HTML Geolocation API

The HTML Geolocation API is used to locate a user's position.

Locate the User's Position

The HTML Geolocation API is used to get the geographical position of a user.

Since this can compromise privacy, the position is not available unless the user approves it.

Note: Geolocation is most accurate for devices with GPS, like smartphones.

Browser Support

The numbers in the table specify the first browser version that fully supports Geolocation.

API					
Geolocation	5.0 - 49.0 (http) 50.0 (https)	9.0	3.5	5.0	16.0

Note: As of Chrome 50, the Geolocation API will only work on secure contexts such as HTTPS. If your site is hosted on an non-secure origin (such as HTTP) the requests to get the users location will no longer function.

Using HTML Geolocation

The `getCurrentPosition()` method is used to return the user's position.

The example below returns the latitude and longitude of the user's position:

Example

```
<script>
var x = document.getElementById("demo");
```

```
function getLocation() {
  if (navigator.geolocation) {
    navigator.geolocation.getCurrentPosition(showPosition);
  } else {
    x.innerHTML = "Geolocation is not supported by this browser.";
  }
}

function showPosition(position) {
  x.innerHTML = "Latitude: " + position.coords.latitude +
  "<br>Longitude: " + position.coords.longitude;
}
</script>
```

Example explained:

- Check if Geolocation is supported
- If supported, run the getCurrentPosition() method. If not, display a message to the user
- If the getCurrentPosition() method is successful, it returns a coordinates object to the function specified in the parameter (showPosition)
- The showPosition() function outputs the Latitude and Longitude

The example above is a very basic Geolocation script, with no error handling.

Handling Errors and Rejections

The second parameter of the `getCurrentPosition()` method is used to handle errors. It specifies a function to run if it fails to get the user's location:

Example

```
function showError(error) {
  switch(error.code) {
    case error.PERMISSION_DENIED:
      x.innerHTML = "User denied the request for Geolocation."
      break;
    case error.POSITION_UNAVAILABLE:
      x.innerHTML = "Location information is unavailable."
      break;
    case error.TIMEOUT:
      x.innerHTML = "The request to get user location timed out."
      break;
    case error.UNKNOWN_ERROR:
```

```
    x.innerHTML = "An unknown error occurred."
    break;
  }
}
```

Displaying the Result in a Map

To display the result in a map, you need access to a map service, like Google Maps.

In the example below, the returned latitude and longitude is used to show the location in a Google Map (using a static image):

Example

```
function showPosition(position) {
  var latlon = position.coords.latitude + "," + position.coords.longitude;

  var img_url = "https://maps.googleapis.com/maps/api/staticmap?center=
  "+latlon+"&zoom=14&size=400x300&sensor=false&key=YOUR_KEY";

  document.getElementById("mapholder").innerHTML = "<img src='"+img_url+"'>";
}
```

Location-specific Information

This page has demonstrated how to show a user's position on a map.

Geolocation is also very useful for location-specific information, like:

- Up-to-date local information
- Showing Points-of-interest near the user
- Turn-by-turn navigation (GPS)

The getCurrentPosition() Method - Return Data

The `getCurrentPosition()` method returns an object on success. The latitude, longitude and accuracy properties are always returned. The other properties are returned if available:

Property	Returns
coords.latitude	The latitude as a decimal number (always returned)

coords.longitude	The longitude as a decimal number (always returned)
coords.accuracy	The accuracy of position (always returned)
coords.altitude	The altitude in meters above the mean sea level (returned if available)
coords.altitudeAccuracy	The altitude accuracy of position (returned if available)
coords.heading	The heading as degrees clockwise from North (returned if available)
coords.speed	The speed in meters per second (returned if available)
timestamp	The date/time of the response (returned if available)

Geolocation Object - Other interesting Methods

The Geolocation object also has other interesting methods:

- `watchPosition()` - Returns the current position of the user and continues to return updated position as the user moves (like the GPS in a car).
- `clearWatch()` - Stops the `watchPosition()` method.

The example below shows the `watchPosition()` method. You need an accurate GPS device to test this (like smartphone):

Example

```
<script>
var x = document.getElementById("demo");
function getLocation() {
  if (navigator.geolocation) {
    navigator.geolocation.watchPosition(showPosition);
  } else {
    x.innerHTML = "Geolocation is not supported by this browser.";
  }
}
function showPosition(position) {
  x.innerHTML = "Latitude: " + position.coords.latitude +
  "<br>Longitude: " + position.coords.longitude;
}
</script>
```

HTML Drag and Drop API

In HTML, any element can be dragged and dropped.

Example

Drag the W3Schools image into the rectangle.

