

Developing Mobile Websites

GeoLocation API

Lesson 1, Activity 2: The GeoLocation API

One can easily see the value of location-aware Web sites; increasingly, we rely on our phones to find the nearest gas station, restaurant, or shopping center. As we move through this lesson, we will explore how location-aware code might enhance the user experience for visitors to our *Jazz Calendar* and *Pickup Soccer* sites.

The World Wide Web Consortium (W3C) has worked to standardize a system by which developers can retrieve a user's location. Based in part on Google Gears, the API is now widely supported (see below for specifics) and offers a device-independent system for accessing the user's latitude and longitude via JavaScript.

How It Works

The GeoLocation API can find a user's location from a variety of information sources, using the best available. The W3C specification states:

The Geolocation API defines a high-level interface to location information associated only with the device hosting the implementation, such as latitude and longitude. The API itself is agnostic of the underlying location information sources. Common sources of location information include Global Positioning System (GPS) and location inferred from network signals such as IP address, RFID, WiFi and Bluetooth MAC addresses, and GSM/CDMA cell IDs, as well as user input. [From dev.w3.org/geo/api/spec-source.html#introduction]

If GPS is available, it will be used (though some devices, because of the delay and high energy use of GPS, opt not to use GPS). If GPS is not available, A-GPS (Assistive GPS, triangulated from cell phone towers), WiFi, IP, or another less accurate means of locating the user will be used.

Browser & Device Support

The W3C Geolocation API works for the following desktop and mobile browsers:

Geolocation API Supported Browsers

Platform	Browsers				
desktop	Firefox from version 3.5	Opera 10.6	Google Chrome	Internet Explorer 9.0	Safari 5
mobile	Android (firmware 2.0 +)	iOS	Windows Phone		

JavaScript Implementation

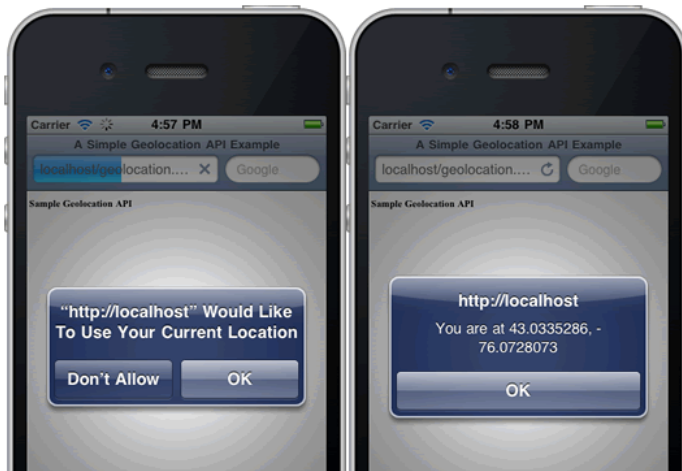
For devices that support the GeoLocation API, no external JavaScript file needs to be linked - the JavaScript functionality is inherent in the browser itself. The core of the code is the `geolocation` object, a child of the `navigator` object:

```
if (navigator.geolocation) {
    navigator.geolocation.getCurrentPosition(successFunction, failureFunction, {enableHighAccuracy:false, maximumAge:30000, timeout:27000});
}
```

The `if` statement tests if the `navigator.geolocation` object exists on this device/browser. If so, we call the object's `getCurrentPosition` function - the first two parameters refer to a function (which we've called `successFunction`, but we could have used any name) to call when `getCurrentPosition` is successful in finding the user's location, and a function (`failureFunction`, in this example) to call when `getCurrentPosition` was not successful in finding the user's location; this parameter is optional. Both `failureFunction` and `successFunction` are functions we will write to handle the success and failure cases. The third parameter, a JSON-formatted hash of options, allows fine control over implementation details; we will omit this optional parameter when calling `getCurrentPosition`.

A Simple Example

Let's look at a simple page (GeoLocationAPI/Demos/findmylocation.html) that pops up a JavaScript message with the user's latitude and longitude location. Users who visit our page will be prompted to explicitly allow our page to use their current location; see the left screen shot below. If they accept, our page will respond with their latitude and longitude, as on the right.



