

## Problem Set 8: Mashup

due by 7:00 P.M. on Friday, 5 December 2008

### Goals.

- Introduce you to XML, DOM, RSS, JavaScript, and Ajax.
- Expose you to classes, objects, methods, and closures.
- Have you learn a real-world API.
- Take the training wheels off.

### Recommended Reading.

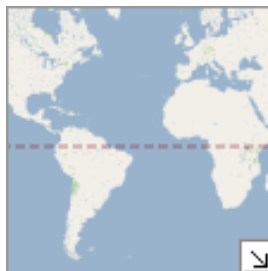
- <http://www.w3schools.com/xml/>
- <http://www.w3schools.com/dom/>
- <http://www.w3schools.com/rss/>
- <http://www.w3schools.com/js/>
- <http://www.w3schools.com/ajax/>

### NOTICE.

For this problem set, you are welcome and encouraged to consult “outside resources,” including books, the Web, strangers, and friends, as you teach yourself more about XML, DOM, RSS, JavaScript, and Ajax, so long as your work overall is ultimately your own. In other words, there remains a line, even if not precisely defined, between learning from others and presenting the work of others as your own.

You may adopt or adapt snippets of code written by others (whether found in some book, online, or elsewhere), so long as you cite (in the form of XHTML, CSS, PHP, or JavaScript comments) the origins thereof.

And you may learn from your classmates, so long as moments of counsel do not devolve into “show me your code” or “write this for me.” You may not, to be clear, examine the source code of classmates. If in doubt as to the appropriateness of some discussion, contact the staff.



## **Academic Honesty.**

All work that you do toward fulfillment of this course's expectations must be your own unless collaboration is explicitly allowed (*e.g.*, by some problem set or the final project). Viewing or copying another individual's work (even if left by a printer, stored in an executable directory, or accidentally shared in the course's virtual terminal room) or lifting material from a book, magazine, website, or other source—even in part—and presenting it as your own constitutes academic dishonesty, as does showing or giving your work, even in part, to another student.

Similarly is dual submission academic dishonesty: you may not submit the same or similar work to this course that you have submitted or will submit to another. Moreover, submission of any work that you intend to use outside of the course (*e.g.*, for a job) must be approved by the staff.

You are welcome to discuss the course's material with others in order to better understand it. You may even discuss problem sets with classmates, but you may not share code. In other words, you may communicate with classmates in English, but you may not communicate in, say, C. If in doubt as to the appropriateness of some discussion, contact the staff.

You may even turn to the Web for instruction beyond the course's lectures and sections, for references, and for solutions to technical difficulties, but not for outright solutions to problems on problem sets or your own final project. However, failure to cite (as with comments) the origin of any code or technique that you do discover outside of the course's lectures and sections (even while respecting these constraints) and then integrate into your own work may be considered academic dishonesty.

All forms of academic dishonesty are dealt with harshly.

## **Grades.**

Your work on this problem set will be evaluated along three primary axes.

*Correctness.* To what extent is your code consistent with our specifications and free of bugs?

*Design.* To what extent is your code written well (*i.e.*, clearly, efficiently, elegantly, and/or logically)?

*Style.* To what extent is your code readable (*i.e.*, commented and indented with variables aptly named)?

## Getting Started.

For this problem set, you will make something out of nothing. Just because you can.

Oh and this is your last problem set ever for CS 50.

: - (

For this problem set, consider downloading and installing **Firefox** followed by **Firebug**, **JavaScript Debugger**, **Live HTTP Headers**, and **Web Developer**, each of which is available on the course's website under **Software**. Once installed, Firebug, JavaScript Debugger, and Live HTTP Headers will appear as options in Firefox's **Tools** menu, and Web Developer will appear as a toolbar beneath Firefox's address bar. See if you can figure out how they each work, simply by playing. For tutorials on some of them, head to the URL below.

<http://code.google.com/support/bin/answer.py?answer=87133&topic=11364>

Odds are you'll find all of these tools valuable.<sup>1</sup>

For this problem set, your work must ultimately render and behave the same on at least two major browsers:

- Google Chrome 0.x
- Firefox 3.x
- Internet Explorer 7.x
- Opera 9.x
- Safari 3.x

Be sure, then, to test your work thoroughly with at least two browsers. It is fine, though, to rely on just one operating system. Make sure that your teaching fellow knows which browsers to use whilst evaluating your work.

