Where in the World Are Carmen and San Diego?

KEEPING DISTANCE

Fasserhaus, photocase.com

Google Maps can pinpoint any location on the globe and more. Here's how to embed the service into any web application. **BY ALBERTO PLANAS AND MARTIN STREICHER**

Google Maps is used pervasively across the web; it is accurate, feature-rich, and easy to integrate with any site. Indeed, it's something of a de facto standard. Google uses its eponymous service to plot results for business and street address searches, and organizations from local government to the media use Google Maps to depict statistical and demographic data. Increasingly, web applications embed Google Maps to chart the locales of everything and anything.

For example, My Tweet Map [1] displays where your Twitter friends live, whereas Physicians Resources [2] plots the location and ranking of area hospitals. One such hospital map, of metropolitan Chicago medical facilities, is shown in Figure 1.

Currently, Google Maps API has two versions: v2 is the latest stable release, and v3 is in latter-phase testing. Both are viable solutions, yet you might prefer one over the other, depending on your requirements.

Version 2 provides an extensive set of JavaScript classes to customize your maps and create applications [3]. For example, v2 provides classes to superimpose polygons and polylines, which are useful for routes and measurements. Version 3 of the API has fewer features so far but streamlines the API [4]. The footprint of v3 is much smaller, which speeds download and load times, especially on handheld devices.

Furthermore, the v3 programmatic interface is a variant of the Model-View-Controller (MVC) pattern, making it more concise and akin to frameworks like Ruby on Rails and CakePHP. Version 2 requires an access key, which you can obtain for no charge after completing a simple form. Version 3 no longer requires an access key.

Here, I'll examine simple applications in v2 and v3. Extensive examples can be found on the Google Maps project page and in the v2 and v3 documentation. Google also provides a new feature called the Code Playground [5], where you can interactively run and debug Google Maps and other code within your browser. The Code Playground has examples of every Google API.

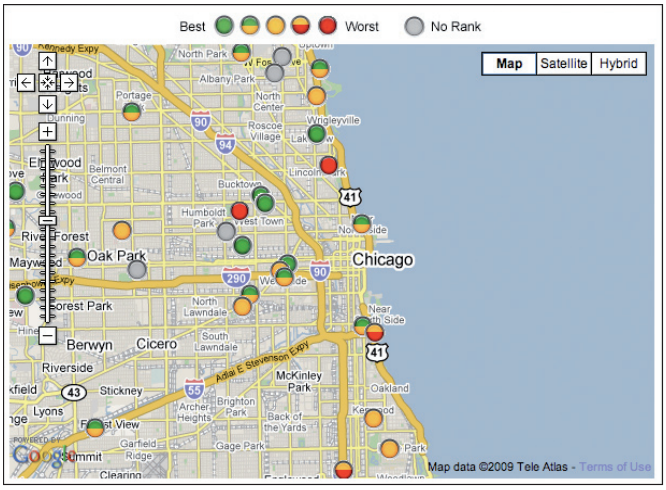

Figure 1: The quality of care among metropolitan Chicago hospitals.

Google Maps Version 2

To use v2, you must acquire an access key. If you want to follow the code examples in this article, acquire an access key and save your key for future reference.

Listing 1 shows a simple v2 application. The map is centered around downtown Ra-

leigh, North Carolina (Figure 2). To avoid problems with mixed encodings, specify the character set explicitly with a *meta* tag. By default, Google Maps uses UTF-8. The *xmlns* attribute in the *html* element and the contents of the *style* section are boilerplates to improve rendering of maps in Internet Explorer.

The string *YOUR_86_CHAR_KEY* should be replaced with the actual access key that Google grants you. This field is required, and your application cannot work without it. v2 also imposes some usage limits, which you can read about when you sign up for your key [6].

The method *GBrowserIsCompatible()* determines whether the browser can run Google Maps. If the browser is unsuitable, nothing appears. Otherwise, the next five lines render the map. The bulk of the work occurs in the *onLoad()* function, which executes after the page is loaded in its entirety.

The phrase *new GMap(document.getElementById("map"))* creates a map and associates it with the named HTML element, *map*. If you glance to the end of the code, you'll see that *map* is the ID of a *div*. The next three lines add the familiar pan, scroll, and zoom control at the top left, the map type control at top right, and the scale legend at bottom left, respectively (Figure 2).

The last line of the function sets the center point of the map (by geographic coordinates) and the zoom scale. Previous versions of the Google Maps API used the class *GPoint* to refer to geographical coordinates. Now you should use *LatLng* for points on the globe and *GPoint* as a point on the map in pixel coordinates.

The v2 API provides a vast number of features to build custom map applications. For example, you can create and place markers, bounding boxes, information boxes, and a lot more. Additionally, you can respond to user interface events, such as mouse clicks and drags.

