Introduction

This document describes how build a Step in a Pipedream workflow to take a set of GPS latitude and longitude coordinates and return the nearest address using an API from the HERE.com location platform. The work in this document is based on this workflow by Raymond Camden but tweaks 1 line to successfully get the API key.

The workflow step described in this document can be done with Here.com's freemium account.

This Pipedream workflow step described here was developed to output additional context (i.e. the nearest address to the received GPS coordinates) to a spreadsheet in the following end-to-end data flow:

Browan Object Locator (BOL) > Helium Hotspot > Helium Console > Pipedream > Google Sheet

At the end of this document is a Bonus section that shows 2 different Pipedream steps to calculate the distance between 2 sets of GPS coordinates.

This document does not explain how to set up the end-to-end data flow described above. If you're interested in setting up that data flow, see Reference 3.1 in the Reference Table. That document uses a TBHH100 temperature and humidity sensor instead of a Browan Object Locator sensor, but the steps are VERY similar. The key differences are that you'll be able to use the Browan Object Decoder function that is already present in the Helium Console instead of having to build your own decoder function & the data elements you're working with will be location data instead of temperature and humidity data.

Table of Contents

Contents

References

The table below lists the learning resources used to support this effort.

Reference Table			
ID	**Topic**	**Reference**	**Description**
1	Pipedream	1. https://pipedream.com/workflows 2. https://pipedream.com/@dangermikeb/browan-object-locator-to-google-sheet-p_vQC3V7	1. Pipedream Web Page where your workflows are found 2. My Pipedream workflow doing the reverse geocode (look at the step called, 'steps.HERE')
2	HERE.com	1. https://developer.here.com/blog/integrating-here-in-pipedream-workflows 2. https://developer.here.com/ 3. https://developer.here.com/documentation/geocoding-search-api/dev_guide/topics/endpoint-reverse-geocode-brief.html	1. Blog post by Raymond Camden in which he explains how he developed a Pipedream workflow using reverse geocoding functionality from HERE.com. In the post, there is a link to his Pipedream workflow 2. HERE.com developer site 3. HERE's Reverse Geocoding documentation
3	Helium	1. https://github.com/mikedsp/helium/blob/master/MyDocuments/HowTo_BrowanTBHH100_to_GoogleSheet-SHARE.pdf	1. Document that describes in detail how to get data from a Browan TBHH100 temperature and humidity sensor to flow in real time to a Google Sheet. The end-to-end data flow is as follows: TBHH100 > Helium Hotspot > Helium Console > Pipedream > Google Sheet

Preconditions

The table below lists what is needed to complete this project.

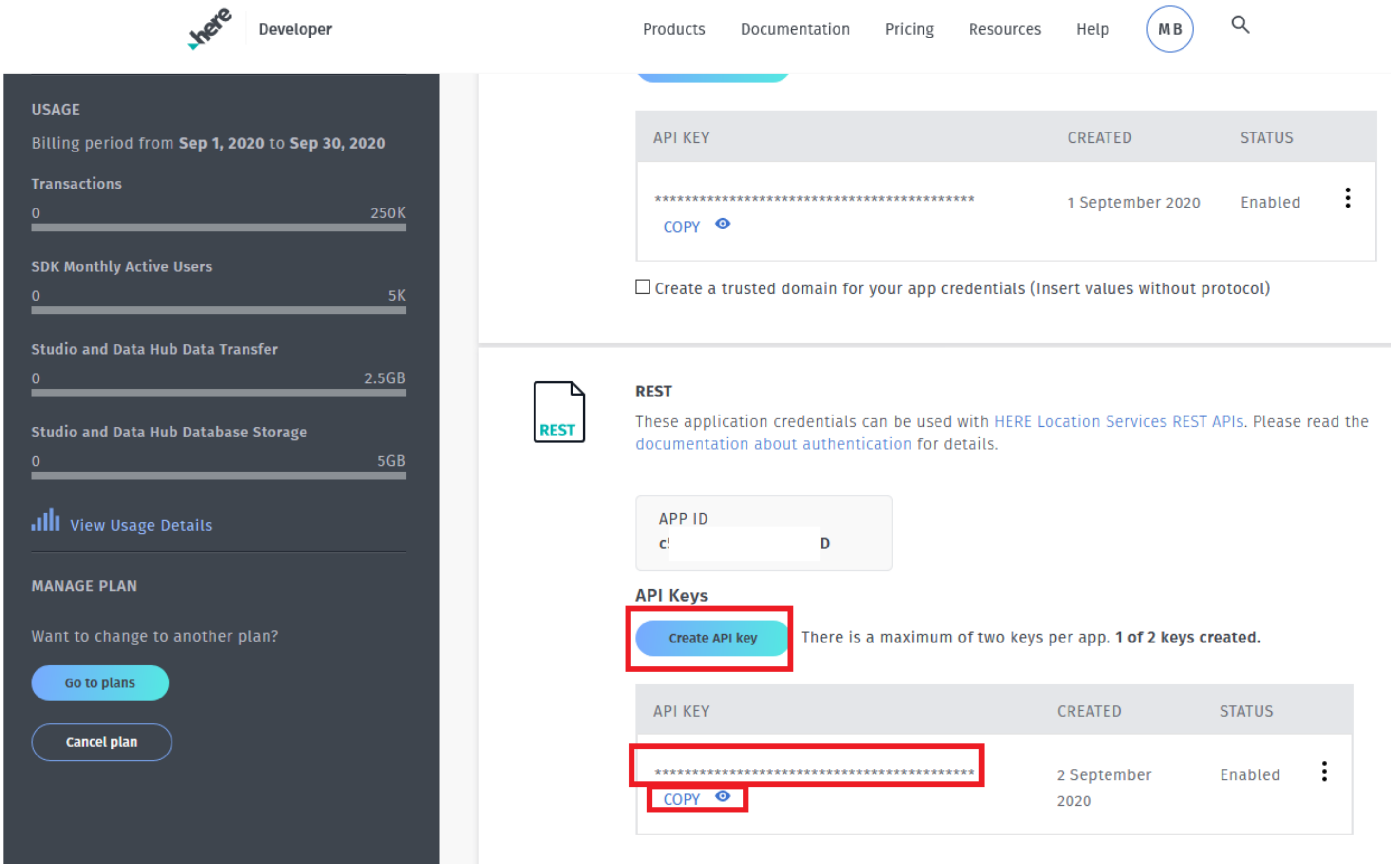
Precondition	Description
HERE.com account	You can sign up for a free account here: https://developer.here.com/
Pipedream Account	You can sign up for a free account here: https://docs.pipedream.com/sign-up/
Pipedream Workflow receiving GPS coordinates	To implement the Step described here, you'll need an existing Pipedream workflow receiving GPS data. In Raymond's blog post (see Reference 2.1) he shared his workflow pulling GPS data from pictures stored in Drop Box. In Reference 3.1, I explain how to build a Helium sensor – to – Google Sheet data flow. If you use those instructions, but use a Browan Object Locator as the sensor instead of the TBHH100 sensor, you can build a Pipedream workflow that receives GPS coordinates and send them (and other data from the sensor) to a Google Sheet.

