* **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. HTML5 is a cooperation between the World Wide Web Consortium (W3C) and the Web Hypertext Application Technology Working Group (WHATWG).

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 Silverlight, and Google Gears.

**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 can help you in building modern websites. Here is a set of some of the most prominent features introduced in HTML5.

1. **New Semantic Elements** − These are like <header>, <footer>, and <section>.
2. **Forms 2.0** − Improvements to HTML web forms where new attributes have been introduced for <input> tag.
3. **Persistent Local Storage** − To achieve without resorting to third-party plugins.
4. **WebSocket** − A next-generation bidirectional communication technology for web applications.
5. **Server-Sent Events** − HTML5 introduces events which flow from web server to the web browsers and they are called Server-Sent Events (SSE).
6. **Canvas** − This supports a two-dimensional drawing surface that you can program with JavaScript.
7. **Audio & Video** − You can embed audio or video on your webpages without resorting to third-party plugins.
8. **Geolocation** − Now visitors can choose to share their physical location with your web application.
9. **Microdata** − This lets you create your own vocabularies beyond HTML5 and extend your web pages with custom semantics.
10. **Drag and drop** − Drag and drop the items from one location to another location on the same webpage.

**Backward Compatibility**

HTML5 is designed, as much as possible, to be backward compatible with existing web browsers. Its new features have been built on existing features and allow you to provide fallback content for older browsers.

* **HTML5 - Syntax**

The HTML 5 language has a "custom" HTML syntax that is compatible with HTML 4 and XHTML1 documents published on the Web, but is not compatible with the more esoteric SGML features of HTML 4. HTML 5 does not have the same syntax rules as XHTML where we needed lower case tag names, quoting our attributes, an attribute had to have a value and to close all empty elements. HTML5 comes with a lot of flexibility and it supports the following features −

* Uppercase tag names.
* Quotes are optional for attributes.
* Attribute values are optional.
* Closing empty elements are optional.
* The DOCTYPE

**DOCTYPE**s 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>

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">

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 simply use the following syntax −

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

**HTML5 Elements:** HTML5 elements are marked up using start tags and end tags. Tags are delimited using angle brackets with the tag name in between. The difference between start tags and end tags is that the latter includes a slash before the tag name. Following is the example of an HTML5 element −

<p>...</p>

HTML5 tag names are case insensitive and may be written in all uppercase or mixed case, although the most common convention is to stick with lowercase. Most of the elements contain some content like <p>...</p> contains a paragraph. Some elements, however, are forbidden from containing any content at all and these are known as void elements. For example, **br, hr, link, meta**, etc.

**HTML5 Attributes**: Elements may contain attributes that are used to set various properties of an element. Some attributes are defined globally and can be used on any element, while others are defined for specific elements only. All attributes have a name and a value and look like as shown below in the example. Following is the example of an HTML5 attribute which illustrates how to mark up a div element with an attribute named class using a value of "example" −

<div class = "example">...</div>

Attributes may only be specified within start tags and must never be used in end tags. HTML5 attributes are case insensitive and may be written in all uppercase or mixed case, although the most common convention is to stick with lowercase.

|  |  |  |
| --- | --- | --- |
| **Attribute** | **Options** | **Function** |
| accesskey | User Defined | Specifies a keyboard shortcut to access an element. |
| align | right, left, center | Horizontally aligns tags |
| background | URL | Places an background image behind an element |
| bgcolor | numeric, hexidecimal, RGB values | Places a background color behind an element |
| class | User Defined | Classifies an element for use with Cascading Style Sheets. |
| contenteditable | true, false | Specifies if the user can edit the element's content or not. |
| contextmenu | Menu id | Specifies the context menu for an element. |
| data-XXXX | User Defined | Custom attributes. Authors of a HTML document can define their own attributes. Must start with "data-". |
| draggable | true,false, auto | Specifies whether or not a user is allowed to drag an element. |
| height | Numeric Value | Specifies the height of tables, images, or table cells. |
| hidden | hidden | Specifies whether element should be visible or not. |
| id | User Defined | Names an element for use with Cascading Style Sheets. |
| item | List of elements | Used to group elements. |
| itemprop | List of items | Used to group items. |
| spellcheck | true, false | Specifies if the element must have it's spelling or grammar checked. |
| style | CSS Style sheet | Specifies an inline style for an element. |
| subject | User define id | Specifies the element's corresponding item. |
| tabindex | Tab number | Specifies the tab order of an element. |
| title | User Defined | "Pop-up" title for your elements. |
| valign | top, middle, bottom | Vertically aligns tags within an HTML element. |
| width | Numeric Value | Specifies the width of tables, images, or table cells. |

**Custom Attributes:** A new feature being introduced in HTML 5 is the addition of custom data attributes. A custom data attribute starts with **data-** and would be named based on your requirement. Here is a simple example −

<div class = "example" data-subject = "physics" data-level = "complex">

...

</div>

The above code will be perfectly valid HTML5 with two custom attributes called *datasubject* and *data-level*. You would be able to get the values of these attributes using JavaScript APIs or CSS in similar way as you get for standard attributes.