SSH to `cloud.cs50.net` and create a directory called `pset8` in `~/public_html/`. (Remember how?) Navigate your way to `~/public_html/pset8/`, and then run `ls`. You should see nothing.<sup>2</sup> Just as you began this semester by writing programs from ~~Scratch~~ scratch, so will you end this semester the same way. Consider the training wheels off!

---

<sup>1</sup> Speaking of tools, if you would like to be adventurous and develop this whole project on your own Mac or PC, you might also like XAMPP, also available on the course's website under **Software**. Just realize that, come submission time, your code and database must ultimately work and live on `cloud.cs50.net`.

<sup>2</sup> See, there's that nothing we mentioned.

- Your mission for this problem set is to implement a mashup that integrates Google Maps with Google News with a MySQL database containing 42,073 unique zip codes. But first, some inspiration!

If you've a friend who owns a Nintendo Wii (that's connected to the Internet), see if you can invite yourself over there to do some "research." Ask your friend to start up the **News Channel**, then click **National News** with the Wiimote, then click any of the articles, then click **Globe** in the screen's bottom-right corner. Notice how you can spin the Earth by clicking **A** and dragging, thereby revealing stacks of articles from different cities and geographic areas. And by zooming in and out with **+** and **-**, you can reveal more or fewer stacks. By clicking a stack's icon, you can then read local news.

If you haven't said friend, watch this instead, paying close attention between 01:04 and 01:50:

<http://www.youtube.com/watch?v=uO6J8ryTKYk>

The challenge ahead isn't to implement precisely that interface but the spirit thereof in the (largely) two-dimensional world of Google Maps! Specifically, your mashup will present users with a Google Map, next to which will be a form. Upon submitting an address via that form, users will be whisked away to that location on the map, which will be sprinkled with markers representing news articles pertaining to the area. Clicking a marker will trigger a balloon to appear, inside of which will be links to those articles.

By problem set's end, then, you'll have a tool whose URL you can share with family and friends back home (that they might actually find useful)!

Alright, where to begin?

- Surf on over to Google Maps at <http://maps.google.com/>. Input something like 02138 into the page's form and click **Search Maps**. You should find yourself whisked away to a familiar location. Well that was easy. Notice, though, that the page's URL did not change. I wonder how they did that!<sup>3</sup>

Now input 28.42, -81.58 instead. Perhaps you'd rather be there?

It looks like Google Maps understands zip codes as well as longitude and latitude. Interesting...

If unfamiliar with longitude and latitude, feel free to read up, albeit in more detail than is necessary for this problem set:

<http://en.wikipedia.org/wiki/Longitude>  
<http://en.wikipedia.org/wiki/Latitude>

---

<sup>3</sup> Cough cough, Ajax.

- Now surf on over to Google News at <http://news.google.com/>. Click the link to **Advanced news search** at top-right, and notice how you can **Return only articles about a local area**. Input a zip code like 02138, and you should see news local to Cambridge. Notice the link to **RSS** on the page's left-hand side. Click it, and you should find yourself at a URL like the below (which, unfortunately, wraps on to two lines):

```
http://news.google.com/news?svnum=10&as_scoring=r&ned=us&as_drrb=q&as_qdr=&as_mind=22&as_minm=10&as_maxd=21&as_maxm=11&geo=02138&aq=f&nolr=1&output=rss
```

Within that otherwise cryptic string should be a familiar value (*i.e.*, 02138). Hm, that could be useful...

- Alright, clearly there's a way to whisk a user away to a particular point on a map, using zip codes or longitude and latitude. But Google News expects only zip codes, so how to find zip codes that are proximal to another? It'd be nice if we had a big list...

