Section 7: Mashups

Project 3: An Outline

In the recently released Project 3, you are expected to create a mashup of news articles (via Google News) onto a dynamic map (via Google Maps). The project will make use of all of the skills you've learned up to this point including XHTML, CSS, PHP, MySQL, Javascript, and Ajax, but will also require you to use the Google Maps API. The project may seem daunting at first, but it is very manageable by breaking it down into multiple parts as described in lecture. One possible breakdown could be:

1. Get your Google Map setup on your domain. This would involve obtaining a Google Maps API key and embedding the appropriate javascript code to get started.
2. Next, add a form to your page, but keep the visible Map as large as possible. Given the CSS requirements in the project spec, this may take more time than it seems. You may have to work for a bit to get everything sized just right!
3. Write the code that re-centers the map based on the search query. Since the form shouldn't POST to a processing page, you will need to implement a Javascript function that is called by the onsubmit attribute. After the search is performed, you should use the setCenter method on the GMap2 object to change the center of the map.
4. Add the ability to display markers on your map and, when clicked, show an information bubble with some text. Rather than being completely random text, you might want to show a link to Google News articles for the search location. You can implement the marker and information bubble with the GMarker class and GInfoWindow method (from the GMap2 class), respectively.
5. Now, do something with that Google News link. Create a PHP document that you query with an Ajax call. This PHP document should retrieve relevant news articles for the user-inputted zip code. Since Google News can return articles in an XML-based RSS feed, you could either parse the XML from the PHP document or simply hand the XML over to your Javascript code that parses it via the DOM. You could implement dropping markers and showing articles in the information bubble at this point.
6. The project requires that you show news articles not only for the searched zip code, but also show some set of articles for surrounding zip codes that are visible within the map window. You'll need to import the table of zip codes and implement a way of finding the zip codes that are visible within the Google Maps area. Using the MySQL procedure should simplify this process a bit, though you will still need to obtain the area visible from the map with the Maps API.
7. Be sure to implement the removal of unnecessary markers and the addition of newly visible ones with the "moveend" and "zoomend" events via a GEvent listener function. That is to say, whenever a user moves or zooms in the map, you must (for performance and memory reasons) remove old markers that are no longer visible and (if applicable), get new articles and show new markers.
8. Finally, give it a thorough testing! You won't want your Javascript to produce any errors or crash for any reason.

This is, of course, only a suggestion. You do not need to follow it and if you decide that there is a better outline for you to follow then feel free to do what is best for you.

Google Maps API

Google Maps is an online application that has become very popular because of its Ajax approach and UI enhancements over other implementations. Google has released an Application Programming Interface (API) for their Maps application that solidifies it as a powerful online mapping system. It is this API that you will use to show and manipulate a dynamic map in your project 3.

First, some useful links:

- **http://code.google.com/apis/maps/documentation/**
  Google Map introduction, table of contents, examples, and resources. Handy! Don't worry about the Mapplets API or Static Maps API. We are only concerned with the "Maps API".

- **http://code.google.com/apis/maps/documentation/reference.html**
  You will probably refer to this reference somewhat frequently as you work on project 3. This page will show you the classes, constructors, methods, controls, and other useful information regarding the options available to you through the API. To find out the proper use of the "setCenter" property, for example, you could search for it on this page. Alternatively, if you forget the name "setCenter", be sure to take a look at all of the methods and functions available under the GMap2 class to be reminded.

- **http://code.google.com/support/bin/topic.py?topic=10028**
  This is a FAQ page for the Google Maps API. It may be useful for getting started.

- **http://groups.google.com/group/Google-Maps-API**
  This is a link to the Google Group for the Maps API. Only try this resource if you have specific questions that are not found in any of the above documentation. Other experienced Maps API developers watch the group and may be able to help you.

Obtaining a Maps API Key

Before you dive into throwing a map on your page, Google requires you to obtain a key for your site. You must obtain a key for your domain, no matter how you develop your site. Otherwise the Maps API will not function properly on your domain. If you develop locally using XAMPP, you may need to obtain a key for http://localhost/. You probably don't need to obtain a key if you develop locally and simply load the XHTML document directly into your browser.

Let's get a key! Go to this website:

http://code.google.com/apis/maps/signup.html

If you are too impatient to read the entire Terms of Use you can read the bulleted points so you are reasonably well-informed as to the appropriate use of the key. Once ready, check the box that agrees to the terms and enter in your domain. You may simply enter "http://domain.tld" rather than fully qualifying "http://project3.domain.tld". The former retrieves a key that works with the latter.

Once you continue, you will need to enter in a Google account. You will have to create one if you do not have one. It's free and easy though!

Finally, you will be given a success page with your key and some sample code. Copy all of this information for future reference, but we'll next take a look at the same code they gave us.

Look Ma, My First Map!

Once you have successfully obtained a Maps API key, you will be presented with this code:

```
<!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>
    <meta http-equiv="content-type" content="text/html; charset=utf-8"/>
    <title>Google Maps JavaScript API Example</title>
    <script src="http://maps.google.com/maps?file=api&amp;v=2&amp;key=YOUR_KEY_HERE"
      type="text/javascript"></script>
    <script type="text/javascript">
      //<![CDATA[

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

      //]]>
    </script>
  </head>
  <body onload="load()" onunload="GUnload()">
    <div id="map" style="width: 500px; height: 300px"></div>
  </body>
</html>
```

You may copy this code and paste it in to your own XHTML page, but realize that it won't work if you put it on your domain because the key won't match. Simply replace the "YOUR_KEY_HERE" value in the "key" argument of the script source with your own key to make it work.