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, et cetera.
* **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 HTML 5 document would look like the following −

<!DOCTYPE html>

<html>

<head>

<meta charset = "utf-8">

<title>...</title>

</head>

<body>

<header>...</header>

<nav>...</nav>

<article>

<section>

...

</section>

</article>

<aside>...</aside>

<footer>...</footer>

</body>

</html>

[Live Demo](http://tpcg.io/g20Tmz)

* **HTML5 - Events**

When users visit your website, they perform various activities such as clicking on text and images and links, hover over defined elements, etc. These are examples of what JavaScript calls **events**.

We can write our event handlers in Javascript or VBscript and you can specify these event handlers as a value of event tag attribute. The HTML5 specification defines various event attributes as listed below −

We can use the following set of attributes to trigger any **javascript** or **vbscript** code given as value, when there is any event that takes place for any HTML5 element.

We would cover element-specific events while discussing those elements in detail in subsequent chapters.

|  |  |  |
| --- | --- | --- |
| **Attribute** | **Value** | **Description** |
| offline | script | Triggers when the document goes offline |
| onabort | script | Triggers on an abort event |
| onafterprint | script | Triggers after the document is printed |
| onbeforeonload | script | Triggers before the document loads |
| onbeforeprint | script | Triggers before the document is printed |
| onblur | script | Triggers when the window loses focus |
| oncanplay | script | Triggers when media can start play, but might has to stop for buffering |
| oncanplaythrough | script | Triggers when media can be played to the end, without stopping for buffering |
| onchange | script | Triggers when an element changes |
| onclick | script | Triggers on a mouse click |
| oncontextmenu | script | Triggers when a context menu is triggered |
| ondblclick | script | Triggers on a mouse double-click |
| ondrag | script | Triggers when an element is dragged |
| ondragend | script | Triggers at the end of a drag operation |
| ondragenter | script | Triggers when an element has been dragged to a valid drop target |
| ondragleave | script | Triggers when an element leaves a valid drop target |
| ondragover | script | Triggers when an element is being dragged over a valid drop target |
| ondragstart | script | Triggers at the start of a drag operation |
| ondrop | script | Triggers when dragged element is being dropped |
| ondurationchange | script | Triggers when the length of the media is changed |
| onemptied | script | Triggers when a media resource element suddenly becomes empty. |
| onended | script | Triggers when media has reach the end |
| onerror | script | Triggers when an error occur |
| onfocus | script | Triggers when the window gets focus |
| onformchange | script | Triggers when a form changes |
| onforminput | script | Triggers when a form gets user input |
| onhaschange | script | Triggers when the document has change |
| oninput | script | Triggers when an element gets user input |
| oninvalid | script | Triggers when an element is invalid |
| onkeydown | script | Triggers when a key is pressed |
| onkeypress | script | Triggers when a key is pressed and released |
| onkeyup | script | Triggers when a key is released |
| onload | script | Triggers when the document loads |
| onloadeddata | script | Triggers when media data is loaded |
| onloadedmetadata | script | Triggers when the duration and other media data of a media element is loaded |
| onloadstart | script | Triggers when the browser starts to load the media data |
| onmessage | script | Triggers when the message is triggered |
| onmousedown | script | Triggers when a mouse button is pressed |
| onmousemove | script | Triggers when the mouse pointer moves |
| onmouseout | script | Triggers when the mouse pointer moves out of an element |
| onmouseover | script | Triggers when the mouse pointer moves over an element |
| onmouseup | script | Triggers when a mouse button is released |
| onmousewheel | script | Triggers when the mouse wheel is being rotated |
| onoffline | script | Triggers when the document goes offline |
| onoine | script | Triggers when the document comes online |
| ononline | script | Triggers when the document comes online |
| onpagehide | script | Triggers when the window is hidden |
| onpageshow | script | Triggers when the window becomes visible |
| onpause | script | Triggers when media data is paused |
| onplay | script | Triggers when media data is going to start playing |
| onplaying | script | Triggers when media data has start playing |
| onpopstate | script | Triggers when the window's history changes |
| onprogress | script | Triggers when the browser is fetching the media data |
| onratechange | script | Triggers when the media data's playing rate has changed |
| onreadystatechange | script | Triggers when the ready-state changes |
| onredo | script | Triggers when the document performs a redo |
| onresize | script | Triggers when the window is resized |
| onscroll | script | Triggers when an element's scrollbar is being scrolled |
| onseeked | script | Triggers when a media element's seeking attribute is no longer true, and the seeking has ended |
| onseeking | script | Triggers when a media element's seeking attribute is true, and the seeking has begun |
| onselect | script | Triggers when an element is selected |
| onstalled | script | Triggers when there is an error in fetching media data |
| onstorage | script | Triggers when a document loads |
| onsubmit | script | Triggers when a form is submitted |
| onsuspend | script | Triggers when the browser has been fetching media data, but stopped before the entire media file was fetched |
| ontimeupdate | script | Triggers when media changes its playing position |
| onundo | script | Triggers when a document performs an undo |
| onunload | script | Triggers when the user leaves the document |
| onvolumechange | script | Triggers when media changes the volume, also when volume is set to "mute" |
| onwaiting | script | Triggers when media has stopped playing, but is expected to resume |

