
        <th>...</th>
        <th>...</th>
        <th>...</th>
        ..........
    </tr>
    <tr>
        <td>...</td.
        ..........
    </tr>
    ........
    ........
</table>
```

cellpadding:

Space between cell content and cell border

cellspacing:

Space between cells's

Using CSS, we can apply border styles (border size, border color, border style)

Note:

For each row or for each column, we can apply different style.

Attributes of <td>

colspan

It enables us to merge multiple columns into a single column

rowspan

It merges multiple rows.

align

left,center,right

valign

top,middle,bottom

For colspan

It is for merging the columns

```
<table border="4">
    <TR>
        <TD>ONE</TD>
        <TD>TWO</TD>
    </TR>
    <TR>
        <TD>THREE</TD>
        <TD>four</TD>
    </TR>
    <TR>
        <TD COLSPAN="2" ALIGN="CENTER">FIVE</TD>
    </TR>
</TABLE>
```

For rowspan

It is for merging the rows

```
<TABLE border="3">
    <TR>
        <TD ROWSPAN="3">ONE</TD>
        <TD>tWO</TD>
    </TR>
    <TR>
            <TD>THREE</TD>
    </TR>
    <tr>
        <TD >FOUR</TD>
    </tr>
</TABLE>
```

Rules attribute in <table> tag

Definition and Usage

The rules attribute specifies which parts of the inside border that should be visible.

For practical reasons, it may be better to not specify any rules, and use CSS instead.

Syntax

```
<table rules="value">
```

Rules="rows"/"cols"/"all" /"none"

Rules for making grid lines visible or not

Note: -

The content in a cell might be any text.

Simple text, hyperlink, image, list, again table and etc..

The content is included in the <td> tag only (in case of headers, <th>)

Rules: It is for in-side borders of the table.

Frame: It is for outside borders of the table.

The IFRAME element

The IFRAME element allows authors to insert a frame(Some HTML Page) within a block of text. Inserting an inline frame within a section of text is much like inserting an object via the OBJECT element: they both allow you to insert an HTML document in the middle of another, they may both be aligned with surrounding text, etc.

Syntax:-

```
<html>
    <body>
        <!--html content goes here-->
        <iframe>
            <!--content of the iframe goes here-->
        </iframe>
    </body>
</html>
```

Note:

We can take multiple Iframe sections in the html page.

Attribute definitions

src = uri

This attribute specifies a link to a long description of the frame. This description should supplement the short description provided using the title attribute, and is particularly useful for non-visual user agents.

name = cdata

This attribute assigns a name to the current frame. This name may be used as the target of subsequent links.

width = length

The width of the inline frame.

height = length

The height of the inline frame

Example:

```
<IFRAME src="foo.html" width="400" height="500"
        scrolling="auto" frameborder="1">
 [Your user agent does not support frames or is currently configured
 not to display frames. However, you may visit
 <A href="foo.html">the related document.</A>]
 </IFRAME>
```

Note:

IFrame is supported in all browsers

Example for : <figure> & <figcaption>

```
<!DOCTYPE html>

<html>

  <head>
     <title>Title name will go here</title>
  </head>

  <body>

    <p>The quick brown fox jumps over a right lazy dog.
      The quick brown fox jumps over a right lazy dog.
      The quick brown fox jumps over a right lazy dog.
      The quick brown fox jumps over a right lazy dog.
      The quick brown fox jumps over a right lazy dog.
      The quick brown fox jumps over a right lazy dog. </p>

    <figure>

       <img src="images/rose.jpg"/>
       <figcaption>This is rose flower.</figcaption>

    </figure>

  </body>

</html>
```

Example for : <embed>

EMBED Element is another media element, which is used to call a media file in browser. A browser pluging may be require to play the media files. **EMBED Element** is specially used for calling a flash movie.

```
<!DOCTYPE html>

<html>

  <head>
     <title>Title name will go here</title>
  </head>

  <body>

     <embed src="media/clock.swf"> </embed>

  </body>

</html>
```

Example for : <details> & <summary> tag

DETAILS Element is used to invoke the show and hide function of **HTML5**. Before the development of **HTML5** we were using the JavaScript for show and hide function. But now it is too easy to use this function with this **DETAILS and SUMMARY Tag of HTML5**.

```
<!DOCTYPE html>

<html>

  <head>
     <title>Title name will go here</title>
  </head>

  <body>

    <details>

      <summary>
      This is the default content.
      </summary>

      <p>This is the main content, which will be shown.</p>

    </details>

  </body>

</html>
```

Example: <audio> element

With the development of **HTML5**, **AUDIO Element** has been introduced for playing the **audio** file in the browser with full user control support.

Yet, before the development of **HTML5 AUDIO Element**, we were unable to play a **audio** file without using the third party browsers plug-in as flash player or quick time.

Now the **AUDIO Element** can be used instead of installing a browser plug-in.

The **AUIDO Element** contains an extra child element is called **SOURCE**, which is used to call the exact media file which we want to be played in the browser.

The **AUDIO Element** can also contains it's properties for controlling the AUDIO File.

```
<!DOCTYPE html>

<html>

  <head>
     <title>Title Name will go here</title>
  </head>

  <body>
    <audio controls>
       <source src="media/simple_audio.mp3" type="audio/mpeg"/>
       <source src="media/simple_audio.ogg" type="audio/ogg"/>
 <!-- This Line Will Be Called When You Are Running An Old Browser -->
       <p>Your file doesn't support the audio element</p>

    </audio>

  </body>

</html>
```

Example on : <video> element

The **VIDEO Element** is used to play a movie or video clip on the website. Before the development of **HTML5 VIDEO Element** we were in compulsion to use the object and embed element and it's lengthy scripts. Now with the help of **VIDEO Element** this **VIDEO** file can be called in easy steps.

```
<!DOCTYPE html>

<html>

  <head>
        <title>Title name will go here</title>
  </head>

  <body>

                <video width="500" height="375" controls="controls">

                        <source src="media/sample_video.mp4"
type="video/mp4"/>
                        <source src="your ogg media file name"
type="video/ogg"/>
                        Your browser does not support the video tag.
                </video>

  </body>
</html>
```

Example on: <video> DOM

Video Dom is such an enhanced feature, which make video playing more reliable and user friendly. With the help of Video Dom now you can customize the Video playing, as we can add additional buttons to play, stop, resize and many more.

```
<!DOCTYPE html>

<html>

  <head>
     <title>Title name will go here</title>
  </head>

  <body>

     <video width="500" height="375" id="video1">

        <source src="media/sample_video.mp4" type="video/mp4" />
        <source src="your ogg media file name" type="video/ogg" />
        Your browser does not support the video tag.