Steps

The steps for creating and using a reverse geocode step in your Pipedream workflow are listed below. Subsequent sections in this document have screenshots of these steps.

1. Get setup in HERE.com
    a. Create a user account
    b. Create a project
    c. In the project, create a REST API key
2. Build the reverse geocode step in your pipedream workflow
3. (optional) Enjoy the sensor data flowing into your Google Sheet
4. (Bonus) Calculate distance between 2 sets of GPS Coordinates

Getting Set Up with Here.com



USAGE

Billing period from **Sep 1, 2020** to **Sep 30, 2020**

Transactions

0 250K

SDK Monthly Active Users

0 5K

Studio and Data Hub Data Transfer

0 2.5GB

Studio and Data Hub Database Storage

0 5GB

View Usage Details

MANAGE PLAN

Want to change to another plan?

Go to plans

Cancel plan

API KEY	CREATED	STATUS
*** COPY	1 September 2020	Enabled

□ Create a trusted domain for your app credentials (Insert values without protocol)

REST

These application credentials can be used with HERE Location Services REST APIs. Please read the documentation about authentication for details.

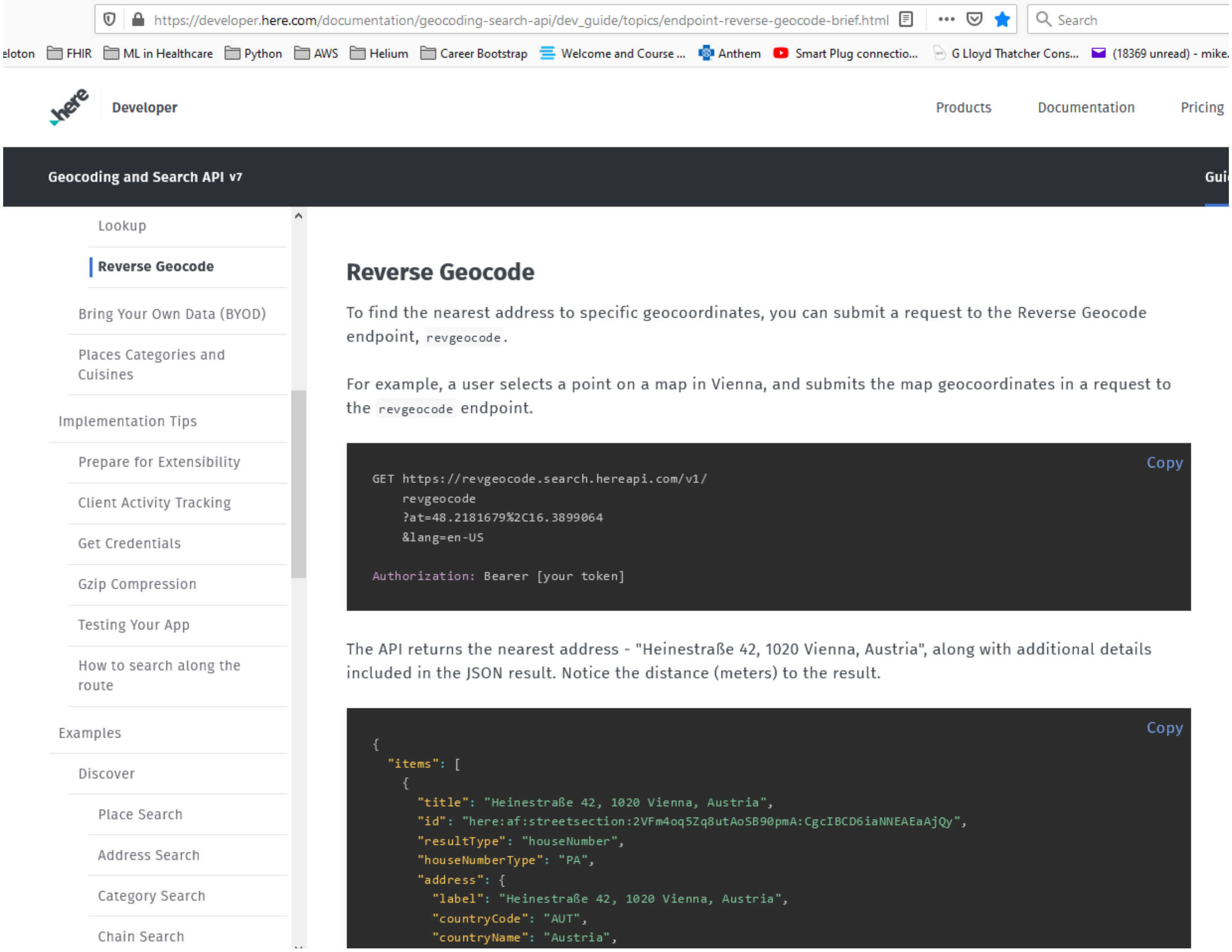
APP ID
c! [redacted] D

API Keys

Create API key There is a maximum of two keys per app. **1 of 2 keys created.**

API KEY	CREATED	STATUS
*** COPY	2 September 2020	Enabled

(above) HERE.com project – generating and copying a REST API key



Reverse Geocode

To find the nearest address to specific geocoordinates, you can submit a request to the Reverse Geocode endpoint, `revgeocode`.

