Using Geolocation in Your Windows* Store Apps

By Bill Sempf

Steve walks down 5th Avenue looking for the shop he was sure was there last year.

“Right there,” he thinks, “across from the pizza shop.”

Frustrated, he stops at a bench and takes out his Ultrabook™ device running Windows 8. After pulling up the Charms bar, he clicks Search and types Shoe repair. Although it is not surprising that nothing is installed on his device that has that search term extant, the number of apps that respond to the term in deep search is somewhat shocking.

Steve taps the new localized app he installed last week, and right away, a blog entry is featured that talks about how the shoe store on 5th moved across town to the new mall.

“How on earth did it know I meant that shoe repair shop?” Steve thinks.

The answer is geolocation.

Bringing Geolocation into Your App

Geolocation is determining the location of a device on the earth by checking sensors and networking details. The primary sensor for geolocation is the Global Positioning System (GPS) sensor. The GPS gets information from triangulating on a series of satellites that were positioned by the U.S. government in the 1980s and were made available for public use in the 1990s.

The GPS network isn’t the only place to get geolocation information, though. It is possible to use your IP address to get a location, and if you have a device that uses the Wi-Fi* network, then triangulation among access points is also possible.

Geolocation was introduced to the Windows operating system in version 7, with Win32* and Microsoft .NET application programming interfaces (APIs). Windows 8 adds a Windows Runtime (WinRT) API for use in Windows Store apps as well as the existing Microsoft .NET and Win32 endpoints. Windows 8 apps also add the ability to use the browser’s geolocation abilities—World Wide Web Consortium (W3C) Geolocation.
**W3C Geolocation**

If you have used a browser to access a website that uses a map, you have probably been asked, “Can this website have access to your location?”

Starting around 2008, browsers started making geolocation available to web developers via the W3C standard for location awareness. This standard gives developers access to not only the current location of the device viewing the web page but also events that occur when the location changes. These standard functions work in Internet Explorer* 10, so they work in your Windows Store apps. All you need to do is use the W3C standard functions for location within your app’s JavaScript* code.

**Get the Current Location**

The Geolocation API specifies the `navigator.geolocation` class in JavaScript, which is an asynchronous function that provides a position object in the callback. Implementation is straightforward in Windows 8 because you can skip the traditional “is this object supported” check. The line in question is line 7 (bold text) in Listing 1.

**Listing 1. Minimal JavaScript code for retrieving geolocation coordinates**

```javascript
(function () {
    "use strict";
    var coordField;
    WinJS.UI.Pages.define("/pages/w3cLocation.html", {
        ready: function (element, options) {
            coordField = document.getElementById('coordinates');
            navigator.geolocation.getCurrentPosition(getCoordinates, err);
        }
    });
    function getCoordinates(position) {
        var lat = position.coords.latitude;
        var long = position.coords.longitude;
        coordField.innerText = 'Current latitude: ' + lat + ' and longitude: ' + long;
    }
    function err(e) {
        console.log(e.message);
    }
})();
```

This code of course assumes a div in the HTML called coordinates.

Create a new blank JavaScript Windows 8 project in Microsoft Visual Studio*, and drop this code in the .js file. (Remember to fix up the `WinJS.UI.Pages.define` function.) Add the div to the HTML file and press F5 to run the app.

Nothing happen? I bet! Check out your JavaScript console back in Visual Studio:

*This site does not have permission to use the Geolocation API.*
The Windows Store Sandbox strikes again! You will need to define **Location** as a capability of the app in the `package.appxmanifest` file for the location to be accessible. To do this, double-click the `package.appxmanifest` file, and click the **Capabilities** tab at the top of the Manifest Designer, as shown in Figure 1. There you will find the check box for **Location**, along with the other capabilities.

![The Capabilities tab in Microsoft Visual Studio® 2012](image)

**Figure 1. The Capabilities tab in Microsoft Visual Studio® 2012**

Now run the app. The Windows 8 environment checks with the user to make sure this app is allowed to use location, as Figure 2 shows.
**Figure 2.** *Windows 8* checks with the user for permission to use geolocation

Allow the feature, and the current coordinates are displayed, as Figure 3 shows.
Watch Position

Finding your users’ locations right where they are standing is always a popular choice, but you may need to do some work when the device moves. `navigator.geolocation` still helps you here, but now you need to `watchPosition()` call. It also takes a callback and error function, just like `getPosition()`.

In fact, you can effectively use the same code (see Listing 2). The two functions can even be used in tandem. It’s possible that you want to get the current location first, just in case the user doesn’t ever move, right?

