Geolocation API

Quick Guides for Masterminds

J.D Gauchat
www.jdgauchat.com

Cover Illustration by **Patrice Garden**
www.smartcreativz.com

The content of this guide is a collection of excerpts from the book HTML5 for Masterminds. For more information, visit www.formasterminds.com.

The source code for this eBook is available at **www.formasterminds.com**

Copyright Registration Number: 1140725

1st Edition 2018

What is Inside

This guide will teach you how to use the Geolocation API to detect the user's location. After reading this guide, you will know how to get the user's current location, how to determine the changes in the location over time, and how to display the information on a map with the Google Maps API.

About this Guide

This guide is a collection of excerpts from the book HTML5 for Masterminds. The information included in this guide will help you understand a particular aspect of web development, but it will not teach you everything you need to know to develop a website or a web application. If you need a complete course on web development, read our book HTML5 for Masterminds. For more information, visit our website at www.formasterminds.com.

What Do You Need

This guide assumes that you have a basic knowledge of HTML, CSS and JavaScript, and you know how to create files and upload them to a server. If you don't know how to program in HTML, CSS or JavaScript, you can download our guides Introduction to HTML, Introduction to CSS, and Introduction to JavaScript. For a complete course on web development, read our book HTML5 for Masterminds.

> **IMPORTANT:** We recommend you to execute the examples in this guide with the latest versions of Google Chrome and Mozilla Firefox (www.google.com/chrome and www.mozilla.com). You can also check the state of the current implementations at www.caniuse.com. To find examples, resources, links and videos, visit our website at www.formasterminds.com.

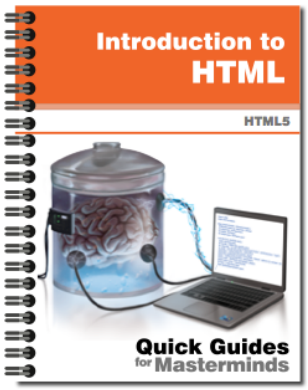
> **The Basics:** To create the files for the examples of this guide, you can use a text editor (Notepad or Text Edit), or a professional editor like Atom (www.atom.io). If you do not have a server to test the files, you can install a server in your computer with a package like MAMP (www.mamp.info). For more information, read our free guide Web Development.

Recommendations

Introduction to JavaScript
Quick Guides for Masterminds

This guide will teach you how to program with JavaScript. After reading this guide, you will know how to create a program in JavaScript, how to define functions and objects, and how to read and modify an HTML document dynamically.

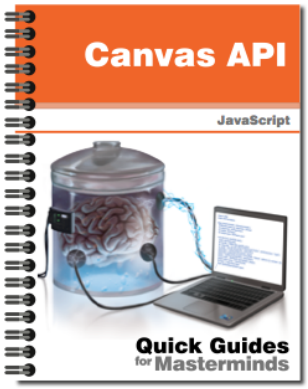
More Information



Introduction to HTML
Quick Guides for Masterminds

This guide will teach you how to create your website's documents with HTML. After reading this guide, you will know how to work with HTML elements, how to define a document's structure, and how to organize its content.

More Information



Canvas API
Quick Guides for Masterminds

This guide will teach you how to create graphic applications for your webiste with the Canvas API. After reading this guide, you will know how to create graphics for the web, how to manipulate images, and how to generate animations and small video games.

More Information

More Guides Available at www.formasterminds.com

Table of Contents

Geolocation API

Locations

The Geolocation API was designed to provide a standard detection mechanism for browsers to let developers determine the user's geographic location. Previously, we only had the option of building an extensive database with IP addresses and programming resource-consuming applications on the server that would only give us an approximate idea of the user's location (usually as vague as the country). This API takes advantage of new systems, such as network triangulation and GPS, to return an accurate location of the device running the application. The information returned may be used to create applications that adapt to the user's location or provide localized information automatically. Three methods are included in the API for this purpose.

- **getCurrentPosition(**location, error, configuration**)**—This method is used for single requests. It can take three attributes: a function to process the location returned, a function to process the errors returned, and an object to configure how the information will be acquired (only the first attribute is required).
- **watchPosition(**location, error, configuration**)**—This method is similar to the previous method except it constantly watches the location to detect and inform the application about changes. It works in a similar way than the **setInterval()** method, repeating the process automatically in a period of time according to the values set by default or the configuration specified by the attributes.
- **clearWatch(**id**)**—This method stops the process started by the **watchPosition()** method. This is similar to the **clearInterval()** method used to stop the process started by **setInterval()**.