Drag and Drop

Drag and drop is a very common feature. It is when you "grab" an object and drag it to a different location.

Browser Support

The numbers in the table specify the first browser version that fully supports Drag and Drop.

API				
Drag and Drop 4.0	9.0	3.5	6.0	12.0

HTML Drag and Drop Example

The example below is a simple drag and drop example:

Example

```
<!DOCTYPE HTML>
<html>
<head>
<script>
function allowDrop(ev) {
  ev.preventDefault();
}

function drag(ev) {
  ev.dataTransfer.setData("text", ev.target.id);
}
```

```
function drop(ev) {
  ev.preventDefault();
  var data = ev.dataTransfer.getData("text");
  ev.target.appendChild(document.getElementById(data));
}
</script>
</head>
<body>

<div id="div1" ondrop="drop(event)" ondragover="allowDrop(event)"></div>

<img id="drag1" src="img_logo.gif" draggable="true" ondragstart="drag(event)" width="336"
height="69">

</body>
</html>
```

It might seem complicated, but lets go through all the different parts of a drag and drop event.

Make an Element Draggable

First of all: To make an element draggable, set the `draggable` attribute to true:

```
<img draggable="true">
```

What to Drag - ondragstart and setData()

Then, specify what should happen when the element is dragged.

In the example above, the `ondragstart` attribute calls a function, drag(event), that specifies what data to be dragged.

The `dataTransfer.setData()` method sets the data type and the value of the dragged data:

```
function drag(ev) {
  ev.dataTransfer.setData("text", ev.target.id);
}
```

In this case, the data type is "text" and the value is the id of the draggable element ("drag1").

Where to Drop - ondragover

The `ondragover` event specifies where the dragged data can be dropped.

By default, data/elements cannot be dropped in other elements. To allow a drop, we must prevent the default handling of the element.

This is done by calling the `event.preventDefault()` method for the ondragover event:

event.preventDefault()

Do the Drop - ondrop

When the dragged data is dropped, a drop event occurs.

In the example above, the ondrop attribute calls a function, drop(event):

```
function drop(ev) {
  ev.preventDefault();
  var data = ev.dataTransfer.getData("text");
  ev.target.appendChild(document.getElementById(data));
}
```

Code explained:

- Call preventDefault() to prevent the browser default handling of the data (default is open as link on drop)
- Get the dragged data with the dataTransfer.getData() method. This method will return any data that was set to the same type in the setData() method
- The dragged data is the id of the dragged element ("drag1")
- Append the dragged element into the drop element

HTML Web Storage API

HTML web storage; better than cookies.

What is HTML 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.

Browser Support

The numbers in the table specify the first browser version that fully supports Web Storage.

API					
Web Storage	4.0	8.0	3.5	4.0	11.5

HTML Web Storage Objects

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)

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..
}
```

The localStorage Object

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.

Example

```
// Store
localStorage.setItem("lastname", "Smith");

// Retrieve
document.getElementById("result").innerHTML = localStorage.getItem("lastname");
```

Example explained:

- Create a localStorage name/value pair with name="lastname" and value="Smith"
- Retrieve the value of "lastname" and insert it into the element with id="result"

The example above could also be written like this:

```
// Store
localStorage.lastname = "Smith";
// Retrieve
document.getElementById("result").innerHTML = localStorage.lastname;
```

The syntax for removing the "lastname" localStorage item is as follows:

```
localStorage.removeItem("lastname");
```

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

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:

Example

```
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).";
```

The sessionStorage Object

The `sessionStorage` object is equal to the localStorage object, **except** that it stores the data for only one session. The data is deleted when the user closes the specific browser tab.

The following example counts the number of times a user has clicked a button, in the current session:

Example

```
if (sessionStorage.clickcount) {
  sessionStorage.clickcount = Number(sessionStorage.clickcount) + 1;
} else {
  sessionStorage.clickcount = 1;
}
document.getElementById("result").innerHTML = "You have clicked the button " +
sessionStorage.clickcount + " time(s) in this session.";
```

HTML Web Workers API

A web worker is a JavaScript running in the background, without affecting the performance of the page.

What is a Web Worker?

When executing scripts in an HTML page, the page becomes unresponsive until the script is finished.

A web worker is a JavaScript that runs in the background, independently of other scripts, without affecting the performance of the page. You can continue to do whatever you want: clicking, selecting things, etc., while the web worker runs in the background.

Browser Support

The numbers in the table specify the first browser version that fully support Web Workers.

API				
Web Workers 4.0	10.0	3.5	4.0	11.5

HTML Web Workers Example

The example below creates a simple web worker that count numbers in the background:

Example

Count numbers:

Check Web Worker Support

Before creating a web worker, check whether the user's browser supports it:

