Table of Contents

Google Maps Hacks By Schuyler Erle, Rich Gibson
ISBN: 0596101619 Publisher: O'Reilly Print Publication Date: 1/1/2006

Chapter 6. API Tips and Tricks

6.1. Hacks 51–61: Introduction

The official Google Maps API makes a lot of things easy. Here are some hacks that push the API in new directions and extend the API with external libraries. Tricks such as filling your whole 21-inch flat panel display with a map, customizing your info windows with an embedded map the way the driving directions work, and integrating Flash with your maps.

You can even add custom labels and photographs on top of your Google Maps. Perhaps this technique reaches the height of elegance (absurdity?) when used to compare the size of Burning Man with New York's Central Park.

The chapter ends with several tricks to allow you to use one developer's key for multiple domains and directories. Do you really want to manage multiple keys because you have the domain http://mydomain.com and you serve the same pages from it and http://www.mydomain.com? I didn't think so!

Hack 51. Make a Fullscreen Map the Right Way

Map too big? Map too small? Flex those pixels into shape!

Imagine you're the proud new owner of a big, shiny 21-inch flat-panel display, and you surf to your favorite Google Maps web site—only to find it still confines you to a tiny little 3-inch-square map. It's an embarrassment, to say the least. Likewise, think about those poor fellows with small monitors who are scrolling until their fingers fall off to get around your gigantic 1600 x 1200–pixel map. Such sites ought to come with warning labels about claustrophobia.

A good Google Map stretches itself to fit comfortably on your screen. In this hack we'll investigate a simple two-step approach to make your map always take up the whole window, then build on the technique to include a fixed right-side panel, similar to the *maps.google.com* web site. We'll also point out some tricks to help you avoid common pitfalls that unsuspecting web developers stumble into when trying to make auto-sizing maps.

6.2.1. Making a Map Take Up the Full Screen

The first step is to include the following style declaration in the `head` section of your page. If your map `div` is named something other than `map`, you'll need to replace `#map` with the appropriate name (e.g., `#mymap`).

Google Maps Hacks By Schuyler Erle, Rich Gibson
ISBN: 0596101619 Publisher: O'Reilly Print Publication Date: 1/1/2006

```
<style type="text/css">
       html, body, #map {
           width: 100%;
           height: 100%;
       }
       html {
           overflow: hidden;
       }
       body {
           margin: 0px 0px 0px 0px;
           padding: 0px 0px 0px 0px;
       }
</style>
```

The `width` and `height` declarations cause the map container to fill the entire document body, and the body to take up all the available window space. Make sure you don't have any inline `width` or `height` attributes on your map element that could override these. The `overflow:hidden` line hides any scrollbars the browser would otherwise display, and the `margin` and `padding` lines get rid of the white border that is shown by default around a document. This leaves us with a map container that stretches to fit all the available window space in the browser (minus that reserved for the toolbars, status bar, etc.).

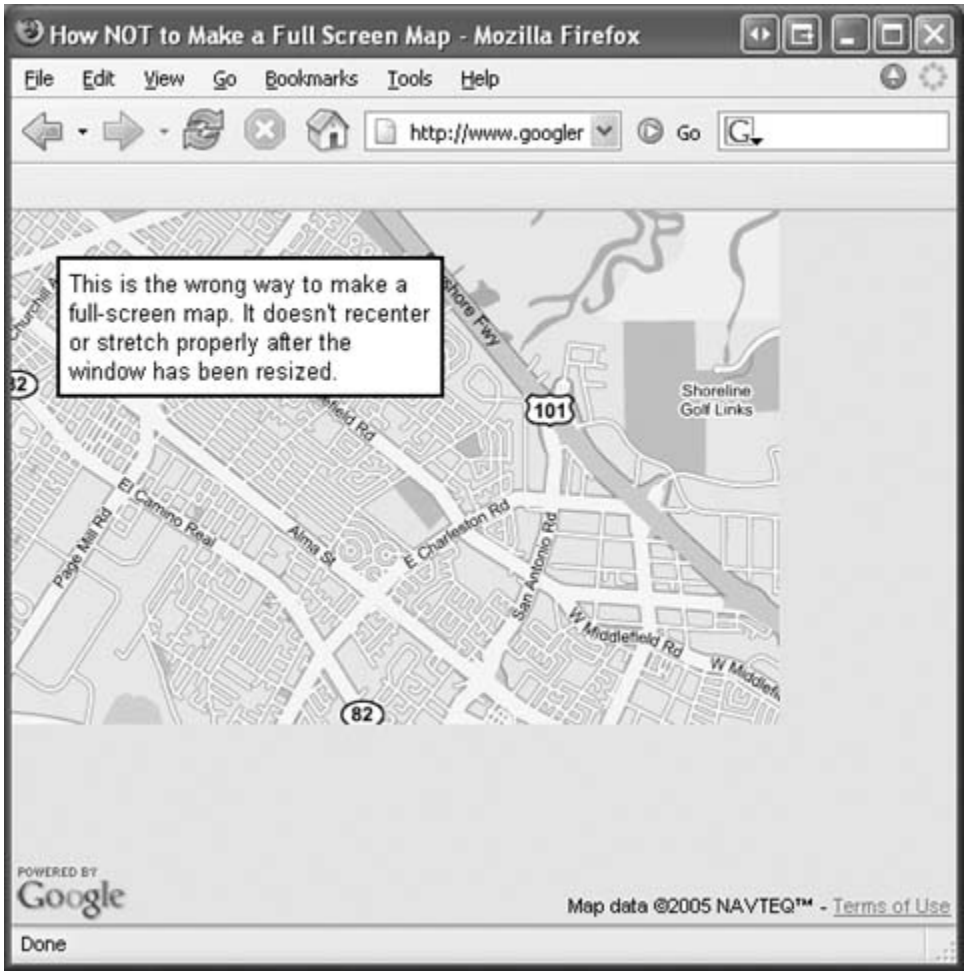
The Google map automatically takes up this whole container when you instantiate it, and most web developers stop here, thinking all is well. However, somewhat less obvious is the fact that the map doesn't detect when its container has been resized—at least, not as of version 1 of the API. Resizing a map that isn't aware when its container size changes can produce some misleading effects, which you can see by going to http://www.googlemappers.com/articles/fullscreenmap/bad.htm.

When you enlarge the browser window, you'll see more of the map revealed and might be tempted to *think* it is stretching to fit your window. But what you are actually seeing is the "off-screen" portion of the map—surrounding tiles that have been preloaded, but aren't supposed to be showing on your screen.

If you open the web page in a very small browser window and then enlarge it enough, you'll run out of off-screen tiles and will start to see the gray background shown in Figure 6-1. Additionally, double-clicking the map will make it pan the point you clicked to the *old* center of the window, based on its dimensions at the time it was created.

Google Maps Hacks By Schuyler Erle, Rich Gibson
ISBN: 0596101619 Publisher: O'Reilly Print Publication Date: 1/1/2006

Figure 6-1. A full-screen map done the wrong way



To make your map resize and re-center properly, you need to let it know when its container has resized. To do this, we make use of an undocumented Google Maps API method called `map.onResize()`, calling it whenever the window dimensions change. Include the following JavaScript code, after you create your map object:

```
if (window.attachEvent) {
    window.attachEvent("onresize", function( ) {this.map.onResize( )} );
    window.attachEvent("onload", function( ) {this.map.onResize( )} );
} else if (window.addEventListener) {
    window.addEventListener("resize",
      function( ) {this.map.onResize( )} , false);
    window.addEventListener("load", function( ) {this.map.onResize( )} ,
false);
}
```

This is the final step needed to obtain a properly working fullscreen map. You can see the complete implementation of these two code snippets at http://www.googlemappers.com/articles/fullscreenmap/default.htm. You may want to check there for the most up-to-date example of the fullscreen map.

Google Maps Hacks By Schuyler Erle, Rich Gibson
ISBN: 0596101619 Publisher: O'Reilly Print Publication Date: 1/1/2006

6.2.2. Adding a Side Panel to the Map

It's easy to add a fixed-width panel alongside an auto-sizing map. Here we show the code needed for a fullscreen map with a panel 300 pixels wide docked to the right side of the screen.

Replace the style declaration used in the `head` section with the following one:

```
<style type="text/css">
       html, body {
           width: 100%;
           height: 100%;
       }
       html {
           overflow: hidden;
       }
       body {
           margin: 0px 0px 0px 0px;
           padding: 0px 0px 0px 0px;
       }
       #map {
           margin-right: 302px;
           height: 100%
       }
       #rightpanel {
           position:  absolute;
           right:          0px;
           top:            0px;
           width:        300px;
           height:        100%;
           overflow:      auto;
           border-left: 2px solid black;
           padding: 0px 5px 0px 10px;
       }
</style>
```

The map's right margin is set to the width of the panel, plus another 2 pixels to account for its left border. The `overflow:auto` attribute in the right panel causes it to show scrollbars if the content is too big to fit in the panel. The border and padding are added just to make things look cleaner.

It's important that the `padding-top` and `padding-bottom` attributes for the right panel be `0px`, or the panel's scrollbar will extend underneath the window's status bar, as shown in Figure 6-2. This has to do with a bottom margin the Google Map API internals add below the map, which unintentionally makes the browser think the window content extends slightly below the screen.

Finally, in the body section declare your side panel right after your map `div`.

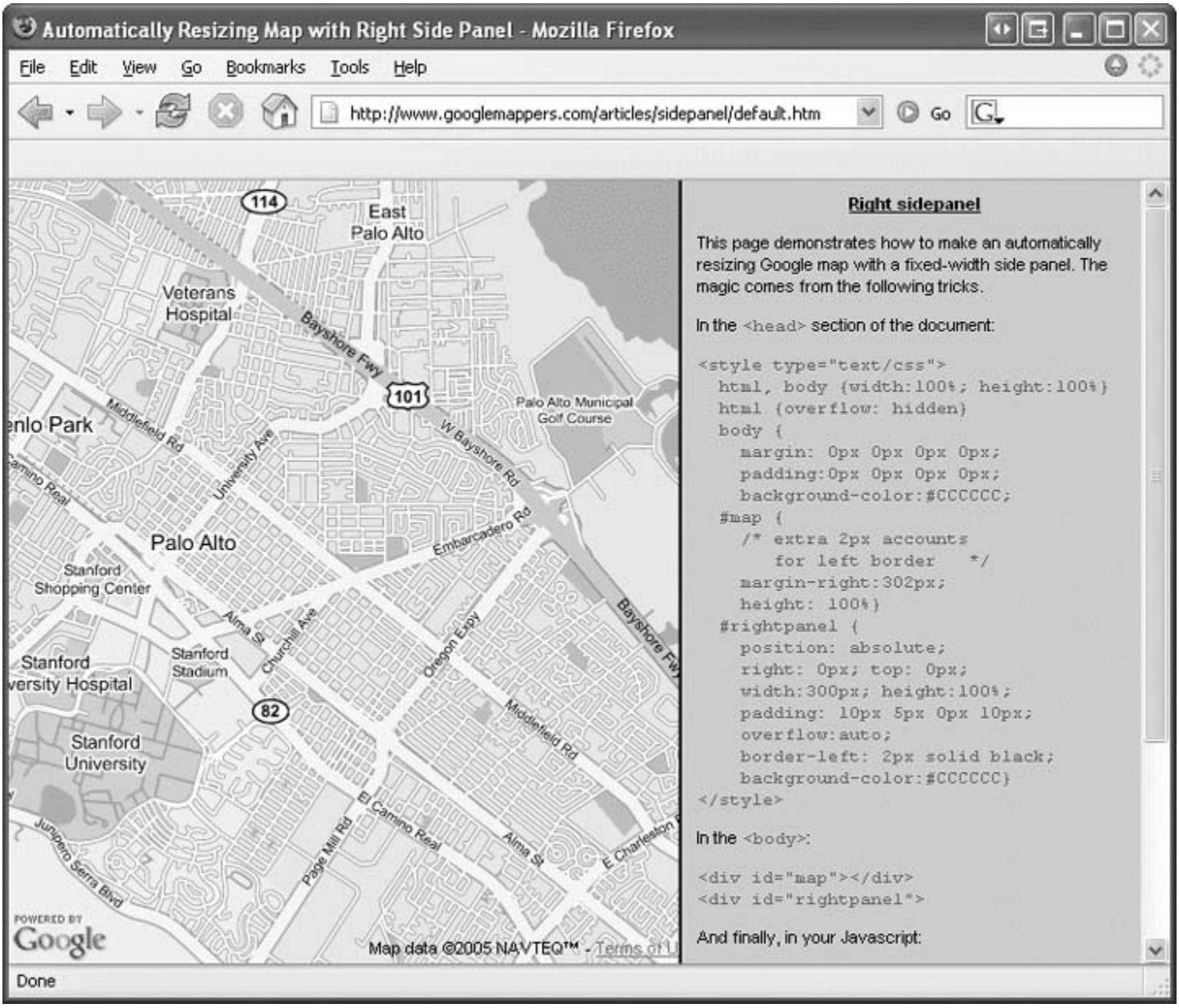
```
<div id="rightpanel"></div>
```

Figure 6-2. The unfortunate effects of including top and bottom side panel padding

Google Maps Hacks By Schuyler Erle, Rich Gibson
ISBN: 0596101619 Publisher: O'Reilly Print Publication Date: 1/1/2006

That's all there is to it! You can fill the side panel with any information you'd like. You can see an up-to-date example at http://www.googlemappers.com/articles/sidepanel/default.htm, which is depicted in Figure 6-3.