Listing 2 shows an application that measures the distance (as the crow flies) between two points.

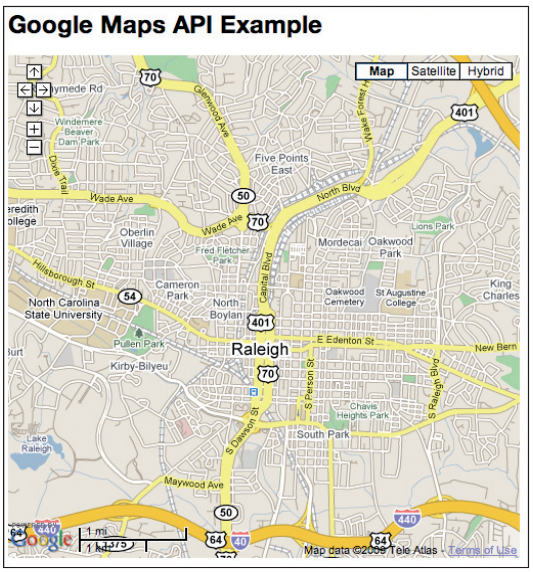

Figure 2: A map of downtown Raleigh, North Carolina.

The application has two status areas. The first area, with ID *latlong*, reflects the geographic coordinates of the center of the map. If you click, drag, and release, the coordinates adjust to reflect the new center. The second status area, with ID *distance*, emits the distance between any arbitrary two points chosen on the map.

Both status areas update when a particular event occurs. The former reacts to a *moveend*, sent when you stop dragging the map. The latter responds whenever you click the mouse. When either event occurs, Google Maps calls the function associated with the event. Each function is called a *listener* and is a common paradigm in user interface programming. The use of *addListener()* associates a listener with an event for a specific map.

The listener associated with the event *moveend* is an anonymous, straightforward function that finds the center of the map and changes the contents of the *latlong* area. The *click* listener is a little more complicated:

```
function(overlay, point) {
if (overlay) {
   removeOverlay(map, points, overlay);
} else if (point) {
   addOverlay(map, points, ↵
              new GMarker(point));
}
```

Listing 1: A Simple Google Maps V2 Application

```
01 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
02     "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
03 <html xmlns="http://www.w3.org/1999/xhtml"
04              xmlns:v="urn:schemas-microsoft-com:vml">
05   <head>
06     <meta http-equiv="content-type" content="text/html;
                        charset=utf-8"/>
07     <title>Listing 1</title>
08     <style type="text/css">
09       v\:* {
10         behavior:url(#default#VML);
11       }
12     </style>
13     <script type="text/javascript"
14       src="http://maps.google.com
              maps?file=api&v=1&key=YOUR_86_CHAR_KEY">
15     </script>
16     <script type="text/javascript">
17       //<![CDATA[
18       function onLoad() {
19         if (GBrowserIsCompatible()) {
20           var map = new GMap(document.
                                 getElementById("map"));
21           map.addControl(new GSmallMapControl());
22           map.addControl(new GMapTypeControl());
23           map.addControl(new GScaleControl());
24           map.centerAndZoom(new LatLng
                                ( 37.4419, -122.1419 ), 4);
25         }
26       }
27       //]]>
28     </script>
29   </head>
30
31   <body onload="onLoad()">
32     <div id="map" style="width: 500px;
        height: 500px"></div>
33   </body>
34 </html>
```

```
polyLine = ➘
   drawLine(map, points, polyLine);
var distance = calcDistance(points);
document.getElementById➘
   ("distance").innerHTML ➘
   = distance + " Km"; 42.}
```

To compute the distance between two points, you must place the two points on the map. To compute a new distance, one or both points must be deleted and then one or both must be re-positioned. That's the purpose of the anonymous function associated with the *click* event.

- When you click on the map, you click either on an overlay (a superimposed element) or a point on the map. Initially, the map has no overlays. If you click on the map, a marker overlay is added. If you click again on the map, a new marker is added. If, however, you click on a marker, it's deleted. The *if* statement implements that logic.
- When you place a second marker on the map, a line is drawn between the first marker and the new marker. The distance between the two markers is calculated by the traditional spherical distance formula and is emitted to the status area.
- If you place a third and subsequent marker, a line is drawn from the penultimate marker to the current marker and the distance is added to the total of all distances shown.
- When you click on a marker and remove it, the line between it and its preceding point (if any) and the line between it and its following point (if any) are removed, and the distances are then recalculated with the markers that remain.

Figure 3 shows the output from the code of Listing 2 with a number of markers added.

Figure 3: Measuring the length of the airport area near Málaga, Spain.

