

Not sure what the difference is ??

#### ERROR ?? Event

The source event that triggered the webhook

{

"name": "TestData1",

"data": "{\"temp\":852 }",

"ttl": 60,

"published\_at": "2018-10-06T01:08:19.064Z",

"coreid": "22002a000647343339373536"

}

void loop() {

tempvalue = analogRead(tempsensor);

String data = String::format( "{\"temp\":%d }", tempvalue );

data = String::format( "{\"lat\":%f , \"lng\":%f}", lat, lng);

Particle.publish("TestData1", data, PRIVATE);

float lat = 39.7391536;

float lng = -104.984703;

data = String::format( "{\"lat\":%f, \"lng\":%f}", lat, lng);

Particle.publish("elevation", data, PRIVATE);

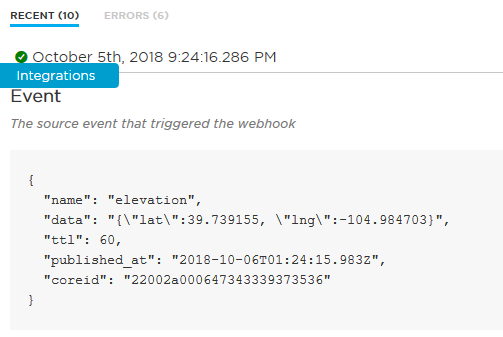
// Wait 60 seconds

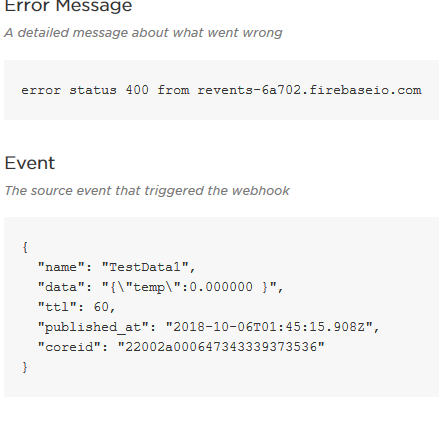
delay(60000);

}









void setup() {

// First, declare all of our pins. This lets our device know which ones will be used for outputting voltage, and which ones will read incoming voltage.

pinMode(led,OUTPUT); // Our LED pin is output (lighting up the LED)

pinMode(photoresistor,INPUT); // Our photoresistor pin is input (reading the photoresistor)

pinMode(power,OUTPUT); // The pin powering the photoresistor is output (sending out consistent power)

// Next, write the power of the photoresistor to be the maximum possible, so that we can use this for power.

digitalWrite(power,HIGH);

// We are going to declare a Particle.variable() here so that we can access the value of the photoresistor from the cloud.

Particle.variable("analogvalue", &analogvalue, INT);

// This is saying that when we ask the cloud for "analogvalue", this will reference the variable analogvalue in this app, which is an integer variable.

// We are also going to declare a Particle.function so that we can turn the LED on and off from the cloud.

Particle.function("led",ledToggle);

// This is saying that when we ask the cloud for the function "led", it will employ the function ledToggle() from this app.

}

float lat = 39.7391536;

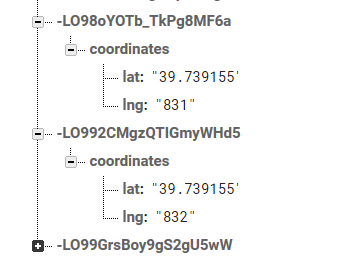
float lng = -104.984703;

data = String::format( "{\"lat\":%f, \"lng\":%d}", lat, tempvalue );

Particle.publish("elevation", data, PRIVATE);

