HTML5 - Server Sent Events

Conventional web applications generate events which are dispatched to the web server. For example, a simple click on a link requests a new page from the server.

The type of events which are flowing from web browser to the web server may be called client-sent events.

## **Web Application for SSE**

To use Server-Sent Events in a web application, you would need to add an <eventsource> element to the document.

The **src** attribute of <eventsource> element should point to an URL which should provide a persistent HTTP connection that sends a data stream containing the events.

<!DOCTYPE HTML>

<html>

<head>

<script type = "text/javascript">

/\* Define event handling logic here \*/

</script>

</head>

<body>

<div id = "sse">

<eventsource src = "/cgi-bin/ticker.cgi" />

</div>

<div id = "ticker">

<TIME>

</div>

</body>

</html>

## **Server Side Script for SSE**

A server side script should send **Content-type** header specifying the type *text/event-stream* as follows.

print "Content-Type: text/event-stream\n\n";

After setting Content-Type, server side script would send an **Event**: tag followed by event name. Following example would send Server-Time as event name terminated by a new line character.

print "Event: server-time\n";

Final step is to send event data using **Data**: tag which would be followed by integer of string value terminated by a new line character as follows −

$time = localtime();

print "Data: $time\n";

Finally, following is complete ticker.cgi written in Perl −

#!/usr/bin/perl

print "Content-Type: text/event-stream\n\n";

while(true) {

print "Event: server-time\n";

$time = localtime();

print "Data: $time\n";

sleep(5);

}

## **Handle Server-Sent Events**

Let us modify our web application to handle server-sent events. Following is the final example.

<!DOCTYPE HTML>

<html>

<head>

<script type = "text/javascript">

document.getElementsByTagName("eventsource")[0].addEventListener("server-time",

eventHandler, false);

function eventHandler(event) {

// Alert time sent by the server

document.querySelector('#ticker').innerHTML = event.data;

}

</script>

</head>

<body>

<div id = "sse">

<eventsource src = "/cgi-bin/ticker.cgi" />

</div>

<div id = "ticker" name = "ticker">

[TIME]

</div>

</body>

</html>

# HTML5 - WebSockets

WebSockets is a next-generation bidirectional communication technology for web applications which operates over a single socket and is exposed via a JavaScript interface in HTML 5 compliant browsers.

Once you get a Web Socket connection with the web server, you can send data from browser to server by calling a **send()** method, and receive data from server to browser by an **onmessage** event handler.

Following is the API which creates a new WebSocket object.

var Socket = new WebSocket(url, [protocal] );

Here first argument, url, specifies the URL to which to connect. The second attribute, protocol is optional, and if present, specifies a sub-protocol that the server must support for the connection to be successful.

## **WebSocket Attributes**

Following are the attribute of WebSocket object. Assuming we created Socket object as mentioned above −

|  |  |
| --- | --- |
| **Sr.No.** | **Attribute & Description** |
| 1 | **Socket.readyState**  The readonly attribute readyState represents the state of the connection. It can have the following values −   * A value of 0 indicates that the connection has not yet been established. * A value of 1 indicates that the connection is established and communication is possible. * A value of 2 indicates that the connection is going through the closing handshake. * A value of 3 indicates that the connection has been closed or could not be opened. |
| 2 | **Socket.bufferedAmount**  The readonly attribute bufferedAmount represents the number of bytes of UTF-8 text that have been queued using send() method. |

## **WebSocket Events**

Following are the events associated with WebSocket object. Assuming we created Socket object as mentioned above −

|  |  |  |
| --- | --- | --- |
| **Event** | **Event Handler** | **Description** |
| open | Socket.onopen | This event occurs when socket connection is established. |
| message | Socket.onmessage | This event occurs when client receives data from server. |
| error | Socket.onerror | This event occurs when there is any error in communication. |
| close | Socket.onclose | This event occurs when connection is closed. |

## **WebSocket Methods**

Following are the methods associated with WebSocket object. Assuming we created Socket object as mentioned above −

|  |  |
| --- | --- |
| **Sr.No.** | **Method & Description** |
| 1 | **Socket.send()**  The send(data) method transmits data using the connection. |
| 2 | **Socket.close()**  The close() method would be used to terminate any existing connection. |

## **WebSocket Example**

A WebSocket is a standard bidirectional TCP socket between the client and the server. The socket starts out as a HTTP connection and then "Upgrades" to a TCP socket after a HTTP handshake. After the handshake, either side can send data.

## **Client Side HTML & JavaScript Code**

At the time of writing this tutorial, there are only few web browsers supporting WebSocket() interface. You can try following example with latest version of Chrome, Mozilla, Opera and Safari.

