Developing Google Mashups

Unit Objectives

After completing this unit, you should be able to:

- Programmatically manipulate a Spry data set
- Create a Google Mashup using Google Maps and data from a ColdFusion server

Unit Topics

- Introduction to Mashups
- Getting Started with Google Maps
- Google Map Basics
- Programmatically Geocoding Addresses
- Making Background Server Requests to CF 8
- Defining Map Markers
- Responding to User Events

Introduction to Mashups

Mashups combine data from different sources into a single tool. They are usually divided into the following four categories:

- Mapping (http://www.sawbuckrealty.com)
- Video and Photo (http://www.flickr.com)
- Search and Shopping (http://www.secretprices.com)
- News (http://debates.news.yahoo.com/)

Mashups can be developed using any development environment that produces an application capable of making HTTP requests. While the vast preponderance of mashups are currently developed using AJAX techniques, a growing number of Flash, Flex, and AIR based mashups are starting to emerge.

The emergence of Mashups has led the major software development companies to embark on a mission to make the creation of mashups easier. Products currently in beta include the following:

- Yahoo Pipes (pipes.yahoo.com)
- Google Mashup Editor
- Microsoft Popfly

This unit focuses on leveraging Google Maps with ColdFusion-based data using Adobe Spry.

Getting Started with Google Maps

Google Maps (http://maps.google.com) is a variation on the Google AJAX Search API that plots locations of search results on a map. It includes a very rich, yet relatively simple to deploy API that enables you to leverage interactive mapping in your web applications. The following solutions were deployed very quickly using the Google maps API:

- See What's Nearby our Training Center: http://training.figleaf.com/Locations/Directions-DC.cfm
- Directions to Solomon Eye Associates: http://www.solomoneyeassociates.com/locations/bowie.htm
- Heart Rhythm Society Find a Specialist: http://www.hrsonline.org/PatientInfo/specialist_locator.cfm
- Find an Orthodontist: http://www.braces.org/locator_search.cfm

Creating a Google Account

You must create a Google account in order to work with most of Google's API's. If you already have a GMail account, then you're ready to proceed. Otherwise, you can apply for a free account at the following URL:

https://www.google.com/accounts/NewAccount

Procuring a Google API License Key

In order to work with Google Maps, or any of Google's other data API's, you need to get a license key. License keys are tied to both your Google account and the domain name where your application will be hosted. You can get a license key at the following URL:

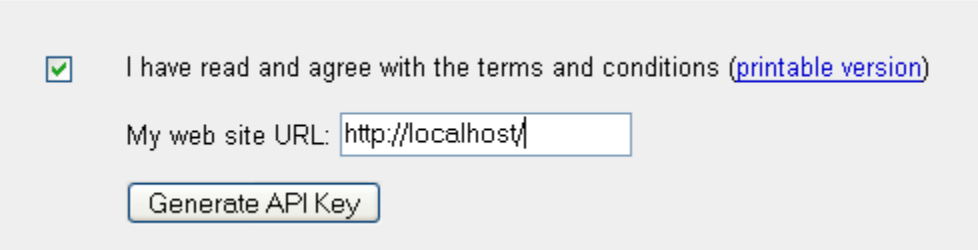
http://code.google.com/apis/maps/signup.html

Your Maps license comes with the following restrictions:

- There is no limit on the number of page views you may generate per day using the Maps API.
- There is a limit on the number of geocode requests per day.
- The Maps API does not include advertising.
- Your service must be freely accessible to end users.
- You may not alter or obscure the logos or attribution on the map.
- Maps should not be used to display illegal activity or reveal personal information

If your service is not freely accessible, you will need to contact Google to license.

The figure below illustrates the process for generating an API key:



As a best practice, you might want to store your API key in a dynamic ColdFusion variable to make it easier to migrate from development into production.