For example, a user selects a point on a map in Vienna, and submits the map geocoordinates in a request to the `revgeocode` endpoint.

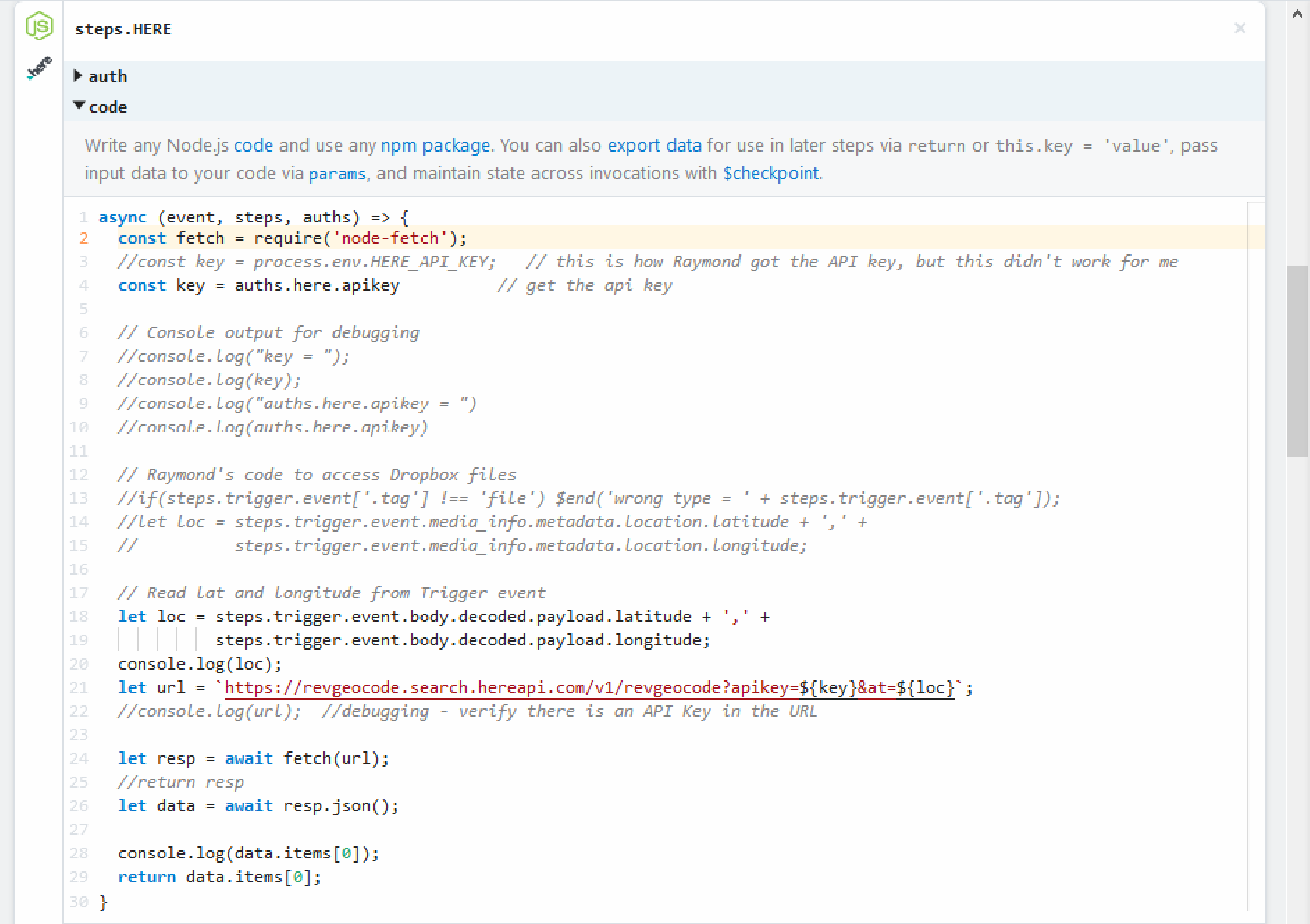
```
GET https://revgeocode.search.hereapi.com/v1/
    revgeocode
    ?at=48.2181679%2C16.3899064
    &lang=en-US

Authorization: Bearer [your token]
```

The API returns the nearest address - "Heinestraße 42, 1020 Vienna, Austria", along with additional details included in the JSON result. Notice the distance (meters) to the result.

```
{
  "items": [
    {
      "title": "Heinestraße 42, 1020 Vienna, Austria",
      "id": "here:af:streetsection:2VFm4oq5Zq8utAoSB90pmA:CgcIBCD6iaNNEAEaAjQy",
      "resultType": "houseNumber",
      "houseNumberType": "PA",
      "address": {
        "label": "Heinestraße 42, 1020 Vienna, Austria",
        "countryCode": "AUT",
        "countryName": "Austria",
```

