Computer Science 75 Lecture 8: November 9, 2009
Fall 2009 Andrew Sellergren
Scribe Notes

Contents

1 Announcements (0:00–2:00, 13:00–20:00)

- Because David didn't make explicit mention of it in lecture and because it's been a few weeks since we handed out the specification, we're extending the deadline for pre-proposals from today to next Monday. But get them in as soon as possible! It's basically just a chance for you to get in touch with your teaching fellow and find out if your idea is of reasonable scope. The specification is very general and welcoming, so be creative! You don't need to combine every piece of knowledge you've gained in this course. Rather, just pick a few which will showcase your skills.
- Also check out the Fun APIs that we've aggregated which may give you an idea for your final project. TextMarks, for example, is a service which assists in the process of sending and receiving SMS messages from the web. Most mobile carriers have e-mail addresses for mobile numbers which automatically forward to mobile devices. David's Shuttleboy service leverages this service to send texts with shuttle schedules. If you send a text to 41411 with SBOY and two stops, you'll get a response with the next shuttle times between those two stops. TextMarks is able to serve multiple customers with the same phone number by asking each customer to choose a unique moniker (SBOY in our case). They then pass along the user's message (a series of arguments) to a URL for that moniker. Once we get those arguments, we can feed them to our own PHP script which will do the lookup of shuttle times and generate a response with the corresponding schedule.

2 Project 3 (2:00–13:00)

- For Project 3, you'll be creating a mashup of Google News and Google Maps. What you'll ultimately present is the familiar Google Maps interface which will allow users to search for addresses. Your application will then whisk the user away to the location they searched for and will display up to five markers for the most populous cities within view. These markers, when clicked, will display links to the top news stories for that city. To see a working example, you can play around with the staff solution.
- To find the five largest cities in view, you'll query a zipcodes database which we purchased here as a CSV file. You'll be writing a command-line PHP script which will parse this file and import it into a MySQL database.
- After poking around a bit on Google News, David found that it offered the ability to search by location and to receive the data in RSS format. This meant there was a good opportunity to automate a request for news and parse the response with PHP.
- On the front-end, JavaScript enables responding to the clicking of the Go button and communicating with the PHP proxy that retrieves the zipcodes and news.

- In order to use the Google Maps API, you'll need to register your domain for a key. You do this by logging into your Google account and submitting the name of your domain. You'll get back a link to the JavaScript source code for Google Maps. Embedded in this link as a GET parameter will be the key that identifies your domain. This key will also work with `localhost` and `127.0.0.1`.

3 JavaScript (20:00–78:00)

3.1 Libraries, Code Analyzers, and Compressors

- Recall from last time this list of popular JavaScript libraries:
  - Dojo
  - Ext JS
  - jQuery
  - MooTools
  - Prototype
  - script.aculo.us
  - YUI

  We won't explore these libraries much in this course, but know that they're available and they add some syntactic sugar to JavaScript which makes it much easier to work with. jQuery and YUI are probably among the best in terms of browser friendliness.

- We've tended to use YUI most often in this course because of the attention it pays to accessibility issues. Generally speaking, however, it is much more verbose than jQuery, which is probably why jQuery is more popular. YUI 3 is an attempt to cut down on the clutter, but for now, it has only been partially released. You've already had experience with the YUI Reset library which attempts to normalize the appearance of web pages across all browsers. `map1.html` is a good example of how even simple pages can appear different in different browsers: Safari and Firefox, for example, show a thin band of whitespace at the top and left of the map while IE does not. The YUI Reset library takes care of some annoying inconsistencies like this. The YUI Fonts library standardizes fonts across different browsers and the YUI Base library restores some of the stylization for tags like `h1` and `h2` which were destroyed by the Reset library. When we introduce Ajax in the weeks to come, we'll make use of the YUI Connection Manager, which greatly simplifies the syntax for making Ajax requests. YUI also makes available a number of widgets like rich text editors and calendar controls which would otherwise be tedious to implement.

- Taking a moment to tout some other libraries, take a look at this Dojo demo of a very cool mouseover effect created with only JavaScript and

CSS. Script.aculo.us also has demos to play around with and jQuery even has a UI-specific library with demos. Feel free to use prefabbed modules like this to build up interesting interfaces for your final project! Be wary of mixing libraries—they don't necessarily play well together.

- The spinning hash tag cloud of HarvardTweets, though implemented with Flash on the back-end, is provided with data via JavaScript.

- All of the libraries listed above are under active development, although Script.aculo.us and Prototype have started to fall by the wayside.

- Speaking of cross-browser differences, htmladdnormallinkQuirks Modehttp://www.quirksmode.org/js/contents.html provides an excellent examination of tiny inconsistencies between browsers. To make sure that your JavaScript code is cross-browser compatible, you might make use of JSLint, a static code analyzer, which will alert you to syntactic gotchas. It would be a good idea to run your Project 3 code through this analyzer, although know that it can be overly anal in some cases, so you don't need to worry about satisfying its every little suggestion.

