Exercise 3: ArcGIS Server Overlays  
In this exercise, we will learn how to configure the following:

* Adding dynamic overlays from an ArcGIS Server instance
* Create a geocode function and integrate it
* Identify a layer within the ArcGIS Server service and display the results

## Step One: Getting Started

To prep for this exercise, do the following:

1. Navigate to the training data, and under the “Ex3” folder, and double-click the “ex3final.html” to open the file in a browser. This is what we’ll be working towards in this exercise. Close the browser.
2. Inside the Ex3 folder, copy the “ex3\_step1.html” file and paste it in the same folder. Rename it “ex3\_<yourname>.html”. This will be the file you will be editing through the rest of the exercise.
3. Right-click “ex3\_<yourname>.html” and choose “Edit”. This should open the page in a text editor.

In this step, you will add a reference to the javascript file I’ve included in the exercise data under the “js” folder. You can download this file for yourself or deep-link it here: http://gmaps-utility-gis.googlecode.com/svn/trunk/arcgislink/src/arcgislink.js. Adding this reference exposes a host of functions to your web map, including the ability to overlay an ArcGIS Server service.

### Add the .js reference

In the file’s header, add the following code below the maps.google.com script tag, and above your custom script section:

<script src="js/arcgislink.js" type="text/javascript"></script>

This links your file to the arcgislink.js file in the “js” folder. Your file must be stored alongside the “js” folder for this relative file source reference to work.

### Add the map overlay

Explore this REST endpoint: <http://server.arcgisonline.com/ArcGIS/rest/services/Demographics/USA_Population_Density/MapServer>

This will be the url you will use as the overlay for this mashup. Below the “map = …” variable definition of the initialize function, add the following code:

**var** url = 'http://server.arcgisonline.com/ArcGIS/rest/services/Demographics/USA\_Population\_Density/MapServer';

**var** agsType = **new** gmaps.ags.MapType(url,{name:'arcgis', opacity:0.5});

map.overlayMapTypes.insertAt(0, agsType);

dynamap = **new** gmaps.ags.MapService(url);

This code establishes the REST URL as a new type of map that can be added with the references found in the arcgislink.js file. The dynamap variable is a simpler definition of the service, which we’ll need for the identify function later.

### Add the geocoder variable to the initialize function

You may have noticed a “var geocoder” statement at the top of your custom script. This global variable is required to access the geocoder throughout the page. At the first line of the initialize function, add the following code:

geocoder = **new** google.maps.Geocoder();

Preview your file in a browser. You should see the Google basemap overlaid with some shaded county polygons indicating population density.

## Step Two: Geocoding

### Add the geocode text box HTML

Now that you have a geocoder function, you need to give the user a place to enter in geocode requests. Inside the side div, before the paragraph defining some text elements, add the following code:

<div id="frmBox">

<form action="" onsubmit="codeAddress(**this**.geocodeTextBox.value); **return** false;" method="post" name="frmGeocode" accept-charset="utf-8">

<p>Address, City, or Zip:

<input id="geocodeTextBox" type="text" name="geocode"/>

</p>

<p><input id="btnFind" type="submit" value="Find"/></p>

</form>

</div>

This form will allow a user to type in an address and hit “Enter” on their keyboard. Upon doing so, the entered text will be sent to the codeAddress function.

### Add the geocode function

Now you need a new “codeAddress” function to take the user input and translate it into coordinates. Google’s geocoder function does just that with the following code. Place this function below the “initialize” function:

**function** codeAddress(address) {

**if** (address) {

geocoder.geocode( { 'address': address}, **function**(results, status) {

**if** (status == google.maps.GeocoderStatus.OK) {

map.setCenter(results[0].geometry.location);

map.fitBounds(results[0].geometry.viewport);

**var** myaddHtml = "Address Found: " +

results[0].formatted\_address;

document.getElementById("dResults").innerHTML = myaddHtml;

} **else** {

document.getElementById("dResults").innerHTML = "Err:"+ status;

}

});

} **else** {

alert("Please enter something into the search box.");

}

}

This code ensures that there is an address, that the geocoder returns an OK status, and then fits the map to the appropriate zoom level and location given the address entered. It also adds some text to the map to indicate the results of the search to the user.

### Add the marker

Usually after performing a geocode or other search, users are conditioned into seeing a marker or other result on the map to indicate where the result was found. The following code clears any previous marker we’ve created, sets the map and establishes a position for the new marker.

marker.setMap(null);

marker.setMap(map);

marker.setPosition(results[0].geometry.location);

marker.setDraggable(true);

Add the code between the map.fitBounds() and var myaddHtml lines of the codeAddress function. Preview your file in a browser and type in some addresses, city names, or zip codes to see if it works as you expect – the map extent and zoom changes, and a marker appears.