<!DOCTYPE HTML>

<html>

<head>

<script type = "text/javascript">

function WebSocketTest() {

if ("WebSocket" in window) {

alert("WebSocket is supported by your Browser!");

// Let us open a web socket

var ws = new WebSocket("ws://localhost:9998/echo");

ws.onopen = function() {

// Web Socket is connected, send data using send()

ws.send("Message to send");

alert("Message is sent...");

};

ws.onmessage = function (evt) {

var received\_msg = evt.data;

alert("Message is received...");

};

ws.onclose = function() {

// websocket is closed.

alert("Connection is closed...");

};

} else {

// The browser doesn't support WebSocket

alert("WebSocket NOT supported by your Browser!");

}

}

</script>

</head>

<body>

<div id = "sse">

<a href = "javascript:WebSocketTest()">Run WebSocket</a>

</div>

</body>

</html>

For installation go to –

<https://www.tutorialspoint.com/html5/html5_websocket.htm>

* Install pywebsocket
* Start the server

# HTML5 - Canvas

HTML5 element <canvas> gives you an easy and powerful way to draw graphics using JavaScript. It can be used to draw graphs, make photo compositions or do simple (and not so simple) animations.

**<canvas>** element which has only two specific attributes **width** and **height** plus all the core HTML5 attributes like id, name and class, etc.

<canvas id = "mycanvas" width = "100" height = "100"></canvas>

You can easily find that <canvas> element in the DOM using *getElementById()* method as follows −

var canvas = document.getElementById("mycanvas");

Simple example on using <canvas> element in HTML5 document.

<!DOCTYPE HTML>

<html>

<head>

<style>

#mycanvas{border:1px solid red;}

</style>

</head>

<body>

<canvas id = "mycanvas" width = "100" height = "100"></canvas>

</body>

</html>

Result:

## **The Rendering Context**

The <canvas> is initially blank, and to display something, a script first needs to access the rendering context and draw on it.

The canvas element has a DOM method called **getContext**, used to obtain the rendering context and its drawing functions. This function takes one parameter, the type of context**2d**.

Following is the code to get required context along with a check if your browser supports <canvas> element −

var canvas = document.getElementById("mycanvas");

if (canvas.getContext) {

var ctx = canvas.getContext('2d');

// drawing code here

} else {

// canvas-unsupported code here

}

## **HTML5 Canvas Examples**

This tutorial covers the following examples related to HTML5 <canvas> element and there are many more.

|  |  |
| --- | --- |
| **Sr.No.** | **Examples & Description** |
| 1 | [**Drawing Rectangles**](https://www.tutorialspoint.com/html5/canvas_drawing_rectangles.htm)  Learn how to draw rectangle using HTML5 <canvas> element |
| 2 | [**Drawing Paths**](https://www.tutorialspoint.com/html5/canvas_drawing_paths.htm)  Learn how to make shapes using paths in HTML5 <canvas> element |
| 3 | [**Drawing Lines**](https://www.tutorialspoint.com/html5/canvas_drawing_lines.htm)  Learn how to draw lines using HTML5 <canvas> element |
| 4 | [**Using Images**](https://www.tutorialspoint.com/html5/canvas_using_images.htm)  Learn how to use images with HTML5 <canvas> element |
| 5 | [**Create Gradients**](https://www.tutorialspoint.com/html5/canvas_create_gradients.htm)  Learn how to create gradients using HTML5 <canvas> element |
| 6 | [**Styles and Colors**](https://www.tutorialspoint.com/html5/canvas_styles_and_colors.htm)  Learn how to apply styles and colors using HTML5 <canvas> element |
| 7 | [**Text and Fonts**](https://www.tutorialspoint.com/html5/canvas_text_fonts.htm)  Learn how to draw amazing text using different fonts and their size. |

# 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.

## **Embedding Video**

Here is the simplest form of embedding a video file in your webpage −

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

Your browser does not support the <video> element.

</video>

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 −

<!DOCTYPE HTML>

<html>

<body>

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

<source src = "/html5/foo.ogg" type ="video/ogg" />

<source src = "/html5/foo.mp4" type = "video/mp4" />

Your browser does not support the <video> element.

</video>

</body>

</html>

## **Video Attribute Specification**

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 = "foo.wav" controls autoplay>

Your browser does not support the <audio> element.

</audio>

Most commonly used audio formats are **ogg, mp3** and **wav**.