     </video>

     <button onclick="playPause()">Play/Pause</button>
     <button onclick="makeBig()">Big</button>
     <button onclick="makeSmall()">Small</button>
     <button onclick="makeNormal()">Normal</button>

     <script type="text/javascript">

        var myVideo=document.getElementById("video1");
        function playPause()
        {
           if (myVideo.paused)
           myVideo.play();
           else
           myVideo.pause();
        }


                function makeBig()
```

```
        {
           myVideo.width=530;
        }

        function makeSmall()
        {
           myVideo.width=320;
        }

        function makeNormal()
        {
           myVideo.width=420;
        }

     </script>

  </body>

</html>
```

Full Screen

```
var video = document.getElementById("video");

// Mozilla
video.mozRequestFullScreen();

// Webkit for video elements only
video.webkitEnterFullScreen();
```

Close Full Screen

```
        // Mozilla
video.mozCancelFullScreen();

// Webkit

video.webkitCancelFullScreen();
```

dotnetsatya@gmail.com

Elements you shouldn't be using:

In HTML5, The life of the following elements have come to an end. They have no role to play in HTML5 because CSS has grab it all.

- <basefont>
- <big>
-
- <font>
- <strike>
- <tt>
- <u>

- <frame>
- <frameset>
- <noframes>

Attributes you shouldn't be using:

Attributes	**Elements**
Align	caption, iframe, img, input, object, legend, table, hr, div, h1, h2, h3, h4, h5, h6, p, col, colgroup, tbody, td, tfoot, th, thead and tr.
alink, link, text and vlink	body
Background	body
Bgcolor	table, tr, td, th and body

Border	table and object
cellpadding and cellspacing	table
char and charoff	col, colgroup, tbody, td, tfoot, th, thead and tr
Clear	br
Compact	dl, menu, ol and ul
Frame	table
Frameborder	iframe
Height	td and th
hspace and vspace	img and object
marginheight and marginwidth	iframe
Noshade	hr
Nowrap	td and th
Rules	table
Scrolling	iframe
Size	hr

Type	li, ol and ul
Valign	col, colgroup, tbody, td, tfoot, th, thead and tr
Width	hr, table, td, th, col, colgroup and pre

Working with DHTML

Dynamic hypertext markup language

It enables us to provide the Interaction between user and the application.

The user can able to provide the input and web page can react based on the input during runtime.

It is a combination of HTML Form, CSS and JavaScript.

We use <form> tag for designing Html page for I/O elements.

We use (<script>) JavaScript to embed client side script.

Example:

user Login

user Register

Search for a specific content.

Get the current date time.

Popups

Form Validations

Animations

Menu

Datepicker

Tabs

Autofill

Etc...

To perform these actions, HTML includes <form /> tag.

To read and write the input fields from/to form, we write some script

(JavaScript)

<form> is used to design the web page with input fields (textbox, button , checkbox, radiobutton, dropdownlist and etc....)

<Script> is used to perform read/write operation dynamically on the <form> using JavaScript.

HTML Forms

An HTML Form allows a user to pass input through the web browser to the web server where it can be processed.

HTML Forms make it possible to build an Intranet/Internet Database Application, specifically they compose the User Interface of the application.

A form allows us to interact with the server.

The user enters the data on form and that is sent to server.

The server will process the user input.

A form is defined with the <form>...</form> tag.

A form must have both an opening tag and a closing tag.

Syntax: -

```
<form>
        <!-- form input types goes here-->
</form>
```

All form elements that belong to a form must be enclosed within the opening and closing tags.

You may have more than one form on a page.

You must NOT have a form within a form.

Syntax: -

```
<form>
    <input type="" name=""/>
</form>
```

Example:

```
<!doctype html>
<html>
    <head>
        <title>form demo</html>
    </head>
    <body>
        <form name="form1" action="demo.asp" method="get">
            <input type="submit" value="sumit"/>
        </form>
    </body>
</html>
```

In the opening form tag, you must define two attributes for the form to work properly, They are: **method** and **action**.

Syntax: -

```
<form action="" method="" name="" id="">
</form>
```

dotnetsatya@gmail.com

Note:

Method and **action** are required for <form> tag to work properly.

The **action** defines where to send the information for processing on submit button.

(we use action attribute to specify the dynamic web form for processing the input: .aspx, .jsp,.php or .asp)

The **method** defines how to send the information to the server. This can be either get or post.

We use **get** method to download data from server (request for data from server)

We use post method to send data to the server.

In case of get method, the form content is sent to server in the form of query string.

Query string is a combination of URL and form content.

The get method sends the form contents in the address like:

http://www.somewhere.com/myphp.php?name=value&name=value.

Example:

http://www.marlabs.com/login.aspx?username=satya&password=satya

This has several limitations:

- Everyone can see what you send in your form
- The form values may not contain non-ASCII characters
- The form values may not exceed 100 characters

The post method sends the form contents in the body of the request. (It uses cookies)

This means that you can send anything you want this way.

The only downside is that most browsers are unable to bookmark post requests.

In most cases we will use the post method although sometimes it is useful to send information to pages via the get method.

Here is an example of the source code of a form:

```
<html>
    <head>
        <title>Sample Form </title>
    </head>
    <body>
        <h1>Sample Form</h1>
        <P>Please enter a  name  </p>
            <form action="displayhello.aspx" method="POST" >
            Name
            <input type="text" name="username" size="10"
            maxlength="8" >
            <input type="submit" value="Click me!">
            </form>
    </body>
```

```
</html
```

Note:

In case of get method and to use query string, we must provide name attribute for each element in the form.

Form elements

Syntax: -

```
<input type="" name="" id=""/>
```

We have many types of elements in form:

For textbox: type="text"

For button: type="button"

For password: type="password"

For radio button: type="radio"

For checkbox: type="checkbox"

For submit button: type="submit"

For reset button: type="reset"

For file upload: type="file"

Button vs. submit

```
<input type="button" value="clickMe"/>
```

In case of simple button, by default no action.

The simple button is used for client side calls (for JavaScript)

```
<input type="submit" value="register"/>
```

In case of submit button, by default page info is sent to action page (server).

It is for server side call.

Note:

For client side call's, we use simple button.

For server side call's, we use submit button.