```
if (typeof(Worker) !== "undefined") {
  // Yes! Web worker support!
  // Some code.....
} else {
  // Sorry! No Web Worker support..
}
```

Create a Web Worker File

Now, let's create our web worker in an external JavaScript.

Here, we create a script that counts. The script is stored in the "demo_workers.js" file:

```
var i = 0;

function timedCount() {
  i = i + 1;
  postMessage(i);
  setTimeout("timedCount()",500);
}

timedCount();
```

The important part of the code above is the `postMessage()` method - which is used to post a message back to the HTML page.

Note: Normally web workers are not used for such simple scripts, but for more CPU intensive tasks.

Create a Web Worker Object

Now that we have the web worker file, we need to call it from an HTML page.

The following lines checks if the worker already exists, if not - it creates a new web worker object and runs the code in "demo_workers.js":

```
if (typeof(w) == "undefined") {
  w = new Worker("demo_workers.js");
}
```

Then we can send and receive messages from the web worker.

Add an "onmessage" event listener to the web worker.

```
w.onmessage = function(event){
  document.getElementById("result").innerHTML = event.data;
};
```

When the web worker posts a message, the code within the event listener is executed. The data from the web worker is stored in event.data.

Terminate a Web Worker

When a web worker object is created, it will continue to listen for messages (even after the external script is finished) until it is terminated.

To terminate a web worker, and free browser/computer resources, use the `terminate()` method:

```
w.terminate();
```

Reuse the Web Worker

If you set the worker variable to undefined, after it has been terminated, you can reuse the code:

```
w = undefined;
```

Full Web Worker Example Code

We have already seen the Worker code in the .js file. Below is the code for the HTML page:

Example

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

<p>Count numbers: <output id="result"></output></p>
<button onclick="startWorker()">Start Worker</button>
<button onclick="stopWorker()">Stop Worker</button>

<script>
var w;

function startWorker() {
  if (typeof(Worker) !== "undefined") {
    if (typeof(w) == "undefined") {
      w = new Worker("demo_workers.js");
    }
    w.onmessage = function(event) {
      document.getElementById("result").innerHTML = event.data;
    };
  } else {
    document.getElementById("result").innerHTML = "Sorry! No Web Worker support.";
  }
}

function stopWorker() {
  w.terminate();
  w = undefined;
}
</script>

</body>
</html>
```

Web Workers and the DOM

Since web workers are in external files, they do not have access to the following JavaScript objects:

- The window object
- The document object
- The parent object

HTML SSE API

Server-Sent Events (SSE) allow a web page to get updates from a server.

Server-Sent Events - One Way Messaging

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

This was also possible before, but the web page would have to ask if any updates were available. With server-sent events, the updates come automatically.

Examples: Facebook/Twitter updates, stock price updates, news feeds, sport results, etc.

Browser Support

The numbers in the table specify the first browser version that fully support server-sent events.

API					
SSE	6.0	79.0	6.0	5.0	11.5

Receive Server-Sent Event Notifications

The EventSource object is used to receive server-sent event notifications:

Example

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

Example explained:

- Create a new EventSource object, and specify the URL of the page sending the updates (in this example "demo_sse.php")
- Each time an update is received, the onmessage event occurs
- When an onmessage event occurs, put the received data into the element with id="result"

Check Server-Sent Events Support

In the tryit example above there were some extra lines of code to check browser support for server-sent events:

```
if(typeof(EventSource) !== "undefined") {
  // Yes! Server-sent events support!
  // Some code.....
} else {
  // Sorry! No server-sent events support..
}
```

Server-Side Code Example

For the example above to work, you need a server capable of sending data updates (like PHP or ASP).

The server-side event stream syntax is simple. Set the "Content-Type" header to "text/event-stream". Now you can start sending event streams.

Code in PHP (demo_sse.php):

```
<?php
header('Content-Type: text/event-stream');
header('Cache-Control: no-cache');

$time = date('r');
echo "data: The server time is: {$time}\n\n";
flush();
?>
```

Code in ASP (VB) (demo_sse.asp):

```
<%
Response.ContentType = "text/event-stream"
Response.Expires = -1
Response.Write("data: The server time is: " & now())
Response.Flush()
%>
```

Code explained:

- Set the "Content-Type" header to "text/event-stream"

- Specify that the page should not cache
- Output the data to send (**Always** start with "data: ")
- Flush the output data back to the web page

The EventSource Object

In the examples above we used the onmessage event to get messages. But other events are also available:

Events	Description
onopen	When a connection to the server is opened
onmessage	When a message is received
onerror	When an error occurs