Listing 2. Extending the code to track changes in coordinates

```javascript
(function () {
    "use strict";
    var coordField;
    WinJS.UI.Pages.define("/pages/w3cLocation.html", {
        ready: function (element, options) {
            coordField = document.getElementById('coordinates');
            navigator.geolocation.getCurrentPosition(getCoordinates, err);
            navigator.geolocation.watchPosition(getCoordinates, err);
        }
    });
})()
```
Using Geolocation in Your Windows® Store App

29 October 2012

```javascript
(function () {
    "use strict";
    var coordField;
    WinJS.UI.Pages.define("/pages/WinrtLocation.html", {
        ready: function (element, options) {
            coordField = document.getElementById('coordinates');
            locator.getGeopositionAsync().done(getCoordinates, err)
        }
    });

    function getCoordinates(position) {
        var lat = position.coords.latitude;
        var long = position.coords.longitude;
        coordField.innerText = 'Current latitude: ' + lat + ' and longitude: ' + long;
    }
    function err(e) {
        console.log(e.message);
    }
})();
```

Adding line 8 (bold text) above updates the div every time the location changes.

**Windows.Devices.Geolocation**

`windows.Devices.Geolocation` is a WinRT API that sits on the Location Sensor API. The Location Sensor is a constantly running service in the Windows operating system that collects geolocation data from any available source and uses accuracy estimates to determine the best location at any given time. Sources can be:

- A driver from a build in the GPS sensor;
- The Windows Location Provider, which gets location data from the Wi-Fi network or IP Address lookup when the device is online; and
- Other drivers that are registered with the service by a device developer.

Although under the covers the Location Sensor API is quite complex, the WinRT Geolocation API makes things quite simple for the developer. It is isolated from the complexities of determining which sensors to read or which location is more accurate.

**Geocoordinates (Latitude and Longitude)**

It really isn't surprising that `windows.Devices.Geolocation.Geolocator` uses practically the same layout as the W3C standard. The idea is to make HTML version 5 (HTML5) developers comfortable, and Microsoft succeeded. The `getPositionAsync()` function not only accepts a callback and error handler but also passes a similar position object to the callback function. You can build an app functionally similar to Listing 2 with only a few changes (see Listing 3).

**Listing 3. JavaScript code that demonstrates asynchronous access to a GPS sensor**

```javascript
(function () {
    "use strict";
    var coordField;
    WinJS.UI.Pages.define("/pages/WinrtLocation.html", {
        ready: function (element, options) {
            coordField = document.getElementById('coordinates');
            locator.getGeopositionAsync().done(getCoordinates, err)
        }
    });

    function getCoordinates(position) {
        var lat = position.coordinate.latitude;
```
You have to instantiate an instance of the Geolocator. It isn’t automatic like the navigator is. Also, the `getPositionAsync()` function uses the Promises style of asynchronous behavior that is standard in all Windows Library for JavaScript (WinJS) apps.

**Civic Addresses**

The greater power of an operating system behind the location service means there are a few more features, though. For instance, it is possible to get some address locations right from WinRT without using an external address service. The city, country, postal code, and state are available for many geopositions throughout the world.

The same `Geoposition` object in the Geolocator provides the `civicAddress` and the coordinates (see Listing 4). You don’t even have to spin up another instance of the class!

**Listing 4. Code updated to retrieve address information**

```javascript
(function () {
    "use strict";
    var coordField;
    var addrField;
    WinJS.UI.Pages.define("/pages/WinrtLocation.html", {
        ready: function (element, options) {
            coordField = document.getElementById('coordinates');
            addrField = document.getElementById('address');
            locator.getGeopositionAsync().done(getCoordinates, err)
        }
    });
    function getCoordinates(position) {
        var lat = position.coordinate.latitude;
        var long = position.coordinate.longitude;
        var city = position.civicAddress.city;
        var state = position.civicAddress.state;
        coordField.innerText = 'Current latitude: ' + lat + ' and longitude: ' + long;
        if (city) {
            addrField.innerText = 'This is in: ' + city + ', ' + state;
        }
    }
    function err(e) {
        console.log(e.message);
    }
})();
```

Notice that I added a `div` to the HTML for the address. The sample HTML now looks like this in my sample `winrtLocation.html` file (see Listing 5).
Listing 5. HTML for the sample Geolocation app

```html
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8" />
  <title>WinrtLocation</title>
  <!-- WinJS references -->
  <link href="//Microsoft.WinJS.1.0/css/ui-dark.css" rel="stylesheet" />
  <script src="//Microsoft.WinJS.1.0/js/base.js"></script>
  <script src="//Microsoft.WinJS.1.0/js/ui.js"></script>
  <link href="WinrtLocation.css" rel="stylesheet" />
  <script src="WinrtLocation.js"></script>
</head>
<body>
<div class="WinrtLocation fragment">
  <header aria-label="Header content" role="banner">
    <button class="win-backbutton aria-label="Back" disabled type="button"></button>
    <h1 class="titlearea win-type-ellipsis">
      <span class="pagetitle">WinRT Geolocation API Sample</span>
    </h1>
  </header>
  <section aria-label="Main content" role="main">
    <div id="coordinates"></div>
    <div id="address"></div>
  </section>
</div>
</body>
</html>
```

