Where am I: Now and Then
Geolocation, Google Maps API, and localStorage

By Jeanine Meyer  jeanine.meyer@purchase.edu   http://faculty.purchase.edu/jeanine.meyer

In this article I will explain how to use HTML5, JavaScript and geolocation to determine the location of the visitor to your site; bring in a map, using the Google Maps API; and store the location on the client computer to be accessed when the application is opened again.

The calculation and use of location information is an exciting, though also controversial, topic today. I mostly will stay away from the controversy and simply describe the mechanics. I created a simple HTML5 JavaScript program that makes a determination of the location of the client computer and brings in a Google Map with the location marked. Figure 1 shows the opening screen in Firefox requesting permission to track location. Applications are not supposed to use location information without the user's granting permission.

The look of the permission request and exactly what is being agreed to vary across the browsers. For example, in Firefox, you can simply click on Share Location, or use the downward pointing arrow to choose from among "Always Share", "Never Share" and "Not Now". I clicked "Share Location" and the screen in Figure 2 appeared. You do need to be careful when granting permissions on computers or other devices, especially granting a blanket permission since it can be tedious to rescind the permission.

The application does not provide an image of a map, but a portal to all Google Maps features, including panning, zooming in or out, and changing to Satellite view. Note: this map is zoomed in all the way. If you zoom out, you also can choose the Terrain view.

My application provides the ability to reset what I term the now location by clicking on the map. This may be appropriate if the geolocation determination is not accurate enough (more on that later). In all cases, the application stores the location either as determined by geolocation or as changed by the user, on the client computer. This information remains on the computer. When the application is re-loaded or the browser closed and re-started or the whole computer shutdown and then started up again another day and the SAME browser started perhaps at a different geographic location, both the old and the new location are shown. I call these positions then and now and if I move the cursor over each of the tiny red x's I've used as markers, the cursor changes to a hand pointing and a title appears as shown in Figure 3.
Figure 3: Two markers on the map, with the title of one indicating then

I now will describe the mechanics of building the application after a brief refresher on the coordinate system used for these maps and background on the determination of physical location.

**Latitude and longitude**

An application featuring geography needs a system for specifying locations. Since my program makes use of the Google Maps Application Programming Interface (API), I use what it uses, namely the standard latitude and longitude. Envision the globe. Latitudes are circles parallel to the equator and they are called parallels. Latitude is specified as an angle, with the equator being 0 and the dot representing the North pole (think of it as a circle with radius zero) as 90. The location representing the South pole has latitude equal to -90. The definition of longitude is arbitrary, but well-fixed in tradition. Longitudes are perpendicular to latitudes. What is arbitrary is the location of the longitude (aka meridians) with value zero. Tradition puts it as going through the town of Greenwich in England. This longitude is also termed the Prime Meridian. Values are positive going east of Greenwich and negative going west. Values are given in degrees, generally using 4 decimal points. Mt. Kisco, NY is at latitude 41.2067 and longitude -73.7272.

**Note:** You will see values given in degrees, minutes and seconds and with designations of North, South, East or West. Looking up Mt. Kisco on www.worldatlas.com produces: 41° 12' N / 73° 43' W

You certainly do not have to memorize values of latitude and longitude, but it is important to appreciate the scale and basic orientation of the units of the coordinate system. A degree of latitude or longitude covers a lot of ground! I advise looking up the coordinates for different places to get a sense of how the units work. For example, Youngsville, NC, where the editor of jsmag resides, is at 36.0279, -78.4748.

**Background on geolocation**

Geolocation is a feature in which the geographic (real-world) location of the user, the person on the client computer or other device, is calculated. Geolocation is not part of HTML or HTML5, but is its own standard. It also is distinct from Google Maps, though in this article, I will describe the JavaScript / Google Maps API implementation for geolocation.

How can software, such as available through the Google Maps API, determine a location? There are several possibilities at this time and there may be more in the future. The Internet Service Provider (ISP) has information on the geographic location of the IP address. This is not as accurate as the other methods. The software can perform send and receive operations to determine nearby cell phone tower and then do a calculation, called triangulation, to determine the location. Alternatively, the software can perform similar operations and calculations on WI-FI hot spots or GPS satellites. These last three may produce 3D locations, that is latitude, longitude plus altitude. They are dependent on the type of device. My desktop computer, which also is the hub of my home WI-FI network, only uses the IP address. However, when I or anyone else uses a laptop in my house, the WI-FI hot spot method is used and if the device is a smart phone, then it will be the cell phone tower or the WI-FI hot spot. Results on these other devices will be more accurate than on my desktop computer.