## Step Three: Identify

This step has the most complicated functions. I myself don’t completely understand these functions, instead I have drawn them from the examples posted by Nianwei at:

<http://google-maps-utility-library-v3.googlecode.com/svn/trunk/arcgislink/docs/examples.html>

### Add the identify function

If you remember from the lesson, the workflow of this web page requires that the geocoding action fires off the identify function. So, back in the codeAddress function, add an “identify” line like this (insert the gray section):

document.getElementById("dResults").innerHTML = myaddHtml;

identify(results[0].geometry.location); //ONLY INSERT THIS LINE!!

} **else** {

This will send the results of the geocode to the identify function. Next, below the codeAddress function, add a new function with the following code:

**function** identify(latLng){

**if** (res) res.length = 0;

dynamap.identify({

geometry : latLng,

tolerance : 3,

layerIds : [ 2 ],

layerOption : 'all',

bounds : map.getBounds(),

width: map.getDiv().offsetWidth,

height: map.getDiv().offsetHeight,

overlayOptions: [],

returnGeometry: false

}, **function** (idresults, err){

**if** (err) {

alert(err.message + err.details.join('\n'));

} **else** {

parseJSON(idresults);

}

});

}

This function takes the lat/long of the geocode and establishes some parameters to use in performing an *identify* on the ArcGIS Server service (“dynamap”). Many of these options can be configured for different services, but for the sake of simplicity, we’re just going to identify one layer and skip getting the geometry of the results.

### Parse the results

Notice the “parseJSON(idresults)” line inside the identify function? This calls a new function to iterate through the results of the identify in order to obtain the pieces we want. Below the identify function, add the following code to accomplish this task:

**function** parseJSON(idresults) {

**var** demoN = "Tract Name: ";

**var** demoT = "Total Population: ";

**var** demoS = "Population Density: ";

res = idresults.results;

layersJSON = { "2": []};

**for** (**var** i = 0; i < res.length; i++) {

**var** thisrec = res[i];

layersJSON[thisrec.layerId].push(thisrec);

}

**var** lyr2 = layersJSON[2];

**switch**(lyr2.length) {

**case** 1:

**var** attributes = lyr2[0].feature.attributes;

document.getElementById("demoName").innerHTML = demoN + attributes["Name"];

document.getElementById("demoTotalPop").innerHTML = demoT + attributes["2010 Total Population"];

document.getElementById("demoPopDens").innerHTML = demoS + attributes["2010 Population Density (Pop per sq mile)"];

document.getElementById("other").innerHTML = "";

**break**;

**case** 0:

**var** othrHTML = "No demographics found in this area. Try zooming, or dragging the marker in the map to a new location.";

document.getElementById("other").innerHTML = othrHTML;

document.getElementById("demoName").innerHTML = "";

document.getElementById("demoTotalPop").innerHTML = "";

document.getElementById("demoPopDens").innerHTML = "";

**break**;

**default**:

**var** othrHTML = "There are "+lyr2.length+" records in the result. Try dragging the marker until you have only one record.";

document.getElementById("other").innerHTML = othrHTML;

document.getElementById("demoName").innerHTML = "";

document.getElementById("demoTotalPop").innerHTML = "";

document.getElementById("demoPopDens").innerHTML = "";

**break**;

}

}

This function establishes some text to use to display our variable name results, iterates through the identify results to get only features from the “Tracts” layer (ID = 2), and then sets up a “switch” statement to make sure we only display attributes from one feature at a time. If the identify returns more than one feature from the layer, the user gets prompted to drag the marker until only one result is found (see the Challenge section). Try “Eagan” to see what happens when more than one feature is found. Try “Fertile” to see a typical single feature result.

All the “document.getElementById” lines inside the switch statements manipulate the HTML on the page to display the results we want to see. One thing we have to be careful with here is to clear the HTML of any text elements at the same time we populate other elements. Preview your file inside a browser to make sure it works as expected.

## Challenge: Dragging the Marker

You may have noticed that in the last step, we prompted users to drag the marker if more than one (or zero) feature was found in the Tracts layer. However, dragging the marker does not actually do anything yet. Examine the “ex3\_final.htm”. What functions are included to enable a new result when the marker is dragged? What additional pieces of the codeAddress function enable this?