```
<input type="reset" value="refresh"/>
```

In case of reset button, the page is reloaded (the content will be reloaded).

<fieldset>...</fieldset>

It is used to create a group box. (Panel)

<legend>...</legend>

It is used to display some header text for the <fieldset>

Syntax:

```
<fieldset>
    <legend>Group header</legend>

    <!--input types goes here-->
</fieldset>
```

Note: -

In html for every element, we include id attribute.

The id attribute will uniquely identity each element.

The value of id attribute should be unique.

We use the value of id attribute in JavaScript for manipulation of the html element.

We use Id for JavaScript DOM manipulation.

We can also use id in CSS to apply styles uniquely to each element.

This ID attribute is applicable for every html element (event <form> also)

But in case of <form> tag, we use id as well name attributes.

The name attribute is for specifying the name/value for any <input /> tag.

This name attribute is applicable for only <form> <input> types.

Whenever we click on submit button, the content of the page is sent to server with all name/value pair.

Note:

You must specify the name attribute for every input type for sending that info to the server (for query string).

Form Elements

Text Fields - For entering one line of text

To put a Text Field into our page we place the following input tag inside our form tags:

```
<input type="text" name="firstname">
```

```
<input type="text" name="country" size="3" maxlength="2"
value="defaultvalue"/>
<input type="text" name="lastname" size="5" />
```

Here, size refers the width of the textbox.

Textarea Fields - For entering multiple lines of text

To put a Textarea Field into our page we place the following textarea tag inside our form tags:

```
<textarea rows="10" cols="30" name="comment">Default Text</textarea>
```

DropDownList

Select Boxes - For selecting a choice from a list

To put a Select Box on our web page we place the following series of tags inside our form tags:

```
<select name="area_code">
   <option value="Cellcom">058</option>
   <option value="Pelephone">051</option>
 </select>
```

Note:

Text

Value

Text is Visible for each item in the Dropdown list

For each item, text is associated with value property.

Example:

```
<select name="gender">
        <option value="0" selected>Choose Gender</option>
        <option value="1">Male</option>
        <option value="2">Female</option>
</select>
```

Note:

We use selected attribute, to specify the default choice.

By default <select> allows single selection.

To enable multiple selections (listbox), we use multiple attribute in the <select> tag.

Syntax: -

```
<select name="" multiple="multiple">
    <option>...</option>
</select>
```

Radio Buttons - For selecting one of a limited number of choices

To put a set of Radio Buttons on our web page we place the following set of input tags inside our form tags:

```
Male: <input type="radio" name="Gender" value="Male" checked />
Female: <input type="radio" name="Gender" value="Female" />
```

Note:

We use checked attribute to specify the default choice.

To group the multiple radio buttons, give the same name.

You can also add multiple radio buttons into a group box or panel. So that they are grouped.

Checkboxes - For selecting one or more of a limited number of choices

To put a Checkbox on our web page, we put the following input tag inside our forms tags:

```
Check me <input type="checkbox" name="check1" value="yes" />
```

Note:

For checkbox also, we use the same checked attribute.

Buttons - For submiting or reseting the form

To put a button in our web page we use the input button tags. There two button types: submit, and reset.

A submit button will take the information in the form and send it to be processed by the action defined by the form using the method defined by the form. Without at least a submit button, your form won't do anything.

A reset button will reset all the values of the fields in your form.

Let's see an example:

```
<input type="submit" value="Click Me!" />
<input type="reset" value="Refresh" />
<input type="button" value="getDate"/>
```

HTML5 - Web Forms 2.0

Web Forms 2.0 is an extension to the forms features found in HTML4.

Form elements and attributes in HTML5 provide a greater degree of semantic mark-up than HTML4 and remove a great deal of the need for tedious scripting and styling that was required in HTML4.

The <input> element in HTML4

HTML4 input elements use the type attribute to specify the data type.

HTML4 provides following types:

Type	Description
Text	A free-form text field, nominally free of line breaks.
Password	A free-form text field for sensitive information, nominally free of line breaks.
Checkbox	A set of zero or more values from a predefined list.
Radio	An enumerated value.
Submit	A free form of button initiates form submission.
File	An arbitrary file with a MIME type and optionally a file name.
Image	A coordinate, relative to a particular image's size, with the extra semantic that it must be the last value selected and initiates form submission.
Hidden	An arbitrary string that is not normally displayed to the user.
Select	An enumerated value, much like the radio type.
Textarea	A free-form text field, nominally with no line break restrictions.
Button	A free form of button which can initiates any event related to button.

The <input> element in HTML5

Apart from the above mentioned attributes, HTML5 input elements introduced several new values for the type attribute.

These are listed below.

NOTE: Try the entire following example using latest version of Opera browser.

Type	Description
Datetime	A date and time (year, month, day, hour, minute, second, fractions of a second) encoded according to ISO 8601 with the time zone set to UTC.
datetime-local	A date and time (year, month, day, hour, minute, second, fractions of a second) encoded according to ISO 8601, with no time zone information.
Date	A date (year, month, day) encoded according to ISO 8601.
Month	A date consisting of a year and a month encoded according to ISO 8601.
Week	A date consisting of a year and a week number encoded according to ISO 8601.
Time	A time (hour, minute, seconds, fractional seconds) encoded according to ISO 8601.
Number	This accepts only numerical value. The step attribute specifies the precision, defaulting to 1.
Range	The range type is used for input fields that should contain a value from a range of numbers.
Email	This accepts only email value. This type is used for input fields that should contain an e-mail address. If you try to submit a simple text, it forces to enter only email address in email@example.com format.
url	This accepts only URL value. This type is used for input fields that should contain a URL address. If you try to submit a simple text, it forces to enter only URL address either in http://www.example.com format or in http://example.com format.

The autofocus attribute

This is a simple one-step pattern, easily programmed in JavaScript at the time of document load, automatically focus one particular form field.

HTML5 introduced a new attribute called autofocus which would be used as follows:

<input type="text" name="search" autofocus/>

The required attribute

Now you do not need to have javascript for client side validations like empty text box would never be submitted because HTML5 introduced a new attribute called required which would be used as follows and would insist to have a value:

<input type="text" name="search" required/>

The placeholder attribute

HTML5 introduced a new attribute called placeholder. This attribute on <input> and <textarea> elements provides a hint to the user of what can be entered in the field. The placeholder text must not contain carriage returns or line-feeds.

Here is the simple syntax for placeholder attribute:

<input type="text" name="search" placeholder="search the web"/>

Example on: <datalist> tag

DATALIST Element is basically use for auto completion of the form. The complete list is put in the option box, and when the user double click the input field the option is dragged down. This is best feature if user doesn't know what to fill in the input box.

