Documentation for Publish What You Pay Canada's Visualization

February 2017

Designed and Developed by OpenNorth

Table of Contents

1. Introduction

This documentation provides guidance for Publish What You Pay Canada for the following: adding data in the Google Sheet, modifying or debugging the visualizations, and adding filters to the visualization. After referring to this documentation, if there are any unresolved concerns/issues regarding the visualization, contact Julia Conzon at juliaconzon@gmail.com.

2. Data

Add Data in Google Sheet

It is important to standardize the data to reduce errors that occur in the visualizations. Not all columns in the Google Sheet are used in the visualization. The following section highlights what data should be included in the used columns and how they should be formatted.

Column A – Id: Provides a unique id number to each row.

Column B – ProjectorGovernment: Indicates whether the payment is for a project or government. P indicates a project, G indicates a government at various geographic levels (e.g., municipal, provincial, federal), and G* indicates an indigenous group.

Column C – Name: Provides the name of the project or government that received a payment.

Column D – AbbreviatedName: Since some names in Column C are too long to be effectively visualized in the payment flow visualization, an abbreviated name can be added to improve visibility. The code will take the abbreviated name and add it to the 'Payment To' section of the visualization, but will keep the full name in the tooltip (i.e., the text that appears when a user hovers over a single path in the visualization).

Column E – CountryLevel: Indicates whether Column C (Name) will be geocoded as a point (latitude/longitude) or as a polygon (geoJSON). Y indicates that it is at the country level, so will be a geoJSON, and N indicates that it has a point location.

Column F – GeoJSON: For country level polygons. If a country is at the country level, then a link to MapIt's API is provided.

Column G – Latitude: If the row's Column E (CountryLevel) is assigned N, then provide the latitude of Column C (Name).

Column H – Longitude: If the row's Column E (CountryLevel) is assigned N, then assign the longitude of Column C (Name).

Column I – Country: Country that the row is geographically in.

Column J – PaymentType: Indicates the type of payment. Ensure that the payment type matches already inputted types. For example, 'Taxes' exists already, so do not input a data as 'Tax' because the code will then add 'Tax' as a new payment type when it is not.

Column K – UnitMeasure: NOT USED

Column L – Volume: NOT USED

Column M – ValuationMethod: NOT USED

Column N – Currency: NOT USED

Column O – Amount: NOT USED (amount in original currency in the original data sources)

Column P – ConversionRate: Rate that was used to convert Column O (Amount) to CAD.

Column Q – CADAmount: The CAD amount of each payment (Column O * Column P). Values should not have any characters other than numbers and decimal points. For example, do not input $1,304,232.231 or 1,304,232.231; instead, input 1304232.231.

Column R – Credit: Indicates whether or not the payment is credit or not. Y indicates it is credit, N indicates it is not credit.

Column S – CompanyName: Name of the extractive company. Ensure that the same company has the same name or else the code will record a company as two different entities. For example, record as 'Royal Dutch Shell Company plc', not 'Shell'.

Column T – CompanySource: Link to company's website.

Column U – CompanyNumber: NOT USED

Column W – Notes: NOT USED

Column X – ReportEndDate: The date reported (y-m-d).

Column Y – Version: NOT USED

Column Z – Source links that allow the user to access the site of the project or government (present in the Map pop ups).

Column AA – ReportNotes: NOT USED

Column AB – FilingDate: NOT USED

Column AC – DataFrom: Indicates the spreadsheet that the data original came from.

How data is pulled from Google Sheet to index.html

Data is pulled from the Google Sheet to the web page using TableTop.

3. Payment Flow Visualization

Description

The payment flow visualization displays the following columns of data: Column C (Name) to represent the Payment To; Column Q (CADAmount) to appear in the hover popup and determine the widths of the paths between Payment From and Payment To; Column S (CompanyName) to represent the Payment From; Column R (Credit) to appear in the hover popup; Column J

(PaymentType) to appear in the hover popup; and Column X (ReportEndDate) to appear in the hover popup. This visualization was developed off a pre-existing design created by Visual Cinnamon. However, Julia Conzon has modified the code to allow for a dynamic database rather than a static database.

Debug Errors

If the visualization is not outputting in the usual format consider the following errors in the Google Sheet's data:

- Column Q (CADAmount) has a negative value ('-') to represent a credit payment. Remove this character and add 'Y' to Column R (Credit)

4. Table

Description

The table is outputted when there is too much data for the payment flow visualization to effectively visualize the data. The table outputs all data that would be present if the data was visualized in a payment flow visualization.

5. Map

Description

The map displays the following columns of data: Column B (ProjectorGovernment) to determine what icon to display the point; Column E (CountryLevel) to indicate whether the row can be assigned to a latitude/longitude or a GeoJSON polygon; Column F (GeoJSON) is a link to MapIt API that provides a GeoJSON of the country; Column G (Latitude) and Column H (Longitude) so the points can be mapped; Column C (Name), Column J (PaymentType), Column Q (CADAmount), Column R (Credit), and Column S (CompanyName) for the popup info. The map is coded in a way that allows it to automatically take all existing and new data from the Google Sheet and output into points or polygons on the map.