* **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 free us from a great deal of 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 −

|  |  |
| --- | --- |
| **Sr.No.** | **Type & Description** |
| 1 | **Text:** A free-form text field, nominally free of line breaks. |
| 2 | **Password:** A free-form text field for sensitive information, nominally free of line breaks. |
| 3 | **Checkbox:** A set of zero or more values from a predefined list. |
| 4 | **Radio:** An enumerated value. |
| 5 | **Submit:** A free form of button initiates form submission. |
| 6 | **File:** An arbitrary file with a MIME type and optionally a file name. |
| 7 | **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. |
| 8 | **Hidden:** An arbitrary string that is not normally displayed to the user. |
| 9 | **Select:** An enumerated value, much like the radio type. |
| 10 | **Textarea:** A free-form text field, nominally with no line break restrictions. |
| 11 | **Button:** A free form of button which can initiates any event related to button. |

**F1.html**

<form>

<p>

Checkibg of form

</p>

</form>

**F2.html**

<form action="http://www.example.com/login.php">

<p>Username:

<input type="text" name="username" size="15" maxlength="10" />

</p>

</form>

**F3.html**

<form action="http://www.example.com/login.php">

<p>Username:

<input type="text" name="username" size="15" maxlength="30" />

</p>

<p>Password:

<input type="password" name="password" size="15" maxlength="30" />

</p>

</form>

**F4.html**

<form action="http://www.example.com/comments.php">

<p>Hello dear, Enter your detail in the following:</p>

<textarea name="comments" cols="60" rows="4">Enter about you...</textarea>

</form>

**F5.html**

<form action="http://www.example.com/profile.php">

<p>Please select your favorite course:

<br />

<input type="radio" name="course" value="MCA" checked="checked" /> MCA

<input type="radio" name="course" value="MBA" /> MBA

<input type="radio" name="course" value="MSc (IT & CA)" /> M.Sc. (IT & CA)

</p>

</form>

**F6.html**

<form action="http://www.example.com/profile.php">

<p>Please select your favorite roti from the given choice:

<br />

<input type="checkbox" name="rotimenu" value="Roti" checked="checked" /> Roti

<input type="checkbox" name=" rotimenu " value="Butter roti" /> Butter Roti

<input type="checkbox" name=" rotimenu " value="Kulcha" /> Kulcha

</p>

</form>

**F7.html**

<form action="http://www.example.com/profile.php">

<p>Which of the following roti do you need</p>

<select name="Roti menu">

<option value="Butter roti"> Butter roti </option>

<option value="Rumal roti" selected=”selected”> Rumal roti </option>

<option value="Missi roti"> Missi roti </option>

</select>

</form>

**F8.html**

<form action="http://www.example.com/profile.php">

<p>Which of the following roti do you need</p>

<select name="Roti menu" multiple=”multiple”>

<option value="Butter roti"> Butter roti </option>

<option value="Rumal roti" selected=”selected”> Rumal roti </option>

<option value="Missi roti"> Missi roti </option>

</select>

</form>

**F9.html**

<form action="http://www.example.com/upload.php" method="post">

<p>Upload your song in MP3 format:</p>

<input type="file" name="user-song" /><br />

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

</form>

**F10.html**

<form action="http://www.example.com/upload.php" method="post">

<p>Upload your song in MP3 format:</p>

<input type="file" name="user-song" /><br />

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

</form>

**The <input> element in HTML5:** HTML5 input elements introduced several new values for the **type** attribute. These are listed below.

|  |  |
| --- | --- |
| **Sr.No.** | **Type & Description** |
| 1 | [**datetime**](https://www.tutorialspoint.com/html5/html5_datetime.htm): 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. |
| 2 | [**datetime-local**](https://www.tutorialspoint.com/html5/html5_datetime_local.htm) : A date and time (year, month, day, hour, minute, second, fractions of a second) encoded according to ISO 8601, with no time zone information. |
| 3 | [**date**](https://www.tutorialspoint.com/html5/html5_date.htm) : A date (year, month, day) encoded according to ISO 8601. |
| 4 | [**month**](https://www.tutorialspoint.com/html5/html5_month.htm) : A date consisting of a year and a month encoded according to ISO 8601. |
| 5 | [**week**](https://www.tutorialspoint.com/html5/html5_week.htm) : A date consisting of a year and a week number encoded according to ISO 8601. |
| 6 | [**time**](https://www.tutorialspoint.com/html5/html5_time.htm) : A time (hour, minute, seconds, fractional seconds) encoded according to ISO 8601. |
| 7 | [**number**](https://www.tutorialspoint.com/html5/html5_number.htm) : It accepts only numerical value. The step attribute specifies the precision, defaulting to 1. |
| 8 | [**range**](https://www.tutorialspoint.com/html5/html5_range.htm) : The range type is used for input fields that should contain a value from a range of numbers. |
| 9 | [**email**](https://www.tutorialspoint.com/html5/html5_email.htm) : It 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. |
| 10 | [**url**](https://www.tutorialspoint.com/html5/html5_url.htm) : It 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. |