Let me Google that for you:

```
http://letmegooglethatforyou.com/?q=zip+code+database+list
```

Among the results should be a link to [zip-codes.com](http://zip-codes.com). Head to this page in particular:

```
http://www.zip-codes.com/zip-code-database.asp
```

Look how much data you can get for \$79.95.<sup>4</sup> Don't worry, we'll pick up the tab. Notice, though, that the data comes in multiple formats, among them CSV.<sup>5</sup> That's perfect, because we can easily parse that programmatically!<sup>6</sup>

To get a sense of what we bought, download this sample:

```
http://www.zip-codes.com/files/sample_database/zip-codes-database-DELUXE-SAMPLE.zip
```

Inside that archive, you'll find a file called `zip-codes-database-DELUXE-SAMPLE.csv` (among others). Go ahead and open it with Excel (or any old text editor). Notice that the database contains 49 fields (*i.e.*, columns), each of which is defined for you in the CSV file's first row. If you open up `ZipCodeDatabaseSpecifications-Deluxe.pdf` from that same archive, you'll find that pages 2 and 3 elaborate on those fields' types. That same PDF, for reference, is available at the URL below.

```
http://www.zip-codes.com/files/documents/ZipCodeDatabaseSpecifications-Deluxe.pdf
```

---

<sup>4</sup> Why go Standard when you can have Deluxe? Actually, we wanted population data, so we got upsold.

<sup>5</sup> [http://en.wikipedia.org/wiki/Comma-separated\\_values](http://en.wikipedia.org/wiki/Comma-separated_values)

<sup>6</sup> Keep an eye out for such things in the future!

You're welcome to examine the actual CSV file that we bought, but you'll find that its 79,960 rows won't fit in Excel:<sup>7</sup>

<http://cs50.net/pub/share/pset8/zip-codes-database-DELUXE.csv>

By default, incidentally, Excel doesn't show leading zeroes in CSV files. And so some of the zip codes in these CSV files' first columns might appear to be fewer than five digits, even though they are not. If you open those same files with any old text editor, you'll see leading zeroes!

- So, thanks to its 79,960 rows and 49 columns, the CSV file we bought is 28 MB. Yet most of those fields you won't even need. Interesting though it may be to know how many Hawaiians lived in 02138 as of the 2000 Census, you don't really need to know that for this problem set.<sup>8</sup> You're only going to waste time later on if you have to sift through so many unneeded bytes while searching for zip codes. So let's "pre-process" the data. It's a pain to manipulate strings in C, so we'll use PHP.

Write, in a file called `import` in `~/public_html/pset8/`, a "script" (*i.e.*, an interpreted program) that imports these fields (and only these fields) into a MySQL database:<sup>9</sup>

- i. `ZipCode`
- ii. `Population`
- iii. `Latitude`
- iv. `Longitude`
- v. `State`
- vi. `City`

The table into which you import these fields must be called `zips`, and it must live in the database called `username_pset8` that we pre-created for you. You may connect to that database using the same username (`DB_USER`) and password (`DB_PASS`) that you used for Problem Set 7.

We leave the design of `zips` up to you, but you might find some good hints in that PDF from `zip-codes.com`. Take care not to allocate more space than you need to for fields. And take care to define, at least, a primary key. Realize, though, that the CSV file actually contains some zip codes multiple times, each of whose rows might have identical values for `Latitude` and `Longitude` but different values for, say, `CityAliasName`. (The file contains 42,073 unique zip codes.) Do not insert any zip code into your table multiple times. We leave it to you to figure out how.

---

<sup>7</sup> Someone decided to limit Excel to  $2^{16} = 65,536$  rows (and  $2^8 = 256$  columns)!

<sup>8</sup> But the answer is 25!

<sup>9</sup> Be sure to import the fields with precisely these names, as defined by the CSV file's first row. If you find, while implementing your mashup, that you would like to make use of other fields from the CSV file, you may modify `import` and `zips` to accommodate. For space's sake, though, do not import more fields than you actually plan to use.

Once you have a schema (*i.e.*, design) in mind, you'll, of course, need to `CREATE` the table, certainly before you can `INSERT` any rows into to with your script. You are welcome to `CREATE` the table "at runtime" via a call to `mysql_query` in your own script or in advance via phpMyAdmin. Recall that phpMyAdmin is available at the URL below.

<http://cloud.cs50.net/phpMyAdmin/>

You'll probably want to use phpMyAdmin quite a bit while developing your script. Odds are you'll make a mistake at least once and want to `ALTER` or `DELETE` rows from your table so as to start fresh. You'll likely find phpMyAdmin's **Browse**, **Structure**, **SQL**, **Empty**, and/or **Drop** tabs of particular help.