Figure 6-3. A properly configured side panel



—Richard Kagerer

Google Maps Hacks By Schuyler Erle, Rich Gibson
ISBN: 0596101619 Publisher: O'Reilly Print Publication Date: 1/1/2006

Hack 52. Put a Map and HTML into Your Info Windows

Add more context to your info windows by including a map and HTML.

There are times you want to include both a map and some text in the same info window. With a little hackery you can get the sort of effect demoed at http://mappinghacks.com/projects/gmaps/map_in_box.html and shown in Figure 6-4.

Figure 6-4. A map and text in the same info window

Google Maps Hacks By Schuyler Erle, Rich Gibson
ISBN: 0596101619 Publisher: O'Reilly Print Publication Date: 1/1/2006

Within the Google Maps API there is an info window class that enables you to click on a point or overlay and get additional information about whatever it was that you clicked on. According to the API documentation, there are really only two types of methods that create info windows. The first type is those that take some kind of HTML in one of their arguments, namely `openInfoWindow(), openInfoWindowHtml( )`, and `openInfoWindowXslt( )`. The second type is really just a single method that takes no HTML as input and produces a blow-up map of some point inside of the info window, but nothing else. This method is `showMapBlowup()`.

There are also times when we need both a map and text in the same window. However, when we look at the full prototype for the `showMapBlowup()` method, namely `showMapBlowup(point, zoomLevel?, mapType?, pixelOffset?, onOpenFn?, onCloseFn?)`, we see there is no way to insert text into a map blowup. How do we get a map and text into an info window?

When we look at the `openInfoWindowHtml( )`, we see that it has the prototype `openInfoWindowHtml(marker, htmlStr, pixelOffset?, onOpenFn?, onCloseFn?)`, where `htmlStr` is *any* string of HTML. Well, since we are using a `div` element to create the main map, why not try putting a `div` element into the `htmlStr`? As it turns out, that works almost wonderfully:

```
GEvent.addListener(testmarker, "click", function ( ) {
 var text = '<p style="text-align: left">';
 text += '<div id="minimap" style="width: 200px; height: 200px"></div>';
 text += 'Here is some text.<br>';
 text += 'This is a link to <a href=

              "http://maps.google.com/">Google Maps</a>';
 testmarker.openInfoWindowHtml(text);
 var minimap = new GMap(document.getElementById(minimap));
 minimap.centerAndZoom(pt,1);
 minimap.addControl(new GSmallMapControl( ));
});
map.addOverlay(testmarker);
```

Here `pt` is a previously defined point, and `testmarker` is a marker that we have to place somewhere. Note that, if you like, you can add your own controls to the blow-up map and put it into a different mode (satellite or hybrid).

However, there is a problem with all this. If we close the info window and then reopen it, we find that the map is gone! We haven't been able to figure out why this happens, but it turns out that there is a workaround:

```
    var count = 0;
    GEvent.addListener(testmarker, "click", function () {
     var text = '<p style="text-align: left">';

     var whichmini = "minimap" + count;
     text += '<div id="';

     text += whichmini + '" style="width: 200px; height: 200px"></div>';
     text += 'Here is some text.<br>';
     text += 'This is a link to <a href=
                 "http://maps.google.com/">Google Maps</a>
<br>';
          text += 'The current value of the infoWindow map counter is ' + count + '.';
          testmarker.openInfoWindowHtml(text);

          var minimap = new GMap(document.getElementById(whichmini));
          minimap.centerAndZoom(pt,1);
          minimap.addControl(new GSmallMapControl( ));

          count++;
         });
         map.addOverlay(testmarker);
```


Google Maps Hacks By Schuyler Erle, Rich Gibson
ISBN: 0596101619 Publisher: O'Reilly Print Publication Date: 1/1/2006

What we have done here is add in a counter. Each time we open the info window, the counter is incremented by one. So if we put this counter into the `id` for the `div` element, then each time that we open the `infoWindow`, we will be opening a new `div` element. Note that we need to reference this new name in the constructor for the mini-map as well.

6.3.1. See Also

- Part of the credit for this should go to april@syclo.com, who came up with the idea of putting a `div` element inside of the HTML in an info window. I came up with the code that added the counter and made it work more than once. See the thread at http://groups-beta.google.com/group/Google-Maps-API/browse_frm/thread/ab3075a8e91f8ecf/ for more details.

—John T. Guthrie

Hack 53. Add Flash Applets to Your Google Maps

Spice up your Google Map info windows with Macromedia Flash animation, and even Java applets.

Integrating Macromedia Flash with a Google Map is a snazzy way to enhance the multimedia experience of your web page. Although the examples given here are toys, no doubt the intrepid reader can find Useful Uses for this sort of integration in the real world.

6.4.1. Flash in the Info Window

As a first step, we'll embed a small Flash animation I've created called *FlashBit.swf* into the pop-up window of a marker. The animation is just some text with a blue ball that bounces across it when clicked, as seen in Figure 6-5.

We're going to use the `openInfoWindowHtml( )` method of the `GMarker` class to add the required HTML to our document. The string we'll be passing is stored in a variable called `flashHtml`.

```
flashHtml = '<object classid="clsid:d27cdb6e-ae6d-'
    + '11cf-96b8-444553540000" '
    + 'codebase="http://fpdownload.macromedia.com/pub/'
    + 'shockwave/cabs/flash/swflash.cab#version=7,0,0,0" '
    + 'width="128" height="53" id="FlashBit" align="middle">'
    + '<param name="allowScriptAccess" value="sameDomain" />'
    + '<param name="movie" value="FlashBit.swf" />'
    + '<param name="quality" value="high" />'
    + '<param name="bgcolor" value="#ffffff" />'
    + '<param name="wmode" value="transparent" />'
    + '<embed src="FlashBit.swf" quality="high" '
    +  'bgcolor="#ffffff" width="128" height="53" '
    +  'name="FlashBit" align="middle" '
```


Google Maps Hacks By Schuyler Erle, Rich Gibson
ISBN: 0596101619 Publisher: O'Reilly Print Publication Date: 1/1/2006

```
+  'allowScriptAccess="sameDomain" '
+  'type="application/x-shockwave-flash" '
+  'wmode="transparent" '
+  'pluginspage="http://www.macromedia.com/go'
+  '/getflashplayer" />'
+  '</object></center>';
```

Figure 6-5. Flash animation in a Google Maps info window

Although the code above may look a little convoluted, it's really nothing more than a typical Flash object declaration. The `object` tag is used by Internet Explorer on Windows platforms, while Netscape and Internet Explorer for the Mac both use the `embed` tag instead. More information on why you need both tags can be found in Macromedia's knowledge base at http://www.macromedia.com/cfusion/knowledgebase/index.cfm?id=tn_4150.

The most important parameters are `movie` and `src`, which should both point to the Flash SWF file. It's also important to set the `wmode` parameter to `transparent` in both the object and embed tags. Otherwise, the Flash object will overlap the map controls when the user scrolls the info window under them, as seen in Figure 6-6.

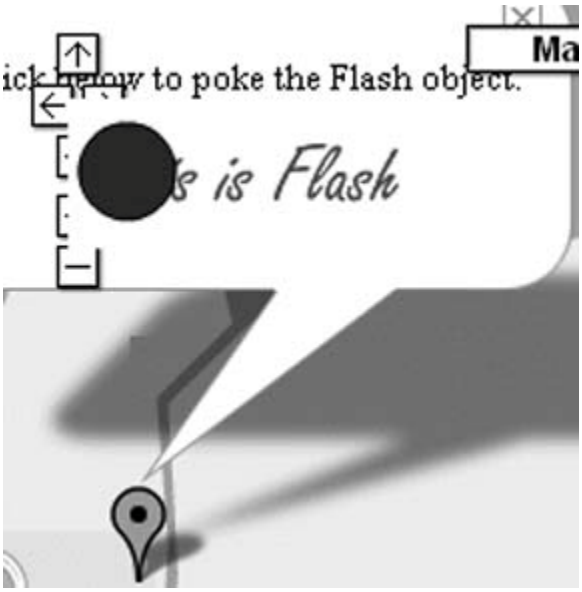
The following straightforward code creates our map. A marker called `flashMarker` is added to the map and, when clicked, it will call the `onFlashMarkerClick()` function:

```
var map = new GMap(document.getElementById("map"));
map.addControl(new GSmallMapControl());
map.addControl(new GMapTypeControl());
map.centerAndZoom(new GPoint(-122.07, 37.44), 4);
```


Google Maps Hacks By Schuyler Erle, Rich Gibson
ISBN: 0596101619 Publisher: O'Reilly Print Publication Date: 1/1/2006

```
var flashMarker = new GMarker(new GPoint(-122.07, 37.433));
map.addOverlay(flashMarker)
GEvent.addListener(flashMarker,'click', onFlashMarkerClick);
```

Figure 6-6. Overlap caused by not including the wmode parameter



Unfortunately, there's a slight bug in that the Flash applet can "jump" out of the window after being clicked if it is inside the very first info window that is opened. You can see the problem by going to http://www.googlemappers.com/articles/embedapplets/default.htm and clicking "Show me the bug."

However, I've found that if the map has to be scrolled to show the info window, the bug doesn't occur. So in the `onFlashMarkerClick()` handler, I've put in a small hack that scrolls the map away from the marker the first time `openInfoWindowHtml()` is called. This way, the Google Map API internals will scroll the map back to the marker before opening the info window, and the bug won't occur.

```
var flashFixed = false;

function onFlashMarkerClick() {

    html = '<center><div style="width:200px; font-size:10pt">'
        + '<br/>Click below to poke the Flash object.</div>';
    html = html + flashHtml;

    if (!flashFixed) {
        flashFixed = true; // only need to do this once
        map.centerAndZoom(new GPoint(-122.07, 37.43), 4);
    }

    // Now open the real one on top
    flashMarker.openInfoWindowHtml(html);

}
```


Google Maps Hacks By Schuyler Erle, Rich Gibson
ISBN: 0596101619 Publisher: O'Reilly Print Publication Date: 1/1/2006

6.4.2. Communication Between Flash and Your Google Map

Integrating Flash and Google Maps becomes much more interesting when you get the two talking to each other. You can make use of the Flash `fscommand( )` method to invoke any JavaScript function on the page, and use the `SetVariable( )` method in your JavaScript code to pass data back into your Flash applet. A complete tutorial is available in a Flash TechNote at http://www.macromedia.com/cfusion/knowledgebase/index.cfm?id=tn_15683#jtfc.

You can also take a look at my scuba-diving log at http://www.leapbeyond.com/ric/scuba/ to see an example of an application that tightly integrates a Flash web site with a Google Map. Just go into the logbook a few pages and hit one of the map icons to get started.

6.4.3. Beyond Flash

The technique described can work with other types of objects in addition to Flash files. For example, you can embed a Java applet into the info window. Figure 6-7 shows how I used Robert Jeppesen's Durius Java applet (available at http://www.durius.com/) to embed a Google logo that ripples with a watery effect when you pass the mouse over it.

Figure 6-7. A Java applet embedded in a Google Map info window

The second marker is added after creating your map:

```
var appletMarker = new GMarker(new GPoint(-122.082667, 37.423269));
map.addOverlay(appletMarker);
GEvent.addListener(appletMarker,'click', onAppletMarkerClick);
```


Google Maps Hacks By Schuyler Erle, Rich Gibson
ISBN: 0596101619 Publisher: O'Reilly Print Publication Date: 1/1/2006

Here's the click handler:

```
function onAppletMarkerClick() {

    html = '<center><div style="width:200px; font-size:10pt">'
        + '<br/>Move your mouse around the image below '
        + 'to poke the Java applet.</div>'
        + '<div style="width:200px; font-size:8pt"><br/>'
        + '<em>Note: Clicking it will leave this page</em></div>';

    // In the html string, create a div containing the applet
    html = html +
        '<div id="applet" style="width:128px; height:53px;">' +
        '       <applet archive="DuriusWaterPic.jar" ' +
        '       width="128px" height="53px" align="bottom" ' +
        '       code="DuriusWaterPic.class">' +
        '           <param name="cabbase" value="DuriusWaterPic.cab" />' +
        '           <param name="image" value="GoogleLogo.gif" />' +
        '           <param name="dotsize" value="3" />' +
        '           <param name="dim" value="2" />' +
        '           <param name="noise" value="0" />' +
        '           <param name="mouse" value="1" />' +
        '           <param name="delay" value="10" />' +
        '           <param name="orientation" value="v" />' +
        '           <param name="bg" value="233423" />' +
        '           <param name="reg" value="43752326" />' +
        '       </applet>' +
        '</div></center>';

    appletMarker.openInfoWindowHtml(html);
}
```

You can see all of these examples at http://www.googlemappers.com/articles/embedapplets/default.htm.