**F11.html**

<form action="http://www.example.org/subscribe.php">

<p>Please enter your email address:</p>

<input type="email" name="email" />

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

</form>

**F12.html**

<form action="http://www.example.com/subscribe.php">

<p>Subscribe to our email list:</p>

<input type="text" name="email" />

<input type="submit" name="subscribe" value="Subscribe" />

</form>

**F13.html**

<form action="http://www.example.org/profile.php">

<p>Please enter your website address:</p>

<input type="url" name="website" />

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

</form>

**F14.html**

<form action="http://www.example.org/search.php">

<p>Search:</p>

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

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

</form>

**F15.html**

<form action="/action\_page.php">

First name:<br>

<input type="text" name="firstname" value="God"><br>

Last name:<br>

<input type="text" name="lastname" value="All mighty"><br><br>

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

<input type="reset">

</form>

**F16.html**

<form>

Select your liking color:

<input type="color" name="mycolor">

</form>

**F17.html**

<form>

Input youyr birth date:

<input type="date" name="bdt">

</form>

**F18.html**

<form>

Enter a date before 01-01-2000:

<input type="date" name="bday" max="01-01-2020"><br>

Enter a date after 2000-01-01:

<input type="date" name="bday" min="01-01-2000"><br>

<input type="submit">

</form>

**F19.html**

<form action="/action\_page.php">

<input list="Roti" name="Roti">

<datalist id="Roti">

<option value="Butter Roti">

<option value="Rumali roti">

<option value="Nan">

<option value="Kulcha">

<option value="Nan">

</datalist>

<input type="submit">

</form>

**F20.html**

<form action="/1.php"

oninput="x.value=parseInt(a.value)+parseInt(b.value)">

0

<input type="range" id="a" name="a" value="50"> 100 +

<input type="number" id="b" name="b" value="50"> =

<output name="x" for="a b"></output>

<br><br>

<input type="submit">

</form>

**F21.html**

<form>

Input yyour Birthday with time (date and time):

<input type="datetime-local" name="bdaytime">

</form>

**F22.html**

<form>

Input your month (month and year):

<input type="month" name="bdaymonth">

</form>

**F23.html**

<form oninput="x.value=parseInt(points.value)">

Input your liking number between 1 to 6:

<input type="number" name="like" min="1" max="6">

<br />min=1 <input type="number" name="like" min="1" >

<br />max=6<input type="number" name="like" max="6">

<br />readonly<input type="number" value=100 name="like" readonly>

<br />disabled<input type="number" value=10 name="like" disabled>

<br/> Range 0 to 10<input type="range" name="points" min="0" max="10">

<br/> Range value is = <output name="x" > </output>

</form>

**F24.html**

<form action="/action\_page.php">

Enter only three letters: <input type="text" name="country\_code" pattern="[A-Z]{3}" title="Three letter in captial only">

<input type="submit">

</form>

**The <output> element**

HTML5 introduced a new element <output> which is used to represent the result of different types of output, such as output written by a script. You can use the **for** attribute to specify a relationship between the output element and other elements in the document that affected the calculation (for example, as inputs or parameters). The value of the for attribute is a space-separated list of IDs of other elements.

[Live Demo](http://tpcg.io/p59YNK)

**F20.html**

<form action="/1.php" oninput="x.value=parseInt(a.value)+parseInt(b.value)">

0

<input type="range" id="a" name="a" value="50"> 100 +

<input type="number" id="b" name="b" value="50"> =

<output name="x" for="a b"></output>

<br><br>

<input type="submit">

</form>

**The placeholder attribute:** HTML5 introduced a new attribute called **placeholder**. This attribute on <input> and <textarea> elements provide 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"/>

This attribute is supported by latest versions of Mozilla, Safari and Crome browsers only.

[Live Demo](http://tpcg.io/IfCb48)

**F25.html**

<form action="/1.php">

<input type="text" name="fname" placeholder="First name">

</form>

**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/>

This attribute is supported by latest versions of Mozilla, Safari and Chrome browsers only.

**F26.html**

<!DOCTYPE HTML>

<html>

<body>

<form action = “1.html" method = "get">

Enter surname : <input type = "text" name = "surname" /> <br/>

Enter name : <input type = "text" name = "Name" autofocus/>

<p>Try to submit using Submit button</p>

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

</form>

</body>

</html>

**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/>

This attribute is supported by latest versions of Mozilla, Safari and Chrome browsers only.

**F27.html**

<!DOCTYPE HTML>

<html>

<body>

<form action = “1.html" method = "get">

Enter surname : <input type = "text" name = "surname" required/> <br/>

Enter name : <input type = "text" name = "Name" autofocus/>

<p>Try to submit using Submit button</p>

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

</form>

</body>

</html>

* **HTML5 - Audio & Video**

HTML5 features include native audio and video support without the need for Flash. The HTML5 <audio> and <video> tags make it simple to add media to a website. You need to set **src** attribute to identify the media source and include a controls attribute so the user can play and pause the media.