(above) HERE.com’s documentation on using the Reverse Geocode API

Building the Reverse Geocode Step in your Pipedream Workflow



steps.HERE

▸ auth

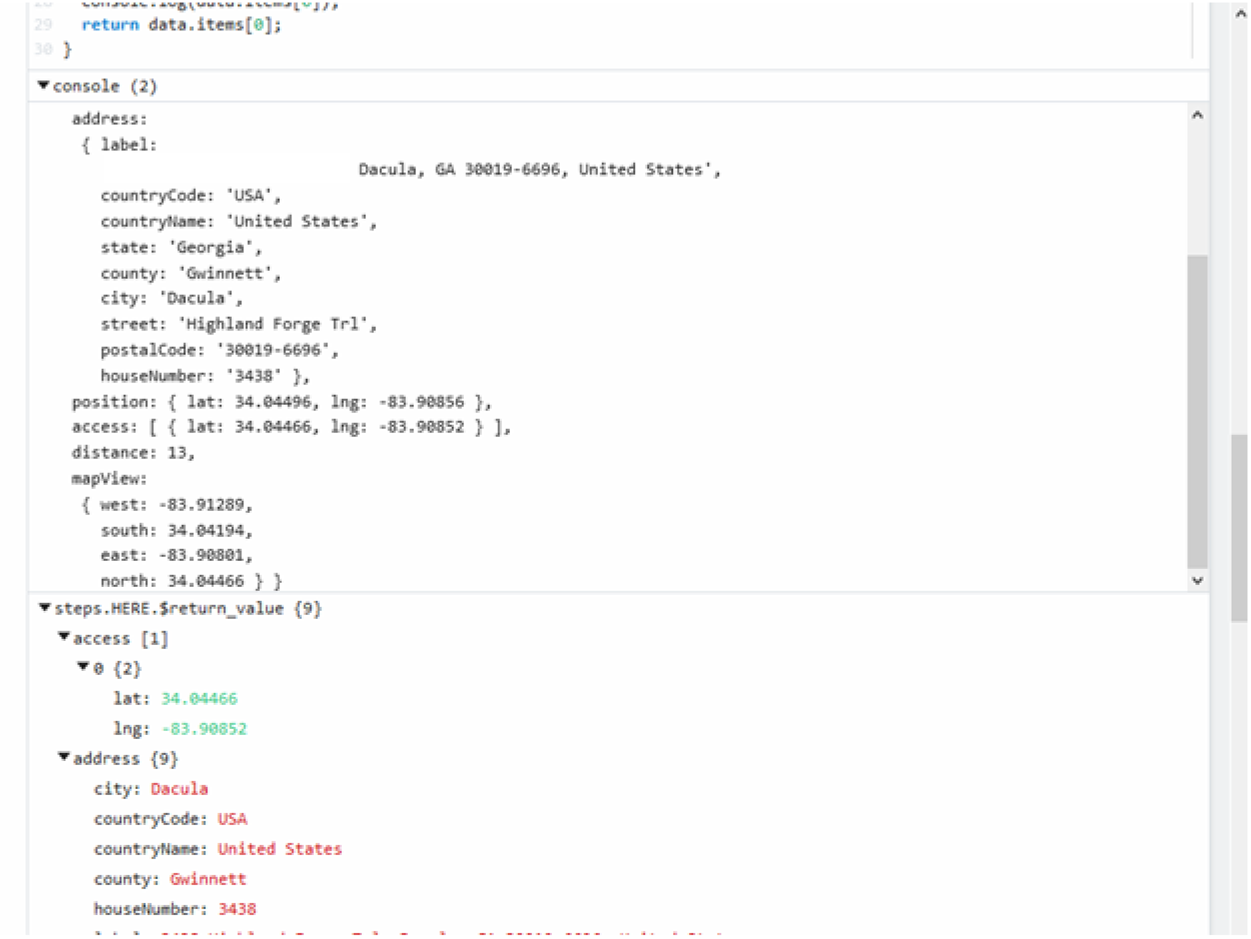
▾ code

Write any Node.js code and use any npm package. You can also export data for use in later steps via `return` or `this.key = 'value'`, pass input data to your code via `params`, and maintain state across invocations with $checkpoint.

```
async (event, steps, auths) => {
  const fetch = require('node-fetch');
  //const key = process.env.HERE_API_KEY;   // this is how Raymond got the API key, but this didn't work for me
  const key = auths.here.apikey          // get the api key

  // Console output for debugging
  //console.log("key = ");
  //console.log(key);
  //console.log("auths.here.apikey = ")
  //console.log(auths.here.apikey)

  // Raymond's code to access Dropbox files
  //if(steps.trigger.event['.tag'] !== 'file') $end('wrong type = ' + steps.trigger.event['.tag']);
  //let loc = steps.trigger.event.media_info.metadata.location.latitude + ',' +
  //          steps.trigger.event.media_info.metadata.location.longitude;

  // Read lat and longitude from Trigger event
  let loc = steps.trigger.event.body.decoded.payload.latitude + ',' +
            steps.trigger.event.body.decoded.payload.longitude;
  console.log(loc);
  let url = `https://revgeocode.search.hereapi.com/v1/revgeocode?apikey=${key}&at=${loc}`;
  //console.log(url);  //debugging - verify there is an API Key in the URL

  let resp = await fetch(url);
  //return resp
  let data = await resp.json();

  console.log(data.items[0]);
  return data.items[0];
}
```