Google Map Basics

Deploying a simple map on a web page is a relatively straightforward affair. You simply have to complete the following steps:

- Use the Google AJAX API Loader to authenticate your application with your license key
- Load the Maps JavaScript library
- Position your map on a web page
- Geocode your starting position on the map

Using the Google AJAX API Loader

Google has four major AJAX API's:

- Maps
- AJAX Search API
- AJAX Feed API
- Data API's

Google Data API's comprise the following services:

- Apps API
- Base data API
- Blogger data API
- Calendar data API
- Code Search data API
- Notebook data API
- Spreadsheets data API
- Picasa Web Albums data API
- Document List data API
- YouTube data API

Authentication and Instantiation

Using the Google AJAX API Loader you can authenticate against all of these API's through a single callout. In addition, you can potentially load each API into your application based on a user-driven event. For example, your application might initially start off using the Maps API, but when a user clicks on a data point, it could dynamically load the search API. This approach can help minimize the initial loading time of your pages.

Note the following:

- Using the AJAX API Loader, all maps API calls are accessible as google.maps.*methodname*
- You can use the google.maps.BrowserIsCompatible() method to determine whether a user's browser is capable of correctly executing the APIs.
- In order to minimize the potential for browser memory leaks you should use the google.maps.Unload() method executed as a body unload event.

In the following example, the web page authenticates itself to Google using a parameterized API key and loads version 2 of the Maps API.

```
<!--- authenticate --->
<script type="text/javascript"
src="http://www.google.com/jsapi?key=#application.mapskey#
"></script>

<!--- load maps API --->
<script language="javascript">
     // load most recent stable version 2
     google.load("maps", "2");
</script>

<!--- unload API --->

<body unload="google.maps.Unload()">
```

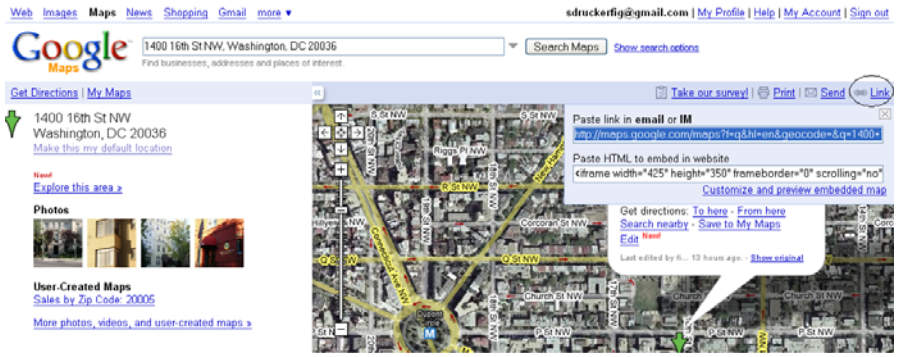
Placing your map on a page

A map can be loaded into any addressable tag set with a unique identifier – typically a . The map will conform to the defined size of the container. A typical map container would resemble the following:

```
<div id="map" style="width: 500px; height: 300px"></div>
```

Geocoding a starting position

Before you can start issuing commands to a map, you must invoke the google.maps.setCenter() method and pass in a latitude/longitude coordinate. You can quickly geocode a starting address by going to maps.google.com, typing in an address, and then clicking the *link* hyperlink. The latitude/longitude of your address will be embedded in the link as the ll parameter as indicated in the following screen shot:



Putting it all together

The full syntax for displaying a simple map is the following:

```
<cfoutput>
<head>
<script type="text/javascript"
src="http://www.google.com/jsapi?key=#application.mapskey#
"></script>

<script language="javascript">
  google.load("maps", "2");
</script>

<script type="text/javascript">
function initialize() {
var map = new
google.maps.Map2(document.getElementById("map"));

map.setCenter(new
google.maps.LatLng(#application.mapsStartingPosition#),
13);
    }
   google.setOnLoadCallback(initialize);
</script>
  </head>
  <body onunload="GUnload()">
    <div id="map" style="width: 500px; height:
300px"></div>
</cfoutput>
```

Customizing the look/feel of a map

Once your map has been instantiated, you can add controls using the google.maps.addControl() method. Supported controls include the following:

- LargeMapControl - a large pan/zoom control used on Google Maps. Appears in the top left corner of the map by default.
- SmallMapControl - a smaller pan/zoom control used on Google Maps. Appears in the top left corner of the map by default.
- SmallZoomControl - a small zoom control (no panning controls) used in the small map blowup windows used to display driving directions steps on Google Maps.
- ScaleControl - a map scale
- MapTypeControl - buttons that let the user toggle between map types (such as Map and Satellite)
- HierarchicalMapTypeControl - a selection of nested buttons and menu items for placing many map type selectors.
- OverviewMapControl - a collapsible overview map in the corner of the screen

Example

```
var map = new
google.maps.Map2(document.getElementById("map"));

/* add a toggle between map/satellite view */
map.addControl(new google.maps.MapTypeControl);
```

Walkthrough 1: Getting Started with Maps

In this walkthrough you will manually geocode an address and display that location on a Google map using the API.