```
<!DOCTYPE html>

<html>

  <head>
     <title>Title name will go here</title>
  </head>

  <body>

     <input list="country">

     <datalist id="country">

        <option value="India">
        <option value="Australia">
        <option value="Sourth Africa">
        <option value="Canada">
        <option value="America">

     </datalist>

     <input type="submit" value="submit"/>

  </body>

</html>
```

Email Validation:

```
<form>
    <input type="email" required /> <br />
    <input type="submit" value="Submit Now!">
</form>
```

Some browsers only look for the @ and other browsers look for at pattern consisting of a @ followed by at least one letter and a dot.

Using patterns to validate email addresses:

Another way to validate email addresses is to use the pattern attribute.

The pattern can be anything you specify and it is based on regular expressions

```
<form>
    <input pattern="/^[a-zA-Z0-9.!#$%&'*+/=?^_`{|}~-]+@[a-zA-Z0-9-]+(?:\.[a-zA-Z0-9-]+)*$/" required />
    <br />
    <input type="submit" value="Submit Now!">
</form>
```

Validating URLs:

```
<form>
    <input type="url" required />
    <input type="submit" value="Submit Now!">
</form>

<form>
    <input type="url" pattern="https?://.+" required />
    <input type="submit" value="Submit Now!">
</form>
```

Validating phone numbers:

```
<form>
    Your phonenumber: <input type="tel" required />
    <input type="submit" value="Submit now" />
</form>
```

Validating dates:

DATE

The format for the date type is YYYY-MM-DD, which means that May 14th 2012 would be rendered 2012-05-12.

```
<form>
  <input type="date">
  <input type="submit" value="Submit Now!">
</form>
```

MONTH

The format for the month type is YYYY-MM, which means that May 2012 would be rendered 2012-05.

```
<form>
  <input type="month">
  <input type="submit" value="Submit Now!">
</form>
```

WEEK

Sometimes all you need is a week, and the format for the week type is YYYY-Www, which means that week 12 in 2012 would be rendered 2012-W12.

```
<form>
  <input type="week">
  <input type="submit" value="Submit Now!">
</form>
```

TIME

Sometimes you need a specific time and the time type is renderes HH:mm:ss.ss but the seconds are optional. This means that 4.30 p.m. would be rendered 16:30 or if you choose to include seconds, 16:30:23.4

```
<form>
  <input type="time">
  <input type="submit" value="Submit Now!">
</form>
```

DATE & TIME

The datetime type has a long format: YYYY-MM-DD THH:mm:ss.s and May 14th 2012 would be rendered 2012-05-12 T16:30:23.4

```
<form>
  <input type="datetime">
  <input type="submit" value="Submit Now!">
</form>
```

Working with autocomplete:

Where autocomplete attribute is used to provide an autocompletion option to user, when user visit the form page. If autocompletion is on, it will autocomplete the form and if autocompletion is off, the user have to fill the form field mannual.

It is possible to have autocomplete "on" and "off" for the form, and "off" and "on" for specific input fields

```
<!DOCTYPE html>

<html>

  <head>
     <title>Title name will go here</title>
  </head>
  <body>
     <form action="demo_form.aspx" autocomplete="on">
      First name:<input type="text" name="fname"><br>
      Last name: <input type="text" name="lname"><br>
      E-mail: <input type="email" name="email" autocomplete="off">
      Phone: <input type="text" name="text" autocomplete="off"><br>
      <input type="submit">
     </form>
  </body>
</html>
```

Working with novalidate:

Where novalidate attribute is used to send the information for not validating the form field. It specifies that form data shouldn't be validated.

```
<!DOCTYPE html>

<html>

  <head>
     <title>Title name will go here</title>
  </head>

  <body>

     <form action="demo_form.aspx" novalidate>
      E-mail: <input type="email" name="email" autocomplete="off">
      <input type="submit">
     </form>

  </body>

</html>
```

Working with formaction:

The formaction attribute is used to send the data through the URL, as we do it with action attribute of <form> element.

The formaction attribute overrides the action attribute of the form element.

The formaction attribute is used with the following type:

✔ type="submit"

✔ type="image"

```
<!DOCTYPE html>
<html>
  <head>
     <title>Title name will go here</title>
  </head>
  <body>
     <form action="demo_form.php">
       First Name: <input type="text" name="fname">
       Last Name: <input type="text" name="lname">
       <input type="submit">
       <input type="submit" formaction="demo_admin.php" value="submit as Admin">
     </form>
  </body>
</html>
```

Working with formmethod:

The formmethod attribute is used to define the HTTP method for sending the data to the server.

The formmethod attribute overrides the method attribute of <form> element.

The formmethod attribute is used with the following type:

✔ type="submit"

✔ type="image"

```
<!DOCTYPE html>
<html>
  <head>
     <title>Title name will go here</title>
  </head>
  <body>
     <form action="demo_form.php" method="get">
       First Name: <input type="text" name="fname">
       Last Name: <input type="text" name="lname">
       <input type="submit">
       <input type="submit" formmethod="post" formaction="demo_post.php">
     </form>
  </body>
</html>
```

Working with formnovalidate:

The formnovalidate attribute is used to prevent the validation of a specific input element.

The formnovalidate attribute overrides the novalidate attribute of the <form> element.

The formmethod attribute is used with the following type:

✔ type="submit"

```
<!DOCTYPE html>
<html>
  <head>
     <title>Title name will go here</title>
  </head>
  <body>
    <form action="demo_form.php">
     First Name: <input type="text" name="fname" required>
     <input type="submit">
     <input type="submit" formnovalidate value="submit without validate">
    </form>
  </body>
</html>
```

Working with formtarget:

The formtarget is used to define the particular landing page where the server shows its response after submission of form.

The formtarget attribute overrides the target attribute of the <form> element.

The formtarget attribute is used with the following type:

✔ type="submit"

✔ type="image"

```
<!DOCTYPE html>
<html>
  <head>
     <title>Title name will go here</title>
  </head>
  <body>
    <form action="demo_form.php">
      First Name: <input type="text" name="fname">
      Last Name: <input type="text" name="fname">
      <input type="submit" value="Submit Normal">
      <input type="submit" formtarget="_blank" value="Submit in New Window">
    </form>
  </body>
</html>
```

Working with height and width attributes in the input fields:

The height and width attribute specify the height and width of an input element.

The height and width attribute is used only with type="image".