—Richard Kagerer

Hack 54. Add a Nicer Info Window to Your Map with TLabel

Callouts, info windows—anything that can fit on a web page—can fit on a map.

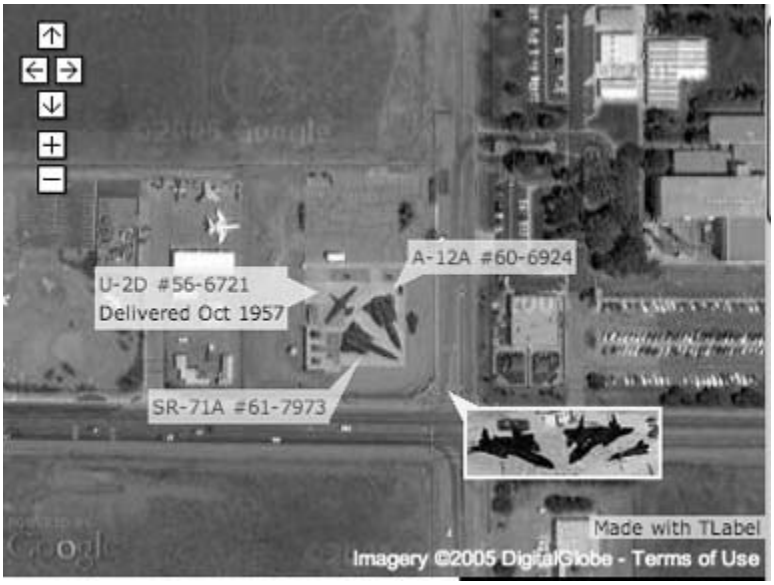
The early Google Maps hacks share a similar affliction: they suffer from red pushpin syndrome. Sometimes the pushpins are green. You can create custom icons, but they are a bit of a challenge, and then you just get a different little symbol.

What you (might) want is the ability to customize the map with unique info windows, images, callouts, and anything else that strikes your fancy. Tom Mangan created the TLabel extension to allow you to embed any DOM object (anything that can appear on a web page) on a Google Map. Figure 6-8 shows Tom's canonical example from the TLabel page at http://gmaps.tommangan.us/tlabel.html, with a Google satellite image of two U.S.

Google Maps Hacks By Schuyler Erle, Rich Gibson
ISBN: 0596101619 Publisher: O'Reilly Print Publication Date: 1/1/2006

Air Force SR-71 Blackbirds and a U-2 spy plane at the Blackbird Air Park, in Palmdale, CA. (For the curious, this is at 34.602975 N, 118.085926 W.)

Figure 6-8. SR-71s at the Blackbird Air Park, with TLabel annotations



Callouts identify each plane. The numbers are links to sites that have more information on each plane. There is also a thumbnail image that was taken from ground level. The image has a tool tip that reads "Click for larger." Surprisingly, clicking on that link leads to a larger image of that thumbnail.

6.5.1. Using TLabel

The TLabel page is at http://gmaps.tommangan.us/tlabel.html. There is a link to a JavaScript file, currently http://gmaps.tommangan.us/tlabel.19.js, but check for the current version. Copy that file onto your own server and include it after you include the Google Maps API.

```
<script src="http://maps.google.com/maps?file=api&v=1&key
      =AOJVHiMoLlyAv…x3sA" type="text/javascript"></script>

<script src="tlabel.10.js" type="text/javascript"></script>
```

Now, create some objects! You control the TLabel object by creating a new object and then setting the properties. There is a demo of some TLabel tricks at http://mappinghacks.com/projects/gmaps/brc_tlabel.html. Here is code to create a label, as shown in Figure 6-9:

```
var label = new TLabel();
label.id = 'mail_box';
label.anchorLatLng = new GPoint (-119.226396, 40.768080);
label.anchorPoint = 'bottomLeft';
label.content = 'Mail Box';
```


Google Maps Hacks By Schuyler Erle, Rich Gibson
ISBN: 0596101619 Publisher: O'Reilly Print Publication Date: 1/1/2006

```
label.percentOpacity = 85;
map.addTLabel(label);
```

Figure 6-9. Black Rock City, annotated with TLabels

First, the code declares the variable named `label` to hold the newly created TLabel object. Next, we set the `id` of the TLabel. The `id` is vital! It must be unique. It is easy, and catastrophic, to end up with a duplicate label (where *catastrophic* means it doesn't work). The `anchorLatLng` is a `GPoint` object, i.e., the location where this object is anchored to the earth. The `anchorPoint` defines which part of the content will touch the anchor lat and long.

In this case, `bottomLeft` means that the bottom-left corner of our marker hits the point in the `anchorLatLng`. The choices are `topLeft, topCenter, topRight, midRight, bottomRight, bottomCenter, bottomLeft, midLeft`, or `center`. The default is `topLeft`. The `content` can be anything that can fit on a web page! In this case, we use the words "Mail Box." Table 6-1 summarizes these properties.

Once you have created the TLabel object and set the properties, there are four useful methods: add the object to your map with `map.addTLabel(label)`, remove it with `map.removeTLabel(label)`, move it around with `label.setPosition(GPoint)`, or change its opacity (100% means you can't see through it at all) with `label.setOpacity(percentage)`.

Google Maps Hacks By Schuyler Erle, Rich Gibson
ISBN: 0596101619 Publisher: O'Reilly Print Publication Date: 1/1/2006

Table 6-1. Useful properties of the TLabel object

Property	Description
`id`	Specifies an ID for the label object. This ID is required and must be unique for each label you embed. The ID is exposed to the DOM, so you can dynamically adjust the label's style if you choose.
`anchorLatLng`	The longitude and latitude where the `anchorPoint` will be pinned to the map. Takes a `GPoint` object.
`anchorPoint`	The point on your embedded object that will be pinned to `anchorLatLng`. `anchorPoint` accepts the following values: `topLeft`, `topCenter`, `topRight`, `midRight`, `bottomRight`, `bottomCenter`, `bottomLeft`, `midLeft`, and `center`. The default is `topLeft`.
`content`	The XHTML code defining the element you wish to embed.
`percentOpacity`	A number between 0 and 100 inclusive, determining the opacity of the image. Default is 100, i.e., completely opaque.

You can replace the content with something more elaborate. This implements an `onMouseOver` event; so moving the mouse over the image pops up more information (i.e., a tool tip).

```
label.content = '<img src="images/blank.png" alt=""
   style="width: 12px; height: 20px;" onmouseover="showFloat(\'sInfo\');"
onmousemove="floatCoords(\'sInfo\',event);"
   onmouseout="hideFloat(\'sInfo\');" />';
```

It is all a question of controlling what is in the `content` variable. You can put the whole thing into one string or split it across lines. If you split it, you need to remember to terminate each line of a multiline section with a quote or you'll get an "unterminated string literal" message.

Check out Tom's site at http://gmaps.tommangan.us/ for more nifty hacks.

—written with assistance from Tom Mangan

Hack 55. Put Photographs on Your Google Maps

Treat photographs as map layers and add them to your Google maps.

While figuring out how to lay my own images into the Google Maps interface, I ended up with some pretty useful JavaScript, so I bundled it up as a tightly integrated extension to the official API. The result is TPhoto. TPhoto allows you to embed alternate aerial photographs inside your Google Maps. The added photos pan and zoom along with the main map view, without interfering with any clicks on the map. Figure 6-10 shows an example of an aerial photo of Groom Lake from 1959 overlaid over the Google Satellite imagery.

Google Maps Hacks By Schuyler Erle, Rich Gibson
ISBN: 0596101619 Publisher: O'Reilly Print Publication Date: 1/1/2006

Figure 6-10. Groom Lake, 1959

Groom Lake. Dreamland Resort. Watertown. The Ranch. For such a secretive place, people certainly know it by a lot of names! The image in Figure 6-10 comes from my Area 51 page at http://gmaps.tommangan.us/groom_lake.html. Figure 6-11 shows a 1968 image over the Google Satellite imagery.

You can add the ability to overlay photos onto your own Google Maps by installing the TPhoto extension.

Download the TPhoto library from http://gmaps.tommangan.us/tphoto.html/. The current version is *tphoto.16.js*, available at http://gmaps.tommangan.us/tphoto.16.js. Place it on your server and include it after you include the Google Maps API in your JavaScript. You could include it directly from the *tommangan.us* server, but it is on very rare occasions not available, so I recommend you download the file to your own server and link it as shown here:

```
<script src="http://maps.google.com/maps?file=api&v=1&key=ApAUl_iS-dEaDI.RA"
      type="text/javascript"></script>
<script src="tphoto.16.js" type="text/javascript"></script>
```

There are two ways to use TPhoto. In the first method, you supply the lat/long of the top left and bottom right corners of your image. In the second method, you supply a single lat/long anchor point, the base zoom level, and the dimensions of your image. The first method looks like:

Google Maps Hacks By Schuyler Erle, Rich Gibson
ISBN: 0596101619 Publisher: O'Reilly Print Publication Date: 1/1/2006

```
var groom1959 = new TPhoto();
groom1959.id = 'groom1959';
groom1959.src = 'images/groom1959.jpg';
photo.percentOpacity = 50;
groom1959.anchorTopLeft = new GPoint(-115.823107,37.248368);
groom1959.anchorBottomRight = new GPoint(-115.801649,37.230123);

map.addTPhoto(photo);
```

Figure 6-11. Area 51 in 1968, over Google Satellite imagery

Here is one way to create a button that will show and hide the overlaid image. This would appear in an HTML form button.

```
<input type="button" value="show 1959" id="show1959" onClick="showImage(); ">
```


Google Maps Hacks By Schuyler Erle, Rich Gibson
ISBN: 0596101619 Publisher: O'Reilly Print Publication Date: 1/1/2006

The button calls the `showImage()` Javascript method. The variable `show` serves as a toggle. It starts at `0` (i.e., don't show the image) and flips state each time the button is clicked. Either the `addTPhoto` method or the `removeTphoto` method will be called depending on the value of `show`. The text displayed on the button is also toggled from "show 1959" to "hide 1959."

```
var show = 0;
function showImage() {
        show = 1-show;
        if (show == 1) {
                map.addTPhoto(groom1959);
                document.getElementById('show1959').value = 'hide 1959';
        } else {
                map.removeTPhoto(groom1959);
                document.getElementById('show1959').value = 'show 1959';
        }
}
```

The second approach takes different parameters:

```
photo = new TPhoto();
photo.id = '[id]';
photo.src = '[src]';
photo.size = new GSize(width,height);
photo.baseZoom = [zoomLevel];
photo.percentOpacity = [percent];
photo.anchorPx = new GPoint(x,y);
photo.anchorLatLng = new GPoint(lng,lat);

map.addTPhoto(photo);
```

There are only a few methods you need to use with TPhoto.

`TPhoto()`

The constructor, which creates a new photo instance. Takes no parameters.

The extension adds two methods to the GMap object:

`addTPhoto` *`(photo)`*

Adds the given photo to the map.

`removeTPhoto` *`(photo)`*

Removes the given photo from the map.

`setOpacity` *`(percentOpacity)`*

Defines the desired opacity of the embedded image. *`percentOpacity`* is a number from 0 to 100, inclusive.

Setting opacity allows you to see through the overlaid image to the base layer. It allows you to do things like compare then and now imagery as shown in Figures 6-12 and 6-13. These are aerial views comparing the overlaid 1994 USGS photo over the Google aerial imagery. The first image shows the Google Aerial base layer view with the 1994 photo overlaying it and set to 20% opacity. You can see through the older image to how things are now (where *now* is when the image was taken).

The right view is at 100% opacity. In the left image you can see buildings that didn't exist in 1994. The two Blackbirds are a constant.

Figure 6-13 is an image of the Blackbird Airpark in Palmdale, California from the Blackbird Sighting page at http://gmaps.tommangan.us/blackbirds.html. The Blackbirds map was my first Google Maps project. Soon after the API was released, I found myself in need of a project that would enable me to go about breaking maps. As it happened, I had just read about Robb Magley's search for the crash site of one of the

Google Maps Hacks By Schuyler Erle, Rich Gibson
ISBN: 0596101619 Publisher: O'Reilly Print Publication Date: 1/1/2006

Blackbirds and thought it would be fun to find out where all of them are now. This is the kind of application that Google Maps has made possible!

—*Tom Mangan*

Figure 6-12. Blackbirds at 20% opacity

Hack 56. Pin Your Own Maps to Google Maps with TPhoto

Google Maps Hacks By Schuyler Erle, Rich Gibson
ISBN: 0596101619 Publisher: O'Reilly Print Publication Date: 1/1/2006

Georeference your own maps and then pin them anywhere.

The first time I visited New York City from the Bay Area, I was excited to see everything, all at once. I figured that Manhatten is small, about 10 1/2 miles from tip to tip, and the boroughs are just a bit larger. How long could it take to go 10 miles?

A temporary New Yorker friend told me, "You can think of Manhattan as being the whole Bay Area. Sure you can drive from Palo Alto to San Francisco, across to Oakland and down the East Bay to San Jose, but would you? Manhattan is physically smaller, but in terms of travel times it is similar."