Code Sample:

[GeoLocationAPI/Demos/findmylocation.html](#)

```
---- CODE OMITTED ----
```

```
---- CODE OMITTED ----
```

We test, with an if statement, for the presence of the `navigator.geolocation` object; if so, we call the `getCurrentPosition` function. We write two functions, `locateSuccess` and `locateFail`, to handle the success and failure cases, respectively.

On successful finding of the user's location, function `locateSuccess` is invoked. Note that, as discussed above, we can name this function anything we like (though, of course, a meaningful name containing "success" makes sense here), as long as the name of the function is the same name we use for the first parameter of the `getCurrentPosition` function. The `loc` parameter exposes a variety of information about the user's location; we assign `loc.coords.latitude` and `loc.coords.longitude` to two local variables, and show those values to the user in a JavaScript alert (popup) window.

If the user's location is not found, function `locateFail` is invoked. We use a JavaScript switch statement to evaluate parameter `geoPositionError`'s `code` property: if 0, something unknown happened; if 1, the user denied permission; etc.

Of course, we could extend this application extensively: we might use Google's [Places](#) library to allow the user to perform a search for "Jazz Clubs" and to return human-friendly addresses (rather than latitude and longitude). Check out the Google Maps [Demo Gallery](#) for some inspiration and code samples.

The parameter passed to the success function, which we named `loc` in our example above, exposes a useful set of properties:

Position Properties

Property	Units
<code>coords.latitude</code>	degrees
<code>coords.longitude</code>	degrees
<code>coords.altitude</code>	meters (may be null)
<code>coords.accuracy</code>	meters
<code>coords.altitudeAccuracy</code>	meters (may be null)
<code>coords.heading</code>	degrees clockwise (may be null)
<code>coords.speed</code>	meters/second (may be null)
<code>timestamp</code>	a date and time

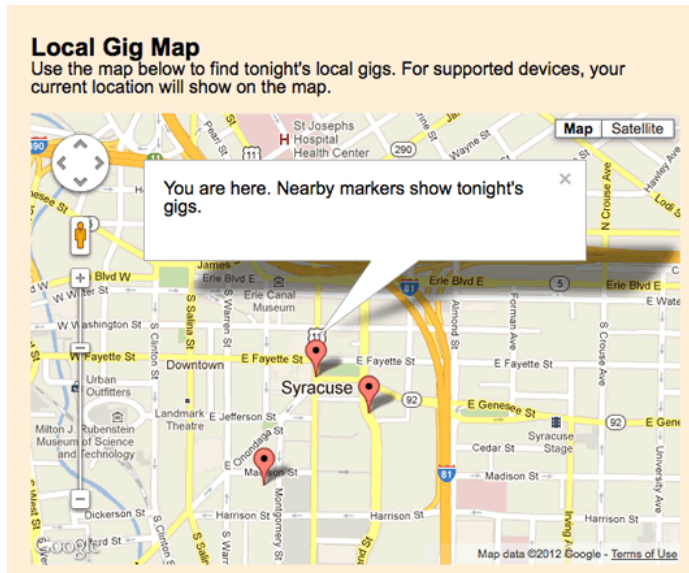
Lesson 1, Activity 4: Google Maps

In the demonstrations below and the next few exercises, we'll make use of the Google Maps API. See the Google Developers [Maps page](#) for lots more information about what one can do to integrate maps into your pages.

Jazz Calendar: Finding Nearby Gigs

Let's build a page for the Jazz Calendar site allowing users to find upcoming concerts near their current location - folks can browse to the page on their smartphone (or desktop computer) and, if they allow their device to send their location, see map pins representing nearby gigs. Of course, we won't actually be listing real gigs; we'll mock this up by creating example locations close to them.