Steps

1. Open a web browser and go to http://maps.google.com
2. Enter an address of 1400 16th Street NW Washington DC
3. Click on Search Maps
4. Click on **Link**
5. Copy and paste the displayed link into your web browser
6. Copy the value of the *ll* url parameter to your clipboard
7. Open /mashups/Application.ini in Dreamweaver
8. Paste the coodinates from your clipboard, overwriting the value of the mapsStartingPosition variable
9. Save and close the file
10. In Dreamweaver, open /walk/Application.cfc and review the code with your instructor. Note that both the Google API license key and the starting position of the map are loaded into application variables.
11. In Dreamweaver, open /walk/walk6-1.cfm
12. Directly under the <title> tags, insert the script declarations to initialize the Google API's and load the maps API. Your code should resemble the following:

```
<script language="javascript"
src="http://www.google.com/jsapi?key=#application.mapskey#
"></script>
<script language="javascript">
  google.load("maps", "2");
</script>
```

13. Where indicated by the comment for step 13, define your location for your map by inserting a tag with the following attributes:

    id = "map"
    style = "width: 500px; height: 300px; float:left;"

14. Underneath the google.load() call from step 3, define a new function named **initialize()**

15. Inside the initialize() function, invoke the google.maps.map2() method to instantiate a map inside of the tag that you defined from step 4. Your syntax should appear as follows:

```
map = new
google.maps.Map2(document.getElementById("map"));
```

16. Underneath the code that you inserted from the prior step, set the focus location of the map to the coordinates stored in the Application.ini file using the setCenter() method. Set a zoom level of 13. Use the following syntax:

```
map.setCenter(
new
google.maps.LatLng(#application.mapsStartingPosition#),13)
);
```

17. Outside of the initialize() function, just before the closing </script> tag, invoke the google.setOnLoadCallback() method to execute the initialize() function. Your code should appear as follows:

```
google.setOnLoadCallback(initialize);
```

18. Save the file and browse. You should see a map appear on screen.
19. Return to Dreamweaver
20. Just before the setCenter() method, display a map type toggle by issuing the following directive:

```
map.addControl(new google.maps.MapTypeControl);
```

21. Save the file and browse. You should see a toggle control in the top-right of the map.
22. Allow the user to zoom in and out of the map by issuing the following directive, right after the code that you inserted from step 11:

```
map.addControl(new google.maps.SmallZoomControl);
```

23. Save the file and test

-- End of Walkthrough --

Programmatically Geocoding Addresses

In order to plot a position on a map, you must know its geocode, or latitude/longitude. You can programmatically determine the geocode of an address by using the google.maps.ClientGeocoder object. Once this object has been instantiated, you can run the getLatLng() method to obtain the coordinates of an address. Note the following:

- The getLatLng() method is server-intensive. As a result, Google does restrict the number of addresses that can be geocoded per day.
- For best results, store geocodes of specific locations in your database
- When calling the getLatLng() method, you must specify a callback function that will receive the calculated coordinates

In the following example, a Google map is centered on a specific address:

```
function plotOrigin(point) {
	if (!point) {
		alert("address not found");
	} else {
		 map.setCenter(point,13);
	}
}

function fnPlotStartPoint(address) {
	var geocoder = new google.maps.ClientGeocoder();
	geocoder.getLatLng(address,plotOrigin);
}

fnPlotStartPoint("1600 Pennsylvania Avenue");
```

Walkthrough 2: Geocoding

In this walkthrough you will programmatically geocode an address and re-center the map on that location.

