Adding Polygons into Tableau Custom Geocoding

(This is not officially supported by Tableau!)

If Tableau’s built-in geocoding data doesn’t contain all of the data that you need for your analysis, there is a mechanism called ‘Custom Geocoding’ built into Tableau to allow you to add in new geographic roles into the hierarchy. Unfortunately, the supported method only allows for the addition of point geometry, not polygons.

For this example, we’ll use a Municipality Shapefile from the U.S. state of New Jersey. The goal will be to add a Municipality geographic role into a custom geocoding database parented to the built-in ‘county’ geographic role.

This document describes how you can use the Custom Geocoding feature AND a custom Python script to load polygons into Tableau’s custom geocoding database.

**High level steps:**

* Create centroids of the boundary Shapefile
* Create polygon WKT .csv
* Convert centroids file to .csv formatted for ‘custom geocoding’
* Create custom geocoding DB in Tableau
* Run Python script to add WKT to custom geocoding DB
* Update .tds file

**Tools Used:**

Polygon dataset (Shapefile or spatial file)

QGIS (or another Desktop GIS)

Python 3.6 +

Tableau Hyper API (Python library)

**Step 1: Prepare the spatial data**

In this example, we’re actually starting with an ArcGIS Endpoint that’s hosting the geographic data. I want to work with a shapefile, so I want to suck the data out of ArcGIS Server and create a shapefile out if it. To do this, I use QGIS to connect to the ArcGIS Server Endpoint and then export/‘Save As’ to create a shapefile.

AGS Endpoint: <https://maps.nj.gov/arcgis/rest/services/Framework/Government_Boundaries/MapServer/2>

In QGIS, use the ‘ArcGIS Feature Service’ connection type, and add a connection to

https://maps.nj.gov/arcgis/rest/services/Framework/Government\_Boundaries/MapServer

Graphical user interface, text, application

Description automatically generated

Expand the tree on the left and add ‘Municipalities’ layer to the QGIS map.

After the Municipality layer is added, right click on the layer -> Export -> Save Feature As…

Choose Esri Shapefile, and set CRS to EPSG: 4326 – WGS 84.

Click OK to export.

Graphical user interface, application

Description automatically generated

The default setting will add the new shapefile as a layer to the map.

From the ‘Vector’ menu, choose Geometry tools and select the ‘centroids’ tool.

Leave the default settings and run it. (Make sure the input layer is your municipalities shapefile layer). QGIS will create a temporary scratch layer with your centoids plus all of the attributes of the original shapefile.

At this point, I actually got an error in QGIS because of an invalid polygon in the shapefile – there is a polygon that has a self-intersecting ring (its boundaries come into contact with itself).

I ran the centroids command again, but used the wrench icon next to the input layer of the centroids dialog to instruct QGIS to skip over invalid geometries. (The setting is ‘Skip (Ignore) Features with Invalid Geometries).

That generates a new layer called Centroids. It’s best to save this to a file. I chose to save as a geojson file. Right click the the Centroids layer -> Export -> Save Features as -> Geojson.

I then found the 1 feature that didn’t have a centroid – the one with the invalid geometry – and I added the centroid manually to the .geojson layer. (To edit a layer, select it in the layers list, then click the pencil icon to begin the edit session. Then click the ‘Add Point feature’ – the 3 dots with the sunburst next to it).

Graphical user interface

Description automatically generated with medium confidence

Click on the map to add the point feature and then fill in the associated attributes. Save your edits.

Next, we need to create a CSV containing a Latitude and Longitude column containing the centroid values. This is what Tableau wants as input.

First, let’s add a Latitude and Longitude column to this .geojson layer. Still in edit mode, right click the layer and choose Open Attribute Table. At the top of the table dialog window, there is a new field icon.

Graphical user interface, application

Description automatically generated

Use the button to add a Latitude column of type Decimal number. Do the same for Longitude.

Now that we have the columns, we need to fill them with the actual values.

Still in the attribute table dialog, choose the column from the dropdown that you want to edit. In the textbox to the right, enter the expression. For Latitude use $y. Then click ‘Update All’.

Graphical user interface

Description automatically generated

Do the same for Longitude, but with the value of $x.

Now we have columns populated with lat and long values. Let’s export this .geojson as a .csv.

