Mashup in a Day

A lesson plan to teach mashup technology

By
Jane McGookey
April, 2009

Mashup in a Day

A lesson plan to teach mashup technology

By
Jane McGookey

A project submitted in partial fulfillment of the requirements for the degree of

Master of Science in

Computer Information Systems

at

Grand Valley State University

April, 2009

_____

Your Professor Date

Abstract

Web 2.0 is the current buzz word for the organic way in which the Web is developing. Jump start your teaching of the programmable Web with "Mashup in a Day", an all-inclusive lesson plan on mashups. Your students will learn not only web terminology but be able to program a Google Map mashup. "Mashup in a Day" clearly shows how Web requests are processed by both the browser and the servers. Notes and slides highlight sections of code that are key to programming a successful mashup. "Mashup in a Day" includes lecture slides and notes, a project assignment, commented sample code and 20 questions (with answers!) for use on quizzes or tests. Come and see why mashup technology is the key to the programmable Web.

Table of Contents

Project Background

The purpose of this project, *Mashup in a Day - A lesson plan for teaching mashup technology*, is two fold. First, I have a passion for teaching computer technology and electronic communication to non-digital natives. This demographic usually is hesitant to learn “yet another” technology so it is necessary to be able to explain things with simple examples and non-technical jargon. Secondly, I have a desire keep up with Web 2.0 developments. During a GVSU masters presentation I learned about a Web technology called ‘mashup’. Immediately I knew this technology was powerful and wanted to learn to develop one myself.

Project Process

Initial research included reading a variety of mashup resourses, including books, websites, and blogs. I recorded progress on my project on my blog at whycompsci.wordpress.com. Meeting with my advisor, Dr Adams, twice a month, always proved worthwhile. His knowledge was freely shared and his suggestions have led a better project.

After the initial research was complete, it was possible to design and code a working mashup. Having a working mashup was crucial in developing the lecture materials for this project.

Step 1: Develop a mashup

As a staff member of the Presbytery of Lake Michigan I saw a need to visually display the locations of our churches. The current online directory displays a text based list of churches based on the selected city. This may or may not be useful for someone looking for a church. This problem can be solved by mashing the church locations with Google maps.

Step 2: Develop a lesson plan

To master a subject is to be able to teach it.

I documented of my progress on my blog, whycompsci.wordpress.com. These experiences and the working mashup code were instrumental in creating the lesson plan materials. The lesson plan consists of lecture slides and notes, class project assignment and sample test questions. The lecture should take one session of instruction.

Project Deliverables

Lecture Slides and Notes

The following pages are the lecture slides with instructor notes. The most up to date course materials can be downloaded from whycompsci.wordpress.com

Mashup in a Day

A lesson plan to teach mashup technology
Computer Science Master's Project

Developed by

Jane McGookey
Grand Valley State University

Lesson Preparation:

1. Make code available for students
2. Register for Google API key
3. Update code
4. Load code on web server for demonstrations
5. Verify code works on your server (GVSU servers do not allow php to get_file_contents from http GET)

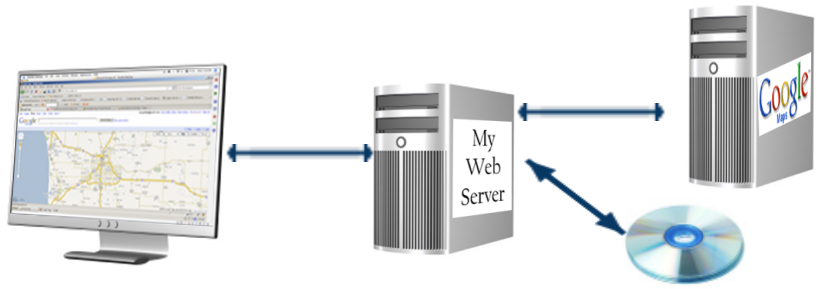
Introduce topic-
Mashup in a Day is a lesson plan to teach mashup technology in a single session.

A Mashup is....

an application that runs on a web server

combining one or more data sources

and displaying the results in a meaningful way.



A mashup is

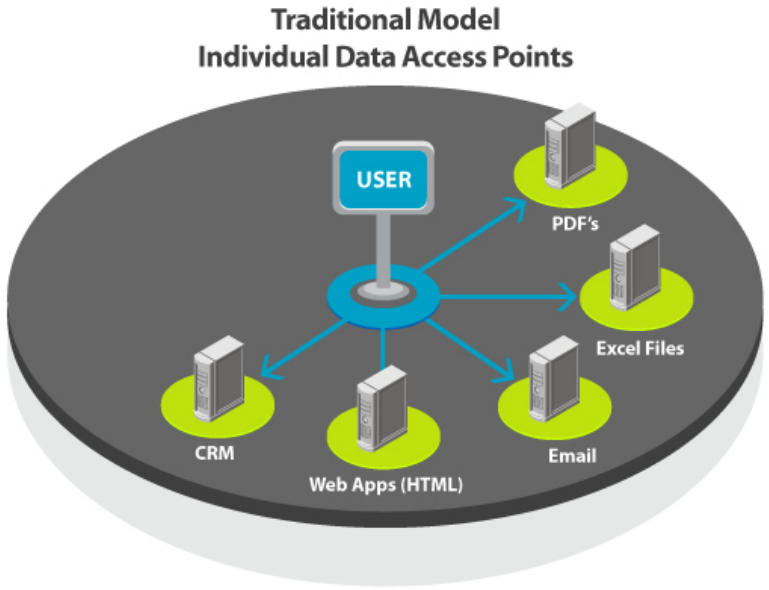
an application that runs on a web server

combining one or more data sources

and displaying the results in a meaningful way.

An example would be to display your favorite fishing holes. The list of locations would be stored on your web server and Google server would be used to generate map.

Web 1.0



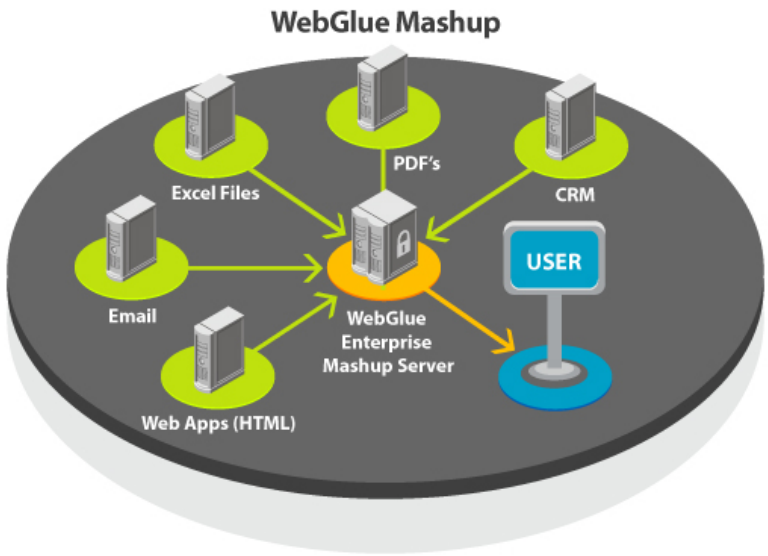
Source: www.rareplay.com/index.php?page=WebGlue

Web 1.0 centers on the user. The user searches for data. The user correlates the data.