Steps

1. In Dreamweaver, open /walk/**walk6-2.cfm**.
2. Review the code with your instructor. This is the solution from the prior walkthrough with minor modifications.
3. Where indicated by the comment, instantiate a google.maps.ClientGeocoder into a local variable named geocoder. Your code should resemble the following:

```
var geocoder = new google.maps.ClientGeocoder();
```

4. Directly under the code from step 3, use the **getLatLng()** method to determine the geocode of the address that a user will enter into the form. Once it is complete, have control transfer to a function named **plotOrigin**. Use the following syntax:

```
geocoder.getLatLng(address,plotOrigin);
```

5. Above the function **fnPlotStartPoint()**, declare a new function named **plotOrigin** that accepts one parameter named **point**.

6. Inside the plotOrigin function, insert an if() block to check if the point parameter is false. Your code should resemble the following:

```
function plotOrigin(point) {
  if (!point) {

  } else {

  }

}
```

7. If the point argument evaluates to false, insert an **alert()** method to indicate that the address was not found.
8. If the point argument does not evaluate to false, use the **setCenter()** method to refocus the map to the location held by the point argument as follows:

```
map.setCenter(point,13);
```

9. Save the file and browse. Try typing in your home address into the address block and click Go. You should see the map recenter to that address.

-- End of Walkthrough --

Working with Spry Datasets

As previously discussed, you can use the Adobe Spry framework to make background data requests to a ColdFusion component. Using the Adobe Spry API, you can loop through resulting query data one row at a time, interactively passing data to the Google Maps API to plot data points.

Configuring your component methods

As mentioned previously, for ColdFusion to return data interactively to Spry you must perform the following steps:

- Supress debugging output
- Set the access attribute of the function to remote
- Output your result in JSON format

These steps have been performed in the following example:

```
<cfcomponent>
   <!--- supress debug output -→
   <cfsetting showdebugoutput="no">

   <!--- set access to remote -→
   <cffunction name="search"
               access="remote"
               returntype="void">

      <cfset var q = querynew("col1,col2")>
      <!--- convert query to array of struct -→
      <cfset var a = query2array(q)>
      <!--- output as JSON -→
      <cfoutput>
          #serializeJSON(q)#
      </cfoutput>
   </cffunction>
</cfcomponent>
```

Programmatically Accessing Server-based Data

Unlike prior examples in this course that relied exclusively on Spry tags to react to changes in a dataset, our Google maps example requires that we programmatically request data from the server and define functions to explicitly handle the results.

In order to monitor the progress of a Spry data server request, you must define an observer function. Once the server transaction has completed you can use JavaScript to loop through the results.

Using observers to monitor download progress

To monitor the progress of the download of a Spry dataset you must add attach an *observer* method. The observer method will be triggered any time the status of a Spry recordset changes. The following three events are supported:

Event Name	Description
onPreLoad	Triggered immediately before Spry makes a server request
onPostLoad	Triggered after the download from the server is complete, but before data is parsed into the dataset.
onDataChanged	Triggered when the server transaction is complete and new data has been loaded into the Spry dataset.

You can add an observer handler to a Spry dataset by using the addObserver() method.

In the following example, a Spry JSON dataset is instantiated with an observer function. The loadData() method initiates a data request to the specified server URL.

```
<script language="JavaScript">
var ds1 = null;

function getData(x,y) {
 ds1 = new Spry.Data.JSONDataSet("http://localhost:8500/
locator.cfc?method=get&x=" + x + "&y=" + y,{useCache:false
});

  ds1.addObserver(loadedDataResponder);
  ds1.loadData();
}
</script>
```

Observer functions are automatically passed the following three arguments:

Argument	Description
notificationType	A text string containing the type of notification – onPreLoad, onPostLoad, or onDataChanged
dataSet	The notifier that spawned the event – in this context, it refers to your Spry data object
data	Any data passed along with the notification

A typical observer method for a data request would resemble the following:

```
function loadedDataResponder(notificationType, dataSet, data)
{
    switch(notificationType) {
        case "onPreLoad":
               window.status = 'Making request to server...';
               break;
        case "onPostLoad":
               window.status = 'Data returned from server';
               break;
        case 'onDataChanged':
               alert("Your data is now ready for output");
               window.status = '';
               break;

     } /* switch */
}
```

Looping through Spry Data

There are a number of Spry data methods that you can invoke in order to loop through a set of records.