Also notice that it is worthwhile to see whether city is empty before you depend on the data. For my current location, all I am getting is country. If I change location in the simulator, though, I can get a broader array of return data. To change locations in the simulator, touch the little globe on the command bar of the tablet and set whatever coordinates you want the simulator to return (see Figure 4).
Looking Up a Location in Data

By itself, getting the coordinates isn’t interesting. You’ll need something a bit more substantial as a feature.

Many services accept latitude and longitude as input parameters and return interesting stuff. One real-world example is finding users on a map based on their coordinates. Microsoft Bing* provides a service that handles this for you. Another neat thing to try is getting points of interest from the Bing API search. Note, however, that doing so requires a Bing Maps Key if you want to try the code samples for yourself. You can get a Bing Maps Key at https://www.bingmapsportal.com.

Finding Yourself on a Map

Bing Maps is a big API with a ton of useful functionality, but the most significant offering is the ability to display a map (with a variety of data) that is centered on the location of the device. To do so, you must set up several asynchronous events and wire them together:

1. When the page is ready, initialize the map entity on the page by attaching a function (initialize) to the DOMContentLoaded event.
2. When the Map module is loaded, have it call getLocation() to start the geolocator (getLocation).

3. When the geolocator is done, populate the map (populateMap).

4. Pass everything up to the screen, and the map is displayed.

Listing 6 shows the code.

**Listing 6. Code to generate a map around the geolocation**

```javascript
(function () {
    "use strict";
    var lat, lon, map;
    var bingMapsKey = 'YourBingMapsKeyHere-Y2pfuS7e2ifm6QE5SKVmNuLNZ79iVF7I44';
    WinJS.UI.Pages.define("/pages/FindOnAMap.html", { 
        ready: function (element, options) { 
        }
    });
    function getLocation() {
        locator.getGeopositionAsync().done(populateMap, err);
    }
    function populateMap(position) {
        lat = position.coordinate.latitude;
        lon = position.coordinate.longitude;
        try {
            var mapOptions = {
                credentials: bingMapsKey,
                center: new Microsoft.Maps.Location(lat, lon),
                mapTypeId: Microsoft.Maps.MapTypeId.road,
                zoom: 8
            };
            map = new Microsoft.Maps.Map(document.getElementById("mapdiv"), mapOptions);
        } catch (e) {
            var md = new Windows.UI.Popups.MessageDialog(e.message);
            md.showAsync();
        }
    }
    function err(e) {
        console.log(e.message);
    }
    function initialize() {
    }
    document.addEventListener("DOMContentLoaded", initialize, false);
}());
```

So, the DOMContentLoaded event starts the initialize function, which starts getLocation, which in turn starts populateMap(). Clear as mud? Good! Let’s get a list of places to eat: I’m starving.
Getting Nearby Points of Interest

Another good use of the user’s geolocation is getting a list of points of interest (POI) that are near the location of the device. A great example of this is a What’s for Lunch–type app that looks at nearby restaurants and randomly picks one.

There are two pieces to this process, and you already have one. The geolocation gets you the latitude and longitude, which is the core of the POI search. You have that already. All you need now is a place to get a list of POIs. The Google Places API, currently in beta, is a great place to get a free list of POIs.

The Google Places API is a Representational State Transfer (REST) service hosted by Google. Use the WinJS.xhr method to call the service and return a list of place objects. The xhr method returns a Promise, which is the WinJS async process, similar to the callback that the W3C Geoposition API accepts. The Promise accepts a callback function (usually provided anonymously), an error function, and a process function (often omitted).

So now, the page.ready handler calls a getPlaces() method when getGeopositionAsync() is done, and getPlaces() populates the placesData, which you can bind to a list on the screen. Listing 7 shows the code.

Listing 7. Code to retrieve POIs from the Google Places API

(function () {
    "use strict";
    var placesData;
    WinJS.UI.Pages.define("/pages/NearbyPlaces.html", {
        ready: function (element, options) {
            locator.getGeopositionAsync().done(getPlaces, err)
        },
    });

    function getPlaces(position) {
        var lat = position.coordinate.latitude;
        var lon = position.coordinate.longitude;
        var constructedURL = 'https://maps.googleapis.com/maps/api/place/search/json?location='+lat+','+lon+'&radius=50&types=restaurant&key='+googleSearchKey;
        WinJS.xhr({ url: constructedURL }).then(function (r) {
            placesData = JSON.parse(r.responseText);
        }, function (err) {
            console.log('Problem with search: ' + err);
        });
    }
    function err(e) {
        console.log(e.message);
    }
});
Considering Privacy

When I first signed up with Google for email, calendar, and such for my consulting company, I looked forward to tying my Google Android* phone into the services. This is back when Google Buzz, the precursor to Google+, was popular, and I logged in to see what the buzz was all about.