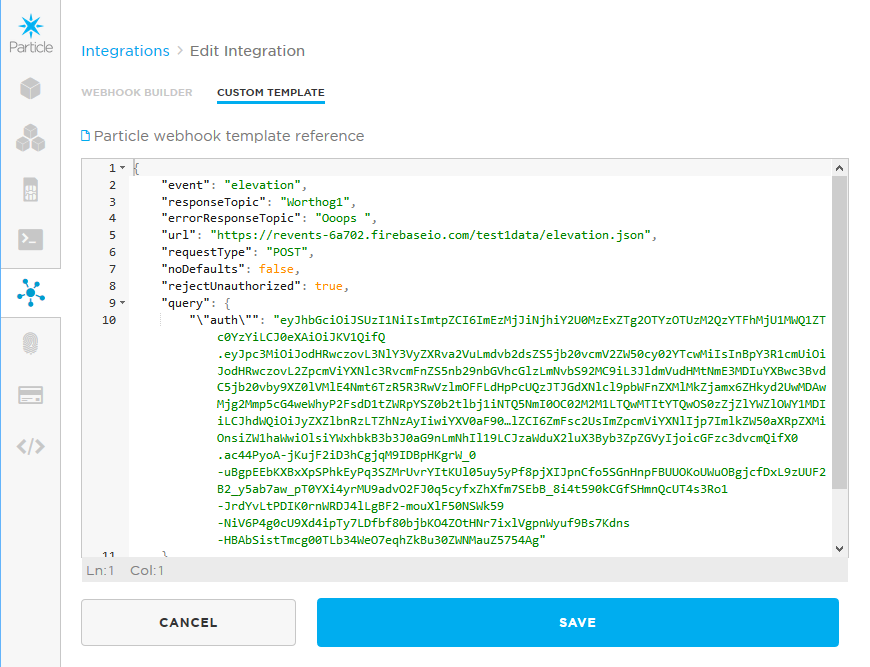
// Wait 60 seconds

delay(60000);

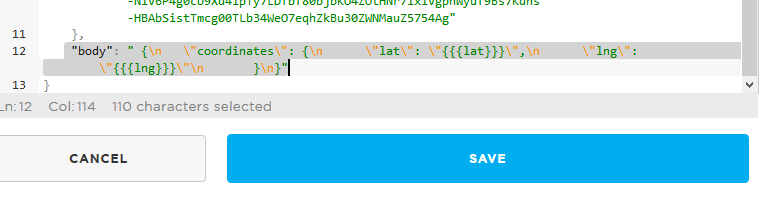


Not sure why this works ???

Check out the custom template



"body": " {\n \"coordinates\": {\n \"lat\": \"{{{lat}}}\",\n \"lng\": \"{{{lng}}}\"\n }\n}"



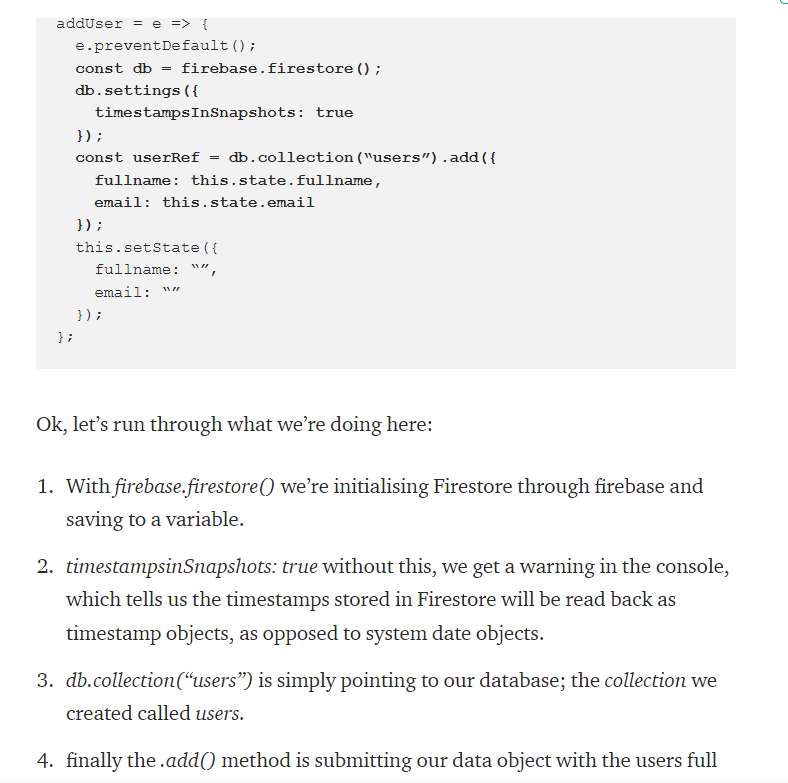
"body": " {\n \"coordinates\": {\n \"lat\": \"{{{lat}}}\",\n \"lng\": \"{{{lng}}}\"\n }\n}"

"body": "{\n \"event\": \"{{{PARTICLE\_EVENT\_NAME}}}\",\n \"data\": \"{{{PARTICLE\_EVENT\_VALUE}}}\",\n \"coreid\": \"{{{PARTICLE\_DEVICE\_ID}}}\",\n \"published\_at\": \"{{{PARTICLE\_PUBLISHED\_AT}}}\",

