# Further Web Mapping

## Introduction

Today we will look at some additional web mapping topics, specifically:

- Geolocation API
- Displaying markers from a web service

## The Geolocation API

### Introduction

The Geolocation API allows you to obtain the current location of the device running the browser from within JavaScript. Even on desktop browsers this will give a result if you are using a wireless network, but its real use is in the mobile web development world. A mobile browser can talk to the GPS chip of the phone and obtain the phone's current location on the earth. Alternatively, if the GPS chip is not available, a rough estimate can be obtained from cell towers or wireless networks.

#### Using the Geolocation API

It is fairly straightforward to use the Geolocation API. Here is an example:

```
<html>
<head>
<title>Geolocation Test</title>
<script type='text/javascript'>

function init()
{
    if(navigator.geolocation)
    {
        navigator.geolocation.getCurrentPosition (proce
    }
    else
    {
        alert("Sorry, geolocation not supported in this
    }
}
```

```
function processPosition(gpspos)
{
    var info = 'Lat: ' + gpspos.coords.latitude + ' lor
    document.getElementById('info').innerHTML = info;
}

function handleError(err)
{
    alert('An error occurred: ' + err.code);
}

</script>
</head>
<body onload='init()'>
<h1>Geolocation Test</h1>
<div id='info'></div>
</body>
</html>
```

How is this working?

- When the page first loads, the ***init()*** function runs. This tests whether geolocation is possible in our browser by checking for the existence of the ***navigator.geolocation*** variable. If it doesn't exist, we display an error. If it ***does*** exist, we tell the geolocation object to obtain the current position with ***getCurrentPosition()***. This takes two parameters, both functions:
    - The first (here, ***processPosition()***) is the function that will run as soon as we get a location back from the GPS chip (or wireless network, etc)
    - The second (here, ***handleError()***) will run if there was an error obtaining the location.
- Note the ***asynchronous*** approach here, just like AJAX. It might take time to obtain a position, so we set up a ***callback*** function to run once we have obtained the position, and continue processing in the meantime. So any statements below the ***getCurrentPosition()*** might run before the callback ***processPosition()*** starts running.
- Considering each of these two functions in turn, let's look at ***processPosition()*** first. This is automatically supplied with one parameter, ***gpspos***, representing the position returned from the GPS device or other location provider. We can obtain the latitude and longitude with ***gpspos.coords.latitude*** and ***gpspos.coords.longitude***. So what this example is doing is placing the coordinates inside the  with the ID of ***info***.

- Moving onto ***handleError()*** which is the function which is run if there is any sort of error in obtaining the position: again, this is automatically supplied with one parameter, ***err***, representing the exact error which occurred. The most meaningful property of ***err*** is a numerical code, ***err.code***, which is what we are displaying here. There are several codes (ref Dive into HTML5, Mark Pilgrim):
  - 1 means that the user denied the browser access to the device's location. For security reasons, a user has to agree to the browser accessing the device's GPS chip, and if they deny that access, the error function will run with a code of 1.
  - 2 means that the location could not be obtained, e.g the GPS satellites or wireless networks are unavailable;
  - 3 means a timeout, i.e. the GPS satellites or network are available but it has taken too long to contact them.

## Watching the position

The above code will simply obtain the current position and stop. In a typical mobile GPS application, however, the user will want to be informed of their location on a regular basis. To do this we use ***watchPosition()*** in place of ***getCurrentPosition()***. Here is an example:

```
<html>
<head>
<title>Geolocation Test</title>
<script type='text/javascript'>

function init()
{
    if(navigator.geolocation)
    {
        navigator.geolocation.watchPosition (processPos
                    {enableHighAccuracy:true, maximumAg
                                        );
    }
    else
    {
        alert("Sorry, geolocation not supported in this
    }
}

function processPosition(gpspos)
{
    var info = 'Lat: ' + gpspos.coords.latitude + ' lor
    document.getElementById('info').innerHTML = info;
}
```

```
function handleError(err)
{
    alert('An error occurred: ' + err.code);
}

</script>
</head>
<body onload='init()'>
<h1>Geolocation Test - Watching Position</h1>
<div id='info'></div>
</body>
</html>
```