Debug Errors

If the map is not outputting in the usual format consider the following errors in the Google Sheet's data:

- Empty cells for latitude and longitude if CountryLevel is 'Y'
- Latitude and longitude values if CountryLevel is 'N'
- Ensure there are number values for latitude and longitude
- The latitude and longitude values were swapped

Design

Icons:

Icon design was created in Google Drawings and saved as .svg files. Once a new design is created and saved as a .svg file, to add new icons for the points, simply move these new files into the

images folder and name and replace the existing files. Thus, ensure the names are ‘gov_icon.svg’, ‘proj_icon.svg’, and ‘indigenous_icon.svg’

Map Skin:
There is an array of skins to use from Mapbox for the Map. Find the desired skin from Mapbox’s library and then replace the existing style in index.html’s var map with the new style.

Ensuring Scalability:
The following section highlights aspects of the map that should be considered once the geographic scope expands from Canada.

Maximum and Minimum Zoom:
The max zoom and minimum zoom should be changed. In index.html search ‘minZoom’ or ‘maxZoom’ to direct you to these map options. Change the number value to the desired tile layer.

Landing Page Center View:
If you prefer the map not center on Canada when the page initially loads, in index.html search ‘center:’ and change to the desired coordinates (latitude, longitude).

Country Level:
At the beginning of the project the main focus was on Canada, but as additional countries are added a new method will need to be consider for incorporating country-level polygons. The initial method was to use MapIt’s API to access a country’s geoJSON (i.e., polygon of the country). This is done through the following combination:

http://global.mapit.mysociety.org/area/ + [OSM area ID]

For example: http://global.mapit.mysociety.org/area/792626 is the link that pulls the geoJSON of Canada.

Since there is only one pull that occurs currently, there are not restrictions; however, the more country links are added to the Google Sheet, the more pulls will occur. This risks restrictions, which could end up deactivating access to the API (through IP address). MapIt requires a donation, view more information here: http://global.mapit.mysociety.org/licensing.

6. Filters

Add Filter
All steps below will require you to modify index.html.

1. Create the filter drop-down in the header

Referring to the image below, copy the same format as either filter-menu for the additional filter, but ensure different id names.

```
<!-- Header filters -->
<ul class="header-top-filters">
    <li class="filter-menu"><span><a class="companyFilter">Extractive Company: <select id="companySelection" class="filter"></select></a></span></li> <!-- Company filter -->
    <li class="filter-menu"><span><a class="paymentTypeFilter">Payment Type: <select id="paymentTypeSelection" class="filter"></select></a></span></li> <!-- Payment type filter
    -->
    <li class="filter-menu"><span><a class="projectFilter">Payment Type: <select id="projectSelection" class="filter"></select></a></span></li> <!-- Payment type filter -->
    <!-- Add additional filter here. Copy paste code line above, rename id (not class) and change filter text info-->
    <li class="filter-button" id="buttonSubmit"><div id="filterSubmit" class="submit" data-toggle="tooltip" title="Click here to filter your selection"> Apply </a></li> <!--
    Submit filter -->
    <li class="filter-button" id="buttonReset"><div class="submit" id="filterReset" data-toggle="tooltip" title="Click here to reset the filters to all data"> Reset </div></li>
    <!-- Reset filter -->
</ul>
```

Now with the filter existing in the HTML, you will have to add the values that will be stored in the drop-down menu of the filter. In the case for the company and payment type filters, data was pulled from the Google Sheet and then all duplicates were removed. For example, all the company names from Column S were pulled, but then a function (seen below) only stores each instance of a company name. This way the drop down menu will only have, for instance, “Centrica plc” once versus x amount of times it is present in Column S. This is why it is important that the text in the Google Sheet for Column J and Column S should remain consistent. For example, if “Centrica plc” was also entered in the Google Sheet as “Centrica” the filter drop-down menu would record both. The code below will have to be replicated, but with different variable names.

```
// This function finds all companies names and removes any duplicates, so that it returns an array of each unique
extractive company
function findCompanies(data) {
    var companies = [];
    var company = [];
    for (var i = 0; i < data.length; i++){
        companies.push((data[i].CompanyName));
    }

    $.each(companies, function(i, el){ // Removes any duplicates
        if($.inArray(el, company) === -1) company.push(el);
    });

    return company
}
```

If the data that is present in the new filter does not require new data being pulled from the Google Sheet, then instead of creating a similar function, you can instead simply create a single array that holds the necessary values for the filter drop-down menu (e.g., var receiver = [“Project”, “Government”, “Indigenous group”]).

Last step, you will have to add the values to the HTML drop-down menu. Copy the code below in the script, but change the variables accordingly (make sure that certain variables match the ones you created already).

```
var allCompanies = findCompanies(data); // Variable that stores the array of unique company names
var companySelection = document.getElementById("companySelection"); // Create a variable that will represent the id location of where the filter will be
stored in the HTML
companySelection.innerHTML = '<option value="all">All</option>'; // Add the "All" option
for (var i = 0; i < allCompanies.length; i++){ // This for loop will run through the array and will output each individual company as an option
    companySelection.innerHTML += '<option value="' + allCompanies[i] + '">' + allCompanies[i] + '</option>';
    if (i == allCompanies.length){
    companySelection.innerHTML += '<option>' + allCompanies[i] + '</option></select>';
    }
}
```

2. Create event and function to filter the data

In index.html, search $("#buttonSubmit").click(function() { and follow the instructions in the comments to add another filter. The importance is copy code, but rename variables and make sure they match where they are supposed to match.