For example, I am interested in researching specific breeds of dogs.

1. On my PC, I (the user) would have a spread sheet containing my list of dogs
2. I would use the web to look up facts about different the breeds
3. I would search the web for photos for variations of the breed.

Web 2.0



Source: www.rareplay.com/index.php?page=WebGlue

With Web 2.0, the web server is central. It allows the user to view data from multiple sources. A Mashup is the web programming needed to collect data from various sources and makes them available dynamically!

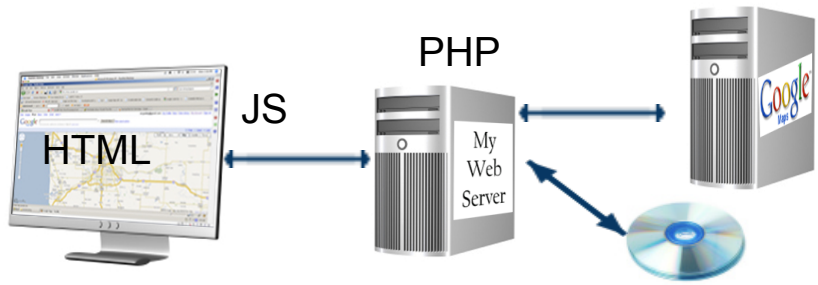
In our dog breed example, the mashup on the web server does the work by storing the list of breeds on the server, the server can be programmed to gather the breed information and photos then present all the information at once!

What you need to be familiar with

Model-View-Controller architecture

Browser/Web Server communications

- HTML, CSS
- JavaScript/AJAX
- PHP/MySQL
- [optional] JS framework



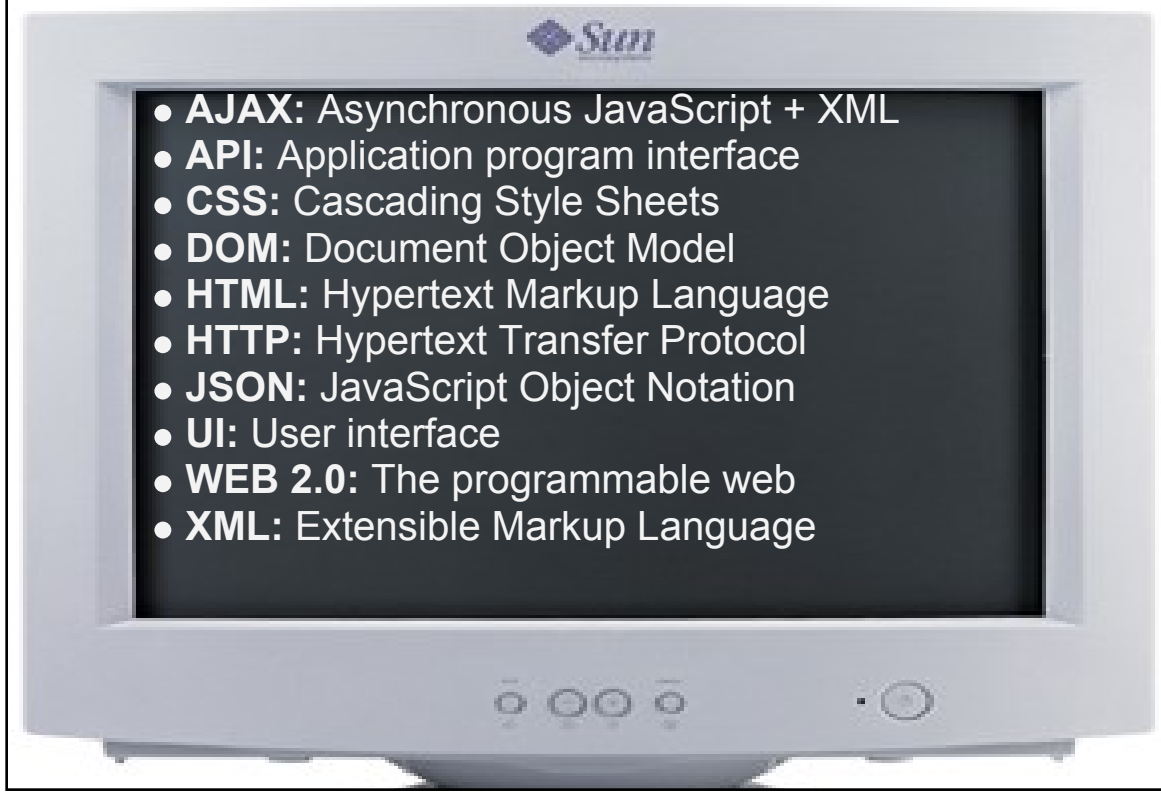
The Model is information gathered from the data sources by php/msql.

The View is handled by HTML and CSS. This is what your browser uses to generate the web page.

The Controller is the Javascript & AJAX that controls how and when the data is transferred to the client.

Walk thru diagram

Terms to know



Comprehension of terminology is important for all students to have a successful learning experience !

Highlight :

AJAX - Asynchronous JavaScript & XML - requests data from server and updates the web dynamically (the user does not have to refresh the browser)

JSON - JavaScript Object Notation which is easily readable by humans and elements are easily accessed programatically

Feel free to google these terms AFTER class

Steps to create a mashup

1. Decide purpose of mashup - what question to answer or problem to solve?
2. Locate the data you need to support #1
    - Do you already control the data?
    - Is the data available on the Internet?
        - Is it downloadable (static)?
        - Is it available programatically?
3. Define user input view
4. Define results view

Remember a mashup is a web application that combines one or more data sources and displays the data in more meaningful ways

1. It is very important to clearly define what the mashup will do.
2. Finding the data is the hardest part.
   Tasks like screen scraping and data entry should be avoided.
   Ideally the data can be found programmatically (API), this will keep your mashup from becoming outdated.
   Search engines can be useful in locating data that can be accessed programatically (API, developer areas)
3. If data can't be found, re-assess the purpose of your mashup.
4. Define user views only when steps 1 & 2 are complete

API Key

Why? To validate requests and replies between Web Server and Third-party servers

How?

1. Human requests key from third-party (e.g Google)
2. Human embeds key into code
3. Executed code sends key with requests to third-party server
4. Request is executed if sending server and key match.

API key provides some amount of assurance that the correct web server is requesting data from the third-party server.

How?

1. Human requests key from third-party (e.g Google)
2. Human embeds key into code.
3. Executed code sends key with request to third-party server.
4. Request is executed if sending server and key match.

Techniques

- Getting data from third party APIs may require knowledge of web protocols (REST, XML-RPC, RSS).
- Timing how long does it take to retrieve data from server

- Cache data on server (php/mysql)
- Defining views for user input and display of data (html, css, javascript, AJAX, flash)
- If you are using maps in your mashup, you need to understand geocoding.

