

Mike Christensen  
Geography 32-611-02  
Class Project Report

**US Postal Service Address Information API**  
**and**  
**Google Maps API “Point-In-Polygon”**

**Introduction**

For my class project, I decided that I wanted to focus more on the web application aspect of the class than on creating maps. In fact, I kept the web application very basic and focused more on functionality than aesthetics. So I set out to combine two APIs into one web application, not knowing whether I could actually put it together.

**Proposal**

The objective of my web application as listed in my proposal is as follows: “Input a street address. Have the Post Office Address Information API determine whether the address is valid (i.e. is an address where mail can be delivered) and format the address in the Post Office standardized format. Have the Google Maps API determine whether the address lies within a pre-determined boundary. Display whether the address is deliverable by the Post Office, display the address in Post Office standardized format, display whether the address lies within a pre-determined boundary, and have the Google Maps API map the address.”

## **Background**

In the Church of Jesus Christ of Latter-day Saints, the basic congregational unit is called a “ward.” Each ward has a geographic boundary, and members of the Church attend the ward of which boundary they live within. Additionally, in areas where there are large populations of single members, YSA (Young Single Adult) wards are established, which overlay multiple “non-single” wards and give singles an increased opportunity to mingle and get married. (As I am still single, it doesn’t always work! LOL) The boundary that I chose for my project is the boundary of the Rose Park YSA Ward, which is the ward that I live within and attend.

### **Step 1: The US Postal Service Address Information API**

As I went to work on this project, I decided that I should tackle the portion that I was least familiar with first. The best working example of the Address Information API is located on the Postal Service’s website: <http://zip4.usps.com/>. Simply put, the US Postal Service has a database of all of the addresses where mail can be delivered, and the API allows anyone to enter an address and have the website validate whether the address is deliverable. The API also returns the address formatted according to standardized format, which is explained in great detail here: <http://pe.usps.com/text/pub28/welcome.htm>.

Creating your own Address Information API, requires signing up for an API key with the US Postal Service. There’s more information about that process and also a technical guide describing how to integrate the API here: <http://www.usps.com/webtools/address.htm>. The technical guide describes the syntax for communicating with the API. Basically the address to be validated must be formatted as an XML file and submitted to the API using either the GET or

the POST method. The API responds with an XML file containing the validated address or an error message.

Unfortunately, the technical guide gives no examples or sample code illustrating how to create a webpage that composes the XML file to be sent to the API or how to parse the XML file returned by the API. I first searched the internet for examples of performing this using only JavaScript but couldn't find any. However, in searching I discovered an example of how to do the task using PHP: <http://joe-riggs.com/blog/2009/10/address-standardization-verification-with-usps-web-tools-and-php/>. Fortunately I had played around with PHP a few years ago, so I had a basic understanding of server-side scripting. After a few hours of tinkering, I was successful in creating a webpage that could communicate with the Address Information API using the PHP code that I had found.

## **Step 2: Point-in-Polygon**

The next portion that I was least familiar with was determining if the address lies within the boundary. After browsing through the Google Maps API Reference, I realized that Google Maps has no built-in function for determining if a point lies within a polygon. This type of query function is something that is standard in GIS, but it would appear that Google Maps has yet to include this kind of functionality. Fortunately after searching online, I discovered that someone had written a JavaScript extension for Google Maps API v3 that performs a point-in-polygon function: <https://github.com/tparkin/Google-Maps-Point-in-Polygon>.

The point-in-polygon code required that the polygon be formatted as an array, but I didn't know how to format it properly. Fortunately I found an example here: [http://www.geocodezip.com/v3\\_polygon\\_example.html](http://www.geocodezip.com/v3_polygon_example.html). In order to keep things simple, I set up

box around my neighborhood as a simple array. After a few hours of tinkering, I was successful in creating a webpage that could determine if a given latitude and longitude falls within the box around my neighborhood.

### **Step 3: The Boundary**

Using ArcMap, I had already drawn the boundary of the Rose Park YSA Ward, but I needed to convert the polygon feature class into any array of latitude and longitude points. I used the toolbox in ArcMap to convert the boundary into a KMZ file. Then I used Google Earth to open the KMZ file and save it as a KML file. I then used Notepad++ to extract just the latitude and longitude points from the KML file. The latitude and longitude points were formatted with longitude followed by a comma followed by latitude and a comma followed by a zero then followed by the next longitude, and all of it was on one long line. The array needed to be latitude and a comma followed by a hard return and then the next latitude. The series of points comprising the boundary turned out to be over 727 points, which was simply too many to edit by hand.