You can use <source> 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 −

<!DOCTYPE HTML>

<html>

<body>

<audio controls autoplay>

<source src = "/html5/audio.ogg" type = "audio/ogg" />

<source src = "/html5/audio.wav" type = "audio/wav" />

Your browser does not support the <audio> element.

</audio>

</body>

</html>

## For Audio Attribute Specification refer Video Attribute specification and exclude Height, poster, width.

## **Handling Media Events**

The HTML5 audio and video tag can have a number of attributes to control various functionalities of the control using JavaScript – and there are many more.

|  |  |
| --- | --- |
| **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. |

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

<!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 = "/html5/foo.mp4">

Your browser does not support the video element.

</video>

<br />

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

</form>

</body>

</html>

HTML5 Geolocation API lets you share your location with your favorite web sites. A JavaScript can capture your latitude and longitude and can be sent to backend web server and do fancy location-aware things like finding local businesses or showing your location on a map.

var geolocation = navigator.geolocation;

The geolocation object is a service object that allows widgets to retrieve information about the geographic location of the device.

## **Geolocation Methods**

The geolocation object provides the following methods −

|  |  |
| --- | --- |
| **Sr.No.** | **Method & Description** |
| 1 | [**getCurrentPosition()**](https://www.tutorialspoint.com/html5/geolocation_getcurrentposition.htm)  This method retrieves the current geographic location of the user. |
| 2 | [**watchPosition()**](https://www.tutorialspoint.com/html5/geolocation_watchposition.htm)  This method retrieves periodic updates about the current geographic location of the device. |
| 3 | [**clearWatch()**](https://www.tutorialspoint.com/html5/geolocation_clearwatch.htm)  This method cancels an ongoing watchPosition call. |

Example:

function getLocation() {

var geolocation = navigator.geolocation;

geolocation.getCurrentPosition(showLocation, errorHandler);

}

Here showLocation and errorHandler are callback methods which would be used to get actual position

## **Location Properties**

Geolocation methods getCurrentPosition() and getPositionUsingMethodName() specify the callback method that retrieves the location information. These methods are called asynchronously with an object **Position** which stores the complete location information.

The **Position** object specifies the current geographic location of the device. The location is expressed as a set of geographic coordinates together with information about heading and speed.

The following table describes the properties of the Position object. For the optional properties if the system cannot provide a value, the value of the property is set to null.

|  |  |  |
| --- | --- | --- |
| **Property** | **Type** | **Description** |
| coords | objects | Specifies the geographic location of the device. The location is expressed as a set of geographic coordinates together with information about heading and speed. |
| coords.latitude | Number | Specifies the latitude estimate in decimal degrees. The value range is [-90.00, +90.00]. |
| coords.longitude | Number | Specifies the longitude estimate in decimal degrees. The value range is [-180.00, +180.00]. |
| coords.altitude | Number | [Optional] Specifies the altitude estimate in meters above the WGS 84 ellipsoid. |
| coords.accuracy | Number | [Optional] Specifies the accuracy of the latitude and longitude estimates in meters. |
| coords.altitudeAccuracy | Number | [Optional] Specifies the accuracy of the altitude estimate in meters. |
| coords.heading | Number | [Optional] Specifies the device's current direction of movement in degrees counting clockwise relative to true north. |
| coords.speed | Number | [Optional] Specifies the device's current ground speed in meters per second. |
| timestamp | date | Specifies the time when the location information was retrieved and the Position object created. |

### **Example**

Following is a sample code which makes use of Position object. Here showLocation method is a callback method −

function showLocation( position ) {

var latitude = position.coords.latitude;

var longitude = position.coords.longitude;

...

}

## **Handling Errors**

Geolocation is complicated, and it is very much required to catch any error and handle it gracefully.

The geolocations methods getCurrentPosition() and watchPosition() make use of an error handler callback method which gives **PositionError** object. This object has following two properties −

|  |  |  |
| --- | --- | --- |
| **Property** | **Type** | **Description** |
| code | Number | Contains a numeric code for the error. |
| message | String | Contains a human-readable description of the error. |

The following table describes the possible error codes returned in the PositionError object.

|  |  |  |
| --- | --- | --- |
| **Code** | **Constant** | **Description** |
| 0 | UNKNOWN\_ERROR | The method failed to retrieve the location of the device due to an unknown error. |
| 1 | PERMISSION\_DENIED | The method failed to retrieve the location of the device because the application does not have permission to use the Location Service. |
| 2 | POSITION\_UNAVAILABLE | The location of the device could not be determined. |
| 3 | TIMEOUT | The method was unable to retrieve the location information within the specified maximum timeout interval. |

### **Example**

Following is a sample code which makes use of PositionError object. Here errorHandler method is a callback method −

function errorHandler( err ) {

if (err.code == 1) {

// access is denied

}

...