The following is the document we are going to use in this guide. It is a simple document with a button inside a **<section>** element that we are going to use to show the information retrieved by the location system.

Listing 1: *Creating a document for testing the Geolocation API*

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title>Geolocation</title>
  <script src="geolocation.js"></script>
</head>
<body>
  <section id="location">
```

```
    <button type="button" id="getlocation">Get my
location</button>
  </section>
</body>
</html>
```

Getting the Location

As we mentioned above, we only need to assign one attribute to the **getCurrentPosition()** method to get a location. This attribute is a function that receives an object called **Position** that contains all the information retrieved by the geolocation systems. The **Position** object has two properties to store the values.

- **coords**—This property contains another object with a set of properties to provide the position of the device. The properties available are **latitude**, **longitude**, **altitude** (in meters), **accuracy** (in meters), **altitudeAccuracy** (in meters), **heading** (in degrees), and **speed** (in meters per second).
- **timestamp**—This property indicates the time when the information was acquired.

The API is defined in an object called **Geolocation**. The **Navigator** object includes the **geolocation** property to offer access to this object. To access the methods of this API, we have to call them from the **geolocation** property of the **navigator** property of the **Window** object, as in **navigator.geolocation.getCurrentPosition()**. The following example implements this method to get the user's current location.

Listing 2: *Getting the location*

```
function initiate() {
  var get = document.getElementById("getlocation");
  get.addEventListener("click", getlocation);
}
function getlocation() {
  navigator.geolocation.getCurrentPosition(showinfo);
}
function showinfo(position) {
  var location = document.getElementById("location");
  var data = "";
  data += "Latitude: " + position.coords.latitude + "<br>";
  data += "Longitude: " + position.coords.longitude +
"<br>";
  data += "Accuracy: " + position.coords.accuracy +
"mts.<br>";
  location.innerHTML = data;
}
```

```
window.addEventListener("load", initiate);
```

In the code in Listing 2, we add a listener for the **load** event to the window to execute a function called **initiate()** as soon as the document is loaded. When the event is fired, this function adds a listener for the **click** event to the button. In consequence, when the button is pressed, the **getCurrentPosition()** method is executed. To process the information produced by the **getCurrentPosition()** method, we define a function called **showinfo()**. When the method is called, a new **Position** object is created with the current information and sent to the **showinfo()** function. We reference that object inside the function with the **position** variable and then use this variable to show the data to the user.

The **Position** object has two important properties: **coords** and **timestamp**. In our example, we use **coords** to access the information we want (latitude, longitude, and accuracy). These values are stored in the **data** variable and then shown on the screen as the new content of the **location** element.

Do It Yourself: Create files with the codes in Listings 1 and 2 and open the document in your browser. When you click the button, the browser asks you whether or not to activate the location system for this application. If you allow the application to access this information, then your latitude, longitude and the accuracy of the data will be shown on the screen (it may take a few seconds to be available).

By adding a second attribute (another function) to the **getCurrentPosition()** method, we are able to capture errors produced in the process, like the error that occurs when the user denies access to the location system to our application.

Along with the **Position** object, the **getCurrentPosition()** method returns the **PositionError** object if an error is detected. The object has two properties, **error** and **message**, to provide the value and a description of the error. The three possible errors are represented by constants.

PERMISSION_DENIED—Value 1. This error occurs when the user denies the Geolocation API access to his or her location information.

POSITION_UNAVAILABLE—Value 2. This error occurs when the position of the device couldn't be determined.

TIMEOUT—Value 3. This error occurs when the position couldn't be determined in the period of time declared in the configuration.

If we want to report errors, we have to create a new function and assign it to the second parameter of the **getCurrentPosition()** method.

Listing 3: *Displaying error messages*

