Computer Science 75
Spring 2009
Scribe Notes

Lecture 8: April 6, 2009
Andrew Sellergren

Contents

1 Projects (0:00–20:00) 2

2 JavaScript, Continued (20:00–54:00) 2

3 Some Examples with JavaScript (54:00–84:00) 5

Computer Science 75 — Lecture 8: April 6, 2009
Spring 2009 — Andrew Sellergren
Scribe Notes

1 Projects (0:00–20:00)

- Congrats to Carl and Rob for topping The Big Board by exploiting the non-real-time stock quotes!
- For your benefit, we have a sample solution for Project 3, as well. This is not meant to be the end-all, be-all, but rather a helpful guide. Feel free to change things around in your own implementation so long as you ultimately meet all the requirements in the project's specification.
- In order to minimize hits on the Google News server we'll actually be caching news items on our own servers for up to 24 hours. If the timestamp of some of your news items seems a little off, this caching is most likely the cause.
- If you take a look at the source code of the sample solution for Project 3, you'll notice that the JavaScript has been *minified.* What we did is to run it through one of YUI's widgets which removes whitespace and comments as well as shortens function and variable names, both to obfuscate the code and to minimize the number of bytes that are transferred over the wire when a request is made for that JavaScript file.
- When we type in a zipcode, what are the steps that are carried out in looking up news items? If we take a look using Live HTTP Headers, we can see that what is actually being passed via GET is a set of GPS coordinates for that city. We get those coordinates from our zipcode database, which provides the latitude and longitude of the city's effective center. We paid a little more this year for the deluxe version of the zipcode database which also includes population for each city. This way, we can prioritize the news items returned by our query such that only news for the 5 biggest cities is returned.
- The motivation for Project 3 is the Wii News Channel, a demonstration of which you can check out here.

2 JavaScript, Continued (20:00–54:00)

- One of the features of the sample Google News mashup (and requirements for Project 3) is dynamic map resizing. How can we accomplish this using JavaScript?
- To set the size of the map when the page loads, we'll want to register a listener for the `onload` property. We can do this like so:

```
window.onload = init;

function init()
{
    //code
}
```

Of course, since we're only going to be calling this function once, when the page loads, why bother giving it a name at all? We can register it as an anonymous function instead:

```
window.onload = function() { //code };
```

- We need to be a bit more clever than this, however, since we want to resize the map anytime that the browser window is resized. Obviously, then, we need to execute code more often than when the page first loads.

- In our HTML, we would do well to identify our map using a `div` like so:

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

  Of course, our `id` doesn't have to be "map," but can be anything we want it to be. Later, when we want to access this `div`, we can execute the following JavaScript code:

  ```
  <script>
  var map = document.getElementById("map");
  </script>
  ```

  Now, if we want to change the map's style attributes, we can do the following:

  ```
  map.style.height = "100px";
  map.style.width = "80%";
  ```

  Of course, this isn't going to accomplish exactly what we want with regard to the dynamic resizing, but it gives you the tools you need to figure it out on your own!

- Question: what happens if we want to register more than one handler for an event? What you *could* do is check if the event already has a handler, copy the pointer reference to that handler, create the new function that you want to add as a handler, and finally chain them together. Needless to say, this is pretty complicated and we won't be doing it on the fly. However, it's a nice segue into discussing the YUI Event Utility.

- The YUI Event Utility allows you to register listeners just as you can do manually in JavaScript. The advantage of the YUI library, however, is that it enables you to chain any number of functions together without having to worry about pointer references. Simply make a call to the `addListener` method and pass it the arguments it requires. It will take care of chaining the functions together, if need be.

- Generally speaking, what approach might we take to accomplish the dynamic map resizing? Obviously, we'll need to grab the total size of the window. From there, we can subtract a certain amount of space for the search bar up top. This can be a value in pixels if we always want it to be the same size. The remainder, then, is the space for the map.

- A sidenote: anytime that an SSL-encrypted page pulls data from a non-SSL-encrypted page, a browser warning (at least in most major browsers) will be issued asking the user if both secure and non-secure items should be displayed. To fix this bug, make sure that all the content on a page is being requested from a `https://` address.

- Another quick gotcha: when you access style elements using JavaScript, they'll be returned to you as strings. Don't simply add a number to them and expect it to update properly!

- Additionally, if the height of an element hasn't been explicitly set, then accessing its `height` property in JavaScript won't return anything. You're better off using a JavaScript library to get this information reliably. In the YUI library, the `getRegion()` method accomplishes this.

- Recall from last time the other JavaScript libraries which are available to you:
  - Dojo
  - Ext JS
  - jQuery
  - MooTools
  - Prototype
  - script.aculo.us
  - YUI

- Scriptaculous, for one, has a site that displays a healthy set of demos of its UI elements here. For jQuery, check out the Datepicker and Dialog widgets. Dojo, also, has a very cool demo called Fisheye which you can play around with here. Be careful, however, not to cram in more bells and whistles than is really useful or necessary!

- For a great set of documentation on cross-browser compatibility issues, check out QuirksMode. A JavaScript verifier like JSLint is invaluable to you, the developer, before you minify your code to make sure that it will work properly once compressed.

- As for compressors, avail yourself of one of the following:
  - JSMin
  - packer

  - ShrinkSafe
  - YUI Compressor

3 Some Examples with JavaScript (54:00–84:00)

- 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.

- 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` element to 100% as well.

- 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.

- 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>
```

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 overlap 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. In this case, it's a link to the Wikipedia page for the Science Center.

- Note that Google only allows the user to have a single info window open at a time.
- Now let's pull a few of these concepts together and even reach back to Project 2 with this next example, `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. It looks like we're opening a `GET` request, checking that it returns an HTTP Status code indicating "OK," then we're grabbing the information we desire from the XML output, in this case, the stock quote for Google. More on Ajax next week!

- More than any of the other projects so far, it will be important for Project 3 for you to learn how to use an external tool (in this case Google Maps) using its API and documentation. We'll walk you through a lot of it, but as is the case in the real world, for a large part of it, you'll be on your own. Try to have fun with it!