}

## **Position Options**

Following is the actual syntax of getCurrentPosition() method −

getCurrentPosition(callback, ErrorCallback, options)

Here third argument is the **PositionOptions** object which specifies a set of options for retrieving the geographic location of the device.

Following are the options which can be specified as third argument −

|  |  |  |
| --- | --- | --- |
| **Property** | **Type** | **Description** |
| enableHighAccuracy | Boolean | Specifies whether the widget wants to receive the most accurate location estimate possible. By default this is false. |
| timeout | Number | The timeout property is the number of milliseconds your web application is willing to wait for a position. |
| maximumAge | Number | Specifies the expiry time in milliseconds for cached location information. |

### **Example**

Following is a sample code which shows how to use above mentioned methods −

function getLocation() {

var geolocation = navigator.geolocation;

geolocation.getCurrentPosition(showLocation, errorHandler, {maximumAge: 75000});

}

# HTML5 - Microdata

Microdata is a standardized way to provide additional semantics in your web pages.

Microdata lets you define your own customized elements and start embedding custom properties in your web pages. At a high level, microdata consists of a group of name-value pairs.

The groups are called **items**, and each name-value pair is a **property**. Items and properties are represented by regular elements.

### **Example**

* To create an item, the **itemscope** attribute is used.
* To add a property to an item, the **itemprop** attribute is used on one of the item's descendants.

Here there are two items, each of which has the property "name" –

<html>

<body>

<div itemscope>

<p>My name is <span itemprop = "name">Zara</span>.</p>

</div>

<div itemscope>

<p>My name is <span itemprop = "name">Nuha</span>.</p>

</div>

</body>

</html>

Result : My name is Zara.

My name is Nuha.

Properties generally have values that are strings but it can have following data types −

## **Global Attributes**

Microdata introduces five global attributes which would be available for any element to use and give context for machines about your data.

|  |  |
| --- | --- |
| **Sr.No.** | **Attribute & Description** |
| 1 | **itemscope**  This is used to create an item. The itemscope attribute is a Boolean attribute that tells that there is Microdata on this page, and this is where it starts. |
| 2 | **itemtype**  This attribute is a valid URL which defines the item and provides the context for the properties. |
| 3 | **itemid**  This attribute is global identifier for the item. |
| 4 | **itemprop**  This attribute defines a property of the item. |
| 5 | **itemref**  This attribute gives a list of additional elements to crawl to find the name-value pairs of the item. |

## **Properties Datatypes**

Properties generally have values that are strings as mentioned in above example but they can also have values that are URLs. Following example has one property, "image", whose value is a URL −

<div itemscope>

<img itemprop = "image" src = "tp-logo.gif" alt = "TutorialsPoint">

</div>

Properties can also have values that are dates, times, or dates and times. This is achieved using the **time** element and its **datetime** attribute.

<html>

<body>

<div itemscope>

My birthday is:

<time itemprop = "birthday" datetime = "1971-05-08">

Aug 5th 1971

</time>

</div>

</body>

</html>

Result :

My birthday is: Aug 5th 1971

Properties can also themselves be groups of name-value pairs, by putting the itemscope attribute on the element that declares the property.

## **Microdata API support**

If a browser supports the HTML5 microdata API, there will be a getItems() function on the global document object. If browser doesn't support microdata, the getItems() function will be undefined.

function supports\_microdata\_api() {

return !!document.getItems;

}

Modernizr does not yet support checking for the microdata API, so you’ll need to use the function like the one listed above.

The HTML5 microdata standard includes both HTML markup (primarily for search engines) and a set of DOM functions (primarily for browsers).

You can include microdata markup in your web pages, and search engines that don't understand the microdata attributes will just ignore them. But if you need to access or manipulate microdata through the DOM, you'll need to check whether the browser supports the microdata DOM API.

## **Defining Microdata Vocabulary**

To define microdata vocabulary you need a namespace URL which points to a working web page. For example https://data-vocabulary.org/Person can be used as the namespace for a personal microdata vocabulary with the following named properties −

* **name** − Person name as a simple string
* **Photo** − A URL to a picture of the person.
* **URL** − A website belonging to the person.

Using about properties a person microdata could be as follows −

<html>

<body>

<div itemscope>

<section itemscope itemtype = "http://data-vocabulary.org/Person">

<h1 itemprop = "name">Gopal K Varma</h1>

<p>

<img itemprop = "photo"

src = "http://www.tutorialspoint.com/green/images/logo.png">

</p>

<a itemprop = "url" href = "#">Site</a>

</section>

</div>

</body>

</html>

Google supports microdata as part of their Rich Snippets program. When Google's web crawler parses your page and finds microdata properties that conform to the http://datavocabulary.org/Person vocabulary, it parses out those properties and stores them alongside the rest of the page data.