(above) reverse geocode workflow step – using auths.here.apikey to read the key

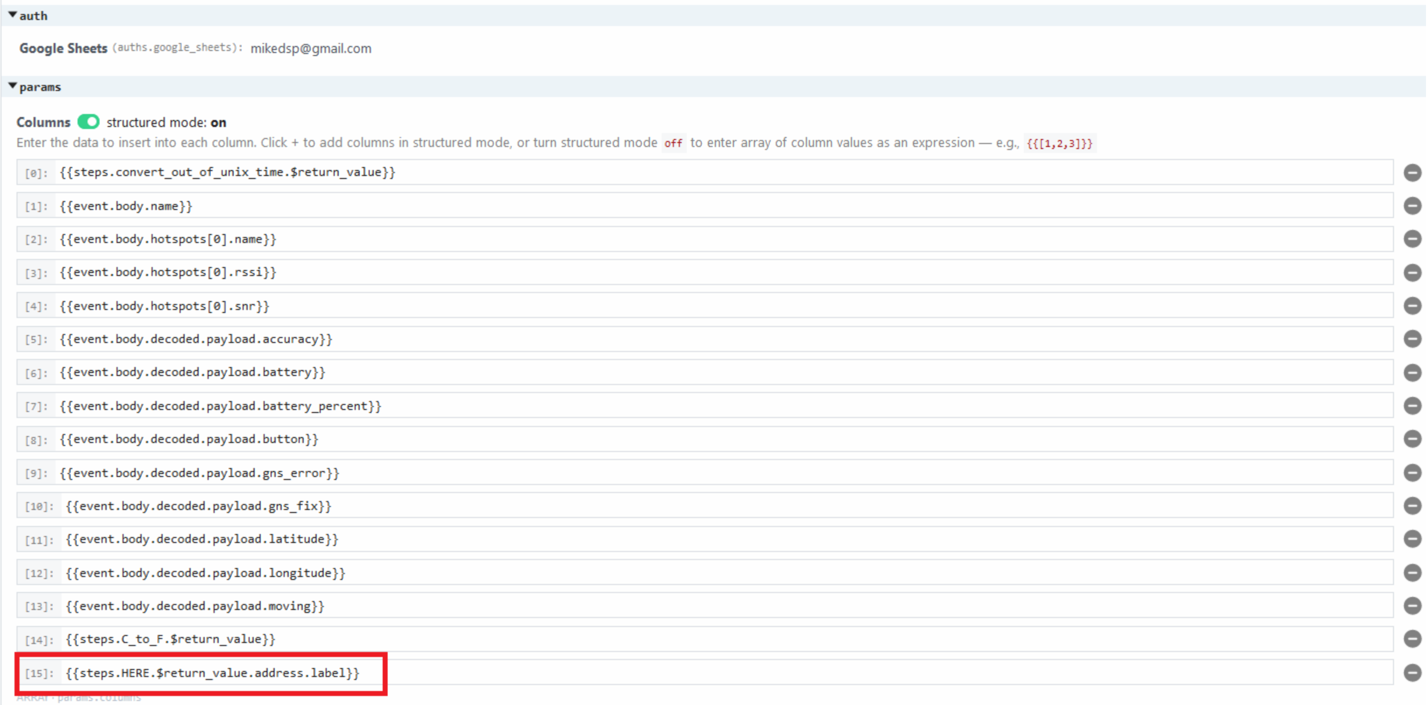
- Raymond Camden shared his Pipedream workflow here
- I've shared my Pipedream workflow here



```
  console.log(data.items[0]);
  return data.items[0];
}
```

```
▾console (2)
    address:
     { label:                            Dacula, GA 30019-6696, United States',
       countryCode: 'USA',
       countryName: 'United States',
       state: 'Georgia',
       county: 'Gwinnett',
       city: 'Dacula',
       street: 'Highland Forge Trl',
       postalCode: '30019-6696',
       houseNumber: '3438' },
    position: { lat: 34.04496, lng: -83.90856 },
    access: [ { lat: 34.04466, lng: -83.90852 } ],
    distance: 13,
    mapView:
     { west: -83.91289,
       south: 34.04194,
       east: -83.90801,
       north: 34.04466 } }
▾steps.HERE.$return_value {9}
  ▾access [1]
    ▾0 {2}
       lat: 34.04466
       lng: -83.90852
  ▾address {9}
     city: Dacula
     countryCode: USA
     countryName: United States
     county: Gwinnett
     houseNumber: 3438
```

(above) Looking at a portion of the output of the reverse geocode step in Pipedream



▾auth

Google Sheets (auths.google_sheets): mikedsp@gmail.com

▾params

Columns structured mode: **on**

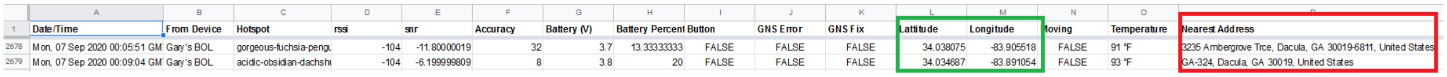
Enter the data to insert into each column. Click + to add columns in structured mode, or turn structured mode `off` to enter array of column values as an expression — e.g., `{{[1,2,3]}}`

```
[0]:  {{steps.convert_out_of_unix_time.$return_value}}
[1]:  {{event.body.name}}
[2]:  {{event.body.hotspots[0].name}}
[3]:  {{event.body.hotspots[0].rssi}}
[4]:  {{event.body.hotspots[0].snr}}
[5]:  {{event.body.decoded.payload.accuracy}}
[6]:  {{event.body.decoded.payload.battery}}
[7]:  {{event.body.decoded.payload.battery_percent}}
[8]:  {{event.body.decoded.payload.button}}
[9]:  {{event.body.decoded.payload.gns_error}}
[10]: {{event.body.decoded.payload.gns_fix}}
[11]: {{event.body.decoded.payload.latitude}}
[12]: {{event.body.decoded.payload.longitude}}
[13]: {{event.body.decoded.payload.moving}}
[14]: {{steps.C_to_F.$return_value}}
[15]: {{steps.HERE.$return_value.address.label}}
```

ARRAY · params.columns

(above) Sending the address.label field from the reverse geocode step to a column in a Google Sheet. The field, ‘address.label’, has all the address components in a single string

Viewing the Address Information in the Google Sheet



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
1	Date/Time	From Device	Hotspot	rssi	snr	Accuracy	Battery (V)	Battery Percent	Button	GNS Error	GNS Fix	Lattitude	Longitude	Moving	Temperature	Nearest Address
2678	Mon, 07 Sep 2020 00:05:51 GMT	Gary's BOL	gorgeous-fuchsia-pengu	-104	-11.80000019	32	3.7	13.33333333	FALSE	FALSE	FALSE	34.038075	-83.905518	FALSE	91 °F	3235 Ambergrove Trce, Dacula, GA 30019-6811, United States
2679	Mon, 07 Sep 2020 00:09:04 GMT	Gary's BOL	acidic-obsidian-dachshu	-104	-6.199999809	8	3.8	20	FALSE	FALSE	FALSE	34.034687	-83.891054	FALSE	93 °F	GA-324, Dacula, GA 30019, United States

(above) Looking at the Google Sheet – the last column shows the nearest address to the Latitude and Longitude values in columns L and M.