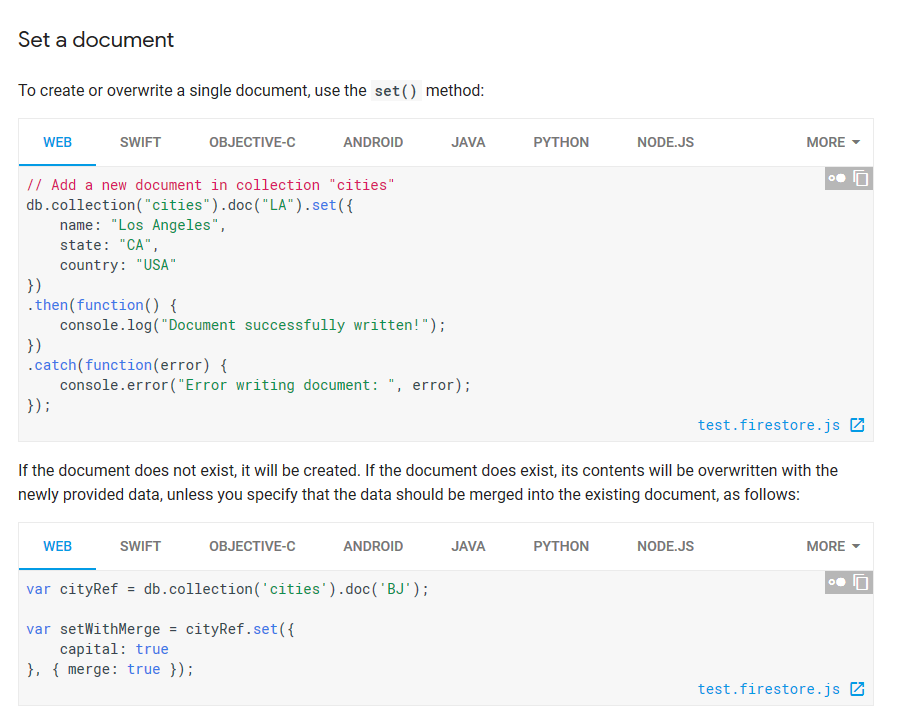
\n \"temperature\" : \ "{{{temp}}} \ "\n

}"

\n \"lng\" : \ "{{{lng}}} \ "\n



<https://firebase.google.com/docs/firestore/manage-data/add-data>



// Add a new document in collection "cities"  
db.collection("cities").doc("LA").set({  
    name: "Los Angeles",  
    state: "CA",  
    country: "USA"  
})  
.then(function() {  
    console.log("Document successfully written!");  
})  
.catch(function(error) {  
    console.error("Error writing document: ", error);  
});



// Add a new document with a generated id.  
db.collection("cities").add({  
    name: "Tokyo",  
    country: "Japan"  
})  
.then(function(docRef) {  
    console.log("Document written with ID: ", docRef.id);  
})  
.catch(function(error) {  
    console.error("Error adding document: ", error);  
});