Google Maps version 2 offers many compelling overlays and is ideal for rich browser-based applications that you might expect a user to run on a desktop. For more ordinary navigation needs,

Listing 2: Measuring Distance Between Two Markers

```
001 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
002   "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
003 <html xmlns="http://www.w3.org/1999/xhtml"
       xmlns:v="urn:schemas-microsoft-com:vml">
004   <head>
005     <title>Calculating distances</title>
006     <style type="text/css">
007       v\:* {
008         behavior:url(#default#VML);
009       }
010     </style>
011
012     <script type="text/javascript"
          src="http://maps.google.com/maps?file
               =api&v=1&key=YOUR_86_CHAR_KEY" >
014     </script>
015     <script type="text/javascript">
016       //<![CDATA[
017       var points = new Array;
018       var polyLine;
019
020       function onLoad() {
021         if (GBrowserIsCompatible()) {
022           var map = new GMap(document.getElementById
                                  ("map"));
023           map.addControl(new GSmallMapControl());
024           map.addControl(new GScaleControl());
025
026           GEvent.addListener(map, 'moveend', function() {
027             var center = map.getCenterLatLng();
028             var latLngStr = '(' + center.y + ', '
                                  + center.x + ')';
029             document.getElementById("latlong").innerHTML
                  = latLngStr; });
030
031           GEvent.addListener(map, 'click',
                                  function(overlay, point) {
032             if (overlay) {
033               removeOverlay(map, points, overlay);
034             } else if (point) {
035               addOverlay(map, points, new GMarker(point));
036             }
037
038             polyLine = drawLine(map, points, polyLine);
039             var distance = calcDistance(points);
040             document.getElementById("distance").innerHTML
                  = distance + " Km"; 42.});
041
042             map.centerAndZoom(new GPoint(-4.48333,
                                          36.66667), 4);
043         }
044       }
045
046       function drawLine(map, points, lastLine) {
047         var p = new Array();
048
049         for (var i = 0; i < points.length; i++) {
050           p.push(new GPoint(points[i].getPoint().x,
                                  points[i].getPoint().y));
```

which you might expect to find on a mobile device, turn to version 3.

Google Maps Version 3

For comparison, Listing 3 and Figure 4 show a simple v3 application for a desktop web browser. When the browser finishes loading the document, *function onLoad()* runs and renders the map. Unlike earlier versions of the Google Maps API, v3 uses the new class *LatLng* to specify points on the globe; *latlng* points to the center of Silicon Valley.

Like other MVC frameworks, v3 uses setters and getters to customize objects. The array *options* provides a shorthand that lets you set a collection of attributes at once. The *zoom* attribute, which ranges from *0* (the entire globe) to *20* (an individual street), is set to *12*; the scale of the map is shown; the map is centered around the coordinates recorded in *latlng*; and the map type is *ROADMAP* (a street map).The other available map types are the self-descriptive *SATELLITE*, *HYBRID*, and *TERRAIN*. The initial zoom value, map center, and map type are required; *scaleControl* is optional and is false by default.

The *new google.maps.Map(document.getElementById("map"), options)* statement renders the map. The HTML element *map* – the 500-square-pixel *div* – provides the page real estate for the map, and *options* describes the initial state of the map.

Each instance of the JavaScript class *Map* represents a single map. Although you can have multiple maps on a page, each must have its own instance. *LatLng* represents only geographic coordinates. To convert an address to a coordinate, or vice versa, use the new *Geocoder* class. Listing 4 shows an example. If you type an address, the map updates to show the location.

Listing 3: A Simple Google Maps V3 Application

```
01 <html>
02   <head>
03     <title>Listing 3: A map of
         Silicon Valley</title>
04     <script type="text/javascript"
       src="http://maps.google.com/maps/
         api/js?sensor=false"></script>
05     <script type="text/javascript">
06       function onLoad() {
07         var latlng = new google.maps.
           LatLng( 37.4419, -122.1419 );
08         var options = {
09           zoom: 12,
10           center: latlng,
             mapTypeId: google.maps.
             MapTypeId.ROADMAP };
12         var map = new google.maps.
         Map( document.getElementById
           ("map"), options );
13       }
14     </script>
15   </head>
16
17   <body onload="onLoad()">
18     <div id="map" style="width:
         500px; height: 500px;"></div>
19   </body>
20 </html>
```

