Overlay data on maps using XSLT, KML, and the Google Maps API, Part 1: Tap into the Google Geocoder Web service

Use the Google Geocoder to look up postal codes and coordinates for storage in a database

Jake Miles

September 02, 2008

Explore the Google Geocoder Web service that takes a street address and returns data about that address including its longitude and latitude. In this two-part article series, you will combine it with the Google Maps API and XSLT to create data overlays for display in Google Maps and Google Earth. You will create an example application for a real-estate brokerage that lets a broker enter listings for apartments through an HTML form, uses Google’s Geocoder Web service to translate those addresses into longitudes and latitude, and then creates KML overlays from the database of apartment listings. In Part 1, you build the first half of the application to collect the apartment listing information from the user, uses the Google Geocoder Web service to turn the street address into its geographical coordinates (longitude and latitude), and stores those coordinates in the database along with the address information.

Google Maps, Google Earth, and the Geocoder

Frequently used acronyms

- API: application programming interface
- HTML: Hypertext Markup Language
- HTTP: Hypertext Transfer Protocol
- KML: Keyhole Markup Language
- URL: Uniform Resource Locator
- XML: Extensible Markup Language
- XSLT: Extensible Stylesheet Transformations

Google Maps has become the ubiquitous map technology on the Web, allowing users to instantly bring up geographical maps and pan and zoom around them, including 360-degree views of the street at eye-level. Google Earth provides a 3D photographic encyclopedia of the Earth, letting you pan and zoom an image of the Earth at various heights. With the Google Maps API, you can
embed Google Maps into your own Web pages. Add KML, an XML language used to describe geographical information such as coordinates on the Earth, and you can overlay your own visual and textual data onto the maps. You can also import KML data into Google Earth, and project your own 3D data onto the Earth as the user pans and zooms.

The other article in this series

- Overlay data on maps using XSLT, KML, and the Google Maps API, Part 2: Transform and use the data

For example, the musical group Nine Inch Nails released their most recent album, "The Slip" through download, and they have posted geographical download data that it produced using Google Earth and KML, as in Figure 1.

Figure 1. Download data for Nine Inch Nails' album "The Slip", rendered using Google Earth and KML

This brilliant concept hints at the possibilities available with Google's API and KML. The height of each spike reflects the number of downloads recorded at that location, and was created (one must assume), by creating a line in KML at the download's longitude and latitude, from altitude 0 to an altitude proportionate to the number of downloads at that location.

One critical detail missing from this visualization is the ability to map customer addresses, or at least postal codes, to their geographical coordinates on the Earth, because all custom KML data is positioned on the Earth using longitude, latitude and altitude coordinates.

To solve this problem, Google recently made available the Google Geocoder Web service, that takes a street address and returns KML data describing the address to whatever accuracy is possible, including its latitude and longitude. Once you have these coordinates you can overlay textual and visual data on 2D maps and the 3D globe as creatively as your imagination allows.
Getting started with the Google Maps API and Geocoder service

To use the Google Maps API and Google's Geocoder Web service, you must first sign up for a Google Maps API key (see Resources for a link), specifying the URL of the Web site that will be issuing API requests (you can sign up for as many keys/URLs as you like). On the results page, you'll find your API key and a starter HTML page containing the JavaScript necessary to display a Google map, the core of which is the JavaScript function `load()` (see Listing 1), which is called when the page loads.

Listing 1. JavaScript function displaying a Google map within the page

```javascript
function load() {
  if (GBrowserIsCompatible()) {
    var map = new GMap2(document.getElementById("map"));
    map.setCenter(new GLatLng(37.4419, -122.1419), 13);
  }
}
```

This function makes sure the browser can display a Google Map, then creates a map object (`GMap2`), providing it with the HTML element in the page to use as the map's container ("map"). It then sets its center to a coordinate on Earth, using the `GLatLng` (latitude/longitude) parameter object, and specifies a zoom depth (height) of 13. Figure 2 shows the resulting map.