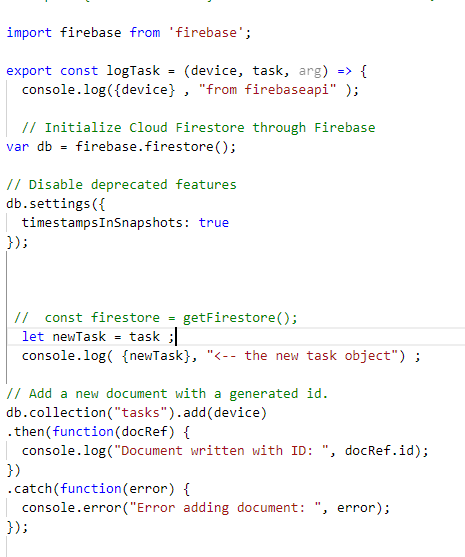
**Use the Cloud Firestore REST API**

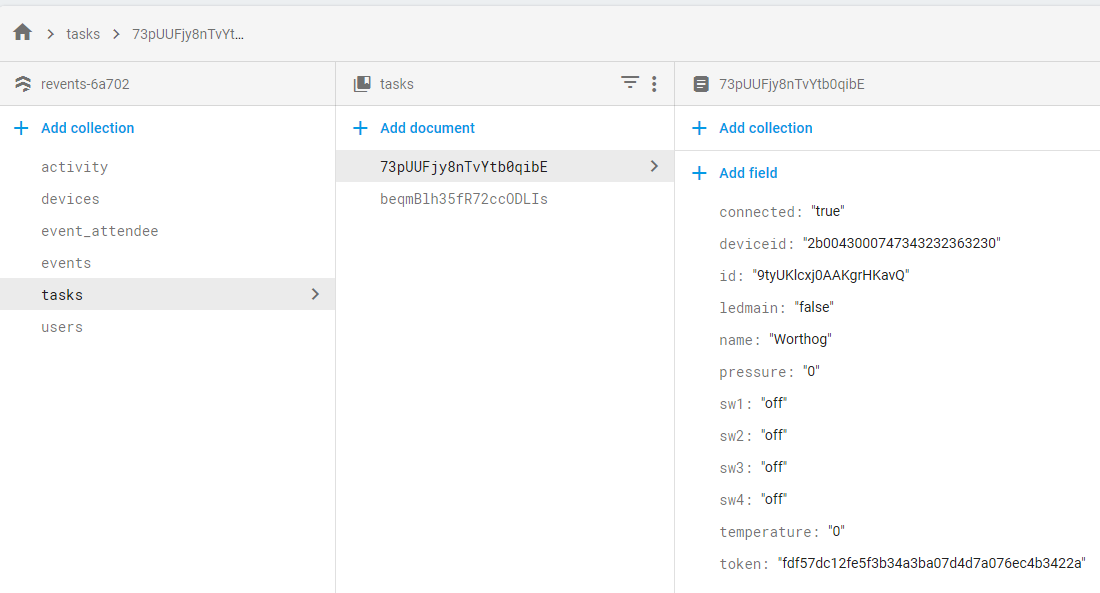
While the easiest way to use Cloud Firestore is to use one of the native client libraries, there are some situations when it is useful to call the REST API directly.

The REST API can be helpful for the following use cases:

* Accessing Cloud Firestore from a resource-constrained environment, such as an internet of things (IoT) device, where running a complete client library is not possible.
* Automating database administration or retrieving detailed database metadata.

If you are using a [gRPC-supported language](https://grpc.io/about/#osp), consider using the [RPC API](https://firebase.google.com/docs/firestore/reference/rpc/) rather than the REST API.

The following code “WORKS”, added the Device object to the Tasks db in Firestore   




<https://firebase.google.com/docs/firestore/query-data/get-data>

Get all documents in a collection

In addition, you can retrieve all documents in a collection by omitting the where() filter entirely:

WebSwiftObjective-CAndroidJavaPythonNode.js

More

db.collection("cities").get().then(function(querySnapshot) {

querySnapshot.forEach(function(doc) {

// doc.data() is never undefined for query doc snapshots

console.log(doc.id, " => ", doc.data());

});

});

Simple queries

The following query returns all cities with state CA:

WebSwiftObjective-CAndroidJavaPythonNode.js

More

// Create a reference to the cities collection

var citiesRef = db.collection("cities");

// Create a query against the collection.

var query = citiesRef.where("state", "==", "CA");

test.firestore.js

The following query returns all the capital cities:

WebSwiftObjective-CAndroidJavaPythonNode.js

More

var citiesRef = db.collection("cities");

var query = citiesRef.where("capital", "==", true);

test.firestore.js

The where() method takes three parameters: a field to filter on, a comparison operation, and a value. The comparison can be <, <=, ==, >, >=, or array\_contains. For iOS, Android, and Java, the comparison operator is explicitly named in the method.

Some example filters:

WebSwiftObjective-CAndroidJavaPythonNode.js

More

citiesRef.where("state", "==", "CA")

citiesRef.where("population", "<", 100000)

citiesRef.where("name", ">=", "San Francisco")

test.firestore.js

Array membership

You can use the array\_contains operator to filter based on array values. For example:

WebSwiftObjective-CAndroidJavaPythonNode.js

More

citiesRef.where("regions", "array-contains", "west\_coast")

test.firestore.js

This query returns every city document where the regions field is an array that contains west\_coast. If the array has multiple instances of the value you query on, the document is included in the results only once.