```
<!DOCTYPE html>
<html>
   <head>
       <title>Title name will go here</title>
   </head>
   <body>
      <form action="demo_form.php">
        First Name: <input type="text" name="fname">
        <input type="image" src="images/tickmark.png" height="50" width="50">
      </form>
   </body>
</html>
```

Working with min and max attributes:

The min and max attribute is used to fix the minimum and maximum value to an input field.

The following input type is support by the min and max attributes:

✔ type="number"

✔ type="range"

✔ type="date"

✔ type="datetime"

✔ type="datetime-local"

✔ type="month"

✔ type="time"

✔ type="week"

```
<!DOCTYPE html>

<html>

  <head>
     <title>Title name will go here</title>
  </head>

  <body>

     <form action="demo_form.php">
       Enter a date before 1988-12-12:
       <input type="date" name="bday" max="1988-12-12"><br>

       Enter a date after 2012-12-12:
       <input type="date" name="bday" min="2012-12-12"><br>

       Quantity (between 1 and 10):
       <input type="number" name="quantity" min="1" max="10"><br>

      <input type="submit">
     </form>

  </body>

</html>
```

Working with multiple attribute:

The multiple attribute is used to give the permission to user selecting or entering more than one value.

The multiple attribute works with the following input types:

✔ type="email"

✔ type="file"

```
<!DOCTYPE html>
<html>
  <head>
     <title>Title name will go here</title>
  </head>
  <body>
    <form action="demo_form.php">
      Select Images:
      <input type="file" name="img" multiple<br>
     <input type="submit">
    </form>
  </body>
</html>
```

Working pattern attribute:

The pattern attribute is used for checking the exact match against pre-defined regular expression.

The following input type is support by the pattern attribute:

✔ type="text"

✔ type="search"

✔ type="url"

✔ type="tel"

✔ type="email"

✔ type="password"

```
<!DOCTYPE html>
<html>
   <head>
       <title>Title name will go here</title>
   </head>
   <body>
      <form action="demo_form.php">
         Village Name:
  <input type="text" name="vname" pattern="[A-Za-z]{5}" title="Valid Name"/>
   <input type="submit">
      </form>
   </body>
</html>
```

Constraint Validation

Native Client Side Validation for Web Forms:

Introduction

Validating forms has notoriously been a painful development experience.

Implementing client side validation in a user friendly, developer friendly, and

accessible way is hard. Before HTML5 there was no means of implementing

validation natively; therefore, developers have resorted to a variety of JavaScript

based solutions.

To help ease the burden on developers, HTML5 introduced a concept known as

constraint validation - a native means of implementing client side validation on

web forms.

What is Constraint Validation?

The core of constraint validation is an algorithm browsers run when a form is submitted to determine its validity. To make this determination, the algorithm utilizes new HTML5 attributes min, max, step, pattern, and required as well as existing attributes maxlength and type.

As an example take this form with an empty required text input:

```
<form>
   <input type="text" required value="" />
   <input type="submit" value="Submit" />
</form>
```

DOM API

The constraint validation API adds the following properties / methods to DOM

nodes.

willValidate property:

The willValidate property indicates whether the node is a candidate for constraint

validation. For submittable elements this will be set to true unless for some reason

the node is barred from constraint validation, such as possessing the disabled

attribute.

```
<div id="one"></div>
<input type="text" id="two" />
```

```
<input type="text" id="three" disabled="disabled" />
<script>
   alert(document.getElementById('one').willValidate); //undefined

   alert(document.getElementById('two').willValidate); //true
   alert(document.getElementById('three').willValidate); //false
</script>
```

validity property:

The validity property of a DOM node returns a ValidityState object containing a

number of boolean properties related to the validity of the data in the node.

example:

```
<input type="text" id="txt1" required="required" />
<script>

   alert(document.getElementById('two').validity.valid); //true

</script>
```

customError: true if a custom validity message has been set per a call to

setCustomValidity():

```
<form id="form-1">
   Password: <input type="password" id="password1" value="" /> <br />
   Confirm Password : <input type="password" id="password2" value="" /> <br />
   <input type="submit" value="Submit" />
</form>

<script>


   var checkValidity=function(){
      if (document.getElementById('password1').value !=
document.getElementById('password2').value) {
         document.getElementById('password1').setCustomValidity('Passwords must match.');
         alert(document.getElementById("password1").validationMessage);
      } else {
         document.getElementById('password1').setCustomValidity('');
      }
   }
   document.getElementById('password1').addEventListener("change", checkValidity, false);
   document.getElementById('password2').addEventListener("change", checkValidity, false);

</script>
```

patternMismatch: true if the node's value does not match its pattern attribute.

```
<input id="foo" title="enter name" pattern="[0-9]{4}" value="1234" />
<input id="bar" title="enter value" pattern="[0-9]{4}" value="ABCD" />
<script>


      alert(document.getElementById('foo').validity.patternMismatch); //false
      alert(document.getElementById('bar').validity.patternMismatch); //true
```

```
</script>
```

rangeOverflow: true if the node's value is greater than its max attribute.

```
<input id="foo" type="number" max="2" value="1" />
<input id="bar" type="number" max="2" value="3" />
<script>
   document.getElementById('foo').validity.rangeOverflow; //false
   document.getElementById('bar').validity.rangeOverflow; //true
</script>
```

rangeUnderflow: true if the node's value is less than its min attribute.

```
<input id="foo" type="number" min="2" value="3" />
<input id="bar" type="number" min="2" value="1" />
<script>
   document.getElementById('foo').validity.rangeUnderflow; //false
   document.getElementById('bar').validity.rangeUnderflow; //true
</script>
```

stepMismatch: true if the node's value is invalid per its step attribute.

```
<input id="foo" type="number" step="2" value="4" />
<input id="bar" type="number" step="2" value="3" />
<script>
   document.getElementById('foo').validity.stepMismatch; //false
   document.getElementById('bar').validity.stepMismatch; //true
</script>
```

typeMismatch: true if an input node's value is invalid per its type attribute.

```
<input id="foo" type="url" value="http://foo.com" />
<input id="bar" type="url" value="foo" />
<input id="foo2" type="email" value="foo@foo.com" />
<input id="bar2" type="email" value="bar" />

<script>
   alert(document.getElementById('foo').validity.typeMismatch); //false
  alert( document.getElementById('bar').validity.typeMismatch); //true
   alert(document.getElementById('foo2').validity.typeMismatch); //false
  alert( document.getElementById('bar2').validity.typeMismatch); //true
</script>
```

valueMissing: true if the node has a required attribute but has no value.

```
<input id="foo" type="text" required="required" value="foo" />
<input id="bar" type="text" required="required" value="" />
<script>
   alert(document.getElementById('foo').validity.valueMissing); //false
   alert(document.getElementById('bar').validity.valueMissing); //true
</script>
```

valid: true if all of the validity conditions listed above are false.

```
<input id="valid-1" type="text" required="required" value="foo" />
<input id="valid-2" type="text" required="required" value="" />
<script>
   alert(document.getElementById('valid-1').validity.valid); //true
   alert(document.getElementById('valid-2').validity.valid); //false
</script>
```

validationMessage

The validationMessage property of a DOM node contains the message the browser displays to the user when a node's validity is checked and fails.

The browser provides a default localized message for this property. If the DOM node is not a candidate for constraint validation or if the node contains valid data validationMessage will be set to an empty string.