Right click on the .geojson layer -> Export -> Save Features as… choose CSV and give the file a name and export the file.

Open the file in Excel or your editor of choice.

Tableau requires the CSV to have a particular structure for this to work properly. Looking at the documentation (<https://help.tableau.com/current/pro/desktop/en-us/custom_geocoding.htm>), I can see that if we want to add Municipality as the role, and we want to parent it to the built-in County role, we need to name and order the .csv columns in a particular way.

This way.

Graphical user interface, text, application, table, Excel

Description automatically generated

The name of the column D (in this case) will become the new geographic role name that you’ll see in Tableau’s geo role menu.

The name of my centroid .csv is municipalies.csv.

\*\*Important! Make sure you save this .csv in a folder ALL BY ITSELF. If you don’t you may get funky results. I did.

**Step 1A – Create Polygon WKT CSV.**

We also need to use QGIS to create a .csv containing WKT (Well Known Text) of the municipality polygons. This is just a string representation of all of the vertices that make up each boundary. We’ll use this later on.

We also need to include the latitude and longitude of the centroid layer, because later on those values will be used to link the polygons to the centroids to correctly update the custom geocoding DB. This assumes that the centroids are uniquely located (no overlapping points).

First, let’s add a ‘Latitude’ and ‘Longitude’ column to the municipality shapefile. That way, when we export it to CSV, it will contain the Lat/Lng values.

In QGIS, seelct the municipalities shapefile layer in the layers list and click the pencil icon to begin an edit session. Right click on the layer and choose ‘Open Attribute Table’. When that dialog opens, click the add new field button. Call it ‘Latitude’ and make it be of type Decimal (precision 7, length 10). Do the same for ‘Longitude’.

Now that the new fields are created, we need to populate them with the x and y values.

To do this, remain in the View Attribute Table window. Select Latitude from the dropdown list of columns. In the text area to the right, use this function to get the Y/Latitude value:

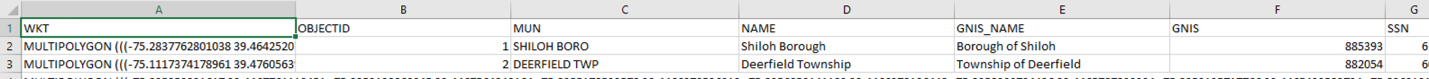
Y(center($geometry))

Then click ‘Update All’. This will give us the Y centroid value for each polygon.

Repeat this step for Longitude/X values.

Now it’s time to export this shapefile as a CSV. To create the WKT csv, right click on the municipalities boundary shapefile layer, then -> Export -> Save Features As -> CSV. Choose a name (you’ll need to reference this name later in the Python script. I called mine municipalities\_wkt.csv).

In the ‘Save Vector Layer as…’ menu, down towards the bottom under Layer Options, choose ‘AS\_WKT’ from the Geometry dropdown. This tells QGIS that we want to convert our polygon data into WKT in the output CSV.



Importantly, the WKT is stored in a column called WKT. We’ll need to reference this column name in the Python script later as well. You should also have Latitude and Longitude columns filled with values.

**Step 2: Import into Tableau**

Now that we have our centroids (points) and polygon WKT (polygon) .csv formatted, we can open Tableau and import it.

Open Tableau.

Click the little white Tableau icon in the top left to switch to the worksheet view.

Click Map -> Geocoding -> Import Custom Geocoding.

Select the folder you created that has the municipality.csv in it all by itself. Click Import. Wait a few minutes while the built-in database gets copied to a new DB that will include your new geo role.

Graphical user interface, text, application

Description automatically generated

Cool.

Note: whenever you open Tableau, it will use this copy of the geocoding DB. (If you want to remove it, or switch back to the built-in copy, you need to choose ‘Remove Custom Geocoding’ from the menu as seen above. Also note – when you publish or share a .twbx, it will pack up this geocoding DB with your .twbx, which can be quite large.

At this point, if you were to load the .geojson file or the .csv containing the municipality names, you could geocode them, but you’d only get points.

Just to try it out, load one of the datasets that contains the municipality names so we can geocode them.

I’m dragging one of my .csv files that has the muni name and the county name.

Click on Sheet 1.