# HTML5 - Drag & drop

Drag and Drop (DnD) is powerful User Interface concept which makes it easy to copy, reorder and deletion of items with the help of mouse clicks. This allows the user to click and hold the mouse button down over an element, drag it to another location, and release the mouse button to drop the element there.

To achieve drag and drop functionality with traditional HTML4, developers would either have to either have to use complex JavaScript programming or other JavaScript frameworks like jQuery etc.

Now HTML 5 came up with a Drag and Drop (DnD) API that brings native DnD support to the browser making it much easier to code up.

## **Drag and Drop Events**

There are number of events which are fired during various stages of the drag and drop operation. These events are listed below −

|  |  |
| --- | --- |
| **Sr.No.** | **Events & Description** |
| 1 | **dragstart**  Fires when the user starts dragging of the object. |
| 2 | **dragenter**  Fired when the mouse is first moved over the target element while a drag is occurring. A listener for this event should indicate whether a drop is allowed over this location. If there are no listeners, or the listeners perform no operations, then a drop is not allowed by default. |
| 3 | **dragover**  This event is fired as the mouse is moved over an element when a drag is occurring. Much of the time, the operation that occurs during a listener will be the same as the dragenter event. |
| 4 | **dragleave**  This event is fired when the mouse leaves an element while a drag is occurring. Listeners should remove any highlighting or insertion markers used for drop feedback. |
| 5 | **drag**  Fires every time the mouse is moved while the object is being dragged. |
| 6 | **drop**  The drop event is fired on the element where the drop was occurred at the end of the drag operation. A listener would be responsible for retrieving the data being dragged and inserting it at the drop location. |
| 7 | **dragend**  Fires when the user releases the mouse button while dragging an object. |

**Note** − Note that only drag events are fired; mouse events such as *mousemove* are not fired during a drag operation.

## **The DataTransfer Object**

The event listener methods for all the drag and drop events accept **Event**object which has a readonly attribute called **dataTransfer**.

The **event.dataTransfer** returns **DataTransfer** object associated with the event as follows −

function EnterHandler(event) {

DataTransfer dt = event.dataTransfer;

.............

}

The *DataTransfer* object holds data about the drag and drop operation. This data can be retrieved and set in terms of various attributes associated with DataTransfer object as explained below −

|  |  |
| --- | --- |
| **Sr.No.** | **DataTransfer attributes and their description** |
| 1 | **dataTransfer.dropEffect [ = value ]**   * Returns the kind of operation that is currently selected. * This attribute can be set, to change the selected operation. * The possible values are none, copy, link, and move. |
| 2 | **dataTransfer.effectAllowed [ = value ]**   * Returns the kinds of operations that are to be allowed. * This attribute can be set, to change the allowed operations. * The possible values are **none, copy, copyLink, copyMove, link, linkMove, move, all** and **uninitialized**. |
| 3 | **dataTransfer.types**  Returns a DOMStringList listing the formats that were set in the dragstart event. In addition, if any files are being dragged, then one of the types will be the string "Files". |
| 4 | **dataTransfer.clearData ( [ format ] )**  Removes the data of the specified formats. Removes all data if the argument is omitted. |
| 5 | **dataTransfer.setData(format, data)**  Adds the specified data. |
| 6 | **data = dataTransfer.getData(format)**  Returns the specified data. If there is no such data, returns the empty string. |
| 7 | **dataTransfer.files**  Returns a FileList of the files being dragged, if any. |
| 8 | **dataTransfer.setDragImage(element, x, y)**  Uses the given element to update the drag feedback, replacing any previously specified feedback. |
| 9 | **dataTransfer.addElement(element)**  Adds the given element to the list of elements used to render the drag feedback. |

## **Drag and Drop Process**

Following are the steps to be carried out to implement Drag and Drop operation −

### **Step 1 - Making an Object Draggable**

Here are steps to be taken −

* If you want to drag an element, you need to set the **draggable**attribute to **true** for that element.
* Set an event listener for **dragstart** that stores the data being dragged.
* The event listener **dragstart** will set the allowed effects (copy, move, link, or some combination).

### **Step 2 - Dropping the Object**

To accept a drop, the drop target has to listen to at least three events.