Obtaining data from other sites programatically may require you to understand certain service oriented architectures and protocols (REST, RSS, XML-RPC)

Be aware of the time it may take data to be retrieve from third-party servers. It may be wise to cache data instead of requesting the data each time the mashup is executed. If Amazon is having a huge sale, it may be slower returning data to your program. Some servers limit the number of requests in a given period of time (minute, day, etc). This information should be available in the API documentation.

Defining user friendly views will increase the likelihood your mashup will be used multiple times. There are a variety of techniques to available to develop user interfaces (UI). The simplest is HTML with JavaScript.

AJAX however can be used to request data from your web server and update the display dynamically.

If you are using maps in your mashup , then you need to understand geocoding.

Geocoding

The process of converting a street address (town, zip code, etc) to a Latitude and Longitude.

1 Campus Dr, Allendale MI 49401
Lat = 42.9615303
Long = -85.8887345

Free Geocoding Services

geocoder.us
webgis.usc.edu
Google Maps
Yahoo Maps

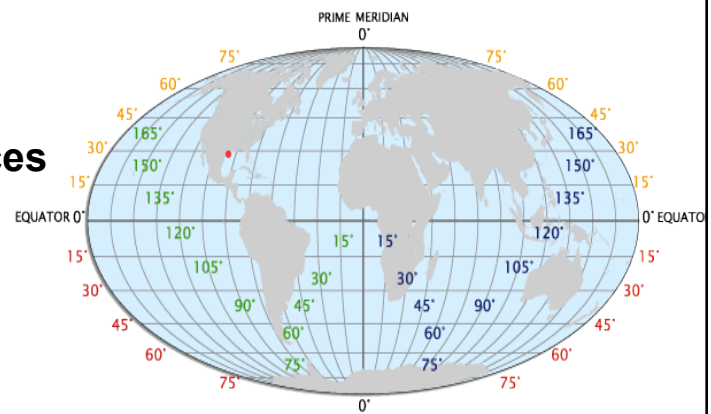


Image source www.worldatlas.com

Geocoding is the process of converting a street address to its latitude and longitude.

Example -
1 Campus Dr, Allendale MI 49401
Lat = 42.9615303
Long = -85.8887345
This slide lists some of the free geocoding resources on the web.

Code Examples

Class project
Quiz

Run the mashup dev.lakemichiganpresbytery.org/map/

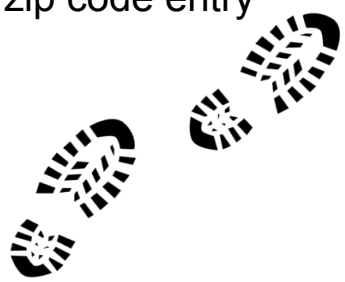
Step 1: Identify the problem: The current system displays text based list of churches by city. The solution is to plot locations on an interactive map showing.

Step 2: Data sources are:

- List of churches provided in XML format from the Presbytery of Lake Michigan (www.lakemichiganpresbytery.org)
- Google Maps & Google Geocoding

Step 3: User input view - simple form for zip code entry

Step 4: Results view - refocus and zoom map based on entered zip code



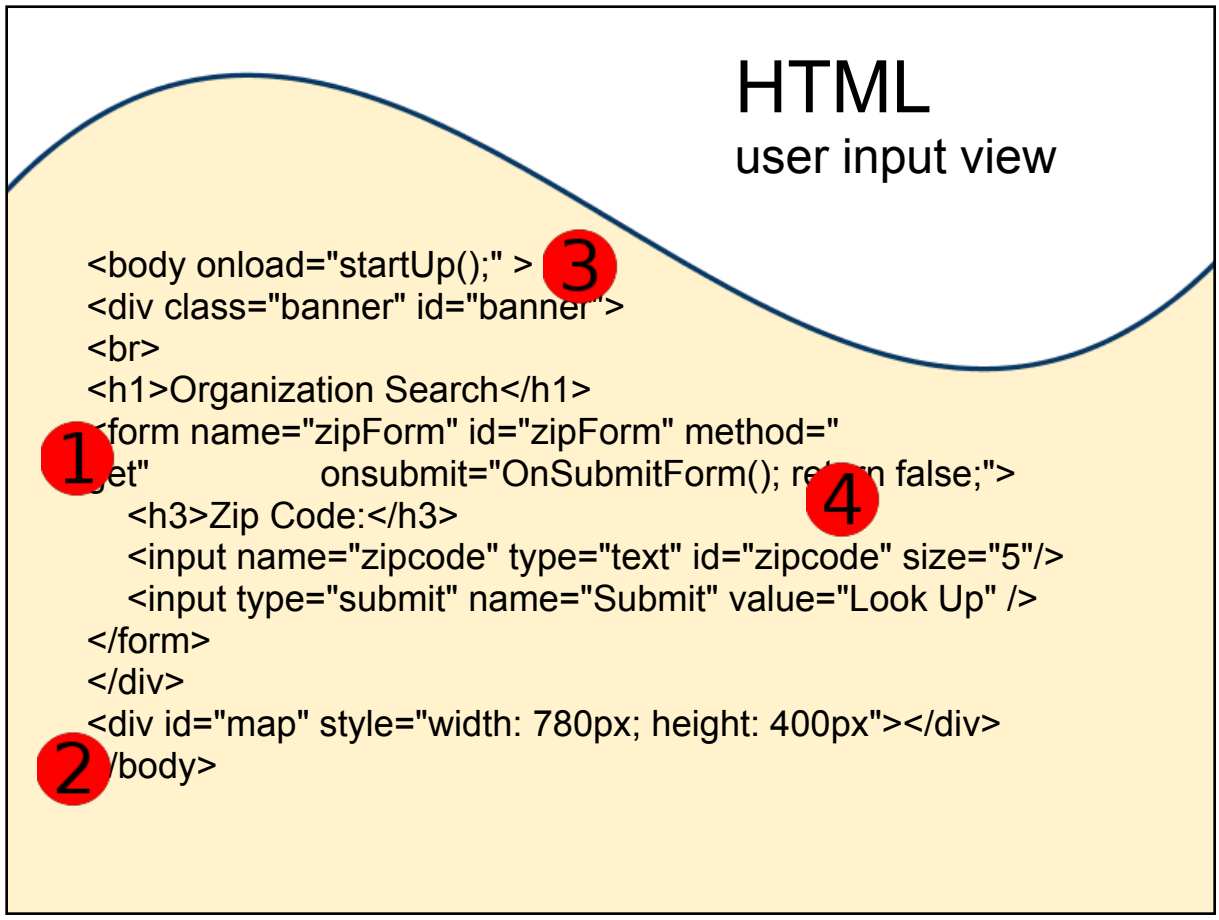
Make code available for students. Run the demo by clicking on the link in the slide.

Step 1: The current online directory displays a text based on list of churches based the selected city. This may or may not be useful for someone looking for a church. This problem can be solved by mashing the church locations with Google maps.

Step 2: Data sources are
1. List of churches provided in XML format from the organization.
2. Google Maps & Google Geocoding

Step 3: User input view - simple form for zip code entry

Step 4: Results view - refocus the map to center on zip code and increase magnification.