So I saved the long line of points as a text file and opened it using WordPerfect. I had WordPerfect search for each comma and zero pair and replace all with a hard return. This put each longitude and latitude pair on a separate line. I saved the reformatted points as a text file and had Excel open it as a CSV file. This put all the longitudes in one column and the latitudes in another. I switched the columns in order to put latitude first and then inserted extra columns populated by cells containing the same text in order to form the correct syntax for the array. I then copied and pasted the newly formed array into the code I was writing in Notepad++.

#### Step 4: Putting It All Together

One thing that I had to teach myself was how to pass variables among HTML, PHP, and JavaScript. Fortunately I found a webpage that explains how to do it:

<http://www.skytopia.com/project/articles/compsci/form.html>. I also have to credit w3 Schools for being a great reference for HTML, XML, PHP, and JavaScript: <http://www.w3schools.com/>.

Once I figured out how to pass variables among the different languages, putting it all together was fairly straightforward. The web application consists of three files: **index.html**, **validate.php**, and **boundary.js**.

**index.html** is simple and consists only of a form that collects the address data. The file is written only in HTML. Upon submitting the form, the form data is sent to the server using the POST method and **validate.php** is loaded.

**validate.php** is complex and is written in HTML, PHP, and JavaScript. The PHP portion of the file collects the address data that was posted to the server, formats it into XML, sends the XML to the US Postal Service Address Information API, receives the XML response from that API, and parses the XML response into variables that will be passed to JavaScript and HTML.

The JavaScript portion of **validate.php** receives the address to be geocoded from the PHP portion, sets up the parameters of the Google Maps window, sends the address to Google Maps to be geocoded, receives the resulting latitude and longitude, evaluates whether that latitude and longitude falls within the boundary of the Rose Park YSA Ward (as defined in **boundary.js**), and formats that result into a variable to be passed to HTML.

The HTML portion of **validate.php** receives the address that was initially sent to the server along with the parsed Address Information API XML response from the PHP portion, receives

the result of the boundary evaluation from the JavaScript portion, and also displays the Google Maps window as defined in the JavaScript portion.

**boundary.js** exists simply because including the boundary array within **validate.php** would have added 727 lines of code to the file, which would have made it difficult to edit.

### **Step 5: Handling Errors from the APIs**

The final portion of creating the web application involves ensuring that errors from the APIs are properly handled. I tested my web application using a variety of different address possibilities and ensured that the proper result was displayed. I've included several examples to illustrate the different possibilities. Feel free to test these examples yourself:

Scenario 1:

Address: **475 N Redwood Rd Unit 50**

City: **Salt Lake City**

State: **UT**

This results in an address that is deliverable, Google Maps can geocode and display the address, and the address lies within the boundary.

Scenario 2:

Address: **475 north redwood road #50**

City: **saLt lAke ciTy**

State: **uT**

This results in an address that is deliverable (note that the Address Information API has reformatted the address according to standardized format), Google Maps can geocode and display the address, and the address lies within the boundary.

Scenario 3:

Address: **475 N Redwood Rd**

City: **Salt Lake City**

State: **UT**

This results in an address that is not deliverable (it is missing “an apartment, suite, or box number”), but Google Maps can geocode and display the address, and the address lies within the boundary.

Scenario 4:

Address: **425 N Redwood Rd**

City: **Salt Lake City**

State: **UT**

This results in an address that is not deliverable (no building exists at 425), but Google Maps can geocode and display the address, and the address still lies within the boundary.

Scenario 5:

Address: **475 S Redwood Rd**

City: **Salt Lake City**

State: **UT**

This results in an address that is deliverable, Google Maps can geocode and display the address, but the address does not lie within the boundary.

Scenario 6:

Address: *empty*

City: **Salt Lake City**

State: **UT**

This results in an address that is not deliverable (“Address Not Found”), but Google Maps can geocode and display the address (it defaults to the center of Salt Lake City), and the location does not lie within the boundary.

Scenario 7:

Address: *empty*

City: *empty*

State: **UT**

This results in an address that is not deliverable (“Invalid City”), but Google Maps can geocode and display the address (it defaults to the center of Utah), and the location does not lie within the boundary.

Scenario 8:

Address: *empty*

City: **New York City**

State: **NJ**



This results in an address that is not deliverable (“Invalid City”), but Google Maps can geocode and display the address (it’s still able to place the marker in lower Manhattan), and the location does not lie within the boundary.

Scenario 9:

Address: *empty*

City: **Paris**

State: *empty*

This results in an address that is not deliverable (“Invalid State Code”), but Google Maps can geocode and display the address (it’s able to place the marker in Paris, France), and the location does not lie within the boundary.

Scenario 10:

Address: *empty*

City: **asdf**

State: *empty*

This results in an address that is not deliverable (“Invalid State Code”), Google Maps cannot geocode and display the address, and since the location cannot be determined, whether it lies within the boundary cannot be evaluated.

**Go Live!**

The web application can be experienced at <http://32611.cascadepeak.com/>!