```
<input type="text" id="foo" required />
<script>
   document.getElementById('foo').validationMessage;
   //Chrome  --> 'Please fill out this field.'
   //Firefox --> 'Please fill out this field.'
   //Safari  --> 'value missing'
   //IE10    --> 'This is a required field.'
   //Opera   --> ''
</script>
```

checkValidity()

The checkValidity method on a form element node (e.g. input, select, textarea)

returns true if the element contains valid data.

On form nodes it returns true if all of the form's children contain valid data.

```
<form id="form-1">
   <input id="input-1" type="text" required />
</form>
<form id="form-2">
   <input id="input-2" type="text" />
</form>
<script>
  alert( document.getElementById('form-1').checkValidity());  //false
   alert(document.getElementById('input-1').checkValidity()); //false

  alert( document.getElementById('form-2').checkValidity());  //true
 alert(  document.getElementById('input-2').checkValidity()); //true
</script>
```

Additionally, every time a form element's validity is checked via checkValidity and fails, an invalid event is fired for that node. Using the example code above if you wanted to run something whenever the node with id input-1 was checked and contained invalid data you could use the following:

```
document.getElementById('input-1').addEventListener('invalid', function() {
   //...
}, false);
```

There is no valid event, however, you can use the change event for notifications of when a field's validity changes.

```
document.getElementById('input-1').addEventListener('change', function(event) {
   if (event.target.validity.valid) {
      //Field contains valid data.
   } else {
      //Field contains invalid data.
   }
}, false);
```

setCustomValidity()

The setCustomValidity method changes the validationMessage property as well as

allows you to add custom validation rules.

Because it is setting the validationMessage passing in an empty string marks the

field as valid and passing any other string marks the field as invalid. Unfortunately there is no way of setting the validationMessage without also changing the validity of a field.

For example, if you had two password fields you wanted to enforce be equal you could use the following:

```
<form id="form-1">
   Password: <input type="password" id="password1" value="" /> <br />
   Confirm Password : <input type="password" id="password2" value="" /> <br />
   <input type="submit" value="Submit" />
</form>

<script>

   var checkValidity=function(){
      if (document.getElementById('password1').value !=

document.getElementById('password2').value) {
         document.getElementById('password1').setCustomValidity('Passwords must

match.');
         alert(document.getElementById("password1").validationMessage);
      } else {
         document.getElementById('password1').setCustomValidity('');
      }
```

```
    }
    document.getElementById('password1').addEventListener("change",
checkValidity, false);
    document.getElementById('password2').addEventListener("change",
checkValidity, false);
</script>
```

HTML Attributes

We've already seen that the maxlength, min, max, step, pattern, and type attributes are used by the browser to constrain data. For constraint validation there are two additional relevant attributes - novalidate and formnovalidate.

Novalidate:

The boolean novalidate attribute can be applied to form nodes. When present this attribute indicates that the form's data should not be validated when it is submitted.

```
<form novalidate>
    <input type="text" required />
```

```
  <input type="submit" value="Submit" />
</form>
```

Because the above form has the novalidate attribute it will submit even though it

contains an empty required input.

Formnovalidate:

The boolean formnovalidate attribute can be applied to button and input nodes to

prevent form validation. For example:

```
<form>
  <input type="text" required />
  <input type="submit" value="Validate" />
  <input type="submit" value="Do NOT Validate" formnovalidate />
</form>
```

Example for setCustomValidity:

```
<form>
  <fieldset>
    <legend>Change Your Password</legend>
    <ul>
      <li>
        <label for="password1">Password 1:</label>
        <input type="password" required="required" id="password1" />
      </li>
      <li>
        <label for="password2">Password 2:</label>
        <input type="password" required="required" id="password2" />
      </li>
    </ul>
    <input type="submit" />
  </fieldset>
</form>

  <script>
var password1 = document.getElementById('password1');
var password2 = document.getElementById('password2');

var checkPasswordValidity = function() {
  if (password1.value != password2.value) {
    password1.setCustomValidity('Passwords must match.');
  } else {
    password1.setCustomValidity('');
  }
```

```
};

password1.addEventListener('change', checkPasswordValidity, false);
password2.addEventListener('change', checkPasswordValidity, false);

</script>
```

CREATING CROSS BROWSER PAGES IN HTML 5

Working with Polyfills for form validation & HTML 5 new elements:

As mentioned earlier, not all browsers and platforms support the new exciting features of HTML5's form validation. Therefore polyfills can come in handy as they improve your users' experience of your website.

Working with Modernizr for Cross browser pages in HTML 5

Modernizr is a small JavaScript Library that detects the availability of native implementations for next-generation web technologies

Modernizr is *"A JavaScript library that detects HTML5 and CSS3 features in the user's browser."*

There are several new features which are being introduced through HTML5 and CSS3 but same time many browsers do not support these news features.

Modernizr provides an easy way to detect any new feature so that you can take corresponding action.

For example, if a browser does not support video feature then you would like to display a simple page.

You can create CSS rules based on the feature availability and these rules would apply automatically on the webpage if browser does not support a new feature.

Setting up Modernizr

At the Modernizr official website, we will find two options to download the file, **Development** and **Production** version.

The Development version is a full and uncompressed version consisting of all the primary feature tests; while in the Production version, we can select the feature tests that we only need.

Syntax:

Before you start using Modernizr, you would have to include its javascript library in your HTML page header as follows:

```
<script src="modernizr.min.js" type="text/javascript"></script>
```

We also need to add `no-js` class to the `<html>` tag

```
<html class="no-js">
```

This class is necessary, in case the user's browser is running without the JavaScript enabled; we can add an appropriate fallback through this class. If it does, Modernizr will replace this class with just js.

Following is the simple syntax to handle supported and non-supported features:

```
/* In your CSS: */
.no-audio #music {
  display: none; /* Don't show Audio options */
}
.audio #music button {
  /* Style the Play and Pause buttons nicely */
}

<!-- In your HTML: -->
<div id="music">
  <audio>
    <source src="audio.ogg" />
    <source src="audio.mp3" />
  </audio>
  <button id="play">Play</button>
  <button id="pause">Pause</button>
</div>
```

Here it is notable that you need to prefix "no-" to every feature/property you want to handle for the browser which does not support them.

Following is the syntax to detect a particular feature through Javascript:

```
if (Modernizr.audio) {
        /* properties for browsers that
      support audio */
        alert("audio supports");
      }else{
        /* properties for browsers that
      does not support audio */

        alert("audio does not support");
      }
```

dotnetsatya@gmail.com

Following is the list of features which can be detected by Modernizr:

Features detected by Modernizr:

Following is the list of features which can be detected by Modernizr:

Feature	CSS Property	JavaScript Check
@font-face	.fontface	Modernizr.fontface
Canvas	.canvas	Modernizr.canvas
Canvas Text	.canvastext	Modernizr.canvastext
HTML5 Audio	.audio	Modernizr.audio
HTML5 Audio formats	NA	Modernizr.audio[format]
HTML5 Video	.video	Modernizr.video
HTML5 Video Formats	NA	Modernizr.video[format]
rgba()	.rgba	Modernizr.rgba
hsla()	.hsla	Modernizr.hsla
border-image	.borderimage	Modernizr.borderimage
border-radius	.borderradius	Modernizr.borderradius
box-shadow	.boxshadow	Modernizr.boxshadow
Multiple backgrounds	.multiplebgs	Modernizr.multiplebgs
opacity	.opacity	Modernizr.opacity
CSS Animations	.cssanimations	Modernizr.cssanimations
CSS Columns	.csscolumns	Modernizr.csscolumns
CSS Gradients	.cssgradients	Modernizr.cssgradients
CSS Reflections	.cssreflections	Modernizr.cssreflections
CSS 2D Transforms	.csstransforms	Modernizr.csstransforms
CSS 3D Transforms	.csstransforms3d	Modernizr.csstransforms3d
CSS Transitions	.csstransitions	Modernizr.csstransitions
Geolocation API	.geolocation	Modernizr.geolocation
Input Types	NA	Modernizr.inputtypes[type]
Input Attributes	NA	Modernizr.input[attribute]
localStorage	.localstorage	Modernizr.localstorage
sessionStorage	.sessionstorage	Modernizr.sessionstorage
Web Workers	.webworkers	Modernizr.webworkers
applicationCache	.applicationcache	Modernizr.applicationcache
SVG	.svg	Modernizr.svg
SVG Clip Paths	.svgclippaths	Modernizr.svgclippaths

SMIL	.smil	Modernizr.smil
Web SQL Database	.websqldatabase	Modernizr.websqldatabase
IndexedDB	.indexeddb	Modernizr.indexeddb
Web Sockets	.websockets	Modernizr.websockets
Hashchange Event	.hashchange	Modernizr.hashchange
History Management	.historymanagement	Modernizr.historymanagement
Drag and Drop	.draganddrop	Modernizr.draganddrop
Cross-window Messaging	.crosswindowmessaging	Modernizr.crosswindowmessaging
addTest() Plugin API	NA	Modernizr.addTest(str,fn)

Examples:

For Autofocus:

```
window.onload = function() {
  // get the form and its input elements
  var form = document.forms[0],
      inputs = form.elements;
  // if no autofocus, put the focus in the first field
  if (!Modernizr.input.autofocus) {
      inputs[0].focus();
  }
  // if required not supported, emulate it
}
```

For Required:

```
window.onload = function() {
   // get the form and its input elements
   var form = document.forms[0],
      inputs = form.elements;
   // if no autofocus, put the focus in the first field
   if (!Modernizr.input.autofocus) {
      inputs[0].focus();
   }

   // if required not supported, emulate it
   if (!Modernizr.input.required) {
      form.onsubmit = function() {
         var required = [], att, val;
         // loop through input elements looking for required
         for (var i = 0; i < inputs.length; i++) {
            att = inputs[i].getAttribute('required');
            // if required, get the value and trim whitespace
            if (att != null) {
               val = inputs[i].value;
               // if the value is empty, add to required array
               if (val.replace(/^\s+|\s+$/g, '') == '') {
                  required.push(inputs[i].name);
               }
            }
         }
         // show alert if required array contains any elements
         if (required.length > 0) {
            alert('The following fields are required: ' +
               required.join(', '));
            // prevent the form from being submitted
            return false;
         }
      };
   }
}
```

Using Modernizr to load external scripts

When creating a custom production version of Modernizr, the option to include `Modernizr.load()` is selected by default.

`Modernizr.load()` is an alias for `yepnope()`, a standalone script loader that was developed with Modernizr in mind

```
var init = function() {
   // get the form and its input elements
   var form = document.forms[0],
      inputs = form.elements;
   // put the focus in the first input field
   inputs[0].focus();

   // check required fields when the form is submitted
   form.onsubmit = function() {
      var required = [], att, val;
      // loop through input elements looking for required
      for (var i = 0; i < inputs.length; i++) {
         att = inputs[i].getAttribute('required');
         // if required, get the value and trim whitespace
         if (att != null) {
            val = inputs[i].value;
            // if the value is empty, add to required array
            if (val.replace(/^\s+|\s+$/g, '') == '') {
               required.push(inputs[i].name);
            }
         }
      }

      // show alert if required array contains any elements
      if (required.length > 0) {
         alert('The following fields are required: ' + required.join(', '));
         // prevent the form from being submitted
         return false;
      }
   };
};
```

A big advantage of Modernizr.load() is that it conditionally loads scripts depending on the results of testing the browser's capabilities—that's why the original is called yepnope().

It loads external scripts asynchronously—in other words, after the Document Object Model (DOM) has loaded in the browser—so it can help speed up the performance of your site.

The basic syntax for Modernizr.load() is to pass it an object with the following properties:

- test: The Modernizr property you want to detect.
- yep: The location of the script you want to load if the test succeeds. Use an array for multiple scripts.
- nope: The location of the script you want to load if the test fails. Use an array for multiple scripts.
- complete: A function to be run as soon as the external script has been loaded (optional).

Both yep and nope are optional, as long as you supply one of them

```
<script>
Modernizr.load({
   test: Modernizr.input.required,
   nope: 'js/check_required.js',
   complete: function() {
      init();
   }
});
</script>
```

For date validation:

```
<!DOCTYPE html>
<html>
<head>
    <title>Demo</title>
    <!--<script src="Scripts/jquery-1.8.2.js"></script>    -->

    <script src="modernizr.custom.52771.js"></script>

   <!-- <script src="Scripts/polyfiller.js"></script>-->

  <!--  <script src="Scripts/jquery-ui-1.8.24.js"></script>
   <script src="Scripts/jquery-ui-1.8.24.min.js"></script>
   <link href="jquery-ui.css" rel="stylesheet" />
   <link href="jquery.ui.datepicker.css" rel="stylesheet" />
   <link href="jquery.ui.theme.css" rel="stylesheet" />-->



     <script>

         ////Using jquery UI datepicker

         //$(document).ready(function () {
         //    $("#txtdob").datepicker();
         //});