- The getRowCount() method returns the number of rows in the data set
- The getRowByRowNumber() method returns a specific row from a data set.

Assuming that you had a Spry recordset containing an array of objects, you could loop through it as indicated below:

```
var count = ds1.getRowCount();
for (var i=0; i<count; i++) {
       row = ds1.getRowByRowNumber(i);
       alert(row.COLUMN);
}
```

Walkthrough 3: Making background server requests with Spry

In this walkthrough you will transmit coordinates to ColdFusion so that it can return query data that will ultimately be plotted on the map.

Please note that the database only contains information for physicians located in a 5 mile radius around 1400 16th Street NW Washington DC.

Steps

Prepare the Component Method

1. In Dreamweaver, open /walk/locator.cfc and review the code with your instructor. Note that the search method had been previously set up to handle requests from ColdFusion using a web 1.0 framework.
2. In the <cffunction> tag for the search() method, change the access attribute to **REMOTE**
3. Set the returntype of the search() method to VOID
4. Delete the <cfreturn> tag in the search() method
5. Just above the closing </cffunction> tag, use <cfoutput> to output the contents of the query, converted to an array. Your syntax should appear similar to the following:

```
<cfouptut>#serializeJSON(query2array(q))#</cfoutput>
```

6. Save the file and browse it using the following URL: http://localhost:8500/ftajax200/walk/locator.cfc?method=search
7. You should see search results in JSON format. Discuss the syntax with your instructor.

Invoke the search method from the web page

8. Return to Dreamweaver
9. Open /walk/walk6-3.cfm. This is the solution from the prior walkthrough.
10. Under the comment for step 10, invoke a user-defined function that you will create in the next steps named getDocs that passes in a single argument – point. Your code should appear as follows:

```
getDocs(point);
```

11. Underneath the comment for step 11, create a function named **getDocs()** that accepts a single parameter named **point**.

12. Inside the getDocs() method, Invoke the getData() method and pass in the following arguments:

    point.lat()
    point.lng(),

Define a Spry Observer to handle returned data from the CFC method

13. Underneath the comment for step 13, insert attach an observer function named loadedData to the ds1 object. Use the following syntax:

```
ds1.addObserver(loadedDataResponder);
```

14. Directly underneath the code that you inserted from the prior step, cause Spry to make a data request to your ColdFusion method by invoking the loadData() method of ds1. Your code should appear as follows:

```
ds1.loadData();
```

15. Underneath the getData() function, define a new function named loadedDataResponder that accepts the following three arguments:

    - notifcationType
    - dataset
    - data

16. Inside the function, create an IF statement to evaluate whether the variable notificationType is equal to the string "onDataChanged". Your code should appear as follows:

```
if (notificationType == "onDataChanged") {

}
```

17. Inside the IF block, define a local variable named **count** and set it equal to the number of rows in the ds1 recordset. Your code should appear similar to the following:

```
count = ds1.getRowCount();
```

18. After the code that you inserted from the previous step, define a FOR loop that iterates from zero to count. Your code should appear similar to the following:

```
for (var i=0; i<count; i++) {

}
```

19. Inside the FOR loop, define a variable named row and set it equal to the ds1 row location defined by the variable i as indicated below. Also, output the contents of the EMAIL address returned in the recordset into an alert() box.

```
var row = ds1.getRowByRowNumber(i);
alert(row.EMAIL);
```

20. Save the file and browse.
21. Enter an address and click GO. You should see a set of email addresses returned to you in the alert box.

-- End of Walkthrough --

Defining Map Markers

Markers identify positions on your map. You can either use the default Google Maps icon, or you may substitute your own image.

Creating a map marker requires that you instantiate a google.maps.marker object and then bind the marker into an overlay.

Overlays

Overlays are like transparent layers on your map that move as a user drags or zooms. Using overlays you can add the following thee types of objects to a map:

- Markers
- Polylines
- Polygons

In addition, your map will automatically be given an info window overlay. The map itself is displayed using a tile overlay.