Check presentation time before continuing, adjust as needed

Client side - HTML for user input view

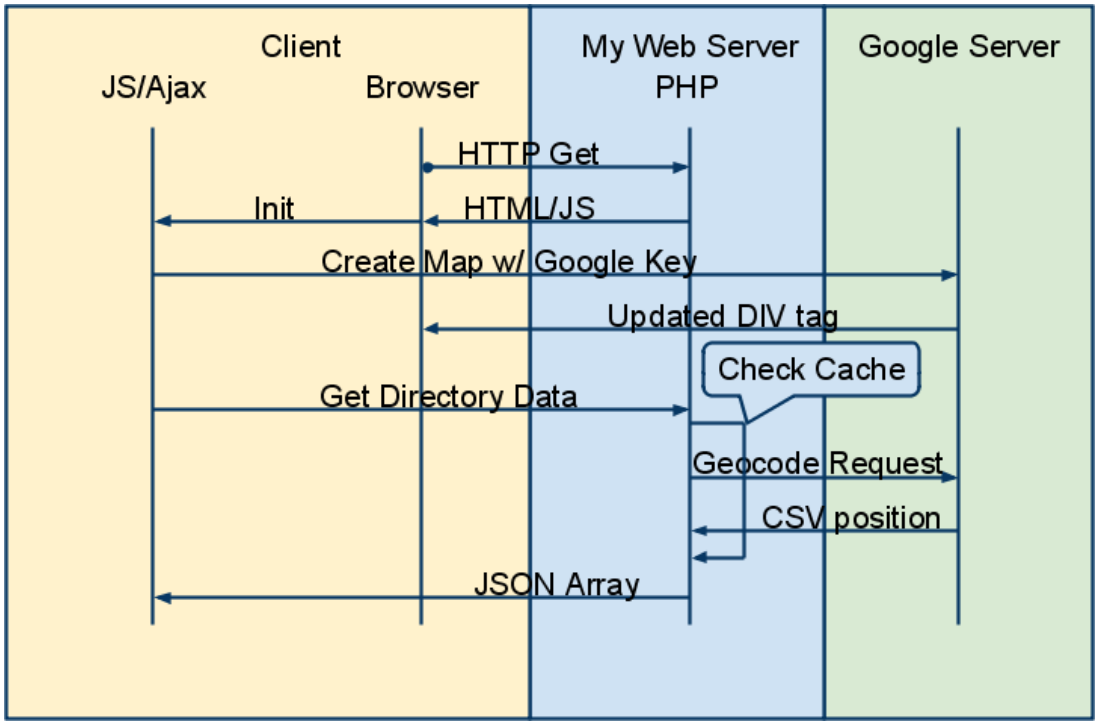
Notice how simple the HTML code is.
1. Set up the form

2. Create an area where the map will be display, this must be defined as a tag with an id called 'map' (Google updates this div element)

3. After the HTML loads, the browser runs the startUp Javascript function. This function requests the information to be displayed on the map (i.e. each church)

4. When the user enters a zip code the OnSubmitForm Javascript function is invoked to recenter and increase magnification on the map.

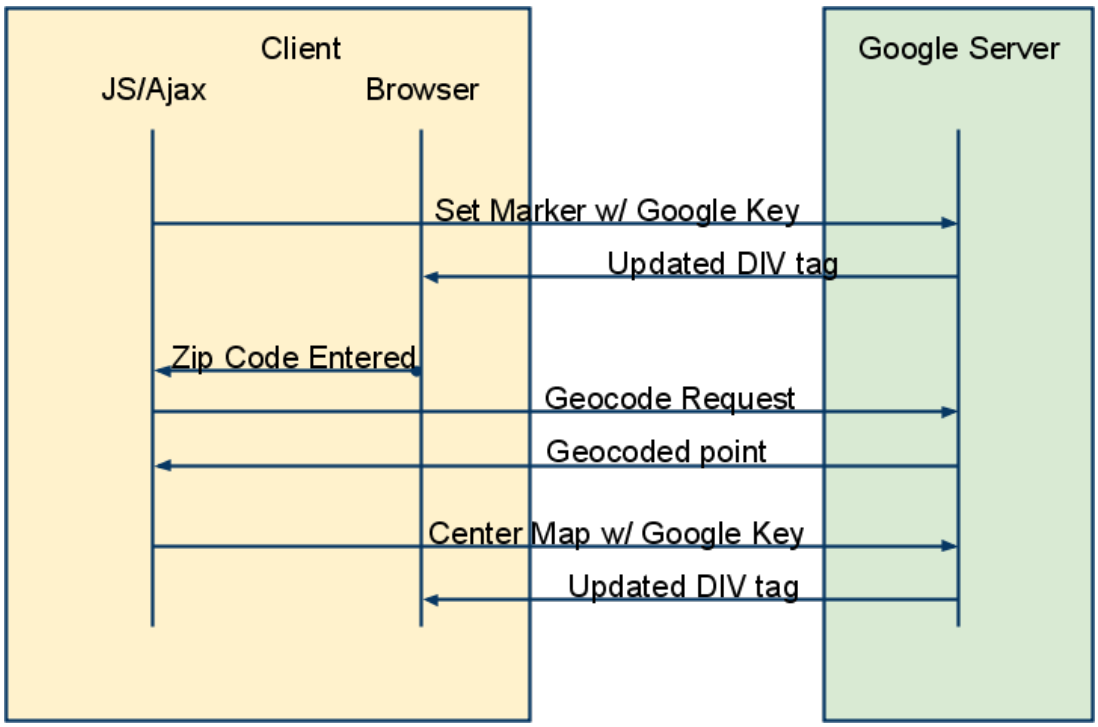
Mashup Timing and Interactions (1/2)



Code samples in this presentation will use the same colors as this diagram. It will help to reinforce request and response concepts..
Client - yellow
My web server - blue
Google - green