See if we can map the table to the tasks in the firestore state…

>>> Old version :

<Table.Body>

{\_.map(data, ({ deviceid, ledmain, name }) => (

<Table.Row key={name}>

<Table.Cell>{name}</Table.Cell>

<Table.Cell>{deviceid}</Table.Cell>

<Table.Cell>{ledmain}</Table.Cell>

</Table.Row>

))}

</Table.Body>

const todosList = !isLoaded(todos)

? 'Loading'

: isEmpty(todos)

? 'Todo list is empty'

: Object.keys(todos).map(

(key, id) => (

<TodoItem key={key} id={id} todo={todos[key]}/>

)

)

return (

</Table.Header>

<Table.Body>

{\_.map(tasks, ({ id, deviceid, ledmain, name }) => (

<Table.Row key={id}>

<Table.Cell>{name}</Table.Cell>

<Table.Cell>{deviceid}</Table.Cell>

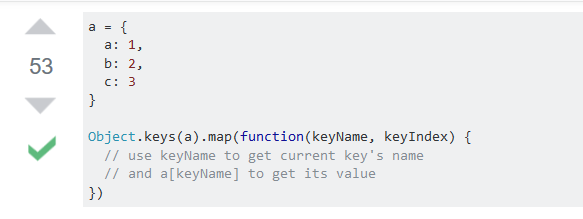
<Table.Cell>{ledmain}</Table.Cell>

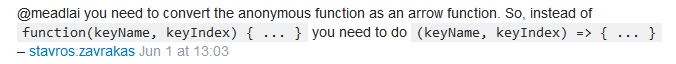
</Table.Row>

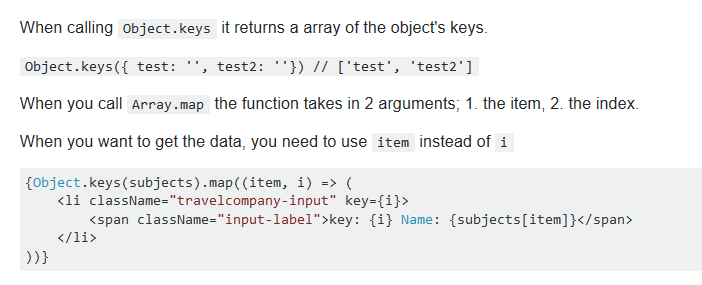
))}

</Table.Body>

</Table>







Map over the keys of the object using Object.keys():

{Object.keys(yourObject).map(function(key) {

return <div>Key: {key}, Value: {yourObject[key]}</div>;

})}

6 down vote accepted

Also, you can't use map for objects like this. You should write

var customer = Object.keys(Customers).map(function(s){ return Customers[s].name });

const arr = Object.keys(bands).map((key) => {

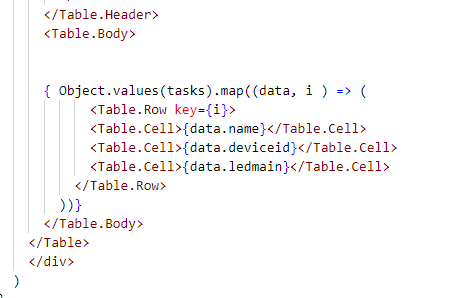
bands[key].\_id = key;

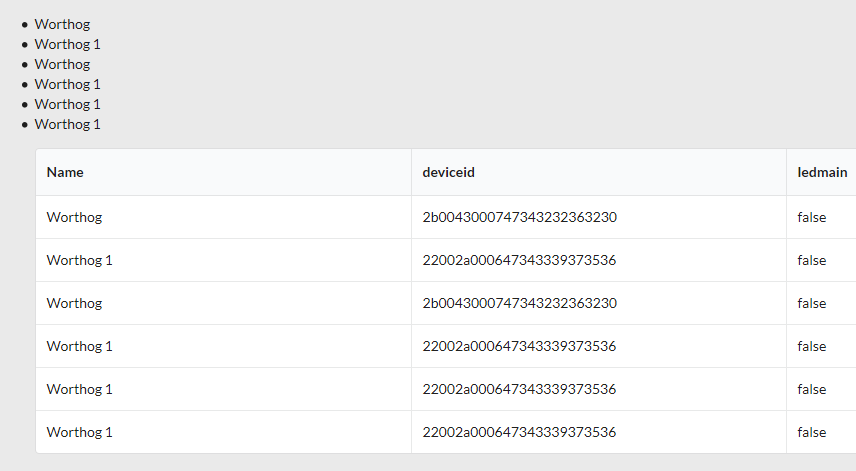
return bands.[key];

});

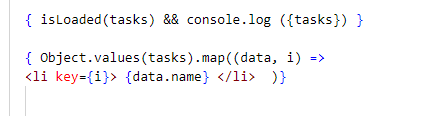
Finally got the map function to work with Firebase and allow us to use the Key value as the key for the react component : see stats.jsx

This version works





Note the debug code:



import { firestoreConnect, isLoaded, isEmpty } from 'react-redux-firebase';

render() {

const { tasks } = this.props;

const { column, data, direction } = this.state

if (!isLoaded(tasks) || isEmpty(tasks)) return <LoadingComponent inverted={true} />;

## MultiLed2 (Worthog1)

<https://console.particle.io/devices/22002a000647343339373536>

// This #include statement was automatically added by the Particle IDE.