You create overlays using the google.maps.Map2.addOverlay() method

You can delete an overlay using either the google.maps.Map2.removeOverlay() method or the google.maps.clearOverlays() method.

Additional overlays that you can add to your maps include the following:

- Traffic
- Driving Directions
- Local Search
- Street View

Markers

Markers identify points on a map and sit on defined overlays as indicated in the following example:

```
var marker = new google.maps.Marker(point);
map.addOverlay(marker);
```

You can create draggable markers.

Google's Marker Manager to avoid clutter when representing hundreds of nodes by allowing you to set which markers will be available at specific zoom levels.

Walkthrough 4: Defining Markers

In this walkthrough you will plot the data points returned from ColdFusion on your map.

Steps

Create a function to return a marker

1. In Dreamweaver, open /walk/walk6-4.cfm and review the code with your instructor. Note that it is the solution from the previous walkthrough.
2. Where indicated by the comment, create a function named **createMarker()** that accepts the following four parameters:

   point
   name
   address
   website

3. Inside the createMarker() function, instantiate a new marker using the following syntax:

```
var marker = new google.maps.Marker(point);
```

4. Return the variable marker to the calling function as described below:

```
return marker;
```

Plot Markers from ColdFusion Data

5. In your loadedDataResponder() function comment out the alert() statement. Where indicated by the comment, define a map position from the returned ColdFusion data using the following syntax:

```
var point = new google.maps.LatLng(row.LAT,row.LONG);
```

6. Directly underneath the code that you inserted from the prior step, invoke the createMarker function, returning the data to a local variable named marker as follows:

```
var marker = createMarker(point,
                          row.FULL_NAME,
                          row.ADDRESS_1,
                          row.WEBSITE);
```

7. Invoke the addOverlay() method to display the marker on the map as follows:

```
map.addOverlay(marker);
```

8. Save the file and browse. Enter an address. You should see additional points plotted around your specified location. Note that you may need to zoom out to see the plotted points.

Responding to User Events

While markers identify positions on your map, you typically would want these markers to be clickable. You can bind custom functions to marker click events using the google.maps.Event.addListener() method.

The following events are supported by the Google maps API and can be applied to either a marker or the map itself:

- mousedown
- mouseup
- click
- dragstart
- drag
- dragend

In addition, events can be programmatically triggered using the google.maps.Event.trigger() method.

Example

In the following example, a click handler is added to a marker object. In this example, clicking on the marker would cause the maps api to display html content in a popup bubble:

```
google.maps.Event.addListener(marker, "click", function()
{
   marker.openInfoWindowHtml(html);
});
```

Walkthrough 5: Responding to Events

In this walkthrough you will make the plotted points on the map clickable. Clicking on a point will reveal the name, address, and contact information for the specified doctor.

Steps

Create an event handler for each marker

1. In Dreamweaver, open /walk/walk6-5.cfm and review the code with your instructor. Note that it is the solution from the previous walkthrough, with one additional function added for your convenience.
2. Where indicated by the comment, initialize a local variable named html so that it is equal to the result of running the createHTML() function. Use the following syntax:

   ```
   var html = createHTML(name,address,website);
   ```

3. After the code that you inserted from the previous step, use the google.maps.Event.addListener() method to define a click event on the marker. Your click event should invoke the openInfoWindowHtml() method of the marker object as indicated below:

   ```
   google.maps.Event.addListener(marker, "click", function() {
                   marker.openInfoWindowHtml(html);
   });
   ```

4. Save the file and test. Try clicking on one of the markers to see detailed information about the doctor.

Unit Summary

- Sign up with Google to get an API license key
- Use the AJAX loader to dynamically load Google API's at runtime
- Use the google.maps.Map2() object to instantiate maps on your pages
- Use Spry to make background data requests to ColdFusion CFC methods.
- Handle Spry callbacks by defining an Observer
- Create map markers using the google.maps.Marker() and assign them to overlays
- You can create custom event handlers to deal with user interactions on a map

Unit Review

1. What do you need to get started using the Google Maps API?

2. Describe the process for placing a map on a web page

3. How do you return JSON data to Spry from ColdFusion

4. Describe the steps for geocoding an address

5. What is the purpose of a Spry Observer?