Maps from a Database: Reading XML data into Leaflet

We are starting off with a "base" HTML file – no map or javascript yet, just something to get you started. The process can be broken into a few steps:

1. Examine the XML file.
2. Write the script to import the XML data, and verify that it's brought in correctly.[1]
3. Write the script to put the data into Leaflet.

1. Examine the XML file

a. Where is the XML file?

We are going to be using the following XML file in class: http://erica.altschul.info/542Tutorial_01.xml. Download and save a copy locally, because web server security requires that AJAX requests be made on the same server.

b. What fields are in the XML file?

Visit the XML file and note what elements we can see. Since an XML element can be named pretty much anything, we won't know what to tell the script to look for without examining the XML first.

```
<?xml version="1.0"?>
<destinations>
	<point name="Tot Lot at Bryan Park">
		<lat>39.15611</lat>
		<long>-86.52664</long>
		<type>outdoors</type>
	</point>
	<point name="Bloomington Bagel Company">
		<lat>39.16716</lat>
		<long>-86.52833</long>
		<type>food</type>
	</point>
```

The root element is *destinations*. The *destination* has child elements *point*. Each *point* has subchild elements *lat*, *long*, and *type*. It also has an attribute *name*. Looking at the contents, we can figure out what each element/attribute is for:

- "name" is intuitively obvious
- "lat" and "long" are the geographic coordinates for the point location
- "type" describes a category for the point

There's no particular requirement about when something should be an element vs. an attribute, and each XML file is likely to be different. It may have different elements, more or less information, and so on. Exploring the XML tells you how to specify the information you want, and also can help you decide what information to ignore. (We won't be using the "type" element, for example, although it is nice to know it's there if we wanted to expand our Leaflet to be able to assign different icons to different types of destinations.)

[1] I found the following tutorial to be a great starting point for this: http://tech.pro/tutorial/877/xml-parsing-with-jquery, even though the syntax and end goal of the script is different.

2. Write the script to import the XML data, verify that it reads correctly

a. Indicate jQuery library location

To simplify the commands to read our XML file, we are going to use jQuery[2] in our script. This is a library of JavaScript functions which are shorter and simpler JavaScript commands. This line:

```
<script src="http://ajax.googleapis.com/ajax/libs/jquery/1.9.1/jquery.min.js"></script>
```

tells your script to use the jQuery library which is hosted by Google. You can download and host it on your own web server, and just change the `src` above to your own filename – that's mostly for future reference, however.

b. Setup

Within a `<script></script>` tag,

```
$(document).ready(function()
{
        $.ajax({
                type: "GET",
                url: "542Tutorial_01.xml",
                dataType: "xml",
                success: parseXml
        });
});
```

What this essentially means is that we are getting (**type**) XML data (**dataType**) from a given location (**url**), and when the script reads that file (**success**), we want to execute a particular function (**parseXml**, which we're about to define). This will happen as soon as the HTML file is opened in a web browser: **$(document).ready**

c. Define the `parseXml` function

```
function parseXml(xml)
{
        $(xml).find("point").each(function()
        {
                . . . . .
        });
}
```

This is a loop that executes once for each instance of "point" it comes across in the XML file. At this point we're just making sure we can correctly read the contents, so for each "point" element, we want to attempt to import the attributes and subchild elements.

1. Reading in an attribute

An attribute of a point can be accessed by using:

```
$(this).attr("name")
```

The (`this`) means "whichever point element the loop is on." The specific attribute (`attr`) is "name".

[2] For more information about jQuery, visit http://www.jquery.com

2. Reading in a subchild element

This is slightly more complicated; first, we need to "find" a particular subchild element, and then we need to bring it in as text.

```
$(this).find("lat").text()
```

Again, we're using (`this`), looking for the subchild element "lat".

d. Do something with what the data

For now, we will just put it into a list within our HTML file so we can verify that it is coming in correctly. Create an empty `<div>` that will be the target (put it at the beginning of the `<body>` section)

```
<div id="output"></div>
```

Each time we read a new attribute or element from a point, we can push it to the "output" `<div>`. We can also include additional text or HTML tags to be a little more informative.

This code will put content into the `<div>` named "output":

```
$("#output").append(. . . . .);
```

So, if we want to put an attribute of each point and a line break into a list, we add following code to the `parseXml` function.

```
$("#output").append("Name: " + $(this).attr("name") + "<br />");
```

And, if we want to put a subchild element of each point and a line break into the list, we would use

```
$("#output").append("Latitude: " + $(this).find("lat").text() + "<br />");
```

e. Bring it all together

So, if we are trying to print out a list of all the elements and attributes in an XML file, we include multiple append statements to cover all the information. If our list looks complete, then we have successfully confirmed that the data imports correctly.[3]

3. Bring it into Leaflet

We can now pass the information to Leaflet[4] and plot it on a map.

a. Leaflet prep

Add this line:

```
<script src="http://cdn.leafletjs.com/leaflet-0.4/leaflet.js"></script>
```

[3] It is good practice to check your work occasionally by opening your HTML file. Chrome has security settings which will make it *look* like something is broken – use Firefox or Internet Explorer instead!

[4] http://leafletjs.com/reference.html

Also, set up the map parameters and location. Add these lines:

```
	<div id="MyMap" style="width: 800px; height: 400px"></div>

	<script>
	//pre-set attributes: initial zoom level and center point
		var zoom = 12;
		var lat = 39.1671;
		var long = -86.5343;[5]

	//Map using Cloudmade tile server
		var MyMap = L.map('MyMap').setView([lat, long], zoom);
		L.tileLayer('http://{s}.tile.cloudmade.com/c445219bea9b43fcac9d57b7a37f46bd/997
	/256/{z}/{x}/{y}.png', {
			maxZoom: 18,
			attribution: 'Map data &copy; <a
href="http://openstreetmap.org">OpenStreetMap</a> contributors, <a
href="http://creativecommons.org/licenses/by-sa/2.0/">CC-BY-SA</a>, Imagery © <a
href="http://cloudmade.com">CloudMade</a>'
		}).addTo(MyMap);
	</script>
```

b. Leaflet Data

In the previous Leaflet lab, we defined each Marker individually. However, for now, we are going to import our data in a loop – the same loop we already made for translating the XML attributes and elements into lines of text.

1. Assign Points

```
mapPoint =
	L.marker([$(this).find("lat").text(),$(this).find("long").text()]).addTo(MyMap);
```

2. Add the Popup Information

```
mapPoint.bindPopup($(this).attr("name"));
```

4. Enjoy your amazing map!

5. Homework

For homework, make your own Leaflet map that reads in data from an XML file. You can look around for an XML file, make one from your own points of interest, or use one of the samples that I made:

- http://erica.altschul.info/542Tutorial_02.xml
- http://erica.altschul.info/542Tutorial_03.xml
- http://erica.altschul.info/542Tutorial_04.xml

Put both a list of points AND a Leaflet map in your final HTML and turn in on Blackboard.

---

[5] Don't forget to change the zoom and/or center point depending on your data location…