Here's a screen shot of the page:



The following JavaScript code, from [GeoLocationAPI/Demos/jazzcalmap.html](#), accomplishes our goal:

```
function locateSuccess(loc) {
    latitude = loc.coords.latitude;
    longitude = loc.coords.longitude;
    var coords = new google.maps.LatLng(latitude, longitude);

    //create two fake gig locations, near to the user's current location:
    var fakegig1_coords = new google.maps.LatLng(latitude - 0.001, longitude + 0.002);
    var fakegig2_coords = new google.maps.LatLng(latitude - 0.003, longitude - 0.002);

    //set options for the Google Map:
    var options = {
        zoom: 15,
        center: coords,
        mapTypeId: google.maps.MapTypeId.ROADMAP
    };

    //create the Google Map:
    var map = new google.maps.Map(document.getElementById("mapcontainer"), options);

    //create display text for the map popup window and the popup window
    var infoWindowText = "You are here. Nearby markers show tonight's gigs.";
    var infowindow = new google.maps.InfoWindow({
        content: infoWindowText
    });

    //create a marker for the user's current position
    var marker = new google.maps.Marker({
        position: coords,
        map: map
    });

    //create markers for the two gigs
    var fakegig1_marker = new google.maps.Marker({
        position: fakegig1_coords,
        map: map
    });
}
```

```

var fakegig2_marker = new google.maps.Marker({
  position: fakegig2_coords,
  map: map
});

//enable clicking on the current-location marker
google.maps.event.addListener(marker, 'click', function() {
  infowindow.open(map,marker);
});

//popup the current-location window when the page loads
google.maps.event.trigger(marker, 'click');
}

```

We use the now-familiar `if (navigator.geolocation)` to test if the user's device supports the GeoLocation API; if so, function `locateSuccess` will be invoked.

We find the user's current latitude and longitude position with `loc.coords.latitude` and `loc.coords.longitude`).

The variables `fakegig1_coords` and `fakegig2_coords` represent the nearby local concerts - we set locations for these by adding or subtracting a little distance from the user's actual location. Of course, in a real application, these locations would be generated from some kind of server-side, database-enabled code (via PHP, ColdFusion, or other, similar, technology), but this example will suffice for our demo now.

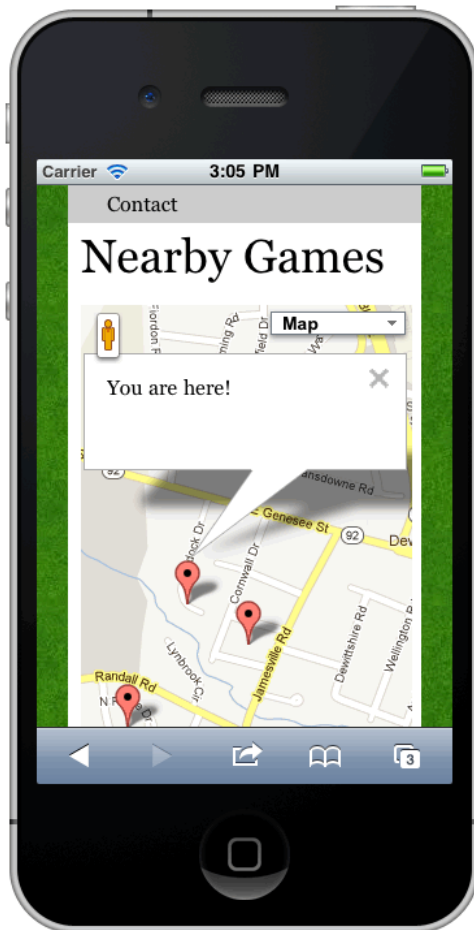
Note that the page ([GeoLocationAPI/Demos/jazzcalmap.html](http://maps.google.com/maps/api/js?sensor=true)) includes a Google Maps JavaScript file (<http://maps.google.com/maps/api/js?sensor=true>) in the headsection. The rest of the JavaScript code sets up and displays the Google Map: we set the zoom level and other options, create the map, create display text and show on a popup window at the user's current location, and create markers to the user's current position and the two fake gig locations.