Note, incidentally, that we did not ask you to name your script `import.php`. Whereas web servers generally require that PHP scripts' names end in `.php` for execution, your shell (*i.e.*, your prompt) only requires that they start with a "shebang," `#` followed by `!`, followed by the full path to `php` (PHP's interpreter).

How to find that full path? Execute the below.

```
which php
```

Aha! You should see that `php` lives in `/usr/bin/php` on `cloud.cs50.net`. Go ahead, then, and put precisely this line atop `import`:<sup>10</sup>

```
#!/usr/bin/php
```

Take care not to put any characters at all (even whitespace) before this shebang. Even with the shebang present, you still need to tell `php` to interpret what follows as actual code (rather than just text, a la XHTML, that should be outputted raw). The second line of your file should thus be:

```
<?php
```

And your last line should be:

```
?>
```

It's everything in between that we now leave to you. Implement `import!` Rather than access that CSV file via its URL on `cs50.net`, though, read from the cloud's local copy in `/home/cs50/pub/share/pset8/` (much like you did `card.raw` for Problem Set 5 and `words` for Problem Set 6). After all, networks are even slower than disks. Odds are, you'll want to befriend PHP functions like `fopen`, `fgetcsv`, and `fclose`. And don't forget old friends like `mysql_connect`, `mysql_select_db`, `mysql_query`, and `mysql_fetch_array`.

---

<sup>10</sup> Alternatively, you could omit the shebang and execute `import` with `~/usr/bin/php import`. But don't.

Once done, you should be able to populate that table called `zips` in `username_pset8` by executing, quite simply, the command below after `chmod'ing import 700`:

```
import /home/cs50/pub/share/pset8/zip-codes-database-DELUXE.csv
```

As this usage implies, you should accept one (and only one) command-line argument: the path to a file to import. You might thus want to read up on, at least, `$argv`:

```
http://us3.php.net/manual/en/reserved.variables.argv.php
```

Incidentally, you might want to try importing a much smaller file than ours first, lest you fill your table with thousands of mistakes. Just fill a text file with some values and commas!

Once you've imported all 42,073 unique zip codes from `zip-codes-database-DELUXE.csv`, leave `zips` alone. You'll need it later.

- Okay, it's now time to turn our (well, your) attention to the Google Maps API (Application Programming Interface), which "lets you embed Google Maps in your own web pages with JavaScript," provided you've signed up for an "API key." Head to

```
http://code.google.com/apis/maps/
```

and follow the link to **Sign up for a Google Maps API key**. When asked for **My web site URL**, provide

```
http://cloud.cs50.net/~username/
```

as your answer, where `username` is your own username, and then click the button labeled **Generate API Key**. On the page that appears next, you should be thanked and informed of your key. Remember that key! Well, don't actually try to memorize it. But copy and paste it somewhere safe.<sup>11</sup> That same page should also provide you with "an example web page to get you started on your way to mapping glory." Well that sounds fun. Go ahead and copy that XHTML (and JavaScript) to your clipboard too, and paste it into a file called `index.html` in `~/public_html/pset8/`.<sup>12</sup> Then visit the URL below.

```
http://cloud.cs50.net/~username/pset8/
```

Damn. Forbidden, right? Well that problem has nothing to do with Google. Go fix! (Remember how?)

- All better now? No? Okay, fine. "Don't forget to `chmod ~/public_html/pset8/` as 711 and `~/public_html/pset8/index.html` as 644." Trying to cut the cord here, you know!

---

<sup>11</sup> If you nonetheless lose track of your key, you can re-generate it via this same process. But it's not nearly as much fun the second time around.

<sup>12</sup> If using Nano, you might want to disable automatic indentation before pasting by hitting Esc followed by i; you can re-enable it after pasting by hitting the same sequence again.

- Okay, at this point you should have a little baby map embedded in an otherwise unremarkable page. But, hey, you can at least drag it around. Let's learn how to make it more interesting. Head over to

<http://code.google.com/apis/maps/documentation/>

and head straight to the section called **Audience**. You should ignore everything else under **Table of Contents**. Follow the link to **Basic Map Objects** and read the whole page! You'll be introduced to not only to cartography but also to "info windows." Oh and by "read" we mean actually read. One aim of this problem set is to prepare you for life after 50.<sup>13</sup> It's time to RTFM! After 50, documentation plus examples will be your best friends when it comes time to learn some new API or some new language altogether. In fact, as you read up on this particular API, be sure to view each and every one of Google's examples. And don't simply play; view each's source! You'll be amazed how much you can infer.