#include "lib1.h"

// -----------------------------------------

// Function and Variable with tempsensor

// -----------------------------------------

// In this example, we're going to register a Spark.variable() with the cloud so that we can read the temperature

// from the TMP36G sensor.

// We'll also register a Spark.function so that we can turn the LED on and off remotely.

// We're going to start by declaring which pins everything is plugged into.

int led0 = D0; // This is where your LED is plugged in. The other side goes to a resistor connected to GND.

int sw1 = D1; // this is a LED, to indicate a future switch/solenoid

int sw2 = D2;

int sw3 = D3;

int tempsensor = A5; // This is where your temp sensor is plugged in.

int tempvalue; // Here we are declaring the integer variable analogvalue

char tempString[20]; // size to largest number of chars to display plus a terminating NULL

// Next we go into the setup function.

void setup() {

// First, declare all of our pins. This lets our device know which ones will be used for outputting voltage, and which ones will read incoming voltage.

pinMode(led0,OUTPUT); // Our LED pin is output (lighting up the LED)

pinMode(sw1,OUTPUT); // Our switch/LED pin is output (lighting up the LED)

pinMode(sw2,OUTPUT); // Our switch/LED pin is output (lighting up the LED)

pinMode(sw3,OUTPUT); // Our switch/LED pin is output (lighting up the LED)

pinMode(tempsensor,INPUT); // Our tempsensor pin is input (reading the TMP36 tempsensor)

// Declare a Spark.variable() here to access the value of the sensor from the cloud.

Spark.variable("tempvalue", &tempvalue, INT);

// This is saying that when we ask the cloud for "analogvalue", this will reference the variable analogvalue in this app, which is an integer variable.

// We are also going to declare a Spark.function so that we can turn the LED on and off from the cloud.

Spark.function("led",ledToggle);

// This is saying that when we ask the cloud for the function "led", it will employ the function ledToggle() from this app.

Spark.function("sw1", sw1Toggle);

Spark.function("sw2", sw2Toggle);

Spark.function("sw3", sw3Toggle);

}

// Next is the loop function...

void loop() {

// check to see what the value of the photoresistor is and store it in the int variable analogvalue

tempvalue = analogRead(tempsensor);

// String temperature = String::format("%d", tempvalue);

String data = String::format( "{\"temp\":%d }", tempvalue );

// sprintf(tempString, "temperature = %2.4", tempvalue );

// String data = String::format("%.2f", tempvalue);

// std::string pi = "pi is " + std::to\_string(3.1415926);

// std::string s = std::to\_string(tempvalue);

// Particle.publish("TestData1", data, PRIVATE);

float lat = 39.7391536;

float lng = -104.984703;

data = String::format( "{\"lat\":%f, \"lng\":%d}", lat, tempvalue );

// Particle.publish("elevation", data, PRIVATE);

// Wait 60 seconds

delay(60000);

}

// Finally, we will write out our ledToggle function, which is referenced by the Spark.function() called "led"

int ledToggle(String command) {

if (command=="on") {

digitalWrite(led0,HIGH);

return 1;

}

else if (command=="off") {

digitalWrite(led0,LOW);

return 0;

}

else {

return -1;

}

}

// Finally, we will write out our ledToggle function, which is referenced by the Spark.function() called "led"

int sw1Toggle(String command) {

if (command=="on") {

digitalWrite(sw1,HIGH);

return 1;

}

else if (command=="off") {

digitalWrite(sw1,LOW);

return 0;

}

else {

return -1;

}

}

int sw2Toggle(String command) {

if (command=="on") {

digitalWrite(sw2,HIGH);

return 1;

}

else if (command=="off") {

digitalWrite(sw2,LOW);

return 0;

}

else {

return -1;

}

}

int sw3Toggle(String command) {

if (command=="on") {

digitalWrite(sw3,HIGH);

return 1;

}

else if (command=="off") {

digitalWrite(sw3,LOW);

return 0;

}

else {

return -1;

}

}

Particle Breathing Green ???

