Exercise 2: KML Overlays  
In this exercise, we will learn how to configure the following:

* Adding Maps-Compatible KMLs
* Using a Legend/TOC

## Step One: Getting Started

To prep for this exercise, do the following:

1. Navigate to the training data, and under the “Ex2” folder, and double-click the “ex2final.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 Ex2 folder, copy the “ex2\_step1.html” file and paste it in the same folder. Rename it “ex2\_<yourname>.html”. This will be the file you will be editing through the rest of the exercise.
3. Right-click “ex2\_<yourname>.html” and choose “Edit”. This should open the page in a text editor.

Find a few KMLs that you want to work with. Search Google for “Minnesota KML” or “KML” plus a topic of interest. Some possibilities:

* Minnesota State Trails: <http://geocommons.com/overlays/138446.kml>
* Wellhead Protection Areas: <http://gis.mda.state.mn.us/kml/Wellhead_Protection_Areas.kmz>
* 6th Congressional District Voting Precincts: <http://www.sos.state.mn.us/Modules/ShowDocument.aspx?documentid=12065>
* Mn/DOT roadside weather stations: <http://www.crh.noaa.gov/images/mpx/geo/mndot.kml>
* All US Congressional Districts: [http://www.google.com/fusiontables/exporttable?query=select+\*+from+207308+&o=kmllink&g=col3](http://www.google.com/fusiontables/exporttable?query=select+*+from+207308+&o=kmllink&g=col3)

### Add the KML to the overlay array

After choosing two or three KMLs from the above list, add the first one to the array that will hold all our KMLs. (Technically you could define each one within the initialize function, but using this method enables more functionality further down the line.) So, directly below the **var** kmlOverlays = []; line, add the following line, using your KML of choice between the single quotes:

kmlOverlays[0] = **new** google.maps.KmlLayer('http://geocommons.com/overlays/138446.kml',{preserveViewport: true}) //Fill in your KML 0 link in between the single quotes

*Bonus question: What do you think “preserveViewport: true” does?*

### Set the Map for the Overlay

After adding one KML to the array, making that KML appear on the map is easy. Below the “map = …” definition line of the initialize function, add the following:

kmlOverlays[0].setMap(map);

Open the file in a browser to see if you succeeded. It may take a few seconds for the KML to load.

## Step Two: Add More KMLs with the Array

### Add two more KMLs to the first array

Now you have one KML in the map, but what if you want more? Using the array, as mentioned, allows for more functionality. So, add entries to the kmlOverlays array at the top of the script, defining each new addition as kmlOverlays[1], kmlOverlays[2], etc. until you have all the KMLs you want. Any more than 4 or 5 is probably too much at this point.

kmlOverlays[1] = //fill in for KML 1 (the second in the array)kmlOverlays[2] = //fill in for KML 2 (the third in the array)

### Add a for…loop to the initialize function

In order to add all of the KMLs to the map, you have to use a Javascript “for…” loop to iterate through the kmlOverlays array and add each one. The nice thing about this is you can then simply add, delete, or otherwise modify your array to control what appears in the map. To enable this:

**Replace** this line:

kmlOverlays[0].setMap(map);

with the **following three lines**:

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

kmlOverlays[i].setMap(map);

}

Open the file in a browser to see if you succeeded. It will probably take even more time for the KMLs to load, depending on how many you added.

## Step Three: Create a TOC

Another reason for using arrays to set up your KMLs is enabling a legend. When you add multiple KMLs to a map, each time you call the “setMap(map)” piece of code, the KML gets added to the top of the map. So multiple KMLs will appear in reverse order of the code. For example:

FirstKML.setMap(map); // This KML will be on the bottom  
SecondKML.setMap(map);  
ThirdKML.setMap(map); //This KML will be on the top

Warning: DO NOT COPY AND PASTE THIS CODE or try to find it! It is an example!

In order to remove the KML, you call kml.setMap(null); and that works fine if you just want to remove a single KML once. But if you want a user to be able to interact with at table of contents (TOC) (also typically functioning as a legend), then you need to use arrays to track their state (part of the map, or not) and to display them on the map in the correct order. Otherwise, you could end up with a different drawing order than the user expects.