If you've any questions about what you've read, simply ask via the course's bulletin board!

Before we move on, let's have you apply what you just learned by making your own map more interesting. It's fine, incidentally, to leave all of your JavaScript in `index.html` or relocate it to some external `.js` file.

Center your map on your hometown instead of Palo Alto. And feel free to alter your map's default zoom level at the same time. If you don't know any longitudes or latitudes offhand, follow the directions in Section 3 of this tutorial to find some using Google Maps itself:

<http://code.google.com/support/bin/answer.py?answer=74725&topic=11364>

Plant an info window at the place you grew up and fill it with some XHTML that includes both some text (*e.g.*, your name) and at least one clickable link (to, *e.g.*, your hometown's website).

Feel free, too, to make your map bigger or perhaps center it in your page. Perhaps supplement the map with some logo or text.

- Head now to

<http://code.google.com/apis/maps/documentation/events.html>

to introduce yourself to "events," "listeners," and "closures." As before, play with and read through each and every example!

---

<sup>13</sup> Pun intended.

- Now head to

<http://code.google.com/apis/maps/documentation/controls.html>

and read up on “controls” and “types.” Once you have a sense of what’s possible, add one or more controls to your map and feel free to alter your map’s type.

- Now teach yourself about “overlays” and “markers” by heading to:

<http://code.google.com/apis/maps/documentation/overlays.html>

You can stop reading once you hit “polylines,” but don’t let us stop you if you’d like to read on.

- Almost done with the reading! Head to

<http://code.google.com/apis/maps/documentation/services.html>

for an introduction to “services” and “geocoding” specifically. You can stop once you hit “Street View,” but, again, feel free to read on.

It’s now time to enhance your own map once more. Add to your map a text field and button. When the latter is clicked, the user should be whisked away (if possible) to whatever location has been typed in the former. There’s no need to plant a marker or info window at that spot, but you’re welcome to do so. You might find the example called `geocoding-simple.html` of particular help.

- Take note of, but don’t (yet) read in its entirety, the API Reference for Google Maps:

<http://code.google.com/apis/maps/documentation/reference.html>

Overwhelming though it may seem at first glance, there’s a lot of good stuff there, as that page defines every one of the API’s classes, types, and functions. It’s not terribly easy to find things there, though, so ctrl-f and the three columns of links atop the page may prove your best friends. Think of this reference as the man page for the whole Google Maps API.

- And now it’s time to bring (well, mash) all this together. Enhance your map in such a way that when the user searches for a location (via that same text field and button you just added), not only is he or she whisked away to that location (if possible), the resulting map is also sprinkled with up to  $n \leq 5$  markers representing the  $n$  largest cities within view. Each city’s marker, when clicked, should trigger an info window to appear containing clickable links to news articles pertinent to the city.

Those articles’ titles and links, of course, should be culled via RSS from Google News based on the zip code(s) within view.

How to make all of this happen? Well, your map already knows how to whisk users away to some location provided via that text field. It sounds like, once done whisking, your map will need to go

GET, wink wink, some articles. Because of “same-origin policies,” your JavaScript code cannot contact Google News directly.<sup>14</sup> But it can contact, say, a PHP file that lives on `cloud.cs50.net` (where your JavaScript came from). Perhaps you could write a PHP script (whose name ends in `.php`) that accepts some HTTP parameters (say, the coordinates of a map’s southwest and northeast corners), figures out which zip codes live within the rectangle defined by those corners, and retrieves RSS news feeds for  $n \leq 5$  of those zip codes? Think of this PHP script as your JavaScript’s proxy to Google News. Of course, some cities have multiple zip codes, and you need to fetch news from the  $n$  largest cities (not zip codes) within view, so you might need to sum a few zip codes’ populations. But that should be doable, perhaps via `GROUP BY` and `SUM` in SQL or via simple arithmetic in PHP.

