Extracted from:

HTML5 and CSS3, Second Edition

Level Up with Today's Web Technologies

The Pragmatic Bookshelf

Dallas, Texas • Raleigh, North Carolina

The Pragmatic Programmers

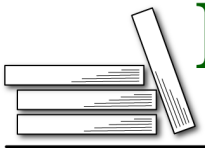
HTML5 and CSS3

Second Edition

Level Up with Today's Web Technologies

Brian P. Hogan

Edited by Susannah Davidson Pfalzer



Pragmatic Bookshelf

Our Pragmatic courses, workshops, and other products can help you and your team create better software and have more fun. For more information, as well as the latest Pragmatic titles, please visit us at *http://pragprog.com*.

The team that produced this book includes:

Susannah Davidson Pfalzer (editor)
Potomac Indexing, LLC (indexer)
Candace Cunningham (copyeditor)
David J Kelly (typesetter)
Janet Furlow (producer)
Juliet Benda (rights)
Ellie Callahan (support)

Printed in the United States of America.
ISBN-13: 978-1-937785-59-8
Encoded using the finest acid-free high-entropy binary digits.
Book version: P1.0—October 2013

Tip 33

Finding Yourself: Geolocation

Geolocation is a technique for discovering where people are, based on their computers' location. Of course, "computer" really can mean smartphone, tablet, or other portable device as well as a desktop or laptop computer. Geolocation determines a person's whereabouts by looking at her computer's IP address, MAC address, Wi-Fi hotspot location, or even GPS coordinates if available. Although it's not strictly part of the HTML5 specification (and never was), Geolocation is often associated with HTML5 because it came on the scene at the same time. Like Web Storage, it's a very useful technology that is already implemented in Firefox, Safari, and Chrome. Let's see how we can use it.

Locating Awesomeness

We've been asked to create a contact page for the AwesomeCo website, and the CIO has asked whether we can show people's location on a map along with the various AwesomeCo support centers. He'd love to see a prototype, so we'll get one up and running quickly.

We'll use Google's Static Map API for this because it doesn't require an API key and we're going to generate a very simple map. When we're done, we'll have something that looks like Figure 36, *Our current location is marked on the map with a Y*, on page 6.

AwesomeCo service centers are located in Portland, Oregon; Chicago, Illinois; and Providence, Rhode Island. Google's Static Map API makes it really easy to plot these points on a map. All we have to do is construct an `img` tag and pass the addresses in the URL, like this:

html5_geolocation/index.html

```
<img id="map" alt="Map of AwesomeCo Service Center locations"
src="http://maps.google.com/maps/api/staticmap?
&amp;size=900x300
&amp;sensor=false
&amp;maptype=roadmap
&amp;markers=color:green|label:A|1+Davol+square,+Providence,+RI+02906-3810
&amp;markers=color:green|label:B|22+Southwest+3rd+Avenue,Portland,+OR
&amp;markers=color:green|label:C|77+West+Wacker+Drive+Chicago+IL">
```

Figure 36—Our current location is marked on the map with a *Y*.

We define the size of the image, and then we tell the Maps API that we did not use any sensor device, such as a client-side geolocation, with the information we're passing to this map. Then we define each marker on the map by giving it a label and an address. We could use comma-separated pairs of coordinates for these markers if we had them, but an address is easier for our demonstration.

How to Be Found

We need to plot our visitor's current location on this map, and we'll do that by providing latitude and longitude for a new marker. We can ask the browser to grab our visitor's latitude and longitude, like this:

html5_geolocation/javascripts/geolocation.js

```
var getLatitudeAndLongitude = function(){
  navigator.geolocation.getCurrentPosition(function(position) {
    showLocation(position.coords.latitude, position.coords.longitude);
  });
};
```

This method prompts the user to provide us with her coordinates. If the visitor allows us to use her location information, we call the `showLocation()` method.

The `showLocation()` method takes the latitude and longitude and reconstructs the image, replacing the existing image source with the new one. Here's how we implement that method:

html5_geolocation/javascripts/geolocation.js

```
Line 1 var showLocation = function(lat, lng){
  2   var fragment = "&markers=color:red|color:red|label:Y|" + lat + "," + lng;
  3   var image = $("#map");
  4   var source = image.attr("src") + fragment;
  5   source = source.replace("sensor=false", "sensor=true");
  6   image.attr("src", source);
  7 };
```

Rather than duplicate the entire image source code, we'll append our location's latitude and longitude to the existing image's source. Before we assign the modified image source back to the document, we need to change the sensor parameter from false to true. We do that on line 5 with the replace() method.

Finally, we call the getLatitudeAndLongitude() method we defined, which kicks everything off.

html5_geolocation/javascripts/geolocation.js

```
getLatitudeAndLongitude();
```

When we bring up the page in our browser, we see our location, marked with a *Y*, among the other locations.

Falling Back

As it stands, visitors without Geolocation support will still see the map with the locations of the AwesomeCo support centers, but they'll get a JavaScript error since there's no Geolocation object available. We need to detect support for Geolocation before we attempt to get the visitor's location. We can use Modernizr for that, but where do we get latitude and longitude if we can't get it from the browser?

Google's Ajax API does location lookup, so it's a great fallback solution.[9]

Our fallback looks like this:

html5_geolocation/javascripts/geolocation.js

```
Line 1 var getLatitudeAndLongitudeWithFallback = function(){
  2   if ((typeof google === 'object') &&
  3       google.loader && google.loader.ClientLocation) {
  4         showLocation(google.loader.ClientLocation.latitude,
  5                      google.loader.ClientLocation.longitude);
  6     }else{
  7       var message = $("<p>Couldn't find your address.</p>");
  8       message.insertAfter("#map");
  9     }
 10 };
```

9. http://code.google.com/apis/ajax/documentation/#ClientLocation

We use Google's ClientLocation() method on line 3 to get a visitor's location and invoke our showLocation() method to plot the location on our map.

Then we tell Modernizr to test for Geolocation. If we have support, we'll call our original method. If we don't have support, we'll use a simplified version of Modernizr.load() to load Google's library and then call our function to plot the coordinates.

html5_geolocation/javascripts/geolocation.js

```
if(Modernizr.geolocation){
  getLatitudeAndLongitude();
}else{
  Modernizr.load({
    load: "http://www.google.com/jsapi",
    callback: function(){
        getLatitudeAndLongitudeWithFallback();
    }
  });
}
```

Unfortunately, Google can't geolocate every IP address out there, so we may still not be able to plot the user on our map; we account for that by placing a message underneath our image on line 7. Our fallback solution isn't foolproof, but it does give us a greater chance of locating our visitor.

Without a reliable method of getting coordinates from the client, we need to come up with a way for the user to provide us with an address, but that's an exercise I'll leave up to you.

Next, let's take a look at HTML5's built-in support for dragging and dropping elements.