We can use TPhoto **[Hack #55]** to take an image of one place and center it in another. At http://www.mappinghacks.com/projects/gmaps/brc_ref.html you see the Google aerial photo of the site of the Burning Man festival in the Black Rock desert of Nevada. If you click "show plan," you'll get the 2005 Burning Man city plan overlaid on the Google Map as shown in Figure 6-14.

Google Maps Hacks By Schuyler Erle, Rich Gibson
ISBN: 0596101619 Publisher: O'Reilly Print Publication Date: 1/1/2006

Figure 6-13. Blackbirds at 100% opacity

But you don't have to overlay the city plan on the Google Map in the proper place! You can scroll the map to another area and then click on a point to overlay the city map on that point. Figure 6-15 shows a hypothetical Burning Apple festival centered on 79th Street in the middle of Central Park.

It is interesting that Manhattan is oriented nearly the same way as the Playa, from southwest to northeast, and the Festival covers about two-thirds of the width of Manhattan. The Metropolitan Museum of Art ends up more or less by the Large Scale Sound installations.

The distance from Center Camp to the perimeter fence is about 27 city blocks—from 72nd Street to 99th Street. And moving out a few zoom levels leaves us in awe at how tremendously big and dense an experience it is to be in New York.

Taking it home, way home to Sebastopol, and centering on O'Reilly, gives us the results in Figure 6-16.

Google Maps Hacks By Schuyler Erle, Rich Gibson
ISBN: 0596101619 Publisher: O'Reilly Print Publication Date: 1/1/2006

This was produced with the single-point method of TPhoto **[Hack #55]**. This makes TPhoto work as a simple georeferencing tool. Georeferencing is the act of mapping the pixel space of an image to a coordinate system that is pegged to the earth's surface.

Figure 6-14. Black Rock City in its proper place

You need to provide the image size, zoom level, and a single point. But we can change the properties of a `TImage` object dynamically. As an example, let's georeference the Black Rock City 2005 Plan. You can download the 65KB PDF image from http://www.burningman.com/preparation/maps/05_maps/. First convert the PDF to a JPEG. You can do this in any image-processing tool. If you have ImageMagick installed, this command works:

```
convert brc_2005.pdf brc_2005.jpg
```

Next, you need to find the pixel coordinates of a recognizable point. I used the GIMP to edit the image. Photoshop would also work. Open the image, then zoom in and place the cursor right over the Man. I get x=917, y=963. Also note the height and width of the image: 1833 x 2274.

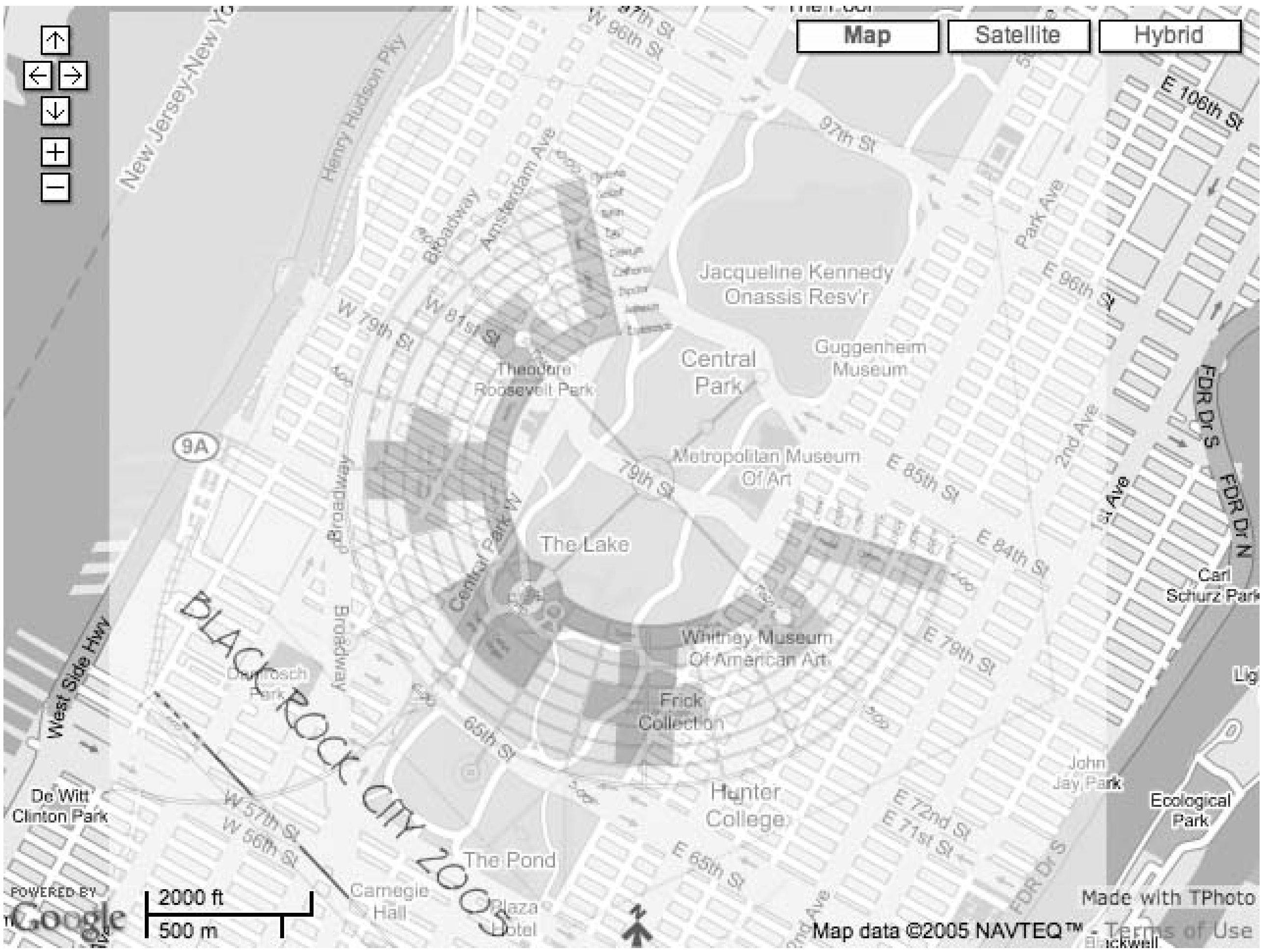
Finally, you need to rotate the image so that North is up. I used the rotate tool to align the grids with the North arrow. It was a -37 degree rotation.

This is the core code that overlays the image:

Google Maps Hacks By Schuyler Erle, Rich Gibson
ISBN: 0596101619 Publisher: O'Reilly Print Publication Date: 1/1/2006

```
var brc = new TPhoto();
brc.id = 'brc';
brc.src = 'images/brc_2005_rotate.jpg';
brc.size = new GSize(2032, 2432);
brc.baseZoom = 1;
brc.percentOpacity = 40;
brc.anchorPx = new GPoint(1110, 948);
brc.anchorLatLng = new GPoint(-119.23616409301758,40.754279292683094);
```

Figure 6-15. Black Rock City centered on Central Park



This loads the image *images/brc_2005_rotate.jpg* and centers it so that the x,y pixel coordinate maps to the passed lat/long. You can adjust the `Gsize` (specified in pixels) and `baseZoom` so that the displayed image takes up the right amount of space. This happens automatically with the two-point use of TPhoto.

You can change the opacity of the overlaid image **[Hack #55]**. This HTML defines two buttons that increase and decrease the photo opacity when they are clicked:

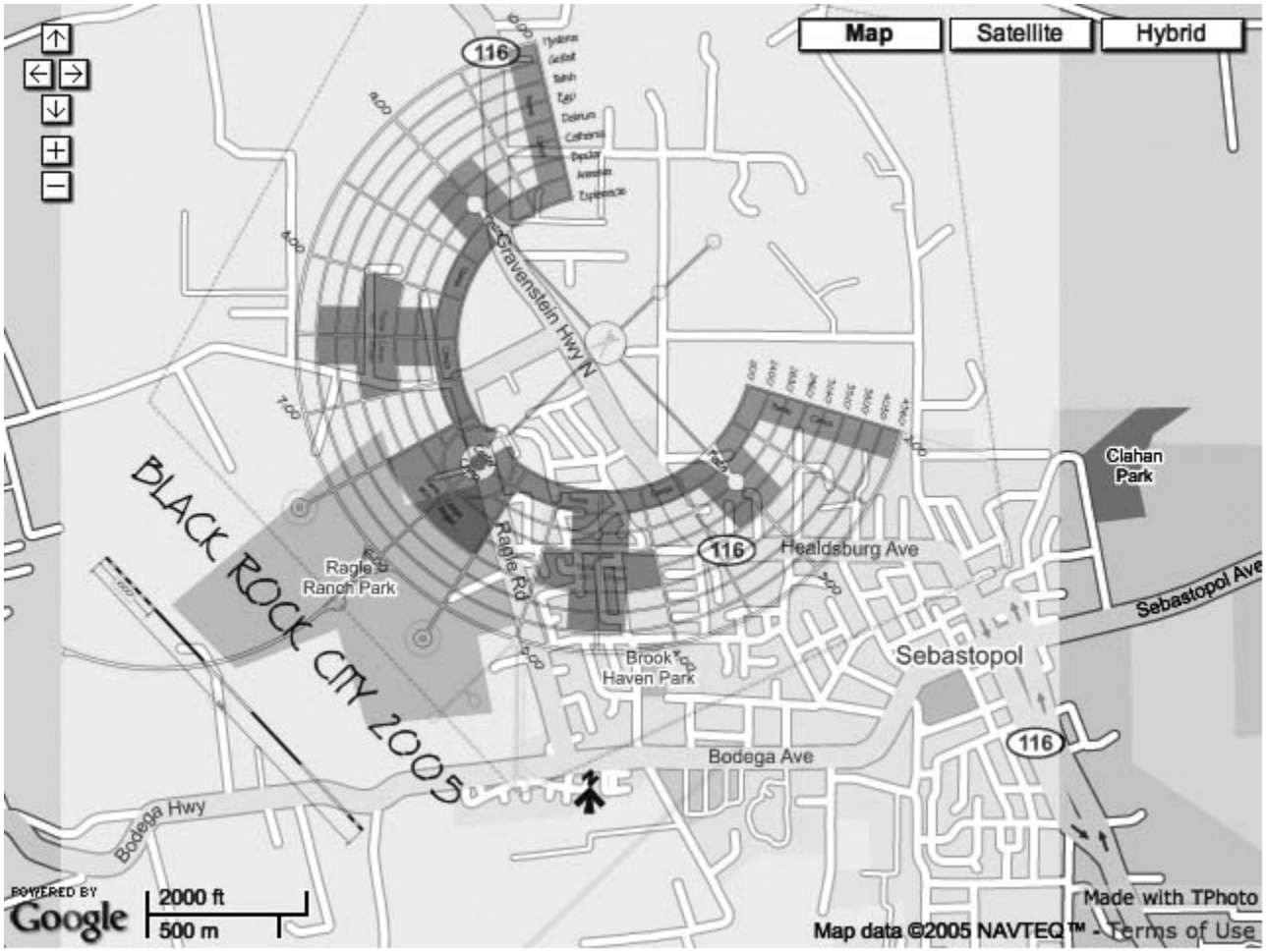
Google Maps Hacks By Schuyler Erle, Rich Gibson
ISBN: 0596101619 Publisher: O'Reilly Print Publication Date: 1/1/2006

```
<input type="button" value="opacity +" id="opacityup"
onClick="changeOpacity(10);">

<input type="button" value="opacity -" id="opacitydown"
onClick="changeOpacity(-10);">
```

When the buttons are clicked, either `+10` or `-10` is passed to the `changeOpacity` method:

Figure 6-16. Black Rock City centered on O'Reilly Media




```
function changeOpacity (z) {
        var temp = brc.percentOpacity;
        temp += z;
        if (temp > 100) {
                temp = 100;
        }
        if (temp < 0) {
                temp = 0;
        }
```


Google Maps Hacks By Schuyler Erle, Rich Gibson
ISBN: 0596101619 Publisher: O'Reilly Print Publication Date: 1/1/2006

```
            map.removeTPhoto(brc);
            brc.percentOpacity = temp;
            map.addTPhoto(brc);
            document.getElementById('opacity').value = brc.percentOpacity;
    }
```

This routine attempts to increase or decrease the opacity by the passed +10 or -10 and then checks that the opacity is not greater than 100 or less than 0. Assuming it is within range, it removes the photo from the active map, sets the opacity to the new value, and then redisplays the image and updates the displayed opacity.

The `showImage` method is called when the user clicks "show plan," and is even more simple:

```
    function showImage() {

            show = 1-show;
            if (show == 1) {
                    map.addTPhoto(brc);
                    document.getElementById('showplan').value = 'hide Plan';
            } else {
                    map.removeTPhoto(brc);
                    document.getElementById('showplan').value = 'show Plan';
            }
    }
```

The variable `show` serves as a toggle. It is initially set to `0`, and then the line `show=1-show` toggles it between `1` and `0`. When it is `1` the photo is displayed and when it is `0` the photo is hidden.

Hack 57. Do a Local Zoom with GxMagnifier

Context is key: GxMagnifier lets you choose the appropriate context.