How to contact that PHP file via JavaScript? Well, you might want to re-examine `xhr-requests.html`, which we’re certain you looked at while reading up on this API’s services:

[http://code.google.com/apis/maps/documentation/services.html#XML\\_Requests](http://code.google.com/apis/maps/documentation/services.html#XML_Requests)

Note that the example relies on `data.xml`, whose structure you can see here:

<http://code.google.com/apis/maps/documentation/examples/include/data.xml>

You won’t, of course, be downloading a static XML file (as the example does `data.xml`), but surely your PHP script could spit out some XML that you yourself have designed? Incidentally, so that your PHP code does not spit out a “MIME type” of `text/html`, as it does by default, you’ll likely want to include a line like the below atop your PHP file:

```
header("Content-type: text/xml");
```

How to retrieve RSS from Google News and return just some articles and URLs in your own XML format? Well, this article you might like:

<http://www.ibm.com/developerworks/library/x-simplexml.html>

Odds are, you’ll need to generate XML that has a bit more structure (*i.e.*, nesting) than that example’s `data.xml`. After all, your PHP script will need to return multiple articles (*i.e.*, titles and URLs) for multiple cities.

How to navigate your PHP script’s XML (or, really, DOM) in JavaScript once returned to the browser? You might find that this example gets you started:

<http://www.captain.at/howto-ajax-process-xml.php>

---

<sup>14</sup>[http://en.wikipedia.org/wiki/Same\\_origin\\_policy](http://en.wikipedia.org/wiki/Same_origin_policy)

And here's an example that is similar in spirit but also discusses the PHP side:

[http://www.w3schools.com/php/php\\_ajax\\_xml.asp](http://www.w3schools.com/php/php_ajax_xml.asp)

Do not hesitate to turn to the Google itself for examples or tutorials beyond these. Or, for that matter, to the course's bulletin board or staff. Just because you've not done something before doesn't mean that you can't figure out how!

- Once you've gotten your map working, go back and ensure that it responds to events like `dragend`, `moveend`, `resize`, and/or `zoomend` by clearing any and all markers and then re-laying up to 5 representing the new area's news. You may want to re-visit the section on event listeners over here:

[http://code.google.com/apis/maps/documentation/events.html#Event\\_Listeners](http://code.google.com/apis/maps/documentation/events.html#Event_Listeners)

- Don't forget that your work must render and behave the same in at least two major browsers!
- Under no circumstances should your mashup generate JavaScript runtime errors. Be sure to check return values and for `null` and `undefined` where appropriate! For help chasing down bugs in your JavaScript code, incidentally, open up **JavaScript Debugger** (assuming you've installed it) under **Tools** in Firefox and watch its **Interactive Session** while playing with your mashup!

### Submitting Your Work.

- Ensure that your work is in `~/public_html/pset8/`. Then execute the command below, where `username` is your own username, in order to dump your database to disk for us; input your database's password (*i.e.*, your `DB_PASS`) when prompted.

```
mysqldump -u username -p username_pset8 > ~/public_html/pset8/username_pset8.sql
```

Now submit your work by executing the command below.

```
cs50submit pset8 ~/public_html/pset8/
```

Thereafter, follow any on-screen instructions until you receive visual confirmation of your work's successful submission. You may re-submit as many times as you'd like; each resubmission will overwrite any previous submission. But take care not to re-submit after the problem set's deadline, as only your latest submission's timestamp is retained.

**CS50 for Your Body.**

- Take one of life's lessons with you. Wear CS50. The course's store is now open for business.<sup>15</sup>

<http://cs50.net/store/>

Orders can be termbilled.

*Couldn't get enough of our problem sets this semester? Want to forever remember those long Friday nights you spent optimizing your binary tree implementation of Mispellings? Yellow border adds extra sexiness.*



<sup>15</sup> Why does a computer science course have its own line of clothing? That is a very good question.

*Surprise your friends with your extreme geekiness. Show them what it takes to be a real athlete!  
Material is soft and guaranteed to make you want to ooh and aah. Six-pack not included.*



*The classic design, now available as a sweatshirt in Navy blue.*



*Property of CS50 Sweatshirt with a hood.*



*Property of CS50 sweatpants to complement your sweatshirt.*



kthxbai