<video src = "11.mp4" width = "300" height = "200" controls>

Your browser does not support the <video> element.

</video>

The current HTML5 draft specification does not specify which video formats browsers should support in the video tag. But most commonly used video formats are −

* **Ogg** − Ogg files with Thedora video codec and Vorbis audio codec.
* **mpeg4** − MPEG4 files with H.264 video codec and AAC audio codec.

You can use <source> tag to specify media along with media type and many other attributes. A video element allows multiple source elements and browser will use the first recognized format –

**V1.html**

<!DOCTYPE HTML>

<html>

<body>

<video width = "300" height = "200" controls autoplay>

<source src = "e:\ckkhtml2016\11.ogg" type ="video/ogg" />

<source src = " e:\ckkhtml2016\11.mp4" type = "video/mp4" />

Your browser does not support the <video> element.

</video>

</body>

</html>

The HTML5 video tag can have a number of attributes to control the look and feel and various functionalities of the control −

|  |  |
| --- | --- |
| **Sr.No.** | **Attribute & Description** |
| 1 | **Autoplay :** This Boolean attribute if specified, the video will automatically begin to play back as soon as it can do so without stopping to finish loading the data. |
| 2 | **Autobuffer:** This Boolean attribute if specified, the video will automatically begin buffering even if it's not set to automatically play. |
| 3 | **Controls:** If this attribute is present, it will allow the user to control video playback, including volume, seeking, and pause/resume playback. |
| 4 | **Height:**This attribute specifies the height of the video's display area, in CSS pixels. |
| 5 | **Loop:**This Boolean attribute if specified, will allow video automatically seek back to the start after reaching at the end. |
| 6 | **Preload:**This attribute specifies that the video will be loaded at page load, and ready to run. Ignored if autoplay is present. |
| 7 | **Poster:**This is a URL of an image to show until the user plays or seeks. |
| 8 | **Src:**The URL of the video to embed. This is optional; you may instead use the <source> element within the video block to specify the video to embed. |
| 9 | **Width:**This attribute specifies the width of the video's display area, in CSS pixels. |

**Embedding Audio:** HTML5 supports <audio> tag which is used to embed sound content in an HTML or XHTML document as follows.

<audio src = " e:\ckkhtml2016\11.wav" controls autoplay>

Your browser does not support the <audio> element.

</audio>

The current HTML5 draft specification does not specify which audio formats browsers should support in the audio tag. But most commonly used audio formats are **ogg, mp3** and **wav**.

You can use <source&ggt; tag to specify media along with media type and many other attributes. An audio element allows multiple source elements and browser will use the first recognized format −

**V2.html**

<!DOCTYPE HTML>

<html>

<body>

<audio controls autoplay>

<source src = " e:\ckkhtml2016\11.mp3" type = "audio/mp3" />

<source src = " e:\ckkhtml2016\11.wav" type = "audio/wav" />

Your browser does not support the <audio> element.

</audio>

</body>

</html>

The HTML5 audio tag can have a number of attributes to control the look and feel and various functionalities of the control –

|  |  |
| --- | --- |
| **Sr.No.** | **Attribute & Description** |
| 1 | **Autoplay:** This Boolean attribute if specified, the audio will automatically begin to play back as soon as it can do so without stopping to finish loading the data. |
| 2 | **Autobuffer:** This Boolean attribute if specified, the audio will automatically begin buffering even if it's not set to automatically play. |
| 3 | **Controls:** If this attribute is present, it will allow the user to control audio playback, including volume, seeking, and pause/resume playback. |
| 4 | **Loop:** This Boolean attribute if specified, will allow audio automatically seek back to the start after reaching at the end. |
| 5 | **Preload:** This attribute specifies that the audio will be loaded at page load, and ready to run. Ignored if autoplay is present. |
| 6 | **Src:** The URL of the audio to embed. This is optional; you may instead use the <source> element within the video block to specify the video to embed. |

**Handling Media Events:** The HTML5 audio and video tag can have a number of attributes to control various functionalities of the control using JavaScript

|  |  |
| --- | --- |
| **S.No.** | **Event & Description** |
| 1 | **Abort:** This event is generated when playback is aborted. |
| 2 | **Canplay:** This event is generated when enough data is available that the media can be played. |
| 3 | **Ended:** This event is generated when playback completes. |
| 4 | **Error:** This event is generated when an error occurs. |
| 5 | **Loadeddata:** This event is generated when the first frame of the media has finished loading. |
| 6 | **Loadstart:** This event is generated when loading of the media begins. |
| 7 | **Pause:** This event is generated when playback is paused. |
| 8 | **Play:** This event is generated when playback starts or resumes. |
| 9 | **Progress:** This event is generated periodically to inform the progress of the downloading the media. |
| 10 | **Ratechange:** This event is generated when the playback speed changes. |
| 11 | **Seeked:** This event is generated when a seek operation completes. |
| 12 | **Seeking:** This event is generated when a seek operation begins. |
| 13 | **Suspend:** This event is generated when loading of the media is suspended. |
| 14 | **Volumechange:** This event is generated when the audio volume changes. |
| 15 | **Waiting:** This event is generated when the requested operation (such as playback) is delayed pending the completion of another operation (such as a seek). |