Figure 2. Google map displayed by starter page

Providing overlay data using KML

You can use the JavaScript Google Maps API to overlay custom data, such as creating several bookmarks at locations or overlaying a few colored polygons and lines. However, for more complex data (that is, data containing many more data points), you will want to use a KML document. The KML document can specify geographic data including addresses, 3D coordinates on the Earth, and custom textual and visual data to be overlaid on maps or 3D models of the Earth.

When you provide KML data to Google Maps, you must use a URL that Google's servers can access to retrieve the KML, so you must provide the server-side KML. In this article series, you'll
use server-side XSLT to transform a database of information into a KML file that you can feed to Google Maps and Google Earth to render custom data.

**Storing Geocoder data in the database**

The longitude and latitude of an address is useful with respect to Google Maps and Google Earth, but to obtain them, you must make a call to the Google Geocoder Web service. This can take some time, so when you store the address in the database it makes sense to store the longitude and latitude of the address along with it. This way you can query the database, directly producing XML, and use XSLT to transform the returned data directly into KML that you can overlay on a map.

To accomplish this, you have two options as to how you call the Geocoder service:

- Call the Geocoder server-side before recording the information in the database
- Use the JavaScript library provided by Google to call the Geocoder before you submit the form that contains the user’s address information

**Calling the Geocoder from PHP**

First you will call the Geocoder from PHP, and use the SimpleXML module to traverse the results. The `recordListing()` function takes the request parameters for recording an apartment listing for a real estate brokerage, calls the Geocoder service, and uses the result to obtain the postal code and geographical coordinates of an address, recording all the information in the database (see Listing 2).

**Listing 2. The `recordListing()` function (PHP)**

```php
function recordListing($address, $aptNumber, $city, $state, $aptType, $rent, $notes) {
    $geocoder = new GoogleGeocoder(GOOGLE_MAPS_API_KEY);
    $result = $geocoder->fetchAddress($address, $city, $state);

    // use the geocoder to make sure the address is accurate enough to use
    if ($result->getAccuracy() < GoogleGeocoderAccuracies::ADDRESS) {
        throw new Exception("Address does not have enough accuracy to record the listing.");
    }

    // store in the database
    createListingInDb ($address, $aptNumber, $city, $state, $result->getZipcode(), $result->getLongitude(), $result->getLatitude(), $aptType, $rent, $notes);
}
```

The code in Listing 2 demonstrates another use of the Geocoder’s result information—a data-integrity check. The Geocoder’s result includes an Accuracy rating, measuring the specificity of the input. Here, `recordListing()` uses this accuracy measurement (explained in more detail after Listing 4 below) to confirm that the information is specific enough to record in the database.
The **GoogleGeocoder** class

To make the code reusable, encapsulate the call to Google's Geocoder in a `GoogleGeocoder` class (see Listing 3).

### Listing 3. The GoogleGeocoder class (PHP)

```php
class GoogleGeocoder {
    private $apiKey;
    private $googleGeocoderUrl = 'http://maps.google.com/maps/geo?';

    public function __construct ($apiKey) {
        $this->apiKey = $apiKey;
    }

    public function fetchAddress ($address, $city, $state) {
        // encode address for google api call (plusses, commas)
        $fullAddress = $this->encodeAddress ($address, $city, $state);

        // create the geocoder API call
        $geocoderCall = $this->googleGeocoderUrl .
                        "q=$fullAddress" .
                        "&key=$this->apiKey" .
                        "&output=xml";

        $result = file_get_contents ($geocoderCall);
        return new GoogleGeocoderResult($result);
    }

    private function encodeAddress($address, $city, $state) {
        return urlencode (join (", ", array ($address, $city, $state)));}

    public function encodeAddress($address, $city, $state) {
        return urlenquote (join (", ", array ($address, $city, $state)));
    }
}
```

The `GoogleGeocoder` class is responsible for calling the geocoder service through HTTP with a given street address. It takes your Google Maps API key in its constructor for use in the HTTP call. `fetchAddress()` first encodes the address properly for the call, then constructs the call's URL, providing the address as the `q` parameter, the Google Maps API key as the `key` parameter, and specifying `output=xml`. Your other output options here are `json` and `csv`, the latter of which will return a simplified comma-delimited response if all you need are the longitude and latitude. Finally, the function calls the URL using `file_get_contents()`, which will read a URL like a file if the string provided begins with a supported protocol such as HTTP.