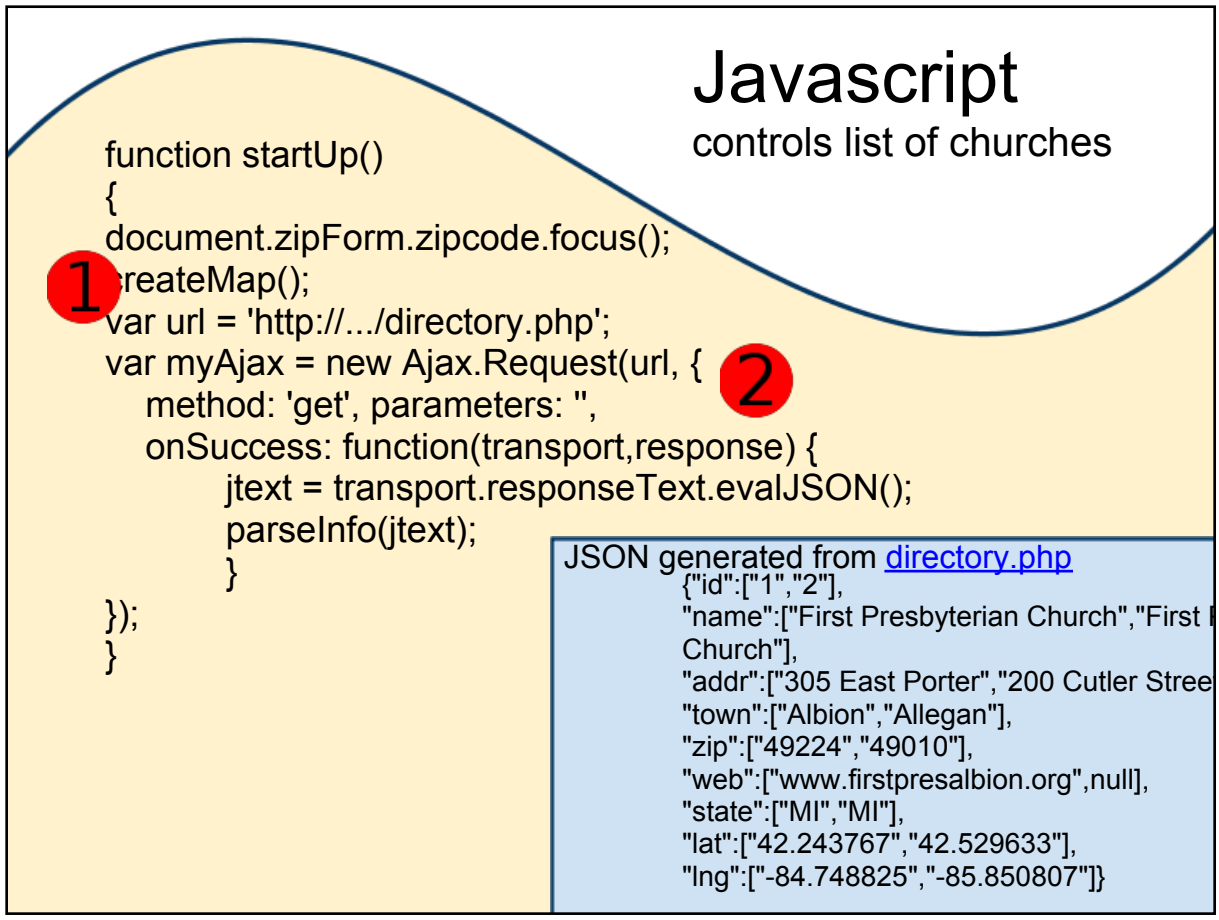
1. Start up - when the URL is typed into the browser an HTTP GET the server to send the web page to the Client.
2. Browser executes the code and displays the zip code form
3. Client requests Google to create the map. Google does this by updating the HTML div element
4. Client uses AJAX to request directory information from web server
5. Web server reads directory information from its local XML file. If position of church is not found in cache then a geocode request is sent to Google.
6. When the last church has been processed, the web server sends display information (name, lat, lng, address, website, etc) to the client as JSON array.

Mashup Timing and Interactions (2/2)



Note: After JSON array is received the client only talks with the Google server

1. The JavaScript in browser decodes the JSON array and requests Google to create the map markers
2. The browser then waits for user to enter a zip code.
3. When user enters a zip code, the browser requests Google to geocode the zip code and then requests Google to recenter the map and to increase the magnification



The browser invokes a JavaScript function called startUp that controls the loading and viewing of the church locations.

1. Request to Google to create a map

2. Create AJAX request which consists of:

a. The URL parameter is the location of the PHP portion of the mashup code on the web server

b. The OnSuccess status initiates a callback function that invokes a JavaScript function called parseInfo which is responsible for creating the map markers

c. The evalJSON method in the callback function is part of the Javascript framework called Prototype. Using a framework hides the details of the AJAX request simplifying the programming. Download the necessary include file from www.**prototype**js.org

Click on directory.php link in slide to demonstrate how to use the browser to test the PHP data service

Javascript

controls Google Marker placement

```
function setMarkerAndText
                    (mapPoint, text)
{
1  var marker = new GMarker (mapPoint);
2  map.addOverlay(marker);
3  GEvent.addListener (marker,
                "click",
                 function () {
                    marker.openInfoWindowHtml (text);
                  } //function
    ); // addListener
}
```

After the JSON array is received the Javascript function setMarkerAndText is invoked. This code segment shows how easy it is to create Google marker.

Demo - click on a marker to display Information Box

To add a marker to the map :

1. Request Google to create a new marker.

2. Ask Google to add the marker to the map overlay

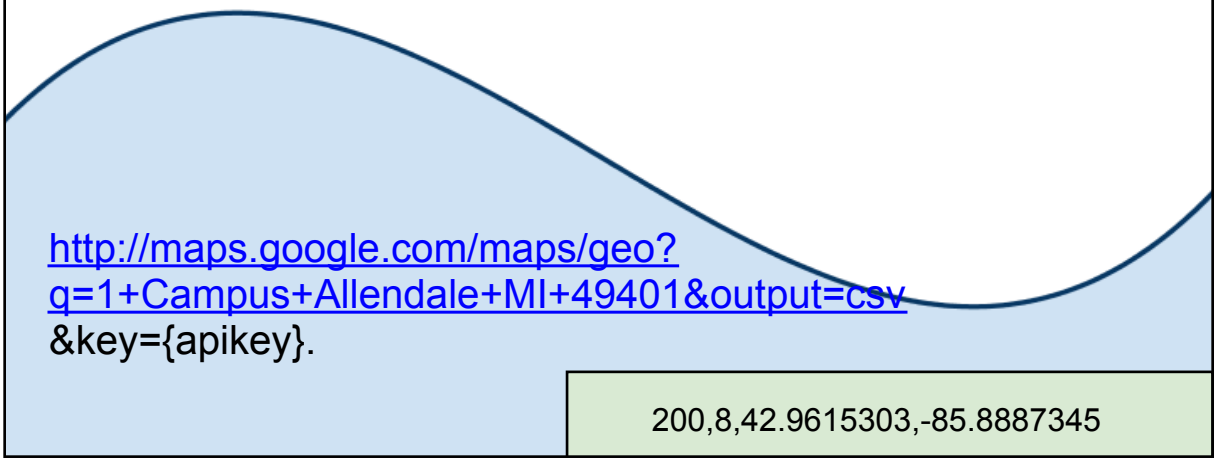
3. Set a Google Event to listen for a mouse click on the marker and define the data to be displayed in the information window

Coding a geocoding request

To retrieve the location of an address, you simply make a standard HTTP GET request.

Example: Geocode the address

1 Campus Dr
Allendale MI 49401



http://maps.google.com/maps/geo?q=1+Campus+Allendale+MI+49401&output=csv&key={apikey}.

200,8,42.9615303,-85.8887345

Server side (blue)

It is useful to echo to the screen your geocde request to verify you are getting the results expected.

Click on the link in the slide to show how the browser can be used to verify the HTTP GET request.