Following is the example which allows to play the given video −

**V3.html**[Live Demo](http://tpcg.io/6lCtEJ)

<!DOCTYPE HTML>

<html>

<head>

<script type = "text/javascript">

function PlayVideo()

{

var v = document.getElementsByTagName("video")[0];

v.play();

}

</script>

</head>

<body>

<form>

<video width = "300" height = "200" src = "e:\ckkhtml2016\11.mp4" controls>

Your browser does not support the video element.

</video>

<br />

<input type = "button" onclick = "PlayVideo();" value = "Play"/>

</form>

</body>

</html>

**Configuring Servers for Media Type**

Most servers don't by default serve Ogg or mp4 media with the correct MIME types, so you'll likely need to add the appropriate configuration for this.

AddType audio/ogg .oga

AddType audio/wav .wav

AddType video/ogg .ogv .ogg

AddType video/mp4 .mp4

* **HTML5 - Web Storage**

With web storage, web applications can store data locally within the user's browser. Before HTML5, application data had to be stored in cookies, included in every server request. Web storage is more secure, and large amounts of data can be stored locally, without affecting website performance. Unlike cookies, the storage limit is far larger (at least 5MB) and information is never transferred to the server. Web storage is per origin (per domain and protocol). All pages, from one origin, can store and access the same data.

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 **session storage** and **local storage** and they would be used to handle different situations. HTML web storage provides two objects for storing data on the client:

* window.localStorage - stores data with no expiration date
* window.sessionStorage - stores data for one session (data is lost when the browser tab is closed)
* **Session Storage**

The *Session Storage* 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.

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, i.e., **session** and as soon as you close the window, the session would be lost.

* **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.

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. The localStorage object stores the data with no expiration date. The data will not be deleted when the browser is closed, and will be available the next day, week, or year.

Before using web storage, check browser support for localStorage and sessionStorage:

if (typeof(Storage) !== "undefined")

{  
    // *Code for localStorage/sessionStorage.*  
}

else

{  
    // Sorry! No Web Storage support..  
}

**l1.html**

<!DOCTYPE html>

<html>

<body>

<div id="result"></div>

<script>

if (typeof(Storage) !== "undefined")

{

localStorage.setItem("your\_name", "CKK GOD");

document.getElementById("result").innerHTML = localStorage.getItem("your\_name");

}

else

{

document.getElementById("result").innerHTML = "Sorry, your browser does not support Web Storage...";

}

</script>

</body>

</html>

* Create a localStorage name/value pair with name="your\_name" and value="Ckk God”
* Retrieve the value of "your\_name" and insert it into the element with id="result"

The example above could also be written like this:

**l2.html**

<!DOCTYPE html>

<html>

<body>

<div id="result"></div>

<script>

if (typeof(Storage) !== "undefined")

{

localStorage.your\_name= "CKK God";

document.getElementById("result").innerHTML = localStorage.your\_name;

}

else

{

document.getElementById("result").innerHTML = "Sorry, your browser does not support Web Storage...";

}

</script>

</body>

</html>

The following example counts the number of times a user has clicked a button. In this code the value string is converted to a number to be able to increase the counter:

**l3.html**

<!DOCTYPE html>

<html>

<head>

<script>

function clickCounter()

{

if(typeof(Storage) !== "undefined")

{

if (localStorage.clickcount)

{

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

}

else

{

localStorage.clickcount = 1;

}

document.getElementById("result").innerHTML = "You have clicked the button " + localStorage.clickcount + " time(s).";

}

else

{

document.getElementById("result").innerHTML = "Sorry, your browser does not support web storage...";

}

}

</script>

</head>

<body>

<p><button onclick="clickCounter()" type="button">Click me!</button></p>

<div id="result"></div>

<p>Click the button to see the counter increase.</p>

<p>Close the browser tab (or window), and try again, and the counter will continue to count (is not reset).</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. The syntax for removing the "your\_name" localStorage item is as follows:

localStorage.removeItem("your\_name");

**Note:** Name/value pairs are always stored as strings. Remember to convert them to another format when needed!

* **HTML5 - Web SQL Database**

The Web SQL Database API isn't actually part of the HTML5 specification but it is a separate specification which introduces a set of APIs to manipulate client-side databases using SQL.Web SQL Database will work in latest version of Safari, Chrome and Opera.

**The Core Methods**

There are following three core

1. **openDatabase** − This method creates the database object either using existing database or creating new one.
2. **transaction** − This method gives us the ability to control a transaction and performing either commit or rollback based on the situation.
3. **executeSql** − This method is used to execute actual SQL query.