**Note:** It was when traveling around picking up the location of WI-FI stations that private information was collected by Google. Even if I installed a password on the WI-FI network in my house, the Street View van still recorded the information on its location, making that information available for geolocation computation.

As you may guess, the geolocation operation is not instantaneous and it may fail. The programmer can choose the degree of accuracy, with more accuracy potentially taking more time and more battery life. These factors mean programming a request is done by setting up what is termed an asynchronous operation. You can do the research to find out more about accuracy, setting time limits and variations of geolocation. The basic code for acquiring location information is shown in Listing 1.

```javascript
var positionopt; positionopt = {
    enableHighAccuracy: true
};
if (navigator.geolocation) {
    navigator.geolocation.getCurrentPosition(handler, problemhandler, positionopt);
} else {
    if (document.getElementById("place")) {
        document.getElementById("place").innerHTML = "I'm sorry but geolocation services are not supported by your browser";
        document.getElementById("place").style.color = "#FF0000";
    }
}
```

Listing 1: Code to initiate geolocation

The condition in the if statement checks if the geolocation method is available. This determination is made instantaneously. The geolocation calculation still could have problems. The first two operands in the getCurrentPosition method indicate two different functions: one if the operation is successful and one if it is not. The definition of my versions of these functions appear in Listing 2. If the operation was successful, the handler function is called with an object that holds the latitude and longitude information. If the
operation attempted but was not successful, the `problemhandler` function is called with an object that holds a code indicating the nature of the problem.

```javascript
function handler(position) {
  var mylat = position.coords.latitude;
  var mylong = position.coords.longitude;
  var result;
  makemap(mylat, mylong);
}
function problemhandler(prob) {
  switch(prob.code) {
    case 1:
      document.getElementById("place").innerHTML = "User declined to share the location information.";
      break;
    case 2:
      document.getElementById("place").innerHTML = "Errors in getting base location.";
      break;
    case 3:
      document.getElementById("place").innerHTML = "Timeout in getting base location.";
  }
}
```

Listing 2: functions for successful and not-successful geolocation calculations

I will describe what is done when geolocation succeeds, namely the function I created and named `makemap`, in the next section. You can see the possible problems indicated by the 3 cases shown in the definition of `problemhandler`.

### Google Maps API

The Google Maps API is a way to bring in Google Maps functionality into your own applications. See the link to the document in the Learn More section. The `<head>` element needs to contain a `<script>` element pointing to the API:

```html
<script type="text/javascript" charset="UTF-8" src="http://maps.google.com/maps/api/js?sensor=false"></script>
```

Note: You will read in the Google Maps API document about acquiring a key. My teaching examples do not use a key, mainly because this did not exist when Version 3.0 started. You can decide for yourself if the use of a key is beneficial. Check out [https://developers.google.com/maps/faq#keysystem](https://developers.google.com/maps/faq#keysystem)

To bring in a map, my code first sets up an associative array with the critical parameters and then calls the `map` method. My `makemap` function uses the location data set in the `handler` function as a result of an operation initiated by the `geolocation` call. The critical statements are shown in Listing 3.

```javascript
function makemap(mylat, mylong) {
  blatlng = new google.maps.LatLng(mylat, mylong);
  myOptions = {
    zoom: 16,
    center: blatlng,
    mapTypeId: google.maps.MapTypeId.ROADMAP
  };
  map = new google.maps.Map(document.getElementById("place"), myOptions);
}
```

Listing 3: Bringing in a Google Map

Notice that a location is a special type of object for the Google Maps API. It is constructed using latitude and longitude values. You should experiment with zoom factors: bigger corresponds to zooming in for more detail. I keep the comment in my code concerning disabling the interface just in case I build an application in which I need to prevent my site visitor from zooming or panning or changing the type of map.

The API provides us a way to put in markers on the map, using the default Google tear drop or an image defined in the code. I created my own small red x in the file "x1.png". This is shown in Listing 4.

```javascript
var xmarker = "x1.png";
markernow = new google.maps.Marker({
  position: blatlng,
  title: "now",
  icon: xmarker,
  map: map });
```

Listing 4: Code for placing a marker

This marker is at the location and will stay on top of the location, moving if it does, if and when the user pans or zooms.