(Bonus) Calculate distance between 2 sets of GPS Coordinates

Problem Statement

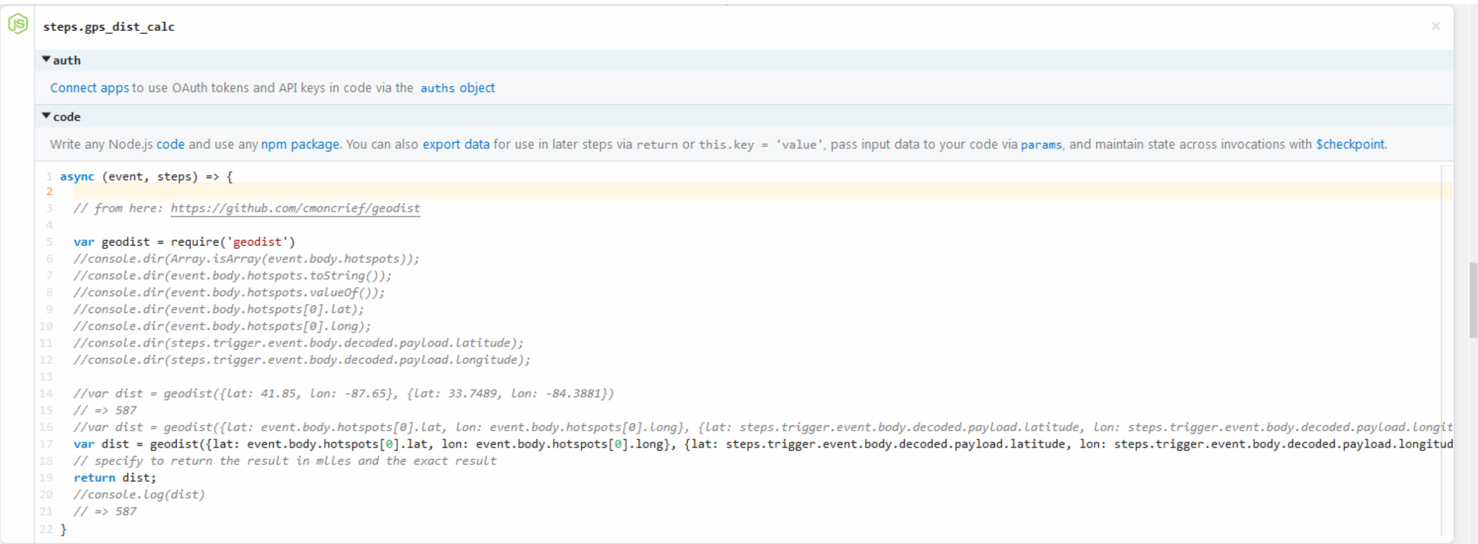
In my data feed from the Helium Network to the Pipedream workflow, there are 2 sets of GPS coordinates: one set for the location of the BOL tracking device and the other set for the location of the Helium hotspot. I wanted to calculate the distance between the 2 sets of coordinates to better understand the coverage area of the hotspot. Or put another way – if the hotspot is able to pick up the reading from the BOL tracking device, then that means that the device is within range of the hotspot. How far away will I be able to pick up readings from the BOL – that's what I wanted to be able to track and see.

Solution

There are lots of Node.JS GPS distance calculations freely available; I chose 2 that had enough documentation to make them reasonable easy to drop into Pipedream. Reference 1.2 has a link to my workflow where you can see or copy the steps. I show the steps in the screenshots below.

The first screenshot is what I consider to be the simple solution because it has just a single, simple function call. That Pipedream step is called, *steps.gps_dist_calc*.

The subsequent screenshots show the more complicated function.



steps.gps_dist_calc

▼auth

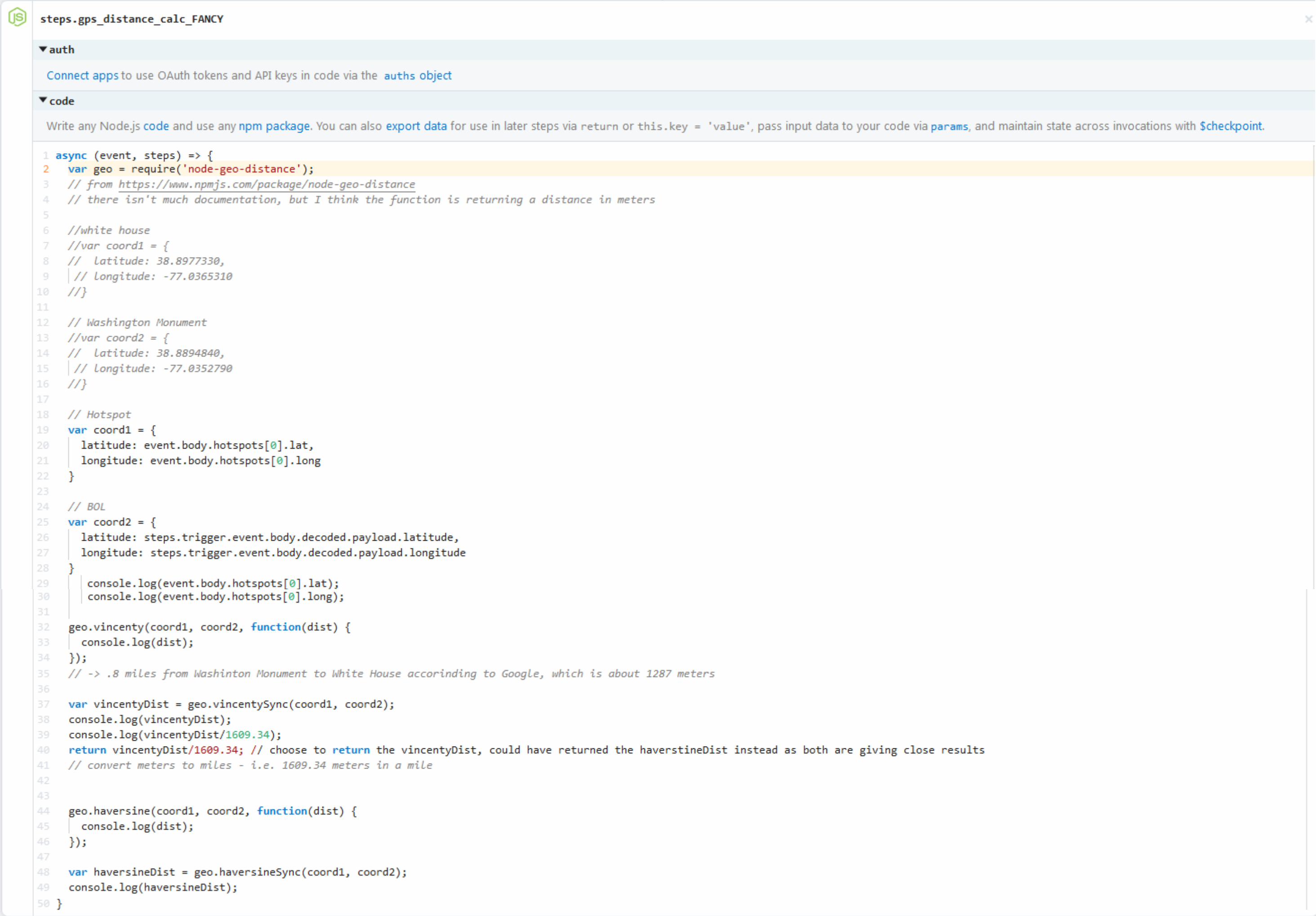
Connect apps to use OAuth tokens and API keys in code via the auths object