XML -
http://maps.google.com.....&output=xml

Notice if you change the output parameter, the information is returned in the desired format

CSV -
http://maps.google.com....&output=csv

JSON (default)
http://maps.google.com...&output=JSON

Code Examples

Class project

Quiz

Class Project

Create a mashup using two or more well-known Web resources. Some examples might are

- Identify and show photos of your favorite places (Google Maps, Flickr & your own data)
- Compare mapping applications (Google, Yahoo, etc)
- Map items for sale on Craigs list, eBay, Amazon, etc
- Average income per zip code (US Census data)
- Search for descriptions of the current New York Times Best Seller list (Amazon, News feed)

Check out these websites for more ideas:
www.programmableweb.com
www.webmashup.com

Distribute Class Project Description Handout
This project will provide students with an experience in designing a mashup, searching for the necessary data and them implementing the mashup.

Examples are:

- Identify and show photos of your favorite places, vacation spots, fishing holes, etc (Google Maps, Flickr & your own data)
- Compare mapping applications (Google, Yahoo, etc)
- Map items for sale on Craigs list, eBay, Amazon, etc
- Average income per zip code (US Census data)
- Search for descriptions of the current New York Times Best Seller list

Resources:
www.programmableweb.com
www.webmashup.com

Class Presentations

Two presentations are will be required.

Presentation #1 shall include:
- Purpose of mashup
- APIs used
- Anticipated Service Oriented Architecture/Protocols (REST, SOAP, RPC)
- Anticipated Data formats to be used (XML, JSON, etc)

Presentation #2 shall include:
- Demonstration
- Code extracts highlighting data procurement
- Lessons learned

Presentation #1 shall include:
- Purpose of mashup
- APIs used
- Anticipated Service Oriented Architecture/Protocols (REST, SOAP, RPC)
- Anticipated Data formats to be used (XML, JSON, etc)

Presentation # 2 shall include:
- Demonstrate working mashup
- Describe Service Oriented Architecture/Protocols used
- Describe data formats used
- Describe and show code samples
- Evaluation of design process
- Describe any lessons learned

Code Examples
Class project
Quiz

Sample quiz questions...

What is the first step in designing a mashup?

The first step in creating a mashup is to define the question to answer or problem to solved

Sample quiz questions...

What is the process called to convert an address to it's Latitude and Longitude?

Geocoding - converts an address to a latitude/longitude position

Sample quiz questions...

What does JSON stand for?

JS - Javascript
O - Object
N - Notation

JSON (JavaScript Object Notation) is a lightweight data-interchange format. It is easy for humans to read and write. It is easy for machines to parse and generate.

Sample quiz questions...

Why is an API key necessary when using third-party data?

Validates requests and replies between the web server and the third-party servers.

Demo Screenshot

Screen shot of the Presbytery of Lake Michigan church directory mashup. Code can be found in Appendix A.

Class Project Description

Purpose:
A mashup is a Web service that takes one or more data sources and correlates and/or displays the data in a more meaningful way. This lab will provide students with experience in designing, finding data sources and coding a mashup. Two presentations are required. The first presentation describes the "what and how" of the mashup. The second presentation requires the demonstration and review of a working mashup.

Task:
Working in a group or as an individual (depending on class size), create a mashup using two or more well-known APIs. Feel free to use sample code as a starting point.

Schedule:
Presentation #1 - 1 week
Presentation #2 - 2 weeks

The following websites may be useful in providing ideas and or resources for your project
www.programmableweb.com
www.webmashup.com

Preparation for presentation #1:

1. Determine the question you want to answer or problem you want to solve.
   Some examples are:
   - Identify and show photos of your favorite places, vacation spots, fishing holes, etc (Google Maps, Flickr & your own data)
   - Compare mapping applications (Google, Yahoo, etc)
   - Map items for sale on Craigs list, eBay, Amazon, etc
   - Average income per zip code (US Census data)
   - Retrieve description for book, CD, DVD by numbering standards (e.g. ISBN) etc for personal library (Amazon)

2. Find your data sources
   Dynamic - register for API keys before proceeding to step 3.
   - Search for APIs
   - Search RSS feeds

   Static Data
   - Search for downloadable data
   - Screenscrape
   - Data entry

If mysql is needed, contact System Administrator to gain access to database

3. Define the user interface. (Make sure you applied for API keys prior to starting this step - it may take several days before you receive your key)

- User input view
- Result view

Presentation #1 shall include:

- Purpose of mashup
- APIs used
- Anticipated Service Oriented Architecture/Protocols (REST, SOAP, RPC)
- Anticipated Data formats to be used (XML, JSON, etc)

Preparation for Presentation #2

1. Implement the mashup

Presentation # 2 shall include:

- Demonstration of mashup
- Describe Service Oriented Architecture/Protocols used
- Data formats used
- Describe and show sample code of data services used
- Evaluation of design process, describe any changes that may have occurred between the design and implementation

Sample Test Questions

1. **What does the term "Web 2.0" mean?** (Web 2.0 is a collaborative web which allows sharing of data securely where the web server does the work and not the user. It is more dynamic than Web 1.0)
2. **What does a mashup do?** (takes data from one or more data sources and presents them in new or different ways)
3. **In Web 1.0 who correlated data from different sources?** (the user)
4. **Name three ways to provide data to the mashup.** (APIs, RSS feeds, download static data, screen scrape, hand enter)
5. **What does AJAX stand for?** (Asynchronous JavaScript + XML requests data from web server and can update view without the user pressing refresh on the browser)
6. **What does XML stand for?** (Extensible Markup Language, data storage format)
7. **Why use a Javascript framework?** (makes writing web apps easier because it hides the details from the programmer - e.g. AJAX states)
8. **Why use an API key?** (The key provides validation of requests and replies between the web server and the third-party server.)
9. **What is geocoding?** (Determining the latitude and longitude of a position)
10. **Determining the latitude and longitude in a mapping mashup is called?** (geocoding)
11. **What does JSON stand for?** (JavaScript Object Notation)
12. **What is the advantage of transmitting data in JSON format?** (JSON is human readable and easily programmable. Using a javascript framework makes it easier to manipulate data using AJAX)
13. **Why would a programmer use php/mysql in a mashup?** (to cache data from other sources or to retrieve data already stored on the server)
14. **Name three well known third party application/data providers** (Google, Amazon, Flkr, ...)
15. **Explain a mashup in terms of the Model-View-Controller architecture.** (The Model is information gathered from the data sources by php/msql. The View is handled by HTML and CSS. This is what generates the web page. The Controller is the Javascript & AJAX that controls how and when the data is transferred to the browser.)

16. **What is the first step in designing a mashup** (Determine the question you want to answer)
17. **What is the hardest step to create a mashup** (Finding the data)
18. **What is the minimum number of web servers required for a mashup?** (one - the web server running your php code)
19. **How can a browser help debug the mashup?** (type the http command into the browser to verify the GET returns the expected data)
20. **In your opinion what is the future of mashup technology?**

Project Evaluation

Mashup in a Day - A lesson plan for teaching mashup technology, was developed as a one session lecture. Many enhancements would be needed to make the lesson plan more robust to represent the diversity of mashup techniques currently available.

Enhancements

RSS – the Web has exploded with data that is available through RSS feeds. The lesson plans should be augmented to demonstrate and dissect an RSS mashup

Screen scraping – there will always be websites that will not expose their data in programmatic ways. This technique should be discussed. Students could be encouraged to find automated solutions to this problem.