The last critical part of this application is setting up the event handling for clicking on the map. The API provides this feature. My code sets up the event handling to be a call to a function I will describe in the next section. Listing 5 shows the single statement to accomplish this. Be careful about the parentheses and brackets. The third argument to the `addEventListener` method is an anonymous function definition.

```javascript
listener = google.maps.event.addListener(map, 'click',
  function(event) {
    resetnow(event.latlng);
  });
```

Listing 5: Setting up event handling for clicking on the map
**HTML5 localStorage**

The basic mechanism of the Web is for people using their own computers, called the client computers, to run programs called browsers, e.g., Firefox, Chrome, that download files from server computers. The browser program interprets the files. This includes executing any JavaScript. The HTML files and any referenced image files are temporarily resident on the client computer but the idea in the early days of the web was to avoid making any permanent changes to the client computer. Commercial applications called for something more. Some person wise in customer relations came up with the friendly term 'cookie' for small files stored on the client computer for later use by programs from the same server. The initial applications of cookies were for saving passwords and other information for the convenience of users. Now, a big use for cookies and variations is for so-called behavioral marketing; that is, saving details of client behavior for companies to customize and target advertising and gather general information about consumer behavior.

Storing information, geographic location or other things such as passwords or items purchased or sites visited on the client computer can be viewed as beneficial for users and some people even appreciate the accuracy of commercial action based on stored information. Others consider it an invasion of privacy and/or just plain creepy. In any case, the European Union has passed what is termed the 'Cookie Law' requiring explicit permissions. People building production websites need to be aware of relevant laws.

The HTML5 `localStorage` feature is the latest facility to provide a way to store relatively small amounts of information on the client computer to be retrieved only by the website that did the initial storing. Information is saved using a key-value pair, with both key and value being strings. For this application, I use the keys: `oldlat` and `oldlng`. The application also places a new marker on the map. This is done in the function `resetnew` shown in Listing 6.

```javascript
function resetnow(clatlng) {
    marker = new google.maps.Marker({
        position: clatlng,
        title: "now",
        icon: xmarker,
        map: map });
    localStorage.oldlat = String(clatlng.lat());
    localStorage.oldlng = String(clatlng.lng());
}
```

Listing 6: Saving values in `localStorage`

Retrieving the data from local storage is performed by referencing the `localStorage.oldlat` and `localStorage.oldlng` values. The strings are converted to numbers, which, in turn, are used to place a marker on the map. The logic of my program is to store the current value in `localStorage` to be used the next time. The coding is shown in Listing 7.

```javascript
if (localStorage.oldlat) {
    var oldlat = Number(localStorage.oldlat);
    var oldlng = Number(localStorage.oldlng);
    priorlatlng = new google.maps.LatLng(oldlat, oldlng);
    markerthen = new google.maps.Marker({
        position: priorlatlng,
        title: "then",
        icon: xmarker,
        map: map });
}
```

Listing 7: Retrieving values from `localStorage`, placing a marker, and saving current values

The expression `localStorage.oldlat` returns `true` if a key-value pair with the key "oldlat" does exist in `localStorage` and `false` otherwise.

**Warning on testing:** Local storage and other cookie variations are intended for use by the website that created them. Check out if your browser retrieves `localStorage` when scripts are tested locally, that is, on a client computer.

The source code for the entire application can be viewed by using the link given in the next section. Applications involving storage of sets of information of variable size are described in my books.

**Learn more**

There are many sources, online and in-print and some sort of e-books, for learning HTML5 and JavaScript techniques along with the standard for geolocation. Here are links to my recent books, the website for this example and the Google Maps API.


- **HTML5 and JavaScript Projects**, [http://www.apress.com/9781430240327](http://www.apress.com/9781430240327). This book is more advanced than the first one. There are three chapters on uses of the Google Maps API, including one on geolocation and reverse geolocation and using php to send email with the information. A chapter describing a jigsaw puzzle explains an elaborate use of `localStorage` to save the current position of each of the pieces of the puzzle.

- To see the application in action and to view the source code, go to [http://faculty.purchase.edu/jeanine.meyer/html5/geolocation/localstorage.html](http://faculty.purchase.edu/jeanine.meyer/html5/geolocation/localstorage.html)


Jeanine Meyer lives just north of New York City and currently teaches at Purchase College/SUNY after many years at IBM, doing research on robotics and manufacturing and consulting on educational grants. She likes providing programming examples for her Mathematics/Computer Science and New Media students.