         //Using Modernizer

         Modernizr.load({
         test: Modernizr.inputtypes.date,
         nope: ['http://ajax.googleapis.com/ajax/libs/jquery/1.4.4/jquery.min.js',
'http://ajax.googleapis.com/ajax/libs/jqueryui/1.8.7/jquery-ui.min.js', 'jquery-ui.css'],
         complete: function () {
              $('input[type=date]').datepicker({
                dateFormat: 'yy-mm-dd'
             });
         }
         });

         ////using webshims

         //$.webshims.polyfill();

    </script>
</head>

<body>
```

```
        <form name="form1" id="form1">
          Date of Birth: <input type="date" id="txtdob" name="txtdob" /><br />
            <input type="submit" value="submit" />
          </form>

</body>
</html>
```

Introducing Webshims

The Webshims Lib is one of the most complete polyfills out there. It is based on jQuery and Modernizr.

Including this one script will enable many features across the whole range of desktop browsers.

Webshims enables HTML5 and CSS3 features like semantic tags, canvas, web storage, geolocation, forms and multimedia.

Reading through this big list, the first thing that might come to mind is how huge this library must be.

Meaning a huge download size and long script execution time.

But here is the kicker, Webshims will automatically detect which features the users browser supports and it only loads what is necessary to simulate everything else.

This way it will not slow down users who already run a modern browser like Firefox or Chrome.

You can even reduce the amount of features to load if you don’t need everything.

How to Use Webshims

To use the webshims lib you need to include the dependencies jQuery and Modernizr alongside Webshims Lib.

```
<script src="scripts/jquery-1.8.2.min.js"></script>
<script src="scripts/modernizr-custom.js"></script>
<script src="scripts/webshim/polyfiller.js"></script>
```

Now you need to initialize Webshims and, if needed, tell it which features you want to use.

```
<script>
   // Polyfill all unsupported features
   $.webshims.polyfill();
</script>

<script>
   // Polyfill only form and canvas features
   $.webshims.polyfill('forms canvas');
</script>
```

And that's it! Webshims will automatically detect and polyfill missing features depending on the user's browser.

Example

Let’s do an example using the <video> tag.
First let's create the basic page without Webshims or any other polyfill.

```
<!DOCTYPE html>
<html>
<head>
  <title>Video native</title>
</head>
<body>
  <video width="480" height="360" controls="controls">
    <source src="Video.mp4" type="video/mp4">
    <source src="Video.webm" type="video/webm">
  </video>
</body>
</html>
```

Modern browsers will display this video correctly, but Internet Explorer 6, 7 or 8 will not.

Now we change the example to embed the Webshims Lib.
You can see that it's not necessary to edit anything else.

```
<!DOCTYPE html>
<html>
<head>
 <title>Video including polyfill</title>
 <script src="scripts/jquery-1.8.2.min.js"></script>
 <script src="scripts/modernizr-custom.js"></script>
 <script src="scripts/webshim/polyfiller.js"></script>
 <script>
  $.webshims.polyfill('mediaelement');
 </script>
</head>
<body>
 <video width="480" height="360" controls="controls">
  <source src="Video.mp4" type="video/mp4">
  <source src="Video.webm" type="video/webm">
 </video>
</body>
</html>
```

The modern browser will display its native <video> tag player but now this functionality is also available in Internet Explorer 6+.

HTML5 - Web Storage

HTML5 introduces two mechanisms, similar to HTTP session cookies, for storing structured data on the client side and to overcome following drawbacks.

Cookies are included with every HTTP request, thereby slowing down your web application by transmitting the same data.

Cookies are included with every HTTP request, thereby sending data unencrypted over the internet.

Cookies are limited to about 4 KB of data . Not enough to store required data.

The two storages are **sessionStorage** and **localStorage** and they would be used to handle different situations.

The latest versions of pretty much every browser supports HTML5 Storage including Internet Explorer.

sessionStorage:

The **sessionStorage** is designed for scenarios where the user is carrying out a single transaction, but could be carrying out multiple transactions in different windows at the same time.

Example:

For example, if a user buying plane tickets in two different windows, using the same site. If the site used cookies to keep track of which ticket the user was buying, then as the user clicked from page to page in both windows, the ticket currently being purchased would "leak" from one window to the other, potentially causing the user to buy two tickets for the same flight without really noticing.

HTML5 introduces the sessionStorage attribute which would be used by the sites to add data to the session storage, and it will be accessible to any page from the same site opened in that window ie session and as soon as you close the window, session would be lost.

Following is the code which would set a session variable and access that variable:

```
<!DOCTYPE HTML>
<html>
<body>

 <script type="text/javascript">
  if( sessionStorage.hits ){
    sessionStorage.hits = Number(sessionStorage.hits) +1;
  }else{
    sessionStorage.hits = 1;
  }
  document.write("Total Hits :" + sessionStorage.hits );
 </script>
 <p>Refresh the page to increase number of hits.</p>
 <p>Close the window and open it again and check the result.</p>

</body>
</html>
```

Local Storage:

The Local Storage is designed for storage that spans multiple windows, and lasts beyond the current session. In particular, Web applications may wish to store megabytes of user data, such as entire user-authored documents or a user's mailbox, on the client side for performance reasons.

Again, cookies do not handle this case well, because they are transmitted with every request.

Example:

HTML5 introduces the **localStorage** attribute which would be used to access a page's local storage area without no time limit and this local storage will be available whenever you would use that page.

Following is the code which would set a local storage variable and access that variable every time this page is accessed, even next time when you open the window:

```
<!DOCTYPE HTML>
<html>
<body>

 <script type="text/javascript">
  if( localStorage.hits ){
    localStorage.hits = Number(localStorage.hits) +1;
```

```
   }else{
     localStorage.hits = 1;
   }
   document.write("Total Hits :" + localStorage.hits );
  </script>
  <p>Refresh the page to increase number of hits.</p>
  <p>Close the window and open it again and check the result.</p>

</body>
</html>
```

Delete Web Storage:

Storing sensitive data on local machine could be dangerous and could leave a security hole.

The Session Storage Data would be deleted by the browsers immediately after the session gets terminated.

To clear a local storage setting you would need to call localStorage.remove('key'); where 'key' is the key of the value you want to remove. If you want to clear all settings, you need to call localStorage.clear() method.

Following is the code which would clear complete local storage:

```
<!DOCTYPE HTML>
<html>
<body>
  <script type="text/javascript">
    localStorage.clear();

    // Reset number of hits.
    if( localStorage.hits ){
      localStorage.hits = Number(localStorage.hits) +1;
    }else{
      localStorage.hits = 1;
    }
    document.write("Total Hits :" + localStorage.hits );
  </script>
  <p>Refreshing the page would not to increase hit counter.</p>
  <p>Close the window and open it again and check the result.</p>

</body>
</html>
```

dotnetsatya@gmail.com