Mashup Editors – Google and others now offer “mashup editors”. As more APIs are developed it should be easier to create more complex mashups using these tools.

Conclusions

While the concept for mashup technology is relatively simple, I underestimated the complicated ways in which third-party data can be obtained. This is the reason the Class Project Assignment has two parts. The first part will allow students to teach each other about a variety of service oriented architectures. The second part will show students how a coding experience may change ones original assumption on how a specific protocol works.

Due to the exponential rate by which web technology is developing it is likely mashup programming as developed for this project will be absorbed into the broader topic of web programming. Mashup editors will become more prevalent and will likely be freely available. A drag and drop mashup editor would not be unrealistic.

Appendix A

Appendix A consists of installation instructions. As well as the code used in the church directory mashup. File included are:

- index.html
- mapfunctions.js
- directory.php

Installation instructions from readme.txt

To install the sample code on your web server you must :

1. have a web server that supports php, mysql - php MUST support call to file_get_contents (Note: GVSU web servers do not allow students to do this)
2. extract the code directory in the mashup_in_a_day.zip file (located on CD or at http://whycompsci.wordpress.com)
3. apply for and update code (directory.php and index.html) with your web server Google map API key (http://code.google.com/apis/maps/signup.html)
4. configure directory.php with your mysql information
5. have in your mysql database create a table called Mash_Locations with text fields for zip and name and float fields for latitude and longitude.
   CREATE TABLE Mash_Locations (zip text, name text, latitude float, longitude float);
6. update the URL in startUp in index.html
7. copy all code and sample data to the web server

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
<!--
    index.html
    Author: Jane McGookey
    Date: April, 2009
    Create the input view for the zipcode lookup mashup
-->
<head>
    <!-- Set the title of the web page -->
    <title>Directory Mashup</title>
    <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />

    <!-- Update the look of your form using the css file -->
    <LINK REL=StyleSheet HREF="style.css" TYPE="text/css">
    <!-- Update the following with your google key -->
    <script src="http://maps.google.com/maps?file=api&amp;v=2&amp;key=URGOOGLE_MAP_API_KEY"
         type="text/javascript"></script>

    <!-- Include mapfunctions.js  to create map, center map and add markers -->
    <script src="mapfunctions.js"> </script>

    <!-- Include Javascript framework to make ajax call easier and to use evalJSON on responseText -->
    <script src="prototype-1.6.0.3.js"> </script>



<script language="javascript">

    // when a zipcode is submitted, refocus the map to that zip and zoom in

    function OnSubmitForm()
    {
        if (document.zipForm.zipcode.value.length != '5' ) {
         alert('Invalid zipcode');
        }
        else {
         centerMap(document.zipForm.zipcode.value);
        }
    }

    // set cursor in form box, generate ajax request to retrieve JSON format of location data

    function startUp()
    {
        document.zipForm.zipcode.focus();  //put cursor in form field
        createMap();
        // update this url with the location of your directory.php file
        var url = 'http://www.URurl.com/directory.php';
//        uncomment the following line and copy then paste the address into your browser to verify you have the correct URL
//        alert (url);
        var jtext;
        var myAjax = new Ajax.Request(
            url,
            {
                method: 'get',
                parameters: '',
                onSuccess: function(transport,response) {
                    jtext = transport.responseText.evalJSON();
                    parseInfo(jtext);
                }
            });
    }

    // parseInfo - parses the JSON fields into information needed for the google marker

    function parseInfo(organization)
    {
        for (var i = 0; i < organization.name.length; i++) {
            generateMarker(organization.lat[i],organization.lng[i],
                        generateMarkerHTML(organization.name[i],organization.addr[i],organization.town[i],organization.state[i],organization.zip[i],organization.web[i]));

        }
    }

    // generateMarkerHTML - generates the HTML used when displaying the information box of the marker

    function generateMarkerHTML (name,addr,town,state,zip,web) {
```

```
        if (web != null) {
        txt = "<div class=marker> <a href=http://"+web+" target=_blank> " +
                 name  +   "</a><br>" +
                 addr  +  "<br>"  +
                 town  +  " " + state  + " "  + zip +
                 "</div>";
        }
        else {
        txt = "<div class=marker>"  +
                 name  +   "<br>" +
                 addr  +  "<br>"  +
                 town  +  " " + state  + " "  + zip +
                 "</div>";
        }

        return txt;
    }



</script>


</head>

<body onload="startUp();" >
    <div class="banner" id="banner">
        <br>
        <h1>Organization Search</h1>
        <form name="zipForm" id="zipForm" method="get" onsubmit="OnSubmitForm(); return false;">
            <h3>Zip Code:
                <input name="zipcode" type="text" id="zipcode" size="5"/>
                <input type="submit" name="Submit" value="Look Up" />
            </h3>
        </form>
    </div>

    <!-- You must have a div called map for google maps to work!  -->
     <div id="map" style="width: 780px; height: 400px"></div>

</body>
</html>
```

```
1  //
2  //    mapfunctions.js
3  //    Author: Jane McGookey
4  //    Date: April, 2009
5  //
6  //    Google Mapping functions for directory mashup
7
8      var map = null;
9      var geocoder = null;
10     var debug = false;
11
12 //    createMap - creates the map, adds zoom controls and centers it North of Lansing MI
13
14     function createMap () {
15         if (GBrowserIsCompatible()) {
16             map = new GMap2(document.getElementById("map"));
17             map.addControl (new GSmallMapControl());
18             map.setCenter(new GLatLng(43, -85.5), 7);  //North of Lansing MI
19             // Create the geocoder
20             geocoder = new GClientGeocoder();
21         }
22     }
23
24
25 //    centerMap - geocodes the given address, then centers the map
26
27     function centerMap (addr) {
28         if (GBrowserIsCompatible()) {
29             if (addr != undefined) {
30                 if (geocoder) {
31                     geocoder.getLatLng(addr,
32                                        function(cpoint) {  //point is a GLatLng point
33                                            if (!cpoint) {
34                                                if (debug) alert ("centerMap  Address Not Found");
35                                            } else {
36                                                map.setCenter(cpoint, 10);
37                                            } // point returned
38                                        } //function
39                     )  //getLatLng - callbck will center map
40                 } else {
41                     alert ("CenterMap Address undefined= "+addr);
42                 } // geocoder
43             }  // addr != undefined
44         } //Browser Compatible
45     }
46
47
48 //    generateMarker - converts Lat/Lng into a google map point
49
50     function generateMarker (lat, lon, addressText) {
51         if (addressText != undefined) {
52             if (geocoder) {
53                 point = new GLatLng(lat, lon);
54                 if (!point) {
55                     alert ("showAddress Address not found in call to getLatLng, "+address);
56                 } else {
57                     setMarkerAndText (point, addressText);
58                 } // point returned
59             };// if geocoder
60         }
61     }    //generateMarker
62
63
64
65 //    setMarkerAndText - takes a google map point and information text and creates a google marker,
66 //                       add event listener for click (so information will be displayed)
67
68
69     function setMarkerAndText (mapPoint, text) {
70         var marker = new GMarker (mapPoint);
71         map.addOverlay(marker);
72         GEvent.addListener (marker,
73             "click",
74                 function () {
75                 marker.openInfoWindowHtml (text);
76                 } //function
77             ); // addListener
78     }
```