**Opening Database**

The *openDatabase* method takes care of opening a database if it already exists, this method will create it if it already does not exist. To create and open a database, use the following code −

var db = openDatabase('mscdb', '1.0', 'Testing database', 2 \* 1024 \* 1024);

The above method took the following five parameters −

Database name

Version number

Text description

Size of database

Creation callback

The last and 5th argument, creation callback will be called if the database is being created. Without this feature, however, the databases are still being created on the fly and correctly versioned.

**Executing queries**

To execute a query you use the database.transaction() function. This function needs a single argument, which is a function that takes care of actually executing the query as follows −

var db = openDatabase('mscdb', '1.0', 'Testing database', 2 \* 1024 \* 1024);

db.transaction(function (ts)

{

ts.executeSql('create table if not exists stud (no unique, name)');

});

The above query will create a table called stud in 'mscdb' database.

**Insert operation**

To create enteries into the table we add simple SQL query in the above example as follows −

var db = openDatabase('mscdb', '1.0', 'Testing database', 2 \* 1024 \* 1024);

db.transaction(function (ts)

{

ts.executeSql('create table if not exists stud (no unique, name)');

ts.executeSql('insert into stud(no,name) VALUES (1, "ckk")');

ts.executeSql('insert into stud(no,name) VALUES (2, "srd")');

});

We can pass dynamic values while creating entering as follows −

var db = openDatabase('mscdb', '1.0', 'Testing database', 2 \* 1024 \* 1024);

db.transaction(function (ts)

{

ts.executeSql('create table if not exists stud (no unique, name)');

ts.executeSql('insert into stud(no,name) values(?,?'),[e\_no,e\_name];

});

Here **e\_no** and **e\_name** are external variables, and executeSql maps each item in the array argument to the "?"s.

**Read operation**

To read already existing records we use a callback to capture the results as follows −

var db = openDatabase('mscdb', '1.0', 'Testing database', 2 \* 1024 \* 1024);

db.transaction(function (ts)

{

ts.executeSql('create table if not exists stud (no unique, name)');

ts.executeSql('insert into stud(no,name) values (1, "ckk")');

ts.executeSql('insert into stud(no,name) values (2, "srd")');

});

db.transaction(function (ts)

{

ts.executeSql('select \* from stud', [], function (ts, results)

{

var len = results.rows.length, i;

msg = "<p>Total number of records : " + len + "</p>";

document.querySelector('#status').innerHTML += msg;

for (i = 0; i < len; i++)

{

alert(results.rows.item(i).no+’ ‘+results.rows.item(i).name );

}

}, null);

});

So finally, let us keep this example in a full-fledged HTML5 document as follows and try to run it with Safari browser.

**l4.html**

<!DOCTYPE HTML>

<html>

<head>

<script type = "text/javascript">

var db = openDatabase('mscdb', '1.0', 'Testing database', 2 \* 1024 \* 1024);

var msg;

db.transaction(function (ts)

{

ts.executeSql('create table if not exists stud (no unique, name)');

ts.executeSql('insert into stud(no,name) values (1, "ckk")');

ts.executeSql('insert into stud(no,name) values (2, "srd")');

msg = '<p>Log message created and row inserted.</p>';

document.querySelector('#status').innerHTML = msg;

});

db.transaction(function (ts)

{

ts.executeSql('select \* from stud', [], function (ts, results)

{

var len = results.rows.length, i;

msg = "<p>Total number of records : " + len + "</p>";

document.querySelector('#status').innerHTML += msg;

for (i = 0; i < len; i++)

{

msg = "<p><b>" + results.rows.item(i).no +" "+results.rows.item(i).name + "</b></p>";

document.querySelector('#status').innerHTML += msg;

}

}, null);

});

</script>

</head>

<body>

<div id = "status" name = "status">Status Message</div>

</body>

</html>

* **HTML5 - IndexedDB**

The indexeddb is a new HTML5 concept to store the data inside user's browser. indexeddb is more power than local storage and useful for applications that requires to store large amount of the data. These applications can run more efficiency and load faster.

**Why to use indexeddb?**

The W3C has announced that the Web SQL database is a deprecated local storage specification so web developer should not use this technology any more. indexeddb is an alternative for web SQL data base and more effective than older technologies.

**Features**

* it stores key-pair values
* it is not a relational database
* IndexedDB API is mostly asynchronous
* it is not a structured query language
* it has supported to access the data from same domain

**IndexedDB**

Before enter into an indexeddb, we need to add some prefixes of implementation as shown below

window.indexedDB = window.indexedDB || window.mozIndexedDB || window.webkitIndexedDB || window.msIndexedDB;

window.IDBTransaction = window.IDBTransaction || window.webkitIDBTransaction ||

window.msIDBTransaction;

window.IDBKeyRange = window.IDBKeyRange || window.webkitIDBKeyRange || window.msIDBKeyRange

if (!window.indexedDB)

{

window.alert("Your browser doesn't support a stable version of IndexedDB.")