The people that I had linked up with for Google Talk all showed on the main screen, and those who had used Buzz had messages showing. It was moderately interesting, but then I noticed “messages near you.”


As it turns out, the guy about 100 addresses down from me was a Buzz user, and there he was—all his messages, right next to an approximation of his address.

I couldn’t uninstall that app fast enough.

The fact is, Google didn’t really do anything wrong. Google tells you that it’s using your location, and we all know what that means. But it’s just shocking to me to see, “Hey, the guy at 335 Main Street thinks the new Spiderman movie is awful.”

You don’t want to shock your users. Make sure you provide a clear expectation of what’s going to happen with their location data. Windows 8 provides tools to make doing so easier, and a few tips will do the rest.

Inform the User That You Are Using Location Information

To make sure you don’t surprise users, tell them you’re using their location data. Give them the option, and make sure you heed their wishes.

Letting the Operating System Handle Location Privacy

In the first example, I showed you how to set up a declaration in your application for using Location services. This isn’t optional: Every time the app is updated, users have to okay the use of the Location services.

This might not always be enough, though. Users can turn off all location services for the entire device at any time from the PC settings dialog box, as shown in Figure 5.
Using Geolocation in Your Windows® Store App

Figure 5. PC settings contains a global on/off for use of location data

Your app must accommodate not having location data, even though users might have agreed to it at the time of installation. Make sure you check the error codes returned from GetGeopositionAsync(). If users have disabled location services globally, calls to the function return an ACCESS_DENIED error. Take this to mean that that user doesn’t want you getting the location, and handle it appropriately in your app.

Sharing Isn’t Always Nice

Just because a user has clicked a button that allows you to access his or her location doesn’t mean that you get to share that information with others. If you are passing it to a weather service and including only an IP number, that’s one thing. If you are sending location information along with an email address to a service, then you might want to consider getting more explicit permissions.

Being More Explicit When You Have To

Sometimes, you have to remind users that you are going to use their location. Steve’s location-aware search app in the story at the beginning of this article should have warned him that it was going to use his location for the search. If you’re concerned about pestering the user, allow him to opt out, but leave the notification on by default.
To alert users, use the Windows.UI.Popups.MessageDialog() function of WinRT. The returned MessageDialog object has values that correspond to the user’s choice: it’s okay to use my location, or not now. Always do what the user requests.

**Don’t Store Location Information Outside the Device**

A general best practice for geolocation is not to store the information outside the device. Even if it would make a use case more solid to call an external service with the user’s location and store it there, don’t. Apple and Google have gotten in a lot of trouble for doing just that, and they have more lawyers than you do.

Instead, if you need to keep track of locations, set up a Windows.Application.Data.localSettings() object, and store a key–value pair set there. You can keep up to 64KB of data—more than enough to keep some location information.

**Dispose of the Location Information After It’s Used**

In line with the last idea of keeping data local rather than remote, get rid of it when it’s no longer needed. The last thing you need is for some curious user to go digging through the file system on his or her device and find a file with everywhere he or she has been for the past 2 years. That isn’t a good way to create trusting users.

The easiest way to do that is to set up code in the WinJS.Application.oncheckpoint event that clears cached location data. You can simply look over any storage you have and delete things if they aren’t needed anymore. Remember, trusting users are happy users.

**About the Author**

Bill Sempf is a software architect. His breadth of experience includes business and technical analysis, software design, development, testing, server management and maintenance, and security. In his 17 years of experience, he has participated in the creation of more than 200 applications for large and small companies, managed the software infrastructure of two Internet service providers, coded complex software in every environment imaginable, and made mainframes talk to cell phones. He is the author of Windows 8 Programming with HTML5 for Dummies and C# 5.0 All in One for Dummies; a coauthor of Effective Visual Studio.NET and many other books; and a frequent contributor to industry magazines. He has been invited to speak for the ACM and IEEE and at CodeMash, DerbyCon, BSides, DevEssentials, the International XML Web Services Expo, and the Association of Information Technology Professionals. Bill also serves on the board of the Columbus branch of the Open Web Application Security Project and is the administrative director of Locksport International.
For More Information

- Google Places API: https://developers.google.com/places/documentation

*Other names and brands may be claimed as the property of others.