<https://community.particle.io/t/photon-breathing-green-after-flashing/16529/13>

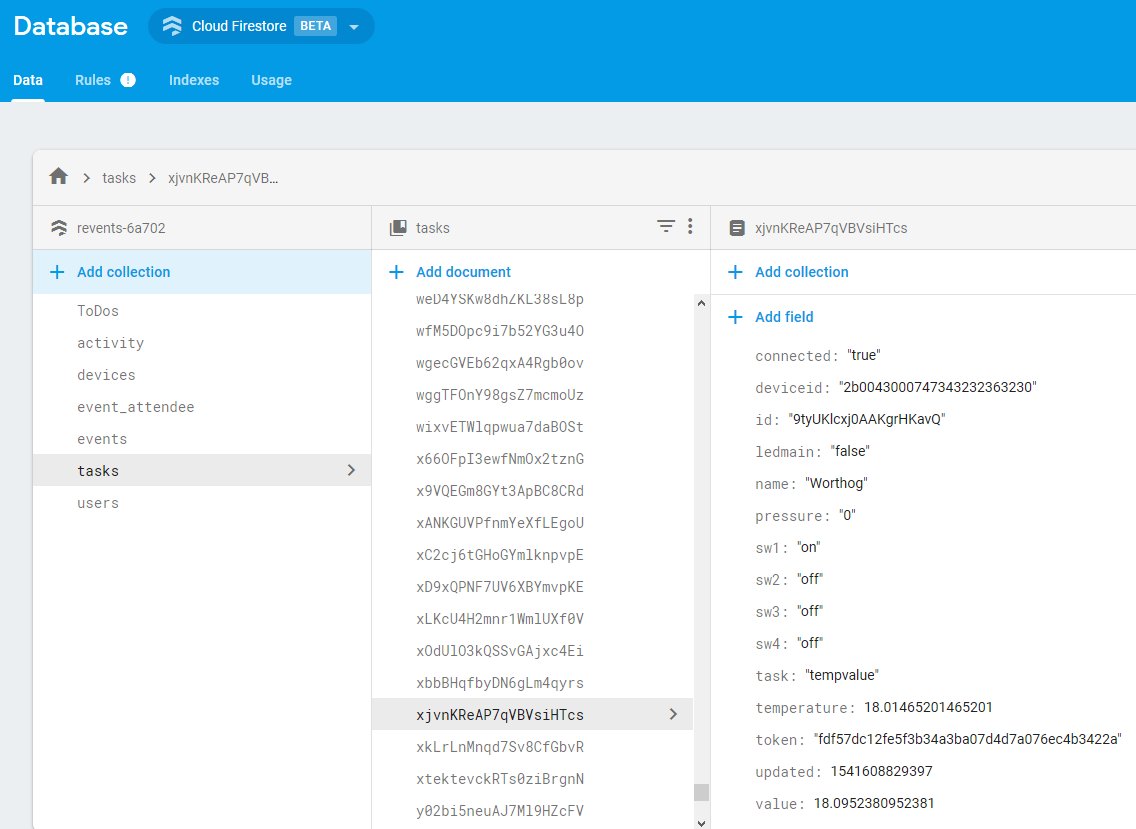
Try Reboot???

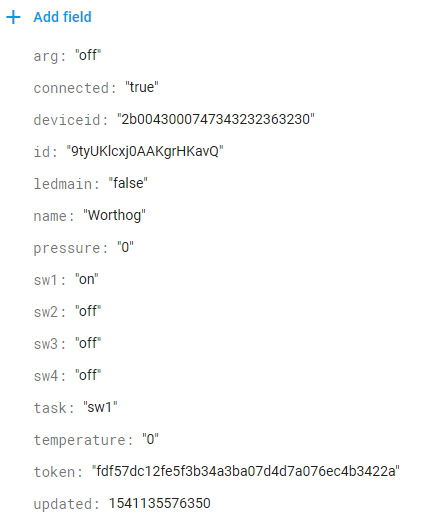
Seemed to work. ( Nov. 17, 2018) , not sure how long device has been running?

Was ok till I went into the Web Build IDE.

## Firestore : Webhook

Now we need the device to send data to Firestore (instead of Firebase) .