▼code

Write any Node.js code and use any npm package. You can also export data for use in later steps via return or this.key = 'value', pass input data to your code via params, and maintain state across invocations with $checkpoint.

```
async (event, steps) => {

  // from here: https://github.com/cmoncrief/geodist

  var geodist = require('geodist')
  //console.dir(Array.isArray(event.body.hotspots));
  //console.dir(event.body.hotspots.toString());
  //console.dir(event.body.hotspots.valueOf());
  //console.dir(event.body.hotspots[0].lat);
  //console.dir(event.body.hotspots[0].long);
  //console.dir(steps.trigger.event.body.decoded.payload.latitude);
  //console.dir(steps.trigger.event.body.decoded.payload.longitude);

  //var dist = geodist({lat: 41.85, lon: -87.65}, {lat: 33.7489, lon: -84.3881})
  // => 587
  //var dist = geodist({lat: event.body.hotspots[0].lat, lon: event.body.hotspots[0].long}, {lat: steps.trigger.event.body.decoded.payload.latitude, lon: steps.trigger.event.body.decoded.payload.longit
  var dist = geodist({lat: event.body.hotspots[0].lat, lon: event.body.hotspots[0].long}, {lat: steps.trigger.event.body.decoded.payload.latitude, lon: steps.trigger.event.body.decoded.payload.longitud
  // specify to return the result in mlles and the exact result
  return dist;
  //console.log(dist)
  // => 587
}
```

(above) Showing the pipedream step, *steps.gps_dist_calc*. There is a link to the GitHub repository where I found the code. I've commented out all the debugging steps. The hardest part was not using the geodist function, but rather figuring out how to read the hotspot's GPS data.



steps.gps_distance_calc_FANCY

▼auth

Connect apps to use OAuth tokens and API keys in code via the auths object

▼code

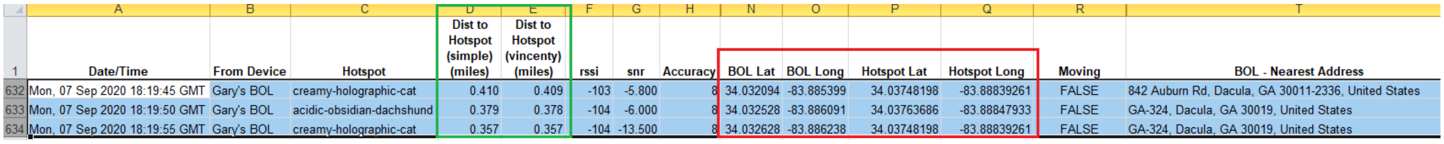
Write any Node.js code and use any npm package. You can also export data for use in later steps via return or this.key = 'value', pass input data to your code via params, and maintain state across invocations with $checkpoint.

```javascript
async (event, steps) => {
  var geo = require('node-geo-distance');
  // from https://www.npmjs.com/package/node-geo-distance
  // there isn't much documentation, but I think the function is returning a distance in meters

  //white house
  //var coord1 = {
  //  latitude: 38.8977330,
  // longitude: -77.0365310
  //}

  // Washington Monument
  //var coord2 = {
  //  latitude: 38.8894840,
  // longitude: -77.0352790
  //}

  // Hotspot
  var coord1 = {
    latitude: event.body.hotspots[0].lat,
    longitude: event.body.hotspots[0].long
  }

  // BOL
  var coord2 = {
    latitude: steps.trigger.event.body.decoded.payload.latitude,
    longitude: steps.trigger.event.body.decoded.payload.longitude
  }
     console.log(event.body.hotspots[0].lat);
     console.log(event.body.hotspots[0].long);

  geo.vincenty(coord1, coord2, function(dist) {
    console.log(dist);
  });
  // -> .8 miles from Washinton Monument to White House accorinding to Google, which is about 1287 meters

  var vincentyDist = geo.vincentySync(coord1, coord2);
  console.log(vincentyDist);
  console.log(vincentyDist/1609.34);
  return vincentyDist/1609.34; // choose to return the vincentyDist, could have returned the haverstineDist instead as both are giving close results
  // convert meters to miles - i.e. 1609.34 meters in a mile


  geo.haversine(coord1, coord2, function(dist) {
    console.log(dist);
  });

  var haversineDist = geo.haversineSync(coord1, coord2);
  console.log(haversineDist);
}
```

(above) Showing the pipedream step, *steps.gps_distance_calc_FANCY.* The function offers 2 different calculation methods: Vincenty's Formula and Haversine. I chose to use Vincenty's formula for no particular reason. Notice that the formula returns the distance in meters. I wanted to see the distance in miles, so divided the result by 1609.34.