* The **dragenter** event, which is used to determine whether or not the drop target is to accept the drop. If the drop is to be accepted, then this event has to be canceled.
* The **dragover** event, which is used to determine what feedback is to be shown to the user. If the event is canceled, then the feedback (typically the cursor) is updated based on the dropEffect attribute's value.
* Finally, the **drop** event, which allows the actual drop to be performed.

Following is the example to drop an object into another object −

<html>

<head>

<style type="text/css">

#boxA, #boxB {

float:left;padding:10px;margin:10px;-moz-user-select:none;

}

#boxA { background-color: #6633FF; width:75px; height:75px; }

#boxB { background-color: #FF6699; width:150px; height:150px; }

</style>

<script type="text/javascript">

function dragStart(ev) {

ev.dataTransfer.effectAllowed='move';

ev.dataTransfer.setData("Text", ev.target.getAttribute('id'));

ev.dataTransfer.setDragImage(ev.target,0,0);

return true;

}

function dragEnter(ev) {

event.preventDefault();

return true;

}

function dragOver(ev) {

return false;

}

function dragDrop(ev) {

var src = ev.dataTransfer.getData("Text");

ev.target.appendChild(document.getElementById(src));

ev.stopPropagation();

return false;

}

</script>

</head>

<body>

<center>

<h2>Drag and drop HTML5 demo</h2>

<div>Try to move the purple box into the pink box.</div>

<div id="boxA" draggable="true" ondragstart="return dragStart(event)">

<p>Drag Me</p>

</div>

<div id="boxB" ondragenter="return dragEnter(event)" ondrop="return dragDrop(event)" ondragover="return dragOver(event)">Dustbin</div>

</center>

</body>

</html>

# HTML5 - Web Workers

JavaScript was designed to run in a single-threaded environment, meaning multiple scripts cannot run at the same time. Consider a situation where you need to handle UI events, query and process large amounts of API data, and manipulate the DOM.

## **What is Web Workers?**

The situation explained above can be handled using **Web Workers** who will do all the computationally expensive tasks without interrupting the user interface and typically run on separate threads.

Web Workers allow for long-running scripts that are not interrupted by scripts that respond to clicks or other user interactions, and allows long tasks to be executed without yielding to keep the page responsive.

Web Workers are background scripts and they are relatively heavy-weight, and are not intended to be used in large numbers. For example, it would be inappropriate to launch one worker for each pixel of a four megapixel image.

When a script is executing inside a Web Worker it cannot access the web page's window object (window.document), which means that Web Workers don't have direct access to the web page and the DOM API. Although Web Workers cannot block the browser UI, they can still consume CPU cycles and make the system less responsive.

## **How Web Workers Work?**

Web Workers are initialized with the URL of a JavaScript file, which contains the code the worker will execute. This code sets event listeners and communicates with the script that spawned it from the main page. Following is the simple syntax −

var worker = new Worker('bigLoop.js');

If the specified javascript file exists, the browser will spawn a new worker thread, which is downloaded asynchronously. If the path to your worker returns an 404 error, the worker will fail silently.

If your application has multiple supporting JavaScript files, you can import them **importScripts()** method which takes file name(s) as argument separated by comma as follows −

importScripts("helper.js", "anotherHelper.js");

Once the Web Worker is spawned, communication between web worker and its parent page is done using the **postMessage()** method. Depending on your browser/version, postMessage() can accept either a string or JSON object as its single argument.

Message passed by Web Worker is accessed using **onmessage** event in the main page. Now let us write our bigLoop example using Web Worker. Below is the main page (hello.htm) which will spawn a web worker to execute the loop and to return the final value of variable **j** −

<!DOCTYPE HTML>

<html>

<head>

<title>Big for loop</title>

<script>

var worker = new Worker('bigLoop.js');

worker.onmessage = function (event) {

alert("Completed " + event.data + "iterations" );

};

function sayHello() {

alert("Hello sir...." );

}

</script>

</head>

<body>

<input type = "button" onclick = "sayHello();" value = "Say Hello"/>

</body>

</html>

Following is the content of bigLoop.js file. This makes use of **postMessage()**API to pass the communication back to main page −

for (var i = 0; i <= 1000000000; i += 1) {

var j = i;

}

postMessage(j);

## **Stopping Web Workers**

Web Workers don't stop by themselves but the page that started them can stop them by calling **terminate()** method.

worker.terminate();

A terminated Web Worker will no longer respond to messages or perform any additional computations. You cannot restart a worker; instead, you can create a new worker using the same URL.