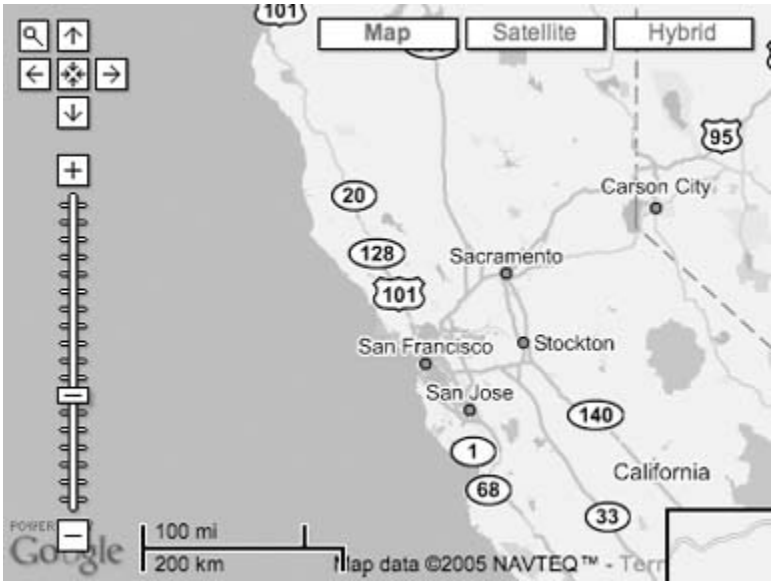
The zoom feature on Google Maps is nice, except when you want to see a zoomed out view so that you have some context and yet still see detail. The normal cartographic answer to that conundrum would be that you are out of luck. Lucky for us, this is no ordinary cartographic environment!

6.8.1. The Hack

GxMagnifier is a free add-in control for Google Maps that creates a moveable, magnified window on top of your map. Figure 6-17 shows GxMagnifier in action.

Google Maps Hacks By Schuyler Erle, Rich Gibson
ISBN: 0596101619 Publisher: O'Reilly Print Publication Date: 1/1/2006

Figure 6-17. GxMagnifier's "Hello World"



Does this look like an ordinary Google Map to you? Click the magnifier icon in the top-left corner of the map. Now, wherever you go on the map, you get a magnified view. This feature can be used with a mere two lines of code. Like a lot of JavaScript libraries, you first load it:

```
<script src="GxMagnifier.1.js" type="text/javascript"></script>
```

Then, add the control in your code:

```
map.addControl(new GxMagnifierControl());
```

Note that we're passing a custom control object to the standard `map.addControl` API call. This is all you need to do to get the basic magnifier control. This is only the most basic example of what GxMagnifier can do! The full documentation, with complete reference to the API can be found here: http://www.googlemappers.com/libraries/gxmagnifier/docs/default.htm.

6.8.2. Doing More with GxMagnifier

In our simple example we instantiated the `GxMagnifierControl` object within our `map.addControl()` call:

```
map.addControl(new GxMagnifierControl());
```

This is a dead simple approach, but it doesn't allow us to control the `GxMagnifierControl` later on. GxMagnifier can do more for us! The next examples will start with this basic framework, with the crucial lines highlighted. Of course, you'll need to use your own developer's key.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/
TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
          <title>GxMagnifier Example Framework</title>
```


Google Maps Hacks By Schuyler Erle, Rich Gibson
ISBN: 0596101619 Publisher: O'Reilly Print Publication Date: 1/1/2006

```
        <script src="http://maps.google.com/
maps?file=api&v=1&key=ilovemykidsmandms" type="text/javascript"></script>

        <script src="GxMagnifier.1.js" type="text/javascript"></script>
    </head>

    <body>
        <div id="map" style="width: 300px; height: 300px"></div>
        <script type="text/javascript">
        //<![CDATA[

        if (GBrowserIsCompatible( )) {
          var map = new GMap(document.getElementById("map"));
          map.centerAndZoom(new GPoint(-122.4618, 37.7902), 4);


          // Create the magnifier
          var magControl = new GxMagnifierControl( );
          map.addControl(magControl);

          // Get reference to the associated GxMagnifier for use in the examples
          var mag = magControl.GxMagnifier;

          // …rest  of the example goes here…
        }

        //]]>
        </script>
    </body>
</html>
```

That looks like a long example, but it is similar to the Google Maps "Hello World" example, with the addition of these lines:

```
var magControl = new GxMagnifierControl();
map.addControl(magControl);
var mag = magControl.GxMagnifier;
```

This defines the variable `magControl`, which we can use to set attributes, and then adds the control to the map. The `GxMagnifierControl` class is just a wrapper, as most of the time we'll be accessing its related `GxMagnifier` object by reading the `GxMagnifierControl.GxMagnifier` property into the variable `mag`.

What can you do with `magControl` now that you have it? Many things!

6.8.2.1. Zooming.

GxMagnifier automatically adjusts its zoom level as your main map is zoomed. Here we make it magnify by 3 zoom levels, instead of the default 2.

```
mag.setMagnification(3);
```

You can also set a negative zoom level, which is useful in conjunction with *docking*, as described below.

Google Maps Hacks By Schuyler Erle, Rich Gibson
ISBN: 0596101619 Publisher: O'Reilly Print Publication Date: 1/1/2006

6.8.2.2. Resizing.

By default, the magnified viewport is one third the size of your main map. You can change its dimensions.

```
mag.setSize(new GSize(150, 80));
```

6.8.2.3. Capturing the cick.

You may not want your magnifier to zoom in on the map when the user clicks it. Here we override the click event to tell us where we clicked and then close the magnifier window. You can use `GEvent.bind( )` for more sophisticated handling.

```
mag.disableDefaultClickHandler( );
GEvent.addListener(mag, "click",
  function(point){
    alert("You clicked " + point.toString( ));
    this.hide( ); // note, "this" is the GxMagnifier instance
  });
```

6.8.2.4. Moving the button.

Here we place the magnifying glass button 10 pixels from the top right corner. Note that `GxMagnifierControl.setDefaultPosition( )` has to be called before adding the control to the map.

```
var magControl = new GxMagnifierControl( );
magControl.setDefaultPosition(new GxControlPositionHack(1, 10, 10));
map.addControl(magControl);
```

6.8.2.5. Automatically panning.

The magnifier can scroll the map when it gets moved near an edge. Note the panning region extends inward from the edges of the main map by half the size of the magnifier window.

```
mag.enableAutoPan();
```

6.8.2.6. Docking.

You can free the magnifier from the map's container and put it anywhere on the page. We do this by creating a `GxMagnifier` instance directly and telling it to use our own HTML `div` element. Here's a fullscreen map, with the magnifier docked to the bottom right corner. The example deviates a bit from our typical ones.

Google Maps Hacks By Schuyler Erle, Rich Gibson
ISBN: 0596101619 Publisher: O'Reilly Print Publication Date: 1/1/2006

Inside the `head` element, we define some custom CSS styles for the page, the map, and our magnifier display:

```
<!-- Set up the containers, and make a "fullscreen" window -->
<style type="text/css">
     html, body {width: 100%; height: 100%}
     body {margin-top: 0px; margin-right: 0px; margin-left: 0px; margin-bottom:
0px}
  #map {position:absolute; width:100%; height:100%}
  #magnifier {position:absolute; right:0px; bottom:0px; width:200px; height:
200px;
  border:2px solid; border-bottom-style:outset}
</style>
```

In the body, we add HTML `div` elements for the map and the magnifier:

```
<div id="map"></div>
<div id="magnifier"></div>
```

And, finally, here's the JavaScript:

```
// Create the magnifier
var div = document.getElementById("magnifier");
div.style.zIndex = 10; // make it stay on top
var mag = new GxMagnifier(map, null, div);
mag.map.setMapType(G_SATELLITE_TYPE)

// Monitor the window resize event and let the map know when it occurs
// This isn't required for GxMagnifier, but it is required to accomplish
// a correct fullscreen Google Map.
if (window.attachEvent) {
  window.attachEvent("onresize", function( ) {this.map.onResize( )} );

} else {
  window.addEventListener("resize", function( ) {this.map.onResize( )} ,
false);
}
```

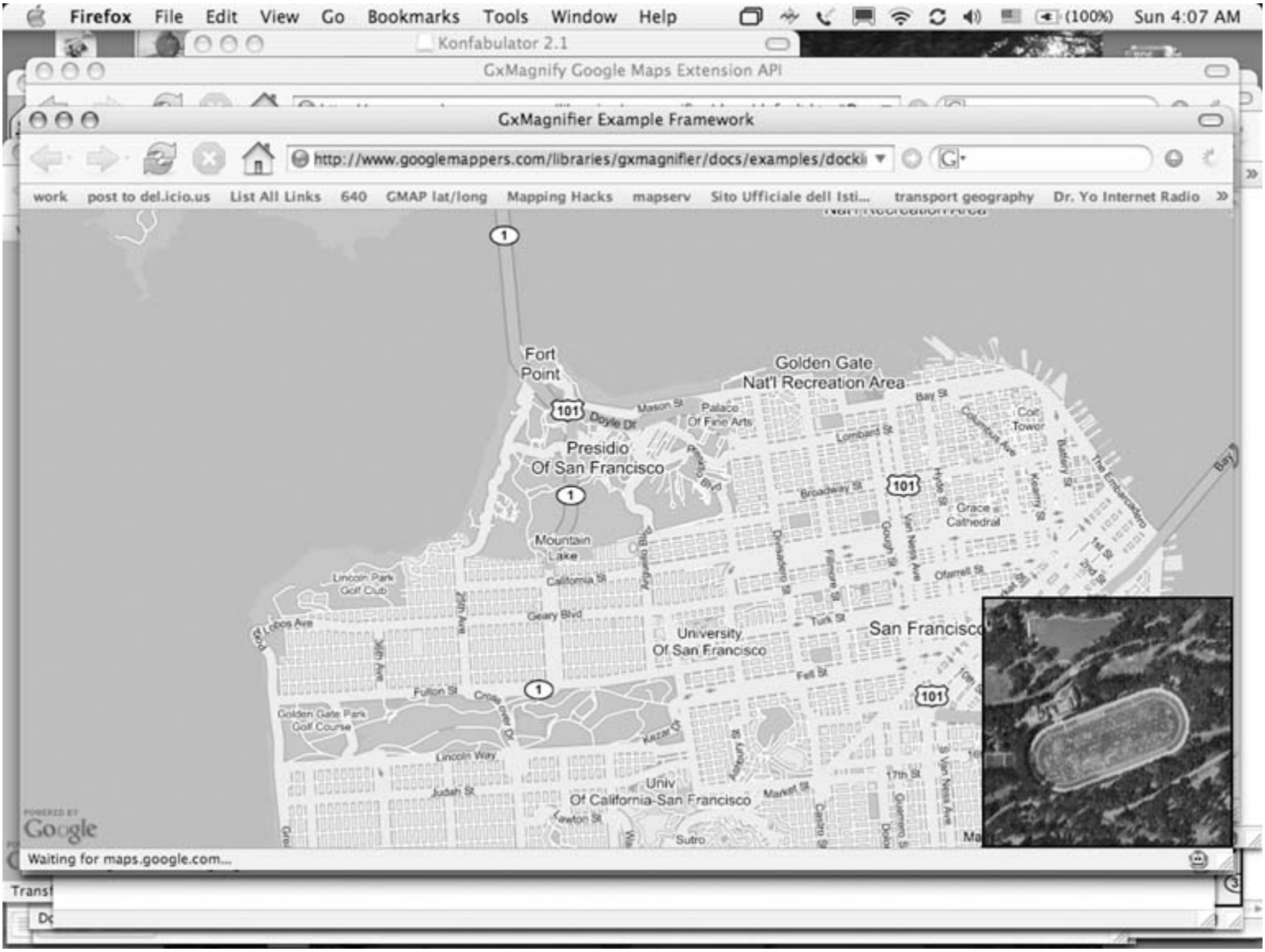
Note the line:

```
mag.map.setMapType(G_SATELLITE_TYPE)
```

The docked map in the corner shows the satellite imagery that corresponds to the point of the mouse, as shown in Figure 6-18. You can find out more about this feature at http://www.googlemappers.com/libraries/gxmagnifier/docs/examples/docking.htm.

Google Maps Hacks By Schuyler Erle, Rich Gibson
ISBN: 0596101619 Publisher: O'Reilly Print Publication Date: 1/1/2006

Figure 6-18. Magnifier docked with a satellite view



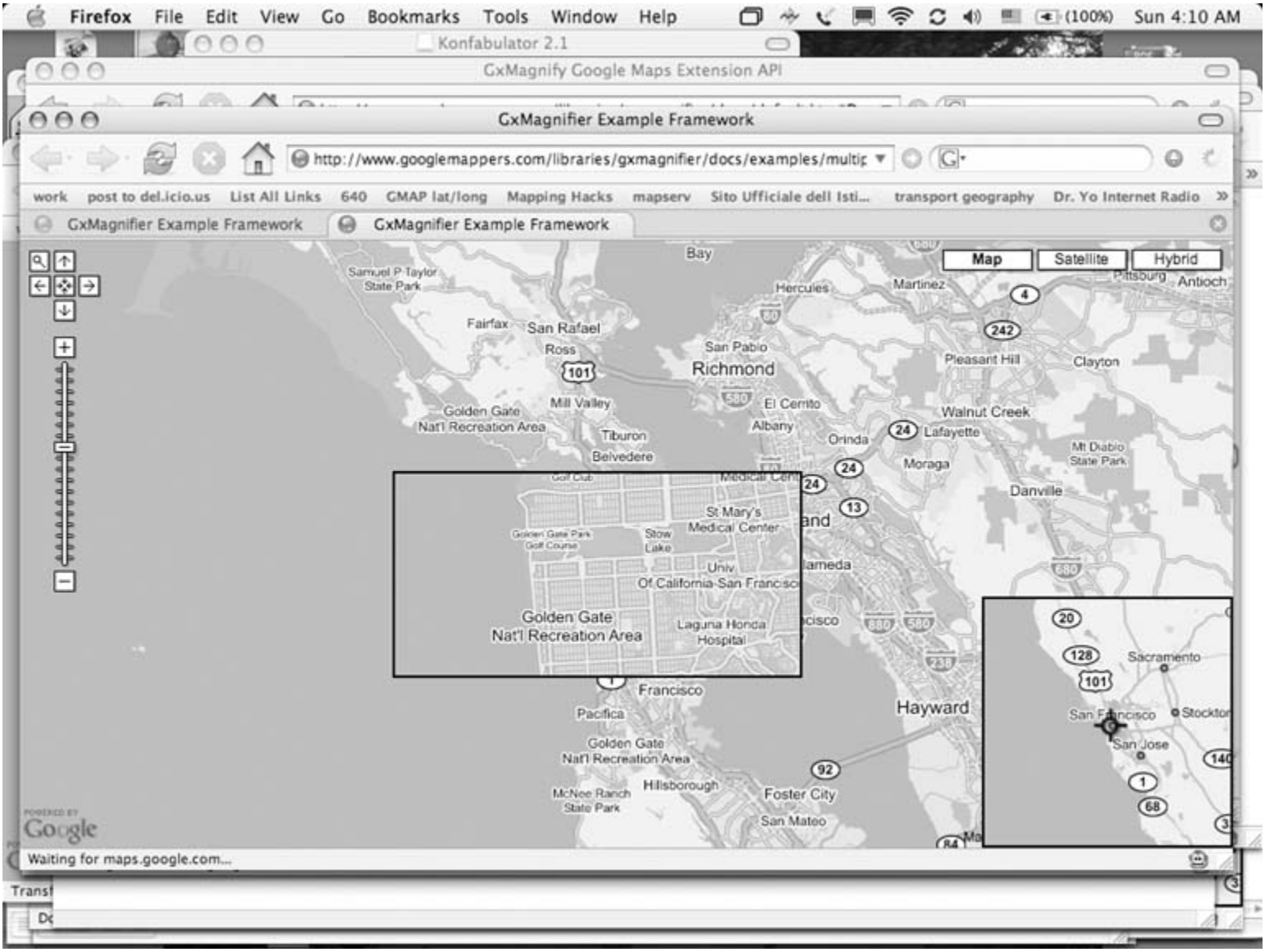
6.8.2.7. Multiple magnifiers and negative zoom.

You can include multiple GxMagnifiers on your map. Figure 6-19 expands on the previous example on docking, and uses a negative magnification factor to create a docked "birds eye view" of where you are on the map. See it in action at http://www.googlemappers.com/libraries/gxmagnifier/docs/examples/multiple.htm.

In case you're curious, I got the idea of trying to make a magnifier control for Google Maps after watching a concept video that shows a similar feature in mapping technology demonstrated under Windows Longhorn (a.k.a. Vista).

Google Maps Hacks By Schuyler Erle, Rich Gibson
ISBN: 0596101619 Publisher: O'Reilly Print Publication Date: 1/1/2006

Figure 6-19. An orientation map and a magnified map



6.8.3. See Also

- GxMagnifier is not supported by Google or part of its official API, and it is beta software, so there may be some bugs. If you run into anything, report it in the Google-Maps-API forum. I try to keep an eye on messages there, but chances are one of the other members will be able to respond with a fix faster than I can.
- The GxMagnifier documentation page has much more detail and lives at http://www.googlemappers.com/libraries/gxmagnifier/docs/default.htm.
- The Google Mappers community is a good place to find out more about custom Google Maps API extensions: http://www.googlemappers.com/.

—Richard Kagerer

Google Maps Hacks By Schuyler Erle, Rich Gibson
ISBN: 0596101619 Publisher: O'Reilly Print Publication Date: 1/1/2006

Hack 58. Find the Right Zoom Level

You've got some points to show on a map. How do you pick the right zoom level to show them all at once?

The Google Maps API is powerful but, as tends to be the case for such toolkits, has some omissions and inconveniences. For example, in many real-world mapping problems, it's common to have a set of points or lines that we wish to display on a map. Typically, we choose the outer four corners, or *extents*, of the map to match the *bounding box* of the data, which is given by the minimum and maximum *x* and *y* coordinates in our data set. By selecting the map extents to match the bounding box of the data, we can guarantee that everything we want to show should be visible in the map display.

The bounding box of a data set is usually very easy to calculate. One way to do so is to simply iterate through the data set, looking for the largest and smallest coordinates in both dimensions. The problem is that the Google Maps API doesn't give us any way to set the map extents based on a bounding box. Instead, the API exports a `centerAndZoom( )` method that accepts a center point of the map, expressed as a latitude and longitude pair, and a zoom level. Google documents the zoom level as an integer, and the examples in their documentation all use `4` as the zoom level, but no definition of the zoom levels themselves is given. Experimentation shows that they correspond with the ticks on the zoom level control.

Our question is then very simple: given a bounding box, how does one determine an appropriate center and zoom level, so that our data is guaranteed to appear on the screen, regardless of screen geometry?

6.9.1. The Brute Force Method

One approach is to look at the current map geometry and see if the screen will hold the bounding box and keep zooming in or out as needed. The zoom levels are approximately powers of two, so it's practical to use a sort of brute-force method, and just multiply or divide the current level by two, and see if it still fits. The following JavaScript does just that:

```
function computeZoom(minX, minY, maxX, maxY) {
  var zl;
  var bounds = map.getBoundsLatLng();
  var cz = map.getZoomLevel();
  var mapWidth = Math.abs(bounds.maxX - bounds.minX);
  var mapHeight = Math.abs(bounds.maxY - bounds.minY);
  var myWidth = Math.abs(maxX - minX);
  var myHeight = Math.abs(maxY - minY);
  var changeZoom = myWidth > mapWidth ? 1 : -1;
  // Just rip through until we find the lowest that will hold our span.
  // For certain geometries (around hemisphere boundaries), this won't work
for(zl=map.getZoomLeve( );zl> 2&&zl<17; zl += changeZoom) {
   width = width * Math.pow(2, changeZoom);
   height = height * Math.pow(2, changeZoom);
   if ((width < lon) || (height < lat))
     return zl - 1;
  }
```


Google Maps Hacks By Schuyler Erle, Rich Gibson
ISBN: 0596101619 Publisher: O'Reilly Print Publication Date: 1/1/2006

```
  return zl;
}
```

This makes setting that center point and zoom level very simple:

```
centerLat = (minLat + maxLat) / 2;
centerLon = (minLon + maxLon) / 2;
var zl = computeZoom(minLon, minLat, maxLon, maxLat);
map.centerAndZoom(new GPoint(centerLon, centerLat), zl);
```

6.9.2. The Analytic Method

However, we can simplify this approach even further. Empirically, we note that at zoom level 17, the resolution of the map is 1.46025 degrees of longitude per pixel. Similarly, we can observe that this resolution halves for each subsequent zoom level, and that this seems to be invariant for the *x* axis of the map. For the *y* axis of the map, this property applies to lower zoom levels of the map, but not the higher zoom levels, because instead of wrapping the map, blank space is shown above and below the poles.

Nevertheless, as a first approximation, we can calculate what the resolution of our map ought to be, given the display geometry of the map in pixels and a bounding box, to get a map that covers the whole box. We can exploit this relationship by taking the logarithm of the ratio of 1.46025 to our desired resolution to the base 2, and then subtracting this value from 17. The following JavaScript performs this operation:

```
function calculateZoom2 (minX, maxX, minY, maxY) {
    var mapElement = document.getElementById("map");
    var degLonPerPixel = Math.abs(maxX - minX) / parseInt(mapElement.style. width);
    var degLatPerPixel = Math.abs(maxY-minY) / parseInt(mapElement.style. height);
    var resolution = Math.max( degLonPerPixel, degLatPerPixel );
    return Math.ceil( 17 - Math.log(1.46025 / resolution) / Math.log(2) );
}
```

Since the JavaScript `Math.log( )` returns the logarithm to the base *e*, we use the mathematical relationship *loga(x) = logb(x) / logb(a)* in the code example above to obtain the logarithm to the base 2. Similarly, we use `Math.ceil()` to find the smallest integer greater than or equal to the result, to ensure that the resulting map is at least as large as our bounding box. As for the difference in resolutions between the *x* and *y* axes for the higher zoom level, we observe empirically that the maximum resolution for the *y* axis at any zoom level is smaller than the resolution of the *x* axis at zoom level 15. At zoom level 15, the entire world is still shown north to south, so this won't be a problem either.

The main advantage of this method is that, unlike the brute force method, it can be applied on the server side as well as the client side. The corresponding bit of Perl code from a CGI script might look like this:

```
my $zoom = int( 17 - log(1.46025 / $resolution) / log(2) ) + 1;
```

This approach requires a little bit less code than the brute-force method, and it feels a lot more elegant. The disadvantage, of course, is that it relies on hardcoded constants that could break if Google decides to change the semantics of its zoom levels without warning, though this seems unlikely at the moment.

6.9.3. The Undocumented API Method

Of course, Google probably needs its own method to determine the right zoom level right? As it happens, the 1.0 version of Google Maps API does provide such a method, albeit an undocumented one. The following code makes use of this undocumented method:

Google Maps Hacks By Schuyler Erle, Rich Gibson
ISBN: 0596101619 Publisher: O'Reilly Print Publication Date: 1/1/2006

```
var center = new GPoint( (minX + maxX) / 2, (minY + maxY) / 2 );
var span = new GSize( maxX - minX, minX - minY );
var zoom = map.spec.getLowestZoomLevel(center, span, map.viewSize);
map.centerAndZoom(center, zoom);
```

This method may be the simplest of all, but it relies on two presently undocumented features of the Google Maps API: the `map.spec.getLowestZoomLevel( )` method and the `map.viewSize` property. Using them runs an even higher risk than the other two methods that your code will break at some point, should Google alter its API. However, we won't be at all surprised if Google decides to document this method, or methods like it in future releases. Read more about this issue on the Google Groups thread at http://xrl.us/getLowestZoomLevel.

One final thought: all these methods try to find the optimal match between the supplied bounding box and a corresponding zoom level. If your bounding box area should happen to match a zoom level exactly, this might not be what you want, because the outliers of your data set will be displayed at the very edge of the map. For this reason, it might be smart to expand the width and height of your bounding box by, say, 10%, to ensure that not only is everything displayed on the map, but everything is displayed well within the map.

—Robert Lipe

Hack 59. Show Lots of Stuff—Quickly

Lots of Google Maps overlays means lots of time spent waiting for them to draw on the map—unless you're smart.

When dealing with nontrivial numbers of overlays to draw on a Google Map—as in more than a hundred or so markers or vertices in a polyline—the drawing time required for a `GMap` object becomes really noticable. The inclination is to want to show markers as quickly as possible, so that the user gets immediate feedback on what's going on, but it's been discovered that if you buffer the objects and then manually trigger the redraw, it is much faster.

6.10.1. The Code

We can accomplish this buffering by adding a custom method of our own to the Google Maps API, which we'll call `addOverlays( )`. By assigning it to the `GMap` class prototype, it becomes available on any `GMap` objects we create. The code looks like this:

```
// This is a bit of a trick for the 1.0 API. The 'addoverlay' method
// is agonizingly slow, so we buffer up the markers in the 'overlays'
// array and then deliver them to GMAP in one shot.