Note, in particular, that we add the Google Map to the `#mapcontainer` div, located (in our markup) before the JavaScript map code: the statement `new google.maps.Map(document.getElementById("mapcontainer"), options)` places the map in that div. In the associated CSS file ([GeoLocationAPI/Demos/css/jazzcalmap.css](#)), we've given the `#mapcontainer` div a width of 100% and height of 400px - this ensures both that it shows up (because it has some guaranteed height) and a flexible width (to scale with its container element).

Lesson 1, Activity 6: Adding a Location-Aware Page to the *Pickup Soccer Site*

Duration: 20 to 30 minutes.

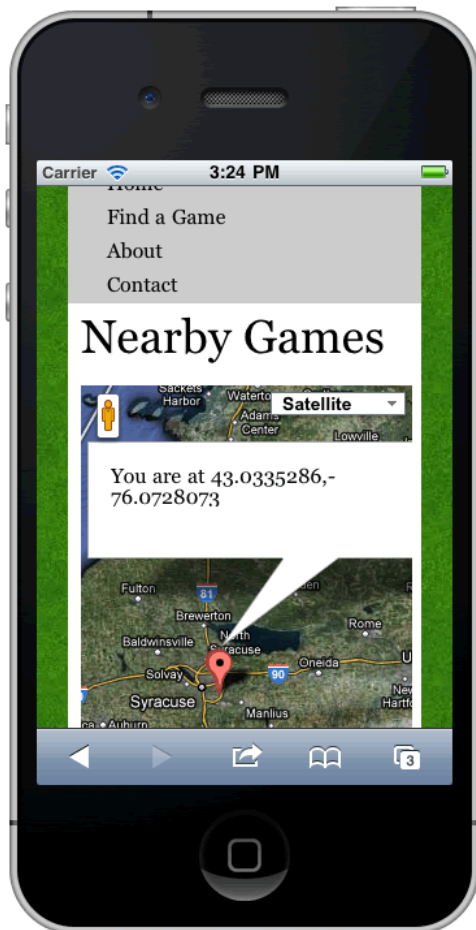
In this exercise, you will use the GeoLocation API to offer visitors to the Pickup Soccer site a way to find pickup games near their current location, as shown in the screen shot:



1. Open [ClassFiles/GeoLocationAPI/Exercises/findgames/index.html](#) in your file editor.
2. Using the code outline given in the exercise file, fill in the missing JavaScript code to find the user's current location.
3. Using the code outline given and referencing the previous example code, fill in the missing JavaScript code to display a Google Map with the user's current location.
4. Write JavaScript code to set locations for at least two sample nearby games; use the same strategy as above: add or subtract a few decimal places' worth of latitude and longitude to the user's current location.
5. Place these sample nearby games on the map.

Challenge

Explore the various Google Map display options and display the user's current location in the info window:



1. Change the map type from ROADMAP to some other option - see the [Map Options](#) section of the Google Maps "Getting Started Guide" for possible values and more information.
2. Display the user's current location in the info window, as shown above.
3. Change the zoom level to display a wider area.

Solution:

[GeoLocationAPI/Solutions/findgames/index.html](#)

```
<!DOCTYPE html>
<html>
<head>
  <title>Soccer Pickup</title>
  <link rel="stylesheet" type="text/css" href="css/reset.css" />
  <link rel="stylesheet" type="text/css" href="css/style.css" />
  <!-- include the Google Maps Javascript file -->
  <script type="text/javascript" src="http://maps.google.com/maps/api/js?sensor=true"></script>
  <meta name="viewport" content="initial-scale=1.0, width=device-width" />
</head>

<body>

---- CODE OMITTED ----

<div id="mapcontainer"></div>
<script type="text/javascript">

  //test whether the user's device supports the GeoLocation API; call getCurrentPosition if so
  if (navigator.geolocation) {
    navigator.geolocation.getCurrentPosition(locateSuccess, locateFail);
  }

  function locateSuccess(loc) {
    //create variables to represent the user's lat and long position
    latitude = loc.coords.latitude;
    longitude = loc.coords.longitude;
    var coords = new google.maps.LatLng(latitude, longitude);
```

```

//create two fake pickup game locations, near to the user's current location:
var fakegame1_coords = new google.maps.LatLng(latitude - 0.001, longitude + 0.002);
var fakegame2_coords = new google.maps.LatLng(latitude - 0.003, longitude - 0.002);

//set options for the Google Map:
var options = {
  zoom: 15,
  center: coords,
  mapTypeId: google.maps.MapTypeId.ROADMAP
};

//create the Google Map:
var map = new google.maps.Map(document.getElementById("mapcontainer"), options);

//create display text for the map popup window and the popup window
var infoWindowText = "<p>You are here!</p>";
var infowindow = new google.maps.InfoWindow({
  content: infoWindowText
});

//create a marker for the user's current position
var marker = new google.maps.Marker({
  position: coords,
  map: map
});

//create markers for the two games
var fakegame1_marker = new google.maps.Marker({
  position: fakegame1_coords,
  map: map
});
var fakegame2_marker = new google.maps.Marker({
  position: fakegame2_coords,
  map: map
});

//enable clicking on the current-location marker
google.maps.event.addListener(marker, 'click', function() {
  infowindow.open(map,marker);
});

//popup the current-location window when the page loads
google.maps.event.trigger(marker, 'click');

}
function locateFail(geoPositionError) {
  //create an alert message when locating the user fails
  alert('An unknown error occurred, sorry');
}
</script>

---- C O D E   O M I T T E D ----

```

We include the Google Maps JavaScript file in the head of the document. We create an empty div (with id mapcontainer), in the body of the page, to hold our map.

We write JavaScript to test whether we can find their location (if (navigator.geolocation)) on their current device. If so, we call if (navigator.geolocation) to get the location.

Function locateSuccess handles the success case, where we have found their location. We create a new Google LatLng object from their location; two more LatLng objects represent the nearby games.

We set options for and create the map, set up the text for and add the InfoWindow, and create markers for the user's position and the two games. Last, we enable clicking on the InfoWindow and ensure (with google.maps.event.trigger(marker, 'click')) that the InfoWindow pops up when the page loads.

Challenge Solution:

GeoLocationAPI/Solutions/findgames/challenge.html

```

---- C O D E   O M I T T E D ----

```

```

function locateSuccess(loc) {

```



```

latitude = loc.coords.latitude;
longitude = loc.coords.longitude;
var coords = new google.maps.LatLng(latitude, longitude);

//create two fake pickup game locations, near to the user's current location:
var fakegame1_coords = new google.maps.LatLng(latitude + (Math.random()/100), longitude - (Math.random()/100));
var fakegame2_coords = new google.maps.LatLng(latitude - (Math.random()/100), longitude + (Math.random()/100));

//set options for the Google Map:
var options = {
  zoom: 8,
  center: coords,
  mapTypeId: google.maps.MapTypeId.HYBRID
};

//create the Google Map:
var map = new google.maps.Map(document.getElementById("mapcontainer"), options);

//create display text for the map popup window and the popup window
var infoWindowText = "<p>You are at "+latitude+", "+longitude+"</p>";
var infowindow = new google.maps.InfoWindow({
  content: infoWindowText
});

//create a marker for the user's current position
var marker = new google.maps.Marker({
  position: coords,
  map: map
});

//create markers for the two games
var fakegame1_marker = new google.maps.Marker({
  position: fakegame1_coords,
  map: map
});
var fakegame2_marker = new google.maps.Marker({
  position: fakegame2_coords,
  map: map
});

//enable clicking on the current-location marker
google.maps.event.addListener(marker, 'click', function() {
  infowindow.open(map,marker);
});

//popup the current-location window when the page loads
google.maps.event.trigger(marker, 'click');

}
---- C O D E   O M I T T E D ----

```

We change the zoom level to a lower number, to show a wider geographic area. Setting `mapTypeId: google.maps.MapTypeId.HYBRID` presents the map as a hybrid roadmap, with both aerial photography and road features. Last, `infoWindowText = "You are at "+latitude+", "+longitude+""` shows the user's latitude/longitude location in the `InfoWindow`, using JavaScript's + string concatenation operator.