Listing 2: Measuring Distance Between Two Markers (cont'd.)

```
051       }
052
053       var newLine = new GPolyline(p);
054       if (lastLine) {
055         map.removeOverlay(lastLine);
056       }
057
058       map.addOverlay(newLine);
059       return newLine;
060     }
061
062     function addOverlay(map, points, overlay) {
063       map.addOverlay(overlay);
064       points.push(overlay);
065     }
066
067     function removeOverlay(map, points, overlay) {
068       map.removeOverlay(overlay);
069       var oi = -1;
070       for (var i = 0; i < points.length; i++) {
071         if (points[i] == overlay) {
072           oi = i;
073           break;
074         }
075       }
076
077       points.splice(oi, 1);
078     }
079
080     function calcDistance(points) {
081       var distance = 0.0;
082       var p1 = points[0];
083       for (var i = 1; i < points.length; i++) {
084         var p2 = points[i];
085         var lat1 = p1.getPoint().y * Math.PI / 180.0;
086         var lon1 = p1.getPoint().x * Math.PI / 180.0;
087         var lat2 = p2.getPoint().y * Math.PI / 180.0;
088         var lon2 = p2.getPoint().x * Math.PI / 180.0;
089         distance += 6378.7 * Math.acos(Math.sin(lat1) *
              Math.sin(lat2) + Math.cos(lat1) *
              Math.cos(lat2) * Math.cos(lon2 - lon1));
092         p1 = p2;
093       }
094
095       return distance;
096     }
097   //]]>
098   </script>
099 </head>
100
101 <body onload="onLoad()">
102   <div id="map" style="width: 500px; height: 500px">
      </div>
103   <div id="latlong"></div>
104   <div id="distance"></div>
105 </body>
106 </html>
```

The function *callAddress()* (from v3 documentation, used with permission) performs the heavy lifting. When the button is pressed, the *Geocoder* object calls the Google Geocoding service to transform the address to one or more coordinates. If the result is *OK* and the result set is non-empty, the marker is placed at the location of the first result. Figure 5 shows a map of Omaha, Nebraska. (If you call the Google Geocoding service through the *Geocoder* class, you do not need an access key. If you call the service directly through HTTP, you must acquire a separate access key.)

Figure 4: A map of Silicon Valley.

Figure 5: A map of the place named in the input box.

Markers are just one of several *overlays* you can add to a map to provide context, cues, and information. Additionally, you can add shadows and windows to display text. At the moment, v3 is not as extensive as v2, but its light weight is nonetheless compelling.

Listing 4: Drawing a Map of Any Address in the World

```
01 <html>
02   <head>
03     <title>Listing 4: A Map of any
        place</title>
04     <script type="text/javascript"
       src="http://maps.google.com/maps/
       api/js?sensor=false"></script>
05     <script type="text/javascript">
06       var geocoder;
07       var map;
08
09       function onLoad() {
10         geocoder =
             new google.maps.Geocoder();
11         var latlng = new google.maps.
           LatLng( 37.4419, -122.1419 );
12         var options = {
             zoom: 12,
             scaleControl: true,
             center: latlng,
             mapTypeId: google.maps.
                MapTypeId.ROADMAP };
13         map = new google.maps.
            Map( document.getElementById
                ("map"),options );
14       }
15
16       function codeAddress() {
17         var address = document.
       getElementById("address").value;
18         geocoder.geocode(
           { address: address },
           function(results, status) {
19           if (status == google.maps.
              GeocoderStatus.OK &&
                 results.length) {
20             if (status != google.maps.
                 GeocoderStatus.
                 ZERO_RESULTS) {
21               map.set_center
                  (results[0].geometry.
                   location);
22               var marker = new google.
                  maps.Marker({
23                 position: results[0].
                     geometry.location,
24                 map: map});
25             }
26           } else {
27             alert("Geocode was
       unsuccessful due to: " + status);
28           }});
29       }
30     </script>
31   </head>
32
33   <body onload="onLoad()">
34     <div id="map" style="width:
        500px; height: 500px;"></div>
39     <div>
40       <input id="address"
          type="textbox"
          value="Palo Alto, CA">
41       <input type="button"
          value="Geocode"
          onclick="codeAddress()">
42     </div>
43   </body>
44 </html>
```

Google Maps is Fun

The Google Maps API makes it easy to generate apps that would otherwise require extensive programming knowledge, special databases, and custom software. The v2 API supplies another group of objects for AJAX. From the client's browser, you can paint terrestrial coordinates stored in a database in many ways. Apps that use this part of the API include Monuments in Paris [7], WikiMap [8], and Traffic in the UK[9]. ■

INFO

[1] My Tweet Map: *http://www.mytweetmap.com*

[2] Physicians Resources: *http://www.netdoc.com/hospital-rankings/*

[3] Google Maps version 2: *http://code.google.com/apis/maps/documentation/reference.html*

[4] Google Maps version 3: *http://code.google.com/apis/maps/documentation/v3/*

[5] Google Code Playground: *http://code.google.com/apis/ajax/playground/*

[6] Google Maps version 2 sign-up: *http://code.google.com/apis/maps/signup.html*

[7] Monuments in Paris: *http://www.kahunablog.de/gmaps.php?map=paris*

[8] WikiMap: *http://www.wikyblog.com/Map/Guest/Home*

[9] Traffic in the UK: *http://www.gtraffic.info*