GMap.prototype.addOverlays=function(a) {
        var b=this;
        for (i=0;i<a.length;i++) {
                try {
                        this.overlays.push(a[i]);
                        a[i].initialize(this);
                        a[i].redraw(true);
                } catch(ex) {
                        alert('err: ' + i + ', ' + ex.toString());
                }
```


Google Maps Hacks By Schuyler Erle, Rich Gibson
ISBN: 0596101619 Publisher: O'Reilly Print Publication Date: 1/1/2006

```
        }
        this.reOrderOverlays();
}
```

The `addOverlays()` method can be used directly in your code or included via a `script` element in your HTML, after you include the Google Maps library.

Using this technique is really easy—instead of calling `map.addOverlay( )` for each marker or polyline we create, we'll push each overlay onto an array, and then pass the array to `map.addOverlays( )`. Here's a fragment of a simple GPX reader that displays waypoint data using this idea:

```
var wpts  = GPX.documentElement.getElementsByTagName("wpt");
var markers = [];
for (var k = 0; k < wpts.length; k++) {
      var point = new GPoint(trkPts[k].getAttribute("lon"),
                             trkPts[k].getAttribute("lat")));
      var marker = new Gmarker(point);
      // … we could do things with the marker here,
      // like set a custom icon or configure an info window …

      markers.push( marker );

}
 // Now let gmap have the waypoints all in one chunk.

map.addOverlays(points);
```

In a similar fashion, if you have multiple contiguous lines to draw, it's smarter to compress them all into a single polyline, rather than ask the `GMap` object to render each one separately:

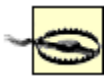
```
// Coalesce each GPX trk/trkseg/trkpts section into a single object.
// Each trkseg is a different object so that the lines don't get all
// glommed together.
var tracks = GPX.documentElement.getElementsByTagName("trk");
var trkpts = [], polylines = [];
for (var i = 0; i < tracks.length; i++) {
  var trkSeg = tracks[i].getElementsByTagName("trkseg");
  for (var j = 0; j < trkSeg.length; j++) {
    var trkPts = trkSeg[j].getElementsByTagName("trkpt");
    for (var k = 0; k < trkPts.length; k++) {
      trkpts.push(new GPoint(trkPts[k].getAttribute("lon"),
                             trkPts[k].getAttribute("lat")));
    }

    polylines.push(new GPolyline(trkpts, trkcolor, 5));
    trkpts = [];
  }
}

            map.addOverlays(polylines);
```

In testing one GPX file with 1,700 trackpoints, using this technique reduced the time required to display from 48 seconds to 6 seconds.

Google Maps Hacks By Schuyler Erle, Rich Gibson
ISBN: 0596101619 Publisher: O'Reilly Print Publication Date: 1/1/2006



Be careful not to get carried away with this idea, as lines will be drawn between every point in a single `GPolyline`. If your intent is to preserve breaks in the drawn line, be sure to preserve them as multiple `GPolyline` objects.

Additional optimizations are possible. For example, are all those track-points really necessary? Tools such as GPSBabel's *arc* filter can simplify the polyline, while retaining its basic shape. Different `GPolyline` objects with different amounts of detail could be chosen based on the current zoom level. As a rule, the less data Google Maps is obliged to work with, the faster it will perform.

6.10.2. See Also

- GPSBabel lives at http://www.gpsbabel.org/.
- "View Your GPS Tracklogs in Google Maps" **[Hack #37]** implements a line simplification algorithm in Ruby and in JavaScript.
- "How Big Is That, Exactly?" **[Hack #28]** also uses a line simplification algorithm to speed up the drawing of state and country shapes.

—Robert Lipe

Hack 60. Make Things Happen When the Map Moves

You can make your maps more interactive by responding to user-initiated events.

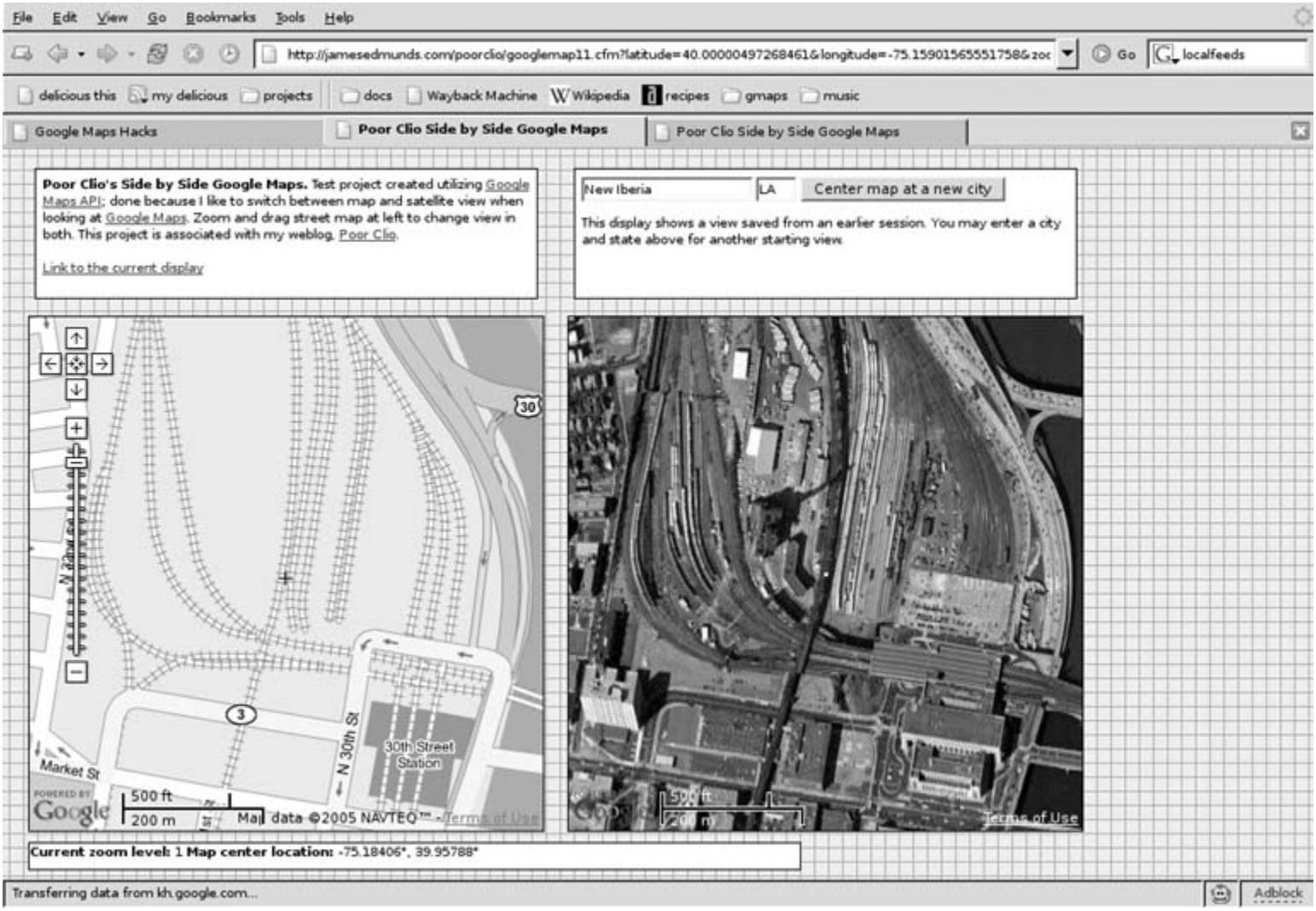
Balancing what to show and what not to show on a Google Map can be tricky. On one hand, for performanc reasons, you want to restrict the overlays displayed on the map to just the ones that would appear within the current view. On the other, if the user should pan or zoom the map, you might want the map's contents to change in response. Fortunately, the Google Maps API offers a way to make this happen automatically.

6.11.1. The Hack

Poor Clio's Side-by-Side Google Maps at http://jamesedmunds.com/poorclio/googlemap11.cfm offers a terrific example of how Google Maps hacks can respond to user actions. The site, built by James Edmunds, shows two Google Maps of the same location side by side. On the left, the regular map mode is shown, while the map on the right shows the same map in satellite mode. Figure 6-20 shows the side-by-side view of the area around 30th Street Station in Philadelphia, with the rail yards extending away to the north.

Google Maps Hacks By Schuyler Erle, Rich Gibson
ISBN: 0596101619 Publisher: O'Reilly Print Publication Date: 1/1/2006

Figure 6-20. Two views of Philadelphia's 30th Street Station, side by side



What's novel about this site is that if you drag the map on the left with your mouse, double-click to recenter it, or zoom in or out, the satellite view on the right recenters and zooms to match. Until Google introduced hybrid mode maps, this was effectively one of the only ways to compare the satellite and map views. The secret to how it works lies in the Google Maps Event API.

6.11.2. The Code

If you've written or looked at JavaScript code that uses the Google Maps API to cause info windows to pop up when the user clicks on a marker, then you've probably seen a method call that looks like this:

```
GEvent.addListener(marker, "click", function( ) {
    marker.openInfoWindowHtml(html);
});
```


Google Maps Hacks By Schuyler Erle, Rich Gibson
ISBN: 0596101619 Publisher: O'Reilly Print Publication Date: 1/1/2006

This method uses the `GEvent` class to register a click event on a marker object. When the marker is clicked, the anonymous function gets called, and the info window is opened. As it happens, the Google Maps API offers several other events for tracking and responding to user interactions.

Here's a snippet of code from the Poor Clio's Side-by-Side page:

```
function onMapMove( ) {
    smap.centerAndZoom(map.getCenterLatLng( ), smap.getZoomLevel( ));
    updateStatus( );
}

function onMapZoom(oldZoom, newZoom) {
    smap.centerAndZoom(map.getCenterLatLng( ), newZoom);
    updateStatus( );
}

GEvent.addListener(map, 'move', onMapMove);
GEvent.addListener(map, 'zoom', onMapZoom);
updateStatus( );
```

These dozen or so lines of JavaScript code handle virtually all of the user interactions needed to keep the two maps in sync. The first thing to note about this code is that, unlike most Google Maps API examples, you'll see *two* `GMap` objects in this code, one called `map` in the JavaScript, and the other called `smap`. Well, why not?

The first of the two functions shown above, `onMapMove( )`, takes the `GPoint` object representing the latitude and longitude of the center of the regular map, and the integer representing the current zoom level of the regular map, and passes them to the `centerAndZoom( )` method of the satellite map. When this happens, the satellite map automatically zooms and recenters to match the center point and zoom level of the regular map. The `onMapZoom( )` function does basically the same thing, but it takes two arguments, representing the old and new zoom levels of the regular map, and uses the new zoom level of the regular map as the new zoom level of the satellite map. The two calls to `GEvent.addListener( )` hook these functions into the move and zoom events on the regular map, respectively, so that when the user changes the view of the regular map, the satellite map follows.

The calls to `updateStatus( )` interspersed through the code above cause the latitude, longitude, and zoom level display below the two maps to keep in sync as well. The code looks like this:

```
function updateStatus()
{
    var point = map.getCenterLatLng();
    var status =
        "<b>Current zoom level:</b> " + map.getZoomLevel() +
        " <b>Map center location:</b> " +
        Math.round(point.x * 100000) / 100000 + "&deg;, " +
        Math.round(point.y * 100000) / 100000 + "&deg;";
    document.getElementById("status").innerHTML = status;
}
```

The use of `Math.round()`, coupled with multiplication and division by a constant power of ten, is a standard trick for rounding a floating-point number to a given number of decimal places. The call to `document.getElementById( )` fetches the HTML Document Object Model element containing the status message, and the assignment to the `innerHTML` property updates the HTML inside with the latest center point and zoom.

This same kind of technique can hypothetically be used to have a "Link to this View" URL track the current view, although Poor Clio's doesn't do it that way. Assuming that you had an HTML `<a>` element with an `id` attribute set to `linkHere`, you could do something like the following to update the URL in the `href` attribute:

Google Maps Hacks By Schuyler Erle, Rich Gibson
ISBN: 0596101619 Publisher: O'Reilly Print Publication Date: 1/1/2006

```
document.getElementById("linkHere").setAttribute( "href",
    "http://some.server.net/gmaps?ll
       =" + point.x + "+" + point.y + "&z=" + zoom );
```

There's one other trick that the Poor Clio's site uses to keep the maps in sync. Because the satellite half of this page doesn't have zoom and pan controls, the only way to move the satellite map independently is to drag it with the mouse. The site uses this line of JavaScript to keep such a thing from happening.