Note that the code is almost the same as the first example, except:

- We use ***watchPosition()*** rather than ***getCurrentLocation()***. The result will be that the GPS chip (or other location provider) will communicate the current location back to ***processPosition()*** every time the location changes, rather than just the once.
- Note also the additional options passed as a third parameter to ***watchPosition()***. On many mobile devices, if we want to use the GPS chip (rather than cell towers, wireless networks, etc) we have to set the ***enableHighAccuracy*** option to ***true***. Also if we want to force the GPS to return a reading every so-many seconds, we have to specify a value for the ***maximumAge*** option, in milliseconds. So the above example will refresh at least every 5 seconds.

## IMPORTANT - Geolocation API now needs HTTPS server

For security reasons (the risk of your location being intercepted by packet-sniffers) the Geolocation API now (recently at the time of writing) requires the use of an HTTPS server (a web server with encrypted communication), on both Chrome and Firefox. The internal servers (Edward, Edward2, Neptune) are unfortunately not yet setup for HTTPS (the Edward2 replacement, due in January, will be); however if you have your own hosting they probably will be, or you will be able to setup HTTPS yourself.

However, if you load the web page locally (directly from the U: drive or similar rather than from a web server) it will work. For that reason:

- Do exercise 1, below, locally - do not upload to Edward or Edward2.
- As a result of the recent changes in browsers and due to the lack of HTTPS availability, **it is not necessary to use the Geolocation API in the coding assignment. However you should still talk about it in the report.**

See Let's Encrypt for information on setting up an HTTPS server.

### Exercise 1 - Geolocation

Do this ***locally***, not on the server - see the above notes on HTTPS.

- Try out the first geolocation example above, to show that it works (it should work, using the university's wifi network).
- Use the Geolocation API to obtain your current position, and display a Leaflet map at zoom 14 centred on that position. How you should approach this is as follows:
  - Add the Leaflet setup code to your init() function for the basic geolocation example above. Set the map's location initially to an arbitrary location e.g. latitude 51, longitude -1.
  - In your ***processPosition()*** method, adjust the map to the new position received. Note that you will need to make your Leaflet ***map*** variable a ***global variable*** (declare it outside any function) so that both your init() function and your processPosition() function can access it.

  You might notice, through reading the Leaflet API, that there is a shortcut way of doing this.... but for practise, use the above method!

## Displaying markers from a web service

A common use of web mapping is to show data from a web service on a map. For example, we might want to show the location of restaurants in a particular city on a web map. This is quite easy - you simply need to make an AJAX request to a web service, parse the JSON (or XML) returned and create markers using the data. The following exercise will allow you to explore this.

### Exercise 2 - Connecting to a web service and displaying markers

In the ***dftitutorials*** database on edward2 is a table called ***artists*** which gives the latitude and longitude of the home towns of selected artists (Oasis, Madonna, the Beatles, David Bowie, Prince, Michael Jackson). There is also a web service at ***http://edward2.solent.ac.uk/wad/artists.php*** which takes an artist as a query string parameter called ***artist*** and returns JSON describing the home town of that artist and the latitude and longitude of that home town.

Modify your HitTastic! AJAX page (Session 7) so that as well as a "Search" button, there is a "Where is this artist from?" button. This should connect to a JavaScript function which sends an AJAX request to this web service. Also write an ***init()*** function to initialise a Leaflet map, as above.

In the callback, parse the JSON returned so that a marker is shown on the map for that artist, and the map is centred at the artist's hometown. When the marker is clicked, the name of the home town of the artist should be shown in a popup.

If no query string parameters are passed to the web service, the home towns of ***all*** the artists are shown. Add a new button to the HitTastic! AJAX page labelled "Show Home Towns Of All Artists". By making an AJAX request to the web service, it should show the location of the home towns of ***all*** the artists on the map, along with a popup giving information about the artist and their home town.

## Further topic - GeoJSON

If you complete all the above, you might want to read these further notes on GeoJSON - a common standard format for representing geographical data.