```
function initiate() {
```

```
  var get = document.getElementById("getlocation");
  get.addEventListener("click", getlocation);
}
function getlocation() {
  navigator.geolocation.getCurrentPosition(showinfo,
showerror);
}
function showinfo(position) {
  var location = document.getElementById("location");
  var data = "";
  data += "Latitude: " + position.coords.latitude + "<br>";
  data += "Longitude: " + position.coords.longitude +
"<br>";
  data += "Accuracy: " + position.coords.accuracy +
"mts.<br>";
  location.innerHTML = data;
}
function showerror(error) {
  alert("Error: " + error.code + " " + error.message);
}
window.addEventListener("load", initiate);
```

The error messages are intended for internal use. The purpose is to provide a mechanism for the application to acknowledge the situation and proceed accordingly. In the code in Listing 3, we add the second parameter to the **getCurrentPosition()** method (another function) and create this function, called **showerror()**, to show the values of the properties **code** and **message**. The value of **code** will be an integer between **0** and **3** according to the number of the error (listed above).

The **PositionError** object is sent to the **showerror()** function and is represented by the variable **error**. We could also check for the errors individually (**error.PERMISSION_DENIED**, for example) and show an alert only if that particular condition is **true**.

The third possible value of the **getCurrentPosition()** method is an object containing up to three properties.

- **enableHighAccuracy**—This is a Boolean property that informs the system that the most accurate location is required. When the value is set to **true**, the browser tries to get information through systems like GPS, for example, to provide the exact location of the device. These are high resource-consuming systems, and their use should be limited to specific circumstances. For this reason, the default value of this property is **false**.

timeout—This property indicates the maximum time for the operation to take place. If the information is not acquired at the deadline, the **TIMEOUT** error is returned. Its value is in milliseconds.

maximumAge—The previous locations are cached in the system. If we consider appropriate to get the last information stored instead of retrieving a new location (to avoid draining the battery or to get a quick response), this property can be set with a time limit. If the last cached location is older than the value of this property, then a new location will be retrieved from the system. Its value is in milliseconds.

The following code tries to get the most accurate location of the device in no more than 10 seconds, but only if there is no previous location in the cache captured less than 60 seconds ago (if there is a previous location, that will be the information returned in the **Position** object).

Listing 4: *Configuring the Geolocation system*

```
function initiate() {
  var get = document.getElementById("getlocation");
  get.addEventListener("click", getlocation);
}
function getlocation() {
  var geoconfig = {
    enableHighAccuracy: true,
    timeout: 10000,
    maximumAge: 60000
  };
  navigator.geolocation.getCurrentPosition(showinfo,
showerror, geoconfig);
}
function showinfo(position) {
  var location = document.getElementById("location");
  var data = "";
  data += "Latitude: " + position.coords.latitude + "<br>";
  data += "Longitude: " + position.coords.longitude +
"<br>";
  data += "Accuracy: " + position.coords.accuracy +
"mts.<br>";
  location.innerHTML = data;
}
function showerror(error) {
  alert("Error: " + error.code + " " + error.message);
}
window.addEventListener("load", initiate);
```

In our example, the object containing the configuration values is stored in the **geoconfig** variable, and this variable is declared as the third attribute of the **getCurrentPosition()** method. Nothing else changed in the rest of the code from the previous example. The **showinfo()** function will show the information on the screen independently of its origin (i.e., if it is cached or new).

From this code, we can see the real purpose of the Geolocation API and what the API was intended for. The most useful features are geared towards mobile devices. For instance, the value **true** for the **enableHighAccuracy** attribute suggests the browser to use systems like GPS to get the most accurate location, which are only available in these types of devices. Also, the methods **watchPosition()** and **clearWatch()**, which we will see next, work with location updates, and this is only useful, of course, when the device running the application is mobile (and it is moving). This brings up two important issues. First, most of our codes will have to be tested in a mobile device to know exactly how they will perform in a real situation. And second, we have to be responsible when using this API. GPS and other location systems consume resources, and in most cases, devices will run out of battery if we are not careful.

Watching the Location

Similar to **getCurrentPosition()**, the **watchPosition()** method takes three attributes and performs the same task: to get the location of the device that is accessing the application. The only difference is that the **getCurrentPosition()** method performs a one-time operation while **watchPosition()** automatically offers new data every time the location changes. The method is watching all the time and sending information to a function when there is a new location to show until we cancel the process with the **clearWatch()** method.

The **watchPosition()** method is implemented the same way than the **getCurrentPosition()** method, as illustrated in the following example.

Listing 5: *Testing the* watchPosition() *method*

```
function initiate() {
  var get = document.getElementById("getlocation");
  get.addEventListener("click", getlocation);
}
function getlocation() {
  var geoconfig = {
    enableHighAccuracy: true,
    maximumAge: 60000
  };
  control = navigator.geolocation.watchPosition(showinfo,
showerror, geoconfig);
}
function showinfo(position) {
```