```
smap.disableDragging();
```

The other way to solve the problem of the maps potentially getting out of sync, of course, would be to make the event handlers a bit more generic and then add two more calls to `addEventListener( )` to make the regular map follow movements on the satellite map as well.

6.11.3. The GEvent API

The key thing to note about the GEvent API is that all its methods are static, which is to say that they're called on the class itself, and not on an instance of the class. This means you say:

```
GEvent.addListener( … ); // RIGHT
```

…as opposed to the following, which you should definitely *not* use:

```
var event = new GEvent();       // WRONG
event.addListener( … ); // WRONG
```

We mention this because it may run contrary to your expectations, as most of the other classes in the Google Maps API require you to instantiate an object of that class before calling methods on it.

The GEvent API shouldn't be mistaken for the JavaScript event API, which works rather similarly, with its `onClick, onMouseOver, onChange` methods, and so on. In general, you'll want to use GEvent handlers for Google Maps API objects, and continue to use JavaScript event handlers for everything else. In particular, as of this writing, there isn't anything like an `onMouseOver` event for Google Maps marker objects, and more's the pity. Perhaps Google will fix this in a future version of its API.

6.11.4. Hacking the Hack

Most Google Maps hacks out there rely on an Update Map button (or the equivalent) to update the overlays on a map after the user zooms or pans. This definitely works, but it's not the most elegant solution for refreshing the map's contents, because it relies on the user to do something that could be done automatically in JavaScript.

However, before you run out and add event handlers to your Google Map, consider the following: the Google Maps API doesn't guarantee exactly *when* the registered event handlers will be called, just that they will be called at some point. If you try dragging the regular map around on the Poor Clio's site, you'll notice that there's sometimes a bit of lag, but (assuming you don't drag too quickly) the satellite map otherwise follows along pretty faithfully. This means that the `move` event on the regular map is getting triggered every so many seconds or milliseconds throughout the drag.

Google Maps Hacks By Schuyler Erle, Rich Gibson
ISBN: 0596101619 Publisher: O'Reilly Print Publication Date: 1/1/2006

If your intention is to have one map track another, this is definitely what you want to have happen. On the other hand, if you're using the event handler to trigger a download of fresh data off the network, you probably don't want to have multiple overlapping data requests slowing everything down while the user is still trying to drag the map.

First, this means you want to use the `moveend` event with the GMap object, and not the `move` event. However, if you really want to be on the safe side, you might do well to combine this with a semaphore and a delay, in case the user does a bit of fine adjustment to recenter or zoom the map to exactly what they want to see. Here's what the code might look like:

```
var updateRequested = 0;

GEvent.addListener( map, "moveend", function() {
    updateRequested++;
    window.setTimeout( beginUpdate, 2500 )
});

function beginUpdate () {
    if (--updateRequested > 0) return;
    // *now* check map.getCenterLatLng() etc.
    // then launch a new GXmlHttp request
    // and do the update handling when the request is complete
}
```

The use of the global `updateRequested` variable acts as a semaphore to keep multiple events from triggering multiple network updates that would then stomp all over each other. When a `moveend` event is triggered, `updateRequested` is incremented from zero to one, and the network update is scheduled to begin 2,500 milliseconds (i.e., 2.5 seconds) later. In the meantime, further `moveend` events will increment `updateRequested` by 1 each time. When `beginUpdate( )` gets called 2.5 seconds after the first `moveend` event, it will decrement `updateRequested` by 1, and then check to see if the semaphore is still greater than zero. If it is, then subsequent movements of the map must have occurred in the meantime, and the update is therefore deferred.

Finally, the last `beginUpdate( )` will trigger 2.5 seconds after last `moveend` event, at which point the user has stopped moving the map around, the `updateRequested` semaphore will be decremented back to zero and, then and only then, the update will occur. Choosing the right delay so that network updates are smooth and non-overlapping, but so that the user experience isn't too terribly degraded, is probably a matter that merits some experimentation.

Hack 61. Use the Right Developer's Key Automatically

What to do when your web site lives at two domains that resolve to the same server.

Adding a Google Map to our web page **[Hack #10]** solved all our problems. Right? Not quite. The map works perfectly when we use http://geocoder.us, but the DNS for the Geocoder site, like that of many sites, is configured to treat http://www.geocoder.us/ as a synonym for http://geocoder.us/. The two domains resolve to the same server, which means that the same page will be served up to people asking for either domain. This works fine, until you put the Google Maps API key into the mix.

When you have the same Google Maps hack on two different URLs, even though they point to the same place, you get the message "The Google Maps API key used on this web site was registered for a different web site. You can generate a new key for this web site at http://www.google.com/apis/maps/." One can be forgiven the odd chuckle at the irony of http://maps.google.com/apis/maps/, as shown in Figure 6-21.

Google Maps Hacks By Schuyler Erle, Rich Gibson
ISBN: 0596101619 Publisher: O'Reilly Print Publication Date: 1/1/2006

Figure 6-21. Getting the developer key right is a tricky thing

How does Google know what server and directory your request is coming from?

For Google Maps to work you need to include the Google Maps JavaScript library into your page as shown in this example:

```
<script
       src="http://maps.google.com/maps?file=api&v=1&key=yourkeyhere"

       type="text/javascript">
</script>
```

A user will load your page by fetching it from your server. His web browser will then look through that page and identify images, scripts, plug-ins, and anything else that was not in the page and so needs to be loaded.

When the browser sees the `script` element, it attempts to load the script specified in the URL listed in the `src` attribute. When a browser asks for information from a web server, it also sends information, including the name of the server and the page that referred the user to get the script. Google Maps is probably using the HTTP `referer` that is automatically sent to the server by your browser.

There are two general ways to solve this problem. Either you maintain more than one developer key and somehow pick the right one to use when you load the Google Maps library, or you force people to always use the URL that is associated with your developer key by silently fixing bad URLs.

Google Maps Hacks By Schuyler Erle, Rich Gibson
ISBN: 0596101619 Publisher: O'Reilly Print Publication Date: 1/1/2006

6.12.1. Pick the Right Key in Perl

If you generate the page from a script, such as with PHP, Python, or Perl, you can maintain a list of server names and directories and the matching developer keys and then dynamically generate the `script` element to load the Google Maps library. This is a Perl example that sets the correct developer's key based on the `HTTP_HOST` that was used to call the script. I've shortened the developer's keys and replaced the actual domain names with placeholders.

```
#!/usr/bin/perl

# These are the keys for www.testingrange.com/pix, www.chilidog.com/pix,
    and www.journalsonline.com/pix

my $key_tr = "ABQIAAAAJhHGNa8WG19AE1v8p0OkZxQpjl…4EDQ";
my $key_jo = "ABQIAAAAJhHGNa8WG19AE1v8p0OkZxSB9G…nOFw";
my $key_cd = "ABQIAAAAJhHGNa8WG19AE1v8p0OkZxQCuE…rtNA";

# which server are we?
my $http_host = $ENV{HTTP_HOST};

my $key = '';

# set the key to match the host
$key = $key_tr if ($http_host =~ /first-domain/);
$key = $key_jo if ($http_host =~ /second-domain/);
$key = $key_cd if ($http_host =~ /third-domain/);

print "Content-type: text/html\n\n";
print qq(
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
      "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
     <head>
          <script src="http://maps.google.com/maps?file=api&v=1&key=$key"
               type="text/javascript"></script><-->
     </head>
     <body>
          <div id="map" style="width: 500px; height: 400px"></div>

          <script type="text/javascript">
          //<![CDATA[
          var map = new GMap(document.getElementById("map"));
          map.addControl(new GSmallMapControl()
          map.centerAndZoom(new GPoint(-122.8288, 38.4025), 3);
          //]]>
          </script>
     </body>
</html>
);
```

This script sets variables for the developer keys for each of our domains, then sets the variable `$key` to one of those values depending on the host-name that was used to call this script. If we have *first-domain.com, second-domain.com*, and *third-domain.com* all pointing to the same web page, then this script will set the correct developer's key.

Google Maps Hacks By Schuyler Erle, Rich Gibson
ISBN: 0596101619 Publisher: O'Reilly Print Publication Date: 1/1/2006

In real life, I would use a template. In this example, the Hello World map is in the script and is printed out with the `print qq( )` command. The `qq( )` operator in Perl automatically does variable interpolation, which means the variable `$key` will be replaced with the correct developer's key.

```
    <script src="http://maps.google.com/maps?file=api&v=1&key=$key"
type
          ="text/javascript"></script><-->
```

This example assumes completely different server names. The more common case is to have http://www.mydomain.com and http://mydomain.com both work. Assuming you've set the keys correctly, these lines will do that.

```
    $key = $key_plain if ($http_host !~ /^www/);
    $key = $key_www if ($http_host =~ /www/);
```

These lines say to use the value of `$key_plain` if the `HTTP_HOST` does *not* start with `www`, and use the value of `$key_www` if the host *does* start with `www`. The important point is to use the right key. If you have scripts in different directories, you'll want to take a look at the variables `$ENV{SCRIPT_NAME}` and `$ENV{REQUEST_URI}`. These variables contain the full path and page. You'll need to strip off the page name so you can compare the host and directory. There is more than one way to do it!

6.12.2. Use JavaScript to Accomplish the Same Goal

You can get the same effect dynamically with JavaScript. This is the Hello World map with a bit of JavaScript that looks to its own `href` to determine which URL it was loaded as, and then loads the map library by dynamically selecting the right developer's key. In this example, I've shortened the full developer's keys because you can't use my keys, and they are just too ugly to be printed in a book.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
      "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
     <head>
                <script type="text/javascript">

                // These are the keys for the servers
                var key_tr = "ABQI…";
                var key_jo = "ABQI…";
                var key_cd = "ABQI…";

                // Three nearly identical code blocks.
                // First define a regular expression that will match my server
                var reg_tr = new RegExp("testingrange");

                // Test the regular expression against the url for the current
                // page, which is available in window.location.href
                // if it contains our pattern then use it to load the script.

                if( reg_tr.test(window.location.href) ) {
                    loadScript(key_tr);
                }

                //same as above for next server
                var reg_jo = new RegExp("journalsonline");
                if( reg_jo.test(window.location.href) ) {
                    loadScript(key_jo);
```


Google Maps Hacks By Schuyler Erle, Rich Gibson
ISBN: 0596101619 Publisher: O'Reilly Print Publication Date: 1/1/2006

```
                }

                //same as above for yet another server
                var reg_cd = new RegExp("chilidog");
                if( reg_cd.test(window.location.href) ) {
                    loadScript(key_cd);
                }

                // build the script tag in the variable src and then
                // write it into the document
                function loadScript(key) {
                    var src ='<' + 'script src=' + '"' +
                    'http://maps.google.com/maps?file=api&v=1&key='+key+'"'+
                    ' type="text/javascript"><'+'/script>';
                    document.write(src);
                }

                //]]>
                </script>
        </head>
        <body>
            Which server is this?
            <div id="map" style="width: 500px; height: 400px"></div>

            <script type="text/javascript">
            //<![CDATA[

            var map = new GMap(document.getElementById("map"));
            map.addControl(new GSmallMapControl());
            map.centerAndZoom(new GPoint(-122.8288, 38.4025), 3);

            //]]>
            </script>
        </body>
    </html>
```

The difference from the original Hello World is the script block in the header. It defines variables for the different developer's keys, and then picks the right key based on the current URL, which is available to JavaScript in the variable `window.location.href`. There is a lot in common between the JavaScript and Perl approaches. They both pick from a set of developer's keys. The next approach lets you use a single key, but it changes the URL that the user sees.

6.12.3. Using Apache's mod_rewrite to Share Keys

If you are using the Apache web server you may be able to use the `mod_rewrite` module to quietly fix your URLs so that they match your developer key. In this example, we will use `mod_rewrite` to silently add `www.` to the front of any calls to http://mappinghacks.com. This technique assumes that your Apache is set up to allow *.htaccess* to override the `FileInfo` setting.

Apache has a lot of options, including options that can be set on per directory basis. The most common way to set them is the Apache configuration file (often */etc/httpd/conf/httpd.conf*) or in a file called *.htaccess* in the directory where you keep your web pages.

The *.htaccess* file will only work if the Apache configuration file is set up to allow it to work. In your *httpd.conf* file, there will be at least one `<Directory>` section. Note that Apache configuration is Byzantine in its complexity, so forgive my simplifications. There needs to be a line in the `<Directory>` section that starts with `AllowOverride` and specifies either `All`, which means the *.htaccess* file can control

Google Maps Hacks By Schuyler Erle, Rich Gibson
ISBN: 0596101619 Publisher: O'Reilly Print Publication Date: 1/1/2006

all aspects of Apache's access to that directory, or `FileInfo`, which allows the *.htaccess* file to control file things, including use of the `mod_rewrite` module.

6.12.4. Edit .htaccess to Rewrite Requests

I generated a key for http://www.mappinghacks.com/projects/gmaps. But I'd like people to be able to use http://mappinghacks.com/projects/gmaps as well. Here is an example of a `mod_rewrite` rule that will redirect requests from http://mappinghacks.com/projects/gmaps to http://www.mappinghacks.com/projects/gmaps. Create a new *.htaccess* file, or edit the existing one, to add these three lines.

```
RewriteEngine on
RewriteCond "%{HTTP_HOST}" "!^www" [NC]
RewriteRule "(.*)" "http://www.%{HTTP_HOST}%{REQUEST_URI}"
```

The first line turns on `mod_rewrite`. The second line looks at the `HTTP_HOST` field, which should be the server name from the URI, `mappinghacks.com` or www.mappinghacks.com. It does a regular expression match against the pattern `!^www`. Bang `(!)` means negation, or not, the caret `(^)` means to match at the start of the string, and `[NC]` says to ignore case (i.e., No Case). The `RewriteRule` tells Apache what to do if the Rewrite Condition is met. So this says "if the HTTP host doesn't start with *www* replace the whole URI with *http://www.* plus the HTTP host, plus the filename, and any parameters that were originally passed." To make it simpler, it just says "make sure the URL starts with *www*."

6.12.5. See Also

- Apache URL Rewriting Guides:
  - Apache Version 1.3: http://httpd.apache.org/docs/1.3/misc/rewriteguide.html
  - Apache Version 2.0: http://httpd.apache.org/docs/2.0/misc/rewriteguide.html

These are almost conversational guides to `mod_rewrite`. From the 2.0 version: "With `mod_rewrite` you either shoot yourself in the foot the first time and never use it again or love it for the rest of your life because of its power."

Google Maps Hacks By Schuyler Erle, Rich Gibson
ISBN: 0596101619 Publisher: O'Reilly Print Publication Date: 1/1/2006