The **response KML of the GoogleGeocoder service**

The Google Geocoder service will return KML that describes the address. For example, for address "123 E. 34th St., New York, NY", the call returns the XML in Listing 4.

### Listing 4. Geocoder response XML for "123 E. 34th St., New York, NY"

```xml
<?xml version="1.0" encoding="UTF-8"?>
<kml xmlns="http://earth.google.com/kml/2.0">
    <Response>
        <name>123 E. 34th St., New York, NY</name>
        <Status>
            <code>200</code>
        </Status>
    </Response>
</kml>
```
The response contains a Status reporting either success (code 200) or one of a number of error codes (see Resources for a link to a table of error codes on the Geocoder service page). The Placemark element represents the address you specified in the call, and provides a structured representation of that street address according to XAL, the eXtensible Address Language that can represent addresses from around the world (which is why the structure may seem so arcane—it has to handle so many different address structures for different countries).

The AddressDetails element also contains an Accuracy attribute, denoting how specifically the XML describes a location on Earth. For example, if you had only specified "New York, NY", you would get a lower Accuracy rating because the Geocoder has less information to work with and would be unable to provide more specific information about the address like the postal code. In that case, it provides a longitude and latitude correct for some point within New York State but not more specific than that. Accuracy is not a rating or degree of confidence in the result, but a measure of the specificity of the input.

For easier use of this XML response, GoogleGeocoder finally creates a GoogleGeocoderResult object to encapsulate the XML traversal as in Listing 5.

**Listing 5. The GoogleGeocoderResult class (PHP)**

```php
class GoogleGeocoderResult {
    public function __construct ($xml) {
```
The `GoogleGeocoderResult` uses PHP's SimpleXML module to make accessing the XML structure nearly trivial. SimpleXML maps an XML document's element tree into a PHP object tree, representing sub-elements as properties. `SimpleXMLElement` takes an XML string and then effectively turns the XML sub-elements into PHP properties of the `SimpleXMLElement` object. If the XML element in question contains attributes, you can also access the `SimpleXMLElement` as a PHP associative array, where each key in the array represents an attribute on the XML element.

First, the `GoogleGeocoderResult` constructor obtains the response code from the `SimpleXMLElement`:

```php
$this->resultCode = intval($this->xml->Response->Status->code);
```

The `$this->xml` portion shown here is the `SimpleXMLElement` created from the XML document. When you access its Response property, it returns a `SimpleXMLElement` object that represents the Response element in the KML response (the top-level `SimpleXMLElement` represents the top-level KML element in the XML tree). If an element contains more than one sub-element of the same name, those sub-elements will be turned into an array property. For example, suppose the Geocoder response returned multiple codes within the State element, you would access the first one with `$this->xml->Response->Status->code[0]`. `SimpleXMLElement` also converts transparently into a PHP string, so that you can pass the code element directly into `intval()` to convert it to a number, rather than call a method on it to obtain its text value.
After obtaining the response code, if it is not 200, this constructor throws a GoogleGeocoderException (see the example code) that maps the Geocoder error code into the appropriate error message (copied directly from the reference table on Google's site).

Then the constructor holds onto the Placemark element (as a SimpleXMLElement object) to represent the specified location on Earth. All other information that you want is from elements under this one. Note that Placemark is accessed here as an array. The Geocoder's response might contain multiple Placemarks for a given address, but SimpleXML lets you access any element as an array because it can't tell from just the XML whether to interpret the element as an array or a single object.

