This will explain how the index.html file “works” to build the map. I recommend having that file open in your text editor of choice (I prefer Sublime Text 2), and following along, while reading the comments in the code.

Hopefully you have at least a passing familiarity with HTML.

At the top of the file, you have the links and scripts for the external API’s, libraries, etc that we are using for the map. Then we have the css styles.

Let’s start with the HTML construction. We’ve got a map div, which is our map. Then, a lengthy section of the “filter\_options” div, which is the user interface for the filtering controls. Within here we have a div for each of the 3 main types (called “type\_filter”, with an id for each type #fire\_filter, etc). This contains the checkbox (“fire\_button”), the dropdown menu (class “chosen-select” and id “property\_usage\_button”), and the date range slider (“slider” “fire\_slider”). Then a short horizontal line break (<hr>), and the div for the next type.

Then, there is a short section for the tooltip info pop\_up called “d3\_tip”. After this is all the Javascript code.

**<script>**

**Map**

First we create the map, using a built in Mapbox function. If you want to use a different map, or your own map, go to Mapbox.com and create an account. Choose the map you want to use, and replace the unique map id with your own, and replace the access token with your own.

You’ll also need to set the .setView coordinates to be the center of whatever city you’re using.

**showText()**

Update this function if you want to change what gets displayed when you hover over a point. If you decide you want to display the incident number for fires, or the number of inspections for current inspections, for instance, you’ll need to do 3 things:

1. Make sure the column exists in the SQL table.
2. Query the table for that column in the SQL\_query.php file
3. Write a new function, like the propertyName(d) functions, like the following:

function incidentNumber(d) {

if (d.properties.in\_fsaf\_yn == “fire”) {

return '</b><br />' + “Incident Number: “ + d.properties.incident\_n;

} else {

return “”; // This means return nothing.

}

4. Then be sure to call incidentNumber(d) as part of the showText(d) function.

showText() then updates the counts of the properties, based on the filters.

In the drawDots(data) function, the actual dots themselves get drawn. Here is where you could change their color, opacity, or shape if you wanted, as well as change what happens when someone mouseOvers them or clicks on the dots. There is a mouseClick() function that can be edited to do things if the user clicks on the dots.

The drawOverlay() function is where the NPU, Battalion, and District overlays get drawn, and this is where you would change the mouseOver and .on(“click”) lines to add functionality to hover and click on the overlays. Perhaps even a zoom function that occurs when the user clicks on the NPU.

Finally, there is a series of functions to listen for events on the filter buttons (clicks or changes), and then run the changing() function to hide or show dots based on the filters selected. A more efficient way of doing this would be to only draw the dots if they meet certain criteria, rather than drawing all of them, but only showing or hiding certain ones. This is also where the getPropertyInfo() function is, which uses an Ajax call to run the SQL\_query.php file to query the SQL database.

This is also where you could change the functions to add new filter criteria if you like.

That’s the end! Thanks for reading! Explore the comments in the code, and change things, break things to find out how they work, and Google everything. There are tons of tutorials out there that explain how things like this work, and they helped me as I was writing it too. This can definitely be modified and improved in many, many ways, and I encourage you to find them, and fork the Github repo with your modifications and improvements.