Tableau detects that my .csv has a Country/State/County hierarchy already, and creates it for me. I want to add Municipality to the hierarchy.

First, right click on the Municipality dimension pill, and choose ‘Geographic Role’ -> Municipality. This tells Tableau that the column in your dataset should be geocoded using the new custom geocoding role we added.

Then drag the Municipality pill up to County in the hierarchy.

If you double click on Municipality now, the map should draw with all points representing municipalities in NJ.

Map

Description automatically generated with low confidence

But we want polygons.

**Step 3: Run Python script to add Polygons**

Right. So.

When Tableau creates the custom geocoding DB like in Step 2, it creates a new folder on your machine. On windows, it’s C:\Users\<my user name>\Documents\My Tableau Repository\Geocoding Data.

Graphical user interface, application, Word

Description automatically generated

In this screenshot, you’ll see the files generated by Tableau when you import a custom geocoding role.

We’re going to use Python to update that GeocodingData.hyper file to include our polygons. This isn’t supported by Tableau, but we’re just adding what the other built-in geographic roles already have – namely, they have 2 additional columns that our municipalities table doesn’t have.

We’ll also modify the municipalities.tds.

**Step 3A: Install and run Python script**

Download the repository from <https://github.com/apollolm/Tableau_Custom_Geocoding_Polygons>

(I assume you have Python 3.6+ installed and know how to install requirements from requirements.txt)

Copy GeocodingData.hyper from C:\Users\<my user name>\Documents\My Tableau Repository\Geocoding Data to the folder where you’ve downloaded the repository.

Also copy the municipalities\_wkt.csv (the one with the WKT in it) to the same folder.

In the command line, run the python script thusly:

Python run -i GeocodingData.hyper -o GeocodingData\_out.hyper -w municipalities\_wkt.csv

The script relies on the input municipalities\_wkt.csv to have the same Latitude/Longitude values as the centroids.csv that was used to import into custom geocoding. It uses the Lat/Long values to make sure the correct polygons are added to the correct row inside of the custom geocoding DB.

1000 things could go wrong when you run the Python script. Wrong input filename, wrong version of Python, wrong GeocodingData.hyper file, etc.

Hopefully though, it works and the script should create a new GeocodingData\_out.hyper file.

Copy this file back into the C:\Users\<my user name>\Documents\My Tableau Repository\Geocoding Data folder, and overwrite the old GeocodingData.hyper file (save a backup first). **Make sure to name it GeocodingData.hyper**. That is the file that Tableau is expecting to read.

**Step 4: Modify the .tds file**

The last step is to open the municipalities.tds file in C:\Users\<my user name>\Documents\My Tableau Repository\Geocoding Data. Create a backup first in case you need to start over again.

A .tds file tells Tableau about the contents of the GeocodingData.hyper file. Since we added a couple of new columns using the Python script, we need to update the .tds to reflect those changes.

Open the municipalities.tds file and make the following changes.

In between lines 82 and 83, insert this text:

<map key='[Geometry (generated)]' value='[LocalData].[Geometry]' />

In between lines 89 and 90, insert this text:

<map key='[MapCode]' value='[LocalData].[MapCode]' />

In between lines 117 and 118, insert this text:

<column datatype='integer' hidden='true' name='[MapCode]' role='dimension' type='quantitative' />

Save the .tds.

Open Tableau.

Step 5: Test

Now that we have the modified .tds and updated GeocodingData.hyper file, Tableau should use it as we try to geocoding new input data. I’m using the Deer Carcas dataset. Drag it into Tableau, and switch to Sheet1.

You should see the dimensions County, Municipality, State, Upper County, Upper Muni, Year.

We need to tell Tableau that the Municipality dimension should be tied to the new geographic role. Right click on the pill -> Geographic Role -> Municipality. The icon next to the pill should change to a globe with a little list icon next to it.

Lastly, we want to make sure there is a hierarchy between municipality and County and State.

Drag the County pill onto/underneath State, and drag Municipality pill onto/underneath County. (If, after dragging the pills to create the hierarchy, they disappear, check that you copied all of the text to the correct places in the .tds according to the previous step.)

Drag Number of Deer Carcases onto the Color shelf.

Graphical user interface, application, map

Description automatically generated