Results

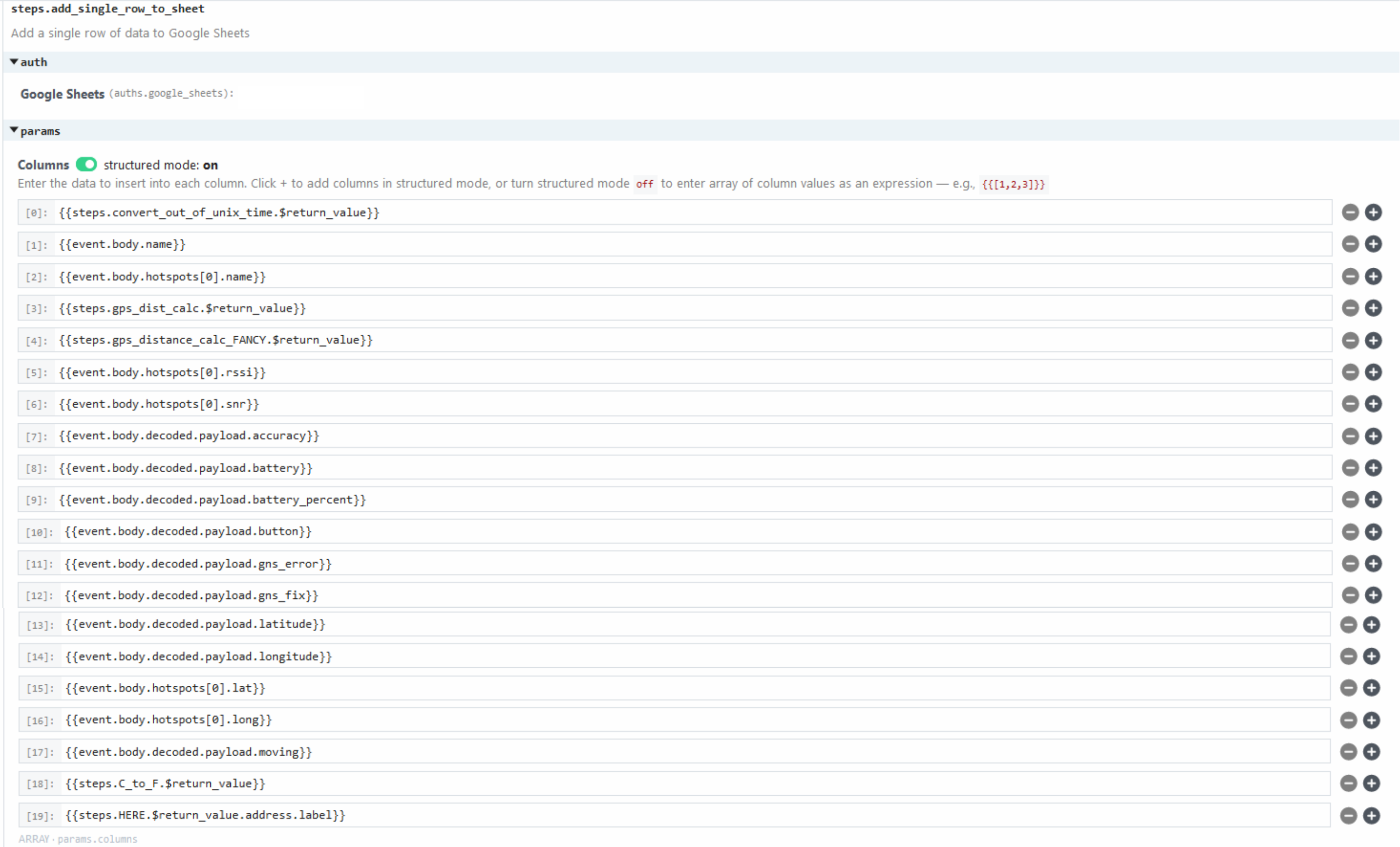
The screenshot below shows the distance calculation results output to the Google Sheet.



	A	B	C	D	E	F	G	H	N	O	P	Q	R	T
1	Date/Time	From Device	Hotspot	Dist to Hotspot (simple) (miles)	Dist to Hotspot (vincenty) (miles)	rssi	snr	Accuracy	BOL Lat	BOL Long	Hotspot Lat	Hotspot Long	Moving	BOL - Nearest Address
632	Mon, 07 Sep 2020 18:19:45 GMT	Gary's BOL	creamy-holographic-cat	0.410	0.409	-103	-5.800	8	34.032094	-83.885399	34.03748198	-83.88839261	FALSE	842 Auburn Rd, Dacula, GA 30011-2336, United States
633	Mon, 07 Sep 2020 18:19:50 GMT	Gary's BOL	acidic-obsidian-dachshund	0.379	0.378	-104	-6.000	8	34.032528	-83.886091	34.03763686	-83.88847933	FALSE	GA-324, Dacula, GA 30019, United States
634	Mon, 07 Sep 2020 18:19:55 GMT	Gary's BOL	creamy-holographic-cat	0.357	0.357	-104	-13.500	8	34.032628	-83.886238	34.03748198	-83.88839261	FALSE	GA-324, Dacula, GA 30019, United States

(above) GPS distance (and reverse geocode) information from a Browan Object Locator tracking device output to a Google Sheet.

I put the 2 distance calculations next to the RSSI and SNR values to make it easy to see how distance from the hotspot affects the RSSI and SNR values.



steps.add_single_row_to_sheet

Add a single row of data to Google Sheets

▼auth

Google Sheets (auths.google_sheets):

▼params

Columns ● structured mode: **on**

Enter the data to insert into each column. Click + to add columns in structured mode, or turn structured mode `off` to enter array of column values as an expression — e.g., `{{[1,2,3]}}`

```
[0]:  {{steps.convert_out_of_unix_time.$return_value}}
[1]:  {{event.body.name}}
[2]:  {{event.body.hotspots[0].name}}
[3]:  {{steps.gps_dist_calc.$return_value}}
[4]:  {{steps.gps_distance_calc_FANCY.$return_value}}
[5]:  {{event.body.hotspots[0].rssi}}
[6]:  {{event.body.hotspots[0].snr}}
[7]:  {{event.body.decoded.payload.accuracy}}
[8]:  {{event.body.decoded.payload.battery}}
[9]:  {{event.body.decoded.payload.battery_percent}}
[10]: {{event.body.decoded.payload.button}}
[11]: {{event.body.decoded.payload.gns_error}}
[12]: {{event.body.decoded.payload.gns_fix}}
[13]: {{event.body.decoded.payload.latitude}}
[14]: {{event.body.decoded.payload.longitude}}
[15]: {{event.body.hotspots[0].lat}}
[16]: {{event.body.hotspots[0].long}}
[17]: {{event.body.decoded.payload.moving}}
[18]: {{steps.C_to_F.$return_value}}
[19]: {{steps.HERE.$return_value.address.label}}
```

ARRAY · params.columns

(above) Showing the parameters being sent to the Google Sheet via the Pipedream workflow