Much of this should seem familiar; it's a standard XHTML page with some Javascript and one lonely div in the page. Let's tear apart how this works:

There is a div element with id "map" where our map will go. We don't put any content in the div though, that will come through the Maps API which is instantiated by the "load()" Javascript function. Note that the load() function is called when the page is loaded via the onload attribute in the body tag.

In the load() function, there is the bare minimum amount of code to get a map visible on your page. First, it tests if the browser is compatible with Maps. If it is, we instantiate a new GMap2 class and refer the map div into it. This constructor will place a copy, so to speak, of Google Maps in the div. Note that the div has a width of 500 pixels and a height of 300 pixels and the map conforms to that restriction.

Finally, we set the map's center with map.setCenter(). Note that the setCenter() method is required after instantiation of the class or you may get an error. setCenter requires one argument, the latitude and longitude of a point via the GLatLng class, and accepts two optional arguments, the zoom level and a map type.

Take a look at this website if you are unfamiliar with latitude and longitude:
http://www.utas.edu.au/spatial/locations/spalatit.html

The GLatLng class constructor requires the latitude and longitude, in that order. Take care not to confuse the two! You may get unexpected results if you swap the two.

Another Example

Let's take a look at a modified load() function:

```
function load() {
  if (GBrowserIsCompatible()) {

        // define the location of Harvard Hall
        var Harvard = new GLatLng(42.374836, -71.118179);

        // instantiate our GMap2 class & set center
        var map = new GMap2(document.getElementById("map"));
        map.setCenter(Harvard, 15);

        // add zoom & pan controls
        map.addControl(new GLargeMapControl());

        // create a marker by instantiating a GMarker class
        var myMarker = new GMarker(Harvard);

        // add the marker by overlaying it on top of the map
        map.addOverlay(myMarker);

        // add a listener to show an information window when the
        // marker is clicked
        GEvent.addListener(myMarker, "click", function() {
                              var html = '<a href="http://cs75.net">cs75.net!</a>';
                              map.openInfoWindowHtml(Harvard, html);
                    });

        // add a listener that checks to see if Harvard Hall
        // has moved outside of the bounds of the map. If it has,
        // we'll make a sly remark to the user.
        GEvent.addListener(map, "moveend", function () {
                            var thisWindow = map.getBounds();
                            if (!thisWindow.containsLatLng(myMarker.getLatLng()))
                                  alert("You must be a distance student!");
                    });


  }
}
```

Some of these lines are similar as they are before, but let's tear this one apart as well:

First we'll define the location of Harvard Hall. The var "Harvard" contains an instantiated GLatLng class that refers to its location in geographic space. We are going to use this several times in this code, so it's easier to just assign it a variable rather than re-type the latitude and longitude coordinates every time.

Next, we do the now-familiar instantiation of the GMap2 class and set the center to Harvard Hall. We can also add the normal Google Maps controls with the addControl method of the GMap2 class, as we've done here.

Next, we'll create a marker. This is useful for pointing at specific locations. It is also possible, through an InfoWindow, to show additional information about the location. We instantiate a marker class and overlay it on top of the map with:

```
var myMarker = new GMarker(Harvard);

map.addOverlay(myMarker);
```

Note that the constructor for the GMarker class takes a GLatLng point. In this case, we'll use the same one we defined before, though we could have used any other point. The addOverlay method in the GMap2 class will show the marker on the map itself.

In order to click on the marker to get an info window, though, we'll need to add a listener event:

```
GEvent.addListener(myMarker, "click", function() {
                    var html = '<a href="http://cs75.net">cs75.net!</a>';
                    map.openInfoWindowHtml(Harvard, html);
              });
```

The addListener method tells the GEvent class to "listen" for a particular event to occur. In this case, we're telling the GEvent class to listen to the myMarker object and when a click occurs (*i.e.*, when a user clicks on the marker), the listener should run the lambda function. This function sets some text in an html var and, using the openInfoWindowHtml method, shows the html in an InfoWindow that points at the Harvard point.

Finally, we will add one more listener. This one, though, will fire a function only when the map has been re-centered; either by clicking-and-dragging, using the arrow keys to navigate around, or, if our page supported it, recentering the map due to code.

```
GEvent.addListener(map, "moveend", function () {
                    var thisWindow = map.getBounds();
                    if (!thisWindow.containsLatLng(myMarker.getLatLng()))
                          alert("You must be a distance student!");
              });
```

Here, we obtain the latitudinal and longitudinal bounds of the visible map, of type GBoundsLatLng, and return it into a thisWindow variable. From there, the GBoundsLatLng class has a handy method called "containsLatLng()" that does exactly what it implies: tests if an inputted latitude and longitude lie within the bounds given. In this case, we feed it the latitude and longitude of the marker. If the bounds of the window does not contain the marker point, we alert the user.