```php
<?php
/*
    directory.php
    Author: Jane McGookey
    Date: April,2009

    Data service to provide information about locations.
    Inputs:
        XML file of locations
        mySql cached Lat/Lng
    Output:
        JSON format of data
*/
header('content-type:text/javascript');

// update mysql database with your information

$DB_Table = 'Mash_Locations';
$DB_User = 'URusername';
$DB_Password = 'URpassword';
$DB_Host = 'localhost';
$DB_Name = 'URdatabaseName';

// Structure of information needed for the location plotted on the map
class aLocation
{
    var $id;
    var $name;
    var $addr;
    var $town;
    var $state;
    var $zip;
    var $web;
    var $lat;
    var $lng;
}
// object containing all locations
class locations
{
    var $aLocation;
}
$directory = new locations;

/*
    readdirectoryInfo - extracts data from the XML file, checks cache for Lat/Lng if not found, query google for Lat/Lng
*/
function readdirectoryInfo () {
  $locResult = array();
  $directoryData = new locations;
  $doc = new DOMDocument();
  $doc->load( 'directory.xml' );

  $oneOrg = $doc->getElementsByTagName( "Results" );
  $index = 0;

  foreach( $oneOrg as $eachdirectory )
  {
    $ids = $eachdirectory->getElementsByTagName( "ID" );
    $directoryData->id[$index] = $ids->item(0)->nodeValue;

    $iname = $eachdirectory->getElementsByTagName( "ChurchName" );
    $directoryData->name[$index] = $iname->item(0)->nodeValue;

    $iaddr = $eachdirectory->getElementsByTagName( "ChurchAddress" );
    $directoryData->addr[$index] = $iaddr->item(0)->nodeValue;

    $itown = $eachdirectory->getElementsByTagName( "Town" );
    $directoryData->town[$index] = $itown->item(0)->nodeValue;

    $izip = $eachdirectory->getElementsByTagName( "Zip" );
    $directoryData->zip[$index] = $izip->item(0)->nodeValue;

    $iweb = $eachdirectory->getElementsByTagName( "ChurchWeb" );
    $directoryData->web[$index] = $iweb->item(0)->nodeValue;

    $directoryData->state[$index] = "MI";

    $locResult = lookupPosition ($directoryData->name[$index],$directoryData->zip[$index]);

    if ($locResult[0] == null) { //only check 1/2 of position either both valid or invalid
```

```
81          //    Look up lat/lon with google
82          $locResult = genLatLon(
83                                  $directoryData->addr[$index],
84                                  $directoryData->town[$index],
85                                  $directoryData->state[$index],
86                                  $directoryData->zip[$index]);
87      }
88
89      $directoryData->lat[$index] = $locResult[0];
90      $directoryData->lng[$index] = $locResult[1];
91
92      $index++;
93
94    }
95    return $directoryData;
96  }
97
98  /*
99      genLatLon - create http request to google for lat/lng of given address
100 */
101 function genLatLon($addr,$town,$state,$zip)
102 {
103     $glatlng = array();
104
105     define("MAPS_HOST", "maps.google.com");
106     // update with your google key
107     define("KEY", "URGOOGLE_MAP_API_KEY");
108     $delay = 0;
109     $base_url_s = "http://" . MAPS_HOST . "/maps/geo?";
110     $base_url_e = "&output=csv&sensor=false&key=" . KEY;
111
112       $geocode_pending = true;
113
114       while ($geocode_pending) {
115         $street = str_replace (" ", "+",$addr);
116         $city = str_replace (" ", "+",$town);
117         $zcode = str_replace ( " ", "+", $zip);
118
119         $gmapAddr[$i] =$address = $street  .",+".$city  .",+".$state.",+".$zcode;
120
121         $request_url = $base_url_s . "q=" . $address . $base_url_e;
122 //        uncomment the following line to debug http request to google (copy the request and paste into brow
    ser)
123 //        echo $request_url."<br>";
124
125         $csv =  file_get_contents ($request_url);
126 //        uncomment the following line to determine if your web server supports the file_get_contents functi
    on
127 //        echo $csv
128         $csvSplit = split(",", $csv);
129         $status = $csvSplit[0];
130         if (strcmp($status, "200") == 0) {
131           // successful geocode
132           $geocode_pending = false;
133           $glatlng[0] = $csvSplit[2];
134           $glatlng[1] = $csvSplit[3];
135           cachePosition($name,$zip,$glatlng[0],$glatlng[1]);
136         } else if (strcmp($status, "620") == 0) {
137           // sent geocodes too fast
138           $delay += 100000;
139         } else {
140           // failure to geocode
141           $geocode_pending = false;
142           $glatlng[0] = null;
143           $glatlng[1] = null;
144           echo "Failed Address request=" . $request_url . " <br>";
145           echo "Received status =|" . $status . "|<br>\n";
146         }
147         usleep($delay);
148       }
149
150       return $glatlng;
151 }
152 /*
153     lookupPosition - check mysql database for cached Lat/Lng
154 */
155 function lookupPosition ($name,$zip) {
156     global $DB_Table;
157     $myPos = array();
158
159     $query = "SELECT latitude, longitude";
```

```
    $query .= " FROM ". $DB_Table;
    $query .= " WHERE zip = '".$zip."' AND name = '".$name."';";

//    echo "Database query = ".$query."<br>";
    $result = mysql_query ($query);
    // Get the data.
    $valid = false;
    $myPos[0] = null;
    $myPos[1] = null;
    if ($result) {
        if ($row = mysql_fetch_array ($result)) {
            $myPos[0] = $row [0];
            $myPos[1] = $row [1];
            $valid = true;
        }
    }

    return $myPos;
}
/*
    cachePosition - store the lat/Lng in the cache with the Name and Zip combo being unique
*/
function cachePosition ($name,$zip,$lat,$lng) {
    global $DB_Table;

    $iquery = "INSERT INTO ". $DB_Table;
    $iquery .= " (name, zip, latitude, longitude) VALUES ";
    $iquery .= "('".$name."','".$zip."',".$lat.",".$lng.");";
//    echo "Database query = ".$iquery."<br>";

    $iresult = mysql_query ($iquery);
    if (!$iresult) {
        echo ("DB Insert fail result =".$iresult."<br>");
    }
}

/*
    Main program
*/


// open database (cached positions) connections
$con = mysql_connect ($DB_Host, $DB_User, $DB_Password);
if (!$con) {
 die ('Could not connect to MYSQL:'.mysql_error());
}
$sel = mysql_select_db($DB_Name, $con);
if (!$sel) {
     die ('Could not select the database:'.mysql_error());
}

// read locations information
$directory = new locations;
echo ('alert("locations read")');;
$directory = readdirectoryInfo();

// close database connection
mysql_close($con);

// output location information in json format
echo json_encode($directory);


?>

```