- To make code more portable and less prone to being stolen, you might choose to compress or minify it before deploying it. A number of compressors are available to do so:

  - JSMin
  - packer
  - ShrinkSafe
  - YUI Compressor

  To varying degrees, these tools will remove whitespace and shorten variable names to make your code harder to read but also easier to transfer over the wire. Our code for the mashup, in `mashup-min.js`, has been obfuscated to discourage prying eyes. You should feel free to examine our XHTML source code for the mashup, however.

- Question: does the compressed code run faster? In theory, yes, but probably not noticeably. It doesn't change the flow of your code and compressors are not the same as compilers. If you were to encrypt your JavaScript (with packer, for example), your code would run slightly slower. If you do want to optimize and compile your JavaScript code, take a look at Google's newly release Closure.

- Question: are there any disadvantages to compressing your code? Yes, you can break it, in fact. If you didn't end all your lines with semicolons, for example, your compressed code probably won't work. That's one of the arguments for using a code analyzer like JSLint.

3.2 Google Maps

- Know that for the examples we're about to go over, you're welcome to run the source code on your own domain, but it probably won't work until you copy and paste in your own API key.
- Google Maps' API is organized into five categories:
  - Basics
  - Events
  - Controls
  - Overlays
  - Services

  Despite all the documentation being on a single page, it's actually quite well-organized at least conceptually. There are two parts to the documentation: one part with step-by-step examples and one part with an exhaustive reference.
- What does it take to embed a Google Map in your website? Once you've registered for an API Key, no more than these lines of code:

```
<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>
```

  As before, we're bracketing the code in `CDATA` tags so that our XHTML will still validate. Then we're calling a function which checks if the user's browser is compatible with Google Maps. If it is, then we instantiate a new `GMap2` object and set its center. That's all there is to it! Well, sort of. We still have to actually *call* this function that we've defined:

```
<body onload="load()" onunload="GUnload()">
```

  There we go.

- Firebug is an invaluable tool for debugging JavaScript. If you click the Script tab and open the JS file you want to debug, you can click to the left of any line of code to place a red dot which represents a breakpoint. Execution of your JS code will pause wherever you have placed a breakpoint so that you can examine the contents of variables, etc. The Console tab will also catch some errors triggered by incorrect syntax. Another useful Firefox addon is JavaScript Debugger. Chrome and Safari have their own built-in debuggers now, as well.

- Our next example, `map2.html`, will fill the entire browser window with the map using CSS style properties:

  ```
  <body onload="load()" onunload="GUnload()"
        style="height: 100%; margin: 0px;">
  <div id="map" style="height: 100%; width: 100%;"></div>
  ```

  We do this simply by setting the `height` and `width` to 100%. Notice, however, that we must set the `height` of the `body` and the `html` elements to 100% as well. This is because the height of the `div` will be set to 100%, but when the page first loads and the map hasn't been inserted, the body will be 0 pixels because it only contains the map, which is still 0 pixels. So essentially we have a race condition where we're asking the map to be 100% of the height of the window, but the window has a height of 0 pixels until the map is filled in.

- So we've filled the viewport with the map, but what happens when we try to add a simple `form` element?

  ```
  <form><input type="text" /><input type="submit" /></form>
  ```

  What happens is that we get an annoying scroll bar that we can never quite get rid of. This is, of course, because the total height of the page will always be greater than 100% of the browser window, by virtue of the additional height of the form. To get rid of this scroll bar, you'll probably need to resort to JavaScript, in particular adding a listener for the `window.resize` event which will do some arithmetic and calculate how large the map should be.

- Our next attempt, `map3.html`, will recreate some of the familiar controls of Google Maps, besides the built-in drag:

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

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

```
            {
                // instantiate map
                var map = new GMap2(document.getElementById("map"));

                // center map on Science Center
                map.setCenter(new GLatLng(42.376649, -71.115789), 13);

                // add control using a local variable
                var typeControl = new GMapTypeControl();
                map.addControl(typeControl);

                // add another control without using a local variable
                map.addControl(new GLargeMapControl());

                // enable scroll wheel and smooth zooming
                map.enableScrollWheelZoom();
                map.enableContinuousZoom();
            }
        }

    //]]>
    </script>
```

As you can see, we add these controls by calling methods from the Google API. We can do this by using a temporary local variable, as in the first case, or simply by instantiating a new object *within* the method call itself. The second way is a little cleaner.