## **Handling Errors**

The following shows an example of an error handling function in a Web Worker JavaScript file that logs errors to the console. With error handling code, above example would become as following −

<script>

var worker = new Worker('bigLoop.js');

worker.onmessage = function (event) {

alert("Completed " + event.data + "iterations" );

};

worker.onerror = function (event) {

console.log(event.message, event);

};

function sayHello() {

alert("Hello sir...." );

}

</script>

## **Checking for Browser Support**

Following is the syntax to detect a Web Worker feature support available in a browser –

<script>

function myFunction() {

if (Modernizr.webworkers) {

alert("Congratulation!! you have web workers support." );

} else {

alert("Sorry!! you do not have web workers support." );

}

}

</script>

# 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**

## **Open an IndexedDB database**

## **Adding the data**

## **Retrieving Data**

## **Removing the data**

## **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>

# HTML5 - Web messaging

Web Messaging is the way for documents to separates browsing context to share the data without Dom. It overrides the cross domain communication problem in different domains, protocols or ports

For example, you want to send the data from your page to ad container which is placed at iframe or voice-versa, in this scenario, Browser throws a security exception. With web messaging we can pass the data across as a message event.

## **Message Event**

Message events fires Cross-document messaging, channel messaging, server-sent events and web sockets.it has described by Message Event interface.

## **Attributes**

|  |  |
| --- | --- |
| **Sr.No.** | **Attributes & Description** |
| 1 | **data**  Contains string data |
| 2 | **origin**  Contains Domain name and port |
| 3 | **lastEventId**  Contains unique identifier for the current message event. |
| 4 | **source**  Contains to A reference to the originating document’s window |
| 5 | **ports**  Contains the data which is sent by any message port |

## **Sending a cross-document message**

Before send cross document message, we need to create a new web browsing context either by creating new iframe or new window. We can send the data using with postMessage() and it has two arguments. They are as −

* **message** − The message to send
* **targetOrigin** − Origin name

### **Examples**

Sending message from iframe to button

var iframe = document.querySelector('iframe');

var button = document.querySelector('button');

var clickHandler = function() {

iframe.contentWindow.postMessage('The message to send.',

'https://www.tutorialspoint.com);

}

button.addEventListener('click',clickHandler,false);

Receiving a cross-document message in the receiving document

var messageEventHandler = function(event){

// check that the origin is one we want.

if(event.origin == 'https://www.tutorialspoint.com') {

alert(event.data);

}

}

window.addEventListener('message', messageEventHandler,false);

## **Channel messaging**

Two-way communication between the browsing contexts is called channel messaging. It is useful for communication across multiple origins.

## **The MessageChannel and MessagePort Objects**

While creating messageChannel, it internally creates two ports to sending the data and forwarded to another browsing context.

* **postMessage()** − Post the message throw channel
* **start()** − It sends the data
* **close()** − It close the ports

In this scenario, we are sending the data from one iframe to another iframe. Here we are invoking the data in function and passing the data to DOM.

var loadHandler = function() {

var mc, portMessageHandler;

mc = new MessageChannel();

window.parent.postMessage('documentAHasLoaded','http://foo.example',[mc.port2]);

portMessageHandler = function(portMsgEvent) {

alert( portMsgEvent.data );

}

mc.port1.addEventListener('message', portMessageHandler, false);

mc.port1.start();

}

window.addEventListener('DOMContentLoaded', loadHandler, false);

Above code, it is taking the data from port 2, now it will pass the data to second iframe

var loadHandler = function() {

var iframes, messageHandler;

iframes = window.frames;

messageHandler = function(messageEvent) {

if( messageEvent.ports.length > 0 ) {

// transfer the port to iframe[1]

iframes[1].postMessage('portopen','http://foo.example',messageEvent.ports);

}

}

window.addEventListener('message',messageHandler,false);

}

window.addEventListener('DOMContentLoaded',loadHandler,false);

Now second document handles the data by using the portMsgHandler function.

var loadHandler() {

// Define our message handler function

var messageHandler = function(messageEvent) {

// Our form submission handler

var formHandler = function() {

var msg = 'add <foo@example.com> to game circle.';

messageEvent.ports[0].postMessage(msg);

}

document.forms[0].addEventListener('submit',formHandler,false);

}

window.addEventListener('message',messageHandler,false);

}

window.addEventListener('DOMContentLoaded',loadHandler,false);

# HTML5 - CORS

**Cross-origin resource sharing (CORS)** is a mechanism to allows the restricted resources from another domain in web browser.

For suppose, if you click on *HTML5- video player* in html5 demo sections. it will ask camera permission. if user allow the permission then only it will open the camera or else it doesn't open the camera for web applications.