```
  var location = document.getElementById("location");
  var data = "";
  data += "Latitude: " + position.coords.latitude + "<br>";
  data += "Longitude: " + position.coords.longitude +
"<br>";
  data += "Accuracy: " + position.coords.accuracy +
"mts.<br>";
  location.innerHTML = data;
}
function showerror(error) {
  alert("Error: " + error.code + " " + error.message);
}
window.addEventListener("load", initiate);
```

We will not notice any change running this example on a static desktop computer, but on a mobile device, new information will be shown every time there is a change in the device's location. How often the information is sent to the **showinfo()** function is determined by the **maximumAge** property. If the new location is retrieved 60 seconds (60000 milliseconds) after the previous one, it is shown; otherwise, the **showinfo()** function will not be called.

Notice that the value returned by the **watchPosition()** method is stored in the **control** variable. This variable is like the id of this operation. If later we want to cancel the process, we have to execute the instruction **clearWatch(control)** and **watchPosition()** will stop updating the information.

Google Maps

In the previous examples, we have shown the location data on the screen exactly the way we receive it. However, these values usually mean nothing to regular people. We cannot immediately state our current locations' latitude and longitude, let alone identify a location in the world from these values. We have two alternatives: use this information internally to calculate position, distance and other variables that let us offer concrete results to the users (such as products or restaurants in the area) or directly show the information retrieved by the Geolocation API in a more comprehensible way, like a map.

A useful API that we can combine with the Geolocation API to achieve this purpose is the Google Maps API. This is an external JavaScript API provided by Google that has nothing to do with HTML5 but is used in modern websites and applications. It offers a variety of ways to work with interactive maps and even real views of locations through the StreetView technology.

The following is a simple example that applies one part of the API called *Static Maps API*. With this API, we can build a URL with the information of the location, and an image of the selected area in a map is returned.

Listing 6: *Representing the location on a map using the Google Maps API*

```
function initiate() {
  var get = document.getElementById("getlocation");
  get.addEventListener("click", getlocation);
}
function getlocation() {
  navigator.geolocation.getCurrentPosition(showinfo,
showerror);
}
function showinfo(position) {
  var location = document.getElementById("location");
  var mapurl =
"http://maps.google.com/maps/api/staticmap?center=" +
position.coords.latitude + "," + position.coords.longitude
+ "&zoom=12&size=400x400&sensor=false&markers=" +
position.coords.latitude + "," + position.coords.longitude;

  location.innerHTML = '<img src="' + mapurl + '">';
}
function showerror(error) {
  alert("Error: " + error.code + " " + error.message);
}
window.addEventListener("load", initiate);
```

The application is simple. We use the **getCurrentPosition()** method and send the information to the **showinfo()** function as usual, but now in this function the values of the **Position** object are added to a Google URL, and then the address is inserted as the source of an **<img>** element to show the image returned on the screen.

Figure 1: *Image returned by the Google Maps API*

Do It Yourself: Test the code of Listing 6 in your browser using the document of Listing 1. You should see a map with your location on the screen. Change the values of the attributes **zoom** and **size** in the URL to modify the map returned by the API. Go

to the Google Maps API's web page to find other alternatives: https://developers.google.com/maps/.

Quick Reference

Methods

getCurrentPosition(location, error, configuration**)**—This method is used for single requests. It can take three attributes: a function to process the location returned, a function to process the errors returned, and an object to configure how the information will be acquired (only the first attribute is required).

watchPosition(location, error, configuration**)**—This method is similar to the previous method except it constantly watches the location to detect and inform the application about changes. It works in a similar way than the **setInterval()** method, repeating the process automatically in a period of time according to the values set by default or the configuration specified by the attributes.

clearWatch(id**)**—This method stops the process started by the **watchPosition()** method. This is similar to the **clearInterval()** method used to stop the process started by **setInterval()**.

Properties

coords—This property contains another object with a set of properties to provide the position of the device. The properties available are **latitude**, **longitude**, **altitude** (in meters), **accuracy** (in meters), **altitudeAccuracy** (in meters), **heading** (in degrees), and **speed** (in meters per second).

timestamp—This property indicates the time when the information was acquired.

enableHighAccuracy—This is a Boolean property that informs the system that the most accurate location is required. When the value is set to **true**, the browser tries to get information through systems like GPS, for example, to provide the exact location of the device. These are high resource-consuming systems, and their use should be limited to specific circumstances. For this reason, the default value of this property is **false**.

timeout—This property indicates the maximum time for the operation to take place. If the information is not acquired at the deadline, the **TIMEOUT** error is returned. Its value is in milliseconds.

maximumAge—The previous locations are cached in the system. If we consider appropriate to get the last information stored instead of retrieving a new location (to avoid draining the battery or to get a quick response), this property can be set with a time limit. If the last cached location is older than the value of this property, then a new location will be retrieved from the system. Its value is in milliseconds.

For Masterminds

Book Series

for more Books and Quick Guides visit

www.formasterminds.com