Next it obtains the Accuracy of the response, obtaining the value of the Accuracy attribute of the AddressDetails element:

```php
$this->accuracy = intval($this->placemark->AddressDetails['Accuracy']);
```

The SimpleXMLElement makes attribute values available as associative array properties on the SimpleXMLElement itself—SimpleXMLElement is effectively a string, an associative array, and a SimpleXMLElement object at once, allowing for such elegant syntax when traversing the XML tree. The constructor then obtains the longitude and latitude coordinates of the location, which appear in the result as `<coordinates>73.980182,40.746595,0</coordinates>`—longitude, latitude, altitude (altitude is always zero when you look up an address).

To obtain the postal code, getZipcode() first makes sure that the response is accurate enough to contain one, using the GoogleGeocoderAccuracies class, which contains constants for each of the Accuracy values found in Google's reference documentation (see Resources).

**Calling the Geocoder from JavaScript**

As an alternative to calling the Geocoder from PHP, you can call it using the GClientGeocoder JavaScript provided in the Google Maps API library. Then, use the response information to set hidden inputs in the form before you submit it to the server. Listing 6 demonstrates how you might do this.

**Listing 6. Calling the Geocoder from JavaScript**

```javascript
function onSubmitAddressForm() {
  var addressInput = document.getElementById('addressInput');
  var cityInput = document.getElementById('cityInput');
  var stateInput = document.getElementById('stateInput');

  var fullAddress = addressInput.value +", " + cityInput.value +", " + stateInput.value;

  var geocoder = new GClientGeocoder();

  geocoder.getLocations(fullAddress, submitAddressFromGeocoderResponse);
}
```
This uses the HTML form found in add-address.php (see Downloads) containing the address, city, and state text inputs to submit, and a button such as <button onclick="onSubmitAddressForm()">Submit</button>, that calls this function to handle the form submission. The onSubmitAddressForm() function obtains the building address, city, and state from the form elements, creates the complete address string from them, creates a new GClientGeocoder object, and calls its getLocations() method, passing in the full address to look up and the callback function to call with the response object (see Listing 7).

**Listing 7. Using the Geocoder callback response in JavaScript**

```javascript
function submitAddressFromGeocoderResponse(response) {
    if (! response || response.Status.code != 200) {
        alert("Geocoder did not recognize address. Code = " + response.Status.code);
    } else {
        var coordinates = response.Placemark[0].Point.coordinates;
        var longitude = coordinates[0];
        var latitude = coordinates[1];
        document.getElementById('longitudeInput').value = longitude;
        document.getElementById('latitudeInput').value = latitude;
        document.getElementById('addressForm').submit();
    }
}
```

GClientGeocoder calls the submitAddressFromGeocoderResponse function when it receives the response of the Geocoder service, which is in JavaScript Object Notation (JSON) format (see Resources for more information). JSON is the increasingly preferred alternative to XML when fetching hierarchical data from JavaScript because it provides a concise, human-readable, and unambiguous representation for a JavaScript object tree that has the same structure as the corresponding XML response data shown earlier. You can see an example of the JSON response structure in Google’s reference documentation (see Resources). Note that the Placemark element is actually an array of Placemarks, as multiple Placemarks might correspond to an address.

SubmitAddressFromGeocoderResponse checks the response’s status code just as the PHP code did, and retrieves the coordinates value using the same object structure as when using SimpleXML. It then sets hidden input values in the page’s form and submits the form programmatically for storage in the database.

**Summary**

You have now completed the first part of your application. You called the Geocoder service and stored the coordinate information in the database. In Part 2 you'll use stored procedures to produce XML data from a MySQL query, XSLT to transform that data into KML overlay data, and the Google Maps API to display the KML on a map embedded in your Web site.
### Downloadable resources

<table>
<thead>
<tr>
<th>Description</th>
<th>Name</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Part 1 example source code</td>
<td>google-maps-series-code.tar</td>
<td>100KB</td>
</tr>
</tbody>
</table>

© Copyright IBM Corporation 2008

**Trademarks**
(www.ibm.com/developerworks/ibm/trademarks/)