### Add toggle states array

The first step in setting up a TOC is to create a new array that will hold the “toggle states” – Boolean values that will signal “on” or “off” – of each KML. Every entry in this array, from [0] to [n], must align and correspond to the entries in the kmlOverlays array. Given the HTML we’re going to use later, each one must be set to “true” initially. Add the following code below the kmlOverlays array to establish this array:

**var** kmlToggleStates = [];

kmlToggleStates[0] = true;

//add more here until the count matches kmlOverlays

Then, add entries to the array until you have the same number of entries in the kmlOverlays array. Each entry in this new array should set the variable to true.

### Add HTML for TOC

Next, you need some HTML in the file to handle the TOC. You may have noticed that in this sample, the map is shoved over to the right of the page. This leaves some room for the TOC in the “side” div of the HTML. Scroll down to the area of the HTML where you see <div id=”side”> and add the following HTML in between the div tags:

<table id="tblTOC" width="150px">

<tr>

<td><input type="checkbox" id="chk2" checked=

"checked" onclick="toggLegend(2);"></td>

<td><strong id="leglbl2">KML 2</strong></td>

<td><img id="leg2" src="http://gis.mda.state.mn.us/images/iconHelp.gif"

alt="Legend 2"></td>

</tr>

<tr>

<td><input type="checkbox" id="chk1" checked=

"checked" onclick="toggLegend(1);"></td>

<td><strong id="leglbl1">KML 1</strong></td>

<td><img id="leg1" src="http://gis.mda.state.mn.us/images/iconHelp.gif"

alt="Legend 1"></td>

</tr>

<tr>

<td><input type="checkbox" id="chk0" checked=

"checked" onclick="toggLegend(0);"></td>

<td><strong id="leglbl0">KML 0</strong></td>

<td><img id="leg0" src="http://gis.mda.state.mn.us/images/iconHelp.gif"

alt="Legend 0"></td>

</tr>

</table>

In the table, each <tr> tag defines a row for each KML you have, with the bottom row corresponding to the first KML in your array. I’ve highlighted in yellow the only text you’ll see in the table, so that you can associate the rows in the table with your array. The above table assumes you have 3 KMLs in your array, so KML 0 is the first KML in your array, KML1 is the second, and KML2 is the third. Add or delete appropriately if you have more or less KMLs in your map.

Preview the file in the browser. Notice I’ve given you a deep-linked image to serve as a placeholder for a column in the table that can hold legend images. In the challenge, you’ll get an opportunity to investigate how to replace these with a legend image after the KML is loaded. If you take this kind of thing back to the office, this might be a place to put a “throbber”, that will indicate to the user that the KML is loading. You can make your own to download here <http://www.ajaxload.info/> or here: <http://www.preloaders.net/en/popular> ...

### Add click TOC function

Notice in the HTML above that each checkbox has a “toggLegend(x);” associated with the onclick event. Each time we click the checkbox, we want this function to be called and passed the ID of the KML layer we’re interested in toggling. Let’s add the function now to the script, as a new function below the initialize function:

**function** toggLegend(index) {

**if** (kmlToggleStates[index]) {

kmlToggleStates[index]=false;

} **else** {

kmlToggleStates[index]=true;

}

loadKML();

}

This function gets the index number passed to it and iterates through the kmlToggleStates array – if the toggle state is currently true, it sets it to false, and vice versa. Then it calls the loadKML function, which we’ll define below.

### Add load KML function

Now that you’ve adjusted the toggle states of each KML, you can remove all of the KMLs from the map and add them back in the expected order with the function below. Add it as a new function below the toggLegend function:

**function** loadKML() {

**var** i;

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

kmlOverlays[i].setMap(null);

}

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

**if** (kmlToggleStates[i]) {

kmlOverlays[i].setMap(map);

}

}

}

Here’s where those two arrays really come in handy. There are probably multiple ways to accomplish this task, but this method ensures that the KMLs appear in the order of the legend. Otherwise, the user might be able to turn them all off themselves, then add them back to the map in the order in which they click, which is probably not what you want.

Now preview the file in a browser, and see if the TOC works as intended. You’ll probably want to replace those  images with static legend images. For many point-based KMLs, you can obtain the legend image by pulling the KML into Google Earth as a network link, then clicking on the symbol to see the deep linked image file URL. If you have time, try the challenge below to see how to use both throbbers and a static legend image.

## Challenge: Dynamic TOC

Launch the ex2final.html file in a browser and watch what happens in the TOC. What is different in the code – both the Javascript and the HTML – to make this happen?