- Getting a little fancier, `map4.html` will finally add one of the red Google markers that you're probably familiar with:

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

        function load()
        {
            if (GBrowserIsCompatible())
            {
                // instantiate map
                var map = new GMap2(document.getElementById("map"));

                // prepare point
                var point = new GLatLng(42.376649, -71.115789);

                // center map on Science Center
                map.setCenter(point, 13);
                // mark Science Center
                var marker = new GMarker(point);
                map.addOverlay(marker);

                // associate info window with marker
                GEvent.addListener(marker, "click", function() {

                    // prepare XHTML
                    var xhtml = "<b>Science Center</b>";
                    xhtml += "<br /><br />";
                    xhtml +=
                    "<a href='http://en.wikipedia.org/wiki/Harvard_Science_Center'
                    target='_blank'>";
                    xhtml += "http://en.wikipedia.org/wiki/Harvard_Science_Center";
                    xhtml += "</a>";

                    // open info window
                    map.openInfoWindowHtml(point, xhtml);

                });

            }
        }

    //]]>
    </script>
```

- In this example, we store the point which represents the Science Center because we need to use it to set the center of the map and to create a marker there.
- As you can see, in the Google Maps API vocabulary, these red markers which, when clicked, pop up a small information bubble, are called overlays. Literally, they're being laid on top of the rest of the content of our webpage, probably using something called the CSS Z-index.
- Once we've added an overlay at the Science Center's GPS coordinates, we register an event handler to the click event of this overlay, which pops up an information window populated with the XHTML content we've specified. We do this using Google's API since we want to avoid mixing libraries whenever possible. The `addEventListener()` method takes three arguments: the object to add the listener to, the event to listen for, and a pointer to the function to call when that event is fired. In the case above, our event handler is an anonymous or lambda function. This is useful here because we're probably not going to need to reference this function ever again, so why waste a variable name on it? There's nothing complicated

about this function—all it does is build an XHTML string and then pass it to the `openInfoWindowHtml` method.

- The syntax we used above to pass a lambda function as an event handler is known as a *closure*. What this means is that the anonymous function has access to all the variables that are in scope at the time it is created. This might cause problems if we were trying to add more than one marker at a time since some of the variable values might be overwritten. We won't delve into this any deeper for now, but if you run into strange bugs like this, be sure to ask the staff about closures.

- Recall from weeks ago the file `quote3.php` which takes a stock symbol as input and returns the current stock price via Yahoo Finance. We're going to make a call to this file in `map5.html`:

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

    function load()
    {
        if (GBrowserIsCompatible())
        {
            // instantiate map
            var map = new GMap2(document.getElementById("map"));

            // prepare point
            var point = new GLatLng(37.4419, -122.1419);

            // center map on Science Center
            map.setCenter(point, 13);

            // mark Science Center
            var marker = new GMarker(point);
            map.addOverlay(marker);

            // associate info window with marker
            GEvent.addListener(marker, "click", function() {

                // prepare Ajax call
                var request = GXmlHttp.create();
                request.open("GET", "quote3.php?symbols=GOOG", true);
                request.onreadystatechange = function() {

                    // only handle successful calls
                    if (request.readyState == 4 && request.status == 200)
                    {
                        // prepare XHTML
```

```
                                var xhtml = "<b>Google</b>";
                                xhtml += "<br /><br />";

                                // get Google's quote
                                var quote =
                                 request.responseXML.getElementsByTagName("quote")[0];

                                // get Google's price
                                var price =
                                quote.getElementsByTagName("price")[0].firstChild.nodeValue;

                                // report price
                                xhtml += "$" + price;

                                // open info window
                                map.openInfoWindowHtml(point, xhtml);
                            }
                        };

                        // get quote
                        request.send(null);

                    });

                }
            }

        //]]>
        </script>
```

Our first foray into Ajax! Now, when we click our red Google marker, we're not just inserting static XHTML. We're actually executing some code. First, we're instantiating a cross-browser Ajax request object via Google's API—an Ajax request is simply an HTTP request created via JavaScript and sent after the original page has already loaded. You've seen these requests before, for example in Facebook's Live Feed. After instantiating this object, it looks like we're opening a GET request, checking that it returns an HTTP Status code indicating "OK," and finally grabbing the information we desire from the XML output, in this case, the stock quote for Google.

- One site that's made heavy use of Ajax is Kayak, which returns hotel and plane ticket search results in real time.
- In our staff mashup, there are actually two Ajax requests being made. One is made to Google's servers to geocode the address the user has provided. The second is to retrieve the five largest cities and their news results. This

second request will pass the coordinates of the southwest and northeast corners of the map and use these to narrow down what cities are returned from the zipcodes database.

- Of course, it might be easier if instead of passing through a proxy PHP file, we could directly query Google News using Ajax. However, this would be a violation of the Same Origin Policy (SOP). For security reasons, JavaScript cannot communicate with files that aren't hosted on the same server. Think of the problems that could arise if someone else's JavaScript were allowed to execute on your server!

- To get around the SOP, your PHP proxy will be the one to contact the Google News servers. You'll probably want to do some parsing of the RSS data before sending it back to your JavaScript. RSS (which stands for Really Simple Syndication or Rich Site Summary), incidentally, looks something like this:

```
<?xml version="1.0" encoding="UTF-8"?>
<rss version="2.0">
    <channel>
        <title></title>
        <description></description>
        <link></link>
        <item>
            <guid></guid>
            <title></title>
            <link></link>
            <description></description>
            <category></category>
            <pubDate></pubDate>
        </item>
        [...]
    </channel>
</rss>
```

  Generally, the `item` element represents a news article, but the RSS format has been co-opted for many different uses, Podcasts being one example.

- More on Ajax next week!