## **Event handles in CORS**

|  |  |
| --- | --- |
| **Sr.No.** | **Event Handler & Description** |
| 1 | **onloadstart**  Starts the request |
| 2 | **onprogress**  Loads the data and send the data |
| 3 | **onabort**  Abort the request |
| 4 | **onerror**  request has failed |
| 5 | **onload**  request load successfully |
| 6 | **ontimeout**  time out has happened before request could complete |
| 7 | **onloadend**  When the request is complete either successful or failure |

## **Example of CORS with handler**

Below example will show the example of makeCorsRequest() and onload handler

// Create the XHR object.

function createCORSRequest(method, url) {

var xhr = new XMLHttpRequest();

if ("withCredentials" in xhr) {

// XHR for Chrome/Firefox/Opera/Safari.

xhr.open(method, url, true);

} else if (typeof XDomainRequest != "undefined") {

// XDomainRequest for IE.

xhr = new XDomainRequest();

xhr.open(method, url);

} else {

// CORS not supported.

xhr = null;

}

return xhr;

}

// Helper method to parse the title tag from the response.

function getTitle(text) {

return text.match('<title>(.\*)?</title>')[1];

}

// Make the actual CORS request.

function makeCorsRequest() {

// All HTML5 Rocks properties support CORS.

var url = 'http://www.tutorialspoint.com';

var xhr = createCORSRequest('GET', url);

if (!xhr) {

alert('CORS not supported');

return;

}

// Response handlers.

xhr.onload = function() {

var text = xhr.responseText;

var title = getTitle(text);

alert('Response from CORS request to ' + url + ': ' + title);

};

xhr.onerror = function() {

alert('Woops, there was an error making the request.');

};

xhr.send();

}

# HTML5 - Web RTC

Web RTC introduced by World Wide Web Consortium (W3C). That supports browser-tobrowser applications for voice calling, video chat, and P2P file sharing.

Web RTC implements three API's as shown below −

* **MediaStream** − get access to the user's camera and microphone.
* **RTCPeerConnection** − get access to audio or video calling facility.
* **RTCDataChannel** − get access to peer-to-peer communication.

## **MediaStream**

The MediaStream represents synchronized streams of media.

stream.getAudioTracks() and stream.VideoTracks(). If there is no audio tracks, it returns an empty array and it will check video stream,if webcam connected, stream.getVideoTracks() returns an array of one MediaStreamTrack representing the stream from the webcam. A simple example is chat applications, a chat application gets stream from web camera, rear camera, microphone.

## **Sample code of MediaStream**

function gotStream(stream) {

window.AudioContext = window.AudioContext || window.webkitAudioContext;

var audioContext = new AudioContext();

// Create an AudioNode from the stream

var mediaStreamSource = audioContext.createMediaStreamSource(stream);

// Connect it to destination to hear yourself

// or any other node for processing!

mediaStreamSource.connect(audioContext.destination);

}

navigator.getUserMedia({audio:true}, gotStream);

## **Screen capture**

It's also possible in Chrome browser with mediaStreamSource and it requires HTTPS.

## **Session Control, Network & Media Information**

Web RTC required peer-to-peer communication between browsers. This mechanism required signaling, network information, session control and media information. Web developers can choose different mechanism to communicate between the browsers such as SIP or XMPP or any two way communications

### **Sample code of createSignalingChannel()**

var signalingChannel = createSignalingChannel();

var pc;

var configuration = ...;

// run start(true) to initiate a call

function start(isCaller) {

pc = new RTCPeerConnection(configuration);

// send any ice candidates to the other peer

pc.onicecandidate = function (evt) {

signalingChannel.send(JSON.stringify({ "candidate": evt.candidate }));

};

// once remote stream arrives, show it in the remote video element

pc.onaddstream = function (evt) {

remoteView.src = URL.createObjectURL(evt.stream);

};

// get the local stream, show it in the local video element and send it

navigator.getUserMedia({ "audio": true, "video": true }, function (stream) {

selfView.src = URL.createObjectURL(stream);

pc.addStream(stream);

if (isCaller)

pc.createOffer(gotDescription);

else

pc.createAnswer(pc.remoteDescription, gotDescription);

function gotDescription(desc) {

pc.setLocalDescription(desc);

signalingChannel.send(JSON.stringify({ "sdp": desc }));

}

});

}

signalingChannel.onmessage = function (evt) {

if (!pc)

start(false);

var signal = JSON.parse(evt.data);

if (signal.sdp)

pc.setRemoteDescription(new RTCSessionDescription(signal.sdp));

else

pc.addIceCandidate(new RTCIceCandidate(signal.candidate));