}

**Open an IndexedDB database**

Before creating a database, we have to prepare some data for the data base. let's start with stud details.

const stud = [ { no: "01", name: "SR Dwivedi" }, { no: "02", name: "PM Dolia"}];

**Adding the data**

Here adding some data manually into the data as shown below −

function add()

{

var request = db.transaction(["stud"], "readwrite").objectStore("stud").add{ no: "01", name: "SR Dwivedi" });

request.onsuccess = function(event)

{

alert("SR Dwivedi has been added to the database.");

};

request.onerror = function(event)

{

alert("Unable to add data\r\nSR Dwivedi is already exist in your database! ");

}

}

**Retrieving Data**

We can retrieve the data from the data base using with get()

function read()

{

var transaction = db.transaction(["stud"]);

var objectStore = transaction.objectStore("stud");

var request = objectStore.get("00-03");

request.onerror = function(event)

{

alert("Unable to retrieve data from database!");

};

request.onsuccess = function(event)

{

if(request.result)

{

alert(“No : “+request.result.no+", Name: " + request.result.name);

}

else

{

alert("Kenny couldn't be found in your database!");

}

};

}

Using with get(), we can store the data in object instead of that we can store the data in cursor and we can retrieve the data from cursor.

function readAll() {

var objectStore = db.transaction("employee").objectStore("employee");

objectStore.openCursor().onsuccess = function(event) {

var cursor = event.target.result;

if (cursor) {

alert("Name for id " + cursor.key + " is " + cursor.value.name + ",

Age: " + cursor.value.age + ", Email: " + cursor.value.email);

cursor.continue();

} else {

alert("No more entries!");

}

};

}

Removing the data

We can remove the data from IndexedDB with remove().Here is how the code looks like

function remove() {

var request = db.transaction(["employee"], "readwrite")

.objectStore("employee")

.delete("02");

request.onsuccess = function(event) {

alert("prasad entry has been removed from your database.");

};

}

HTML Code

To show all the data we need to use onClick event as shown below code −

<!DOCTYPE html>

<html>

<head>

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

<title>IndexedDb Demo | onlyWebPro.com</title>

</head>

<body>

<button onclick = "read()">Read </button>

<button onclick = "readAll()"></button>

<button onclick = "add()"></button>

<button onclick = "remove()">Delete </button>

</body>

</html>

The final code should be as −

<!DOCTYPE html>

<html>

<head>

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

<script type = "text/javascript">

//prefixes of implementation that we want to test

window.indexedDB = window.indexedDB || window.mozIndexedDB ||

window.webkitIndexedDB || window.msIndexedDB;

//prefixes of window.IDB objects

window.IDBTransaction = window.IDBTransaction ||

window.webkitIDBTransaction || window.msIDBTransaction;

window.IDBKeyRange = window.IDBKeyRange || window.webkitIDBKeyRange ||

window.msIDBKeyRange

if (!window.indexedDB) {

window.alert("Your browser doesn't support a stable version of IndexedDB.")

}

const employeeData = [

{ id: "00-01", name: "gopal", age: 35, email: "gopal@tutorialspoint.com" },

{ id: "00-02", name: "prasad", age: 32, email: "prasad@tutorialspoint.com" }

];

var db;

var request = window.indexedDB.open("newDatabase", 1);

request.onerror = function(event) {

console.log("error: ");

};

request.onsuccess = function(event) {

db = request.result;

console.log("success: "+ db);

};

request.onupgradeneeded = function(event) {

var db = event.target.result;

var objectStore = db.createObjectStore("employee", {keyPath: "id"});

for (var i in employeeData) {

objectStore.add(employeeData[i]);

}

}

function read() {

var transaction = db.transaction(["employee"]);

var objectStore = transaction.objectStore("employee");

var request = objectStore.get("00-03");

request.onerror = function(event) {

alert("Unable to retrieve daa from database!");

};

request.onsuccess = function(event) {

// Do something with the request.result!

if(request.result) {

alert("Name: " + request.result.name + ",

Age: " + request.result.age + ", Email: " + request.result.email);

} else {

alert("Kenny couldn't be found in your database!");

}

};

}

function readAll() {

var objectStore = db.transaction("employee").objectStore("employee");

objectStore.openCursor().onsuccess = function(event) {

var cursor = event.target.result;

if (cursor) {

alert("Name for id " + cursor.key + " is " + cursor.value.name + ",

Age: " + cursor.value.age + ", Email: " + cursor.value.email);

cursor.continue();

} else {

alert("No more entries!");

}

};

}

function add() {

var request = db.transaction(["employee"], "readwrite")

.objectStore("employee")

.add({ id: "00-03", name: "Kenny", age: 19, email: "kenny@planet.org" });

request.onsuccess = function(event) {

alert("Kenny has been added to your database.");

};

request.onerror = function(event) {

alert("Unable to add data\r\nKenny is aready exist in your database! ");

}

}

function remove() {

var request = db.transaction(["employee"], "readwrite")

.objectStore("employee")

.delete("00-03");

request.onsuccess = function(event) {

alert("Kenny's entry has been removed from your database.");

};

}

</script>

</head>

<body>

<button onclick = "read()">Read </button>

<button onclick = "readAll()">Read all </button>

<button onclick = "add()">Add data </button>

<button onclick = "remove()">Delete data </button>

</body>

</html>