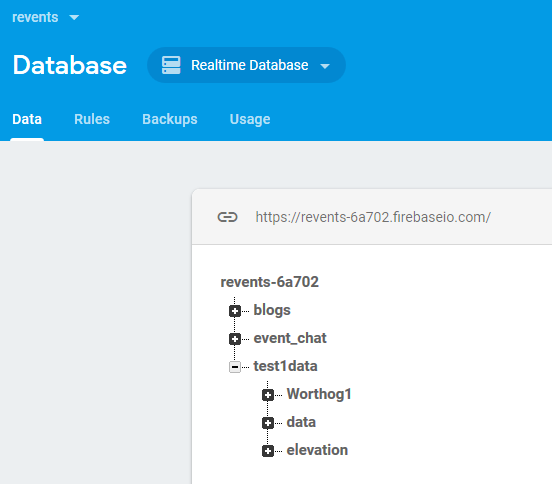


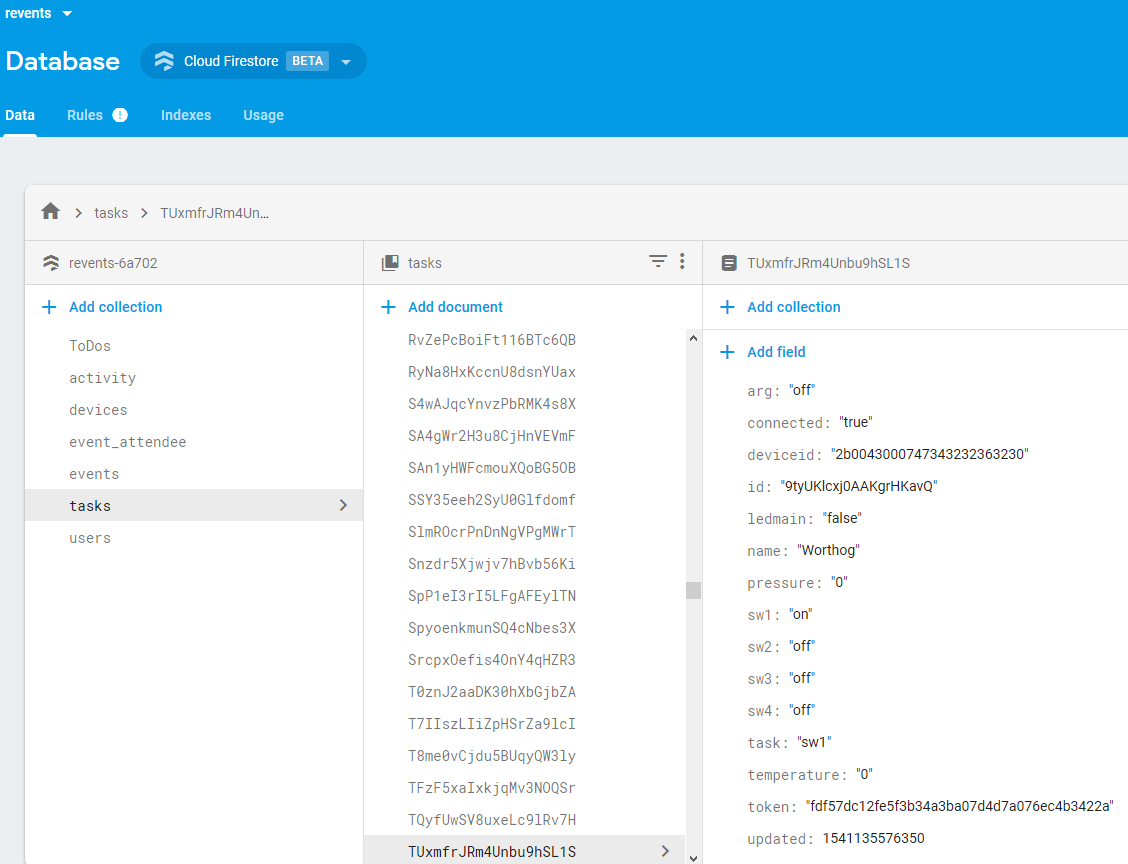
Should we just have a single Status Function that returns the Device Status and all data???

Task : : “status”

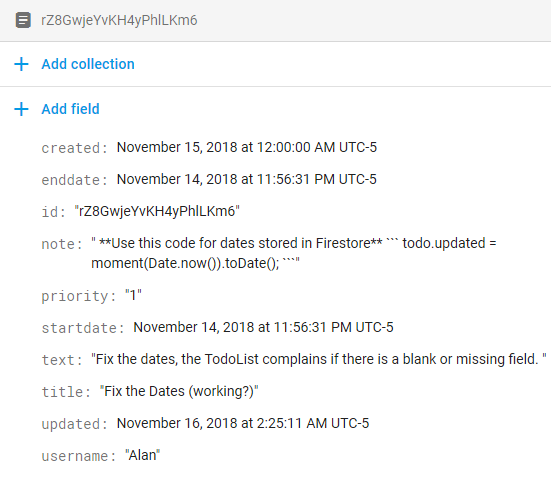
Can drop the Token field

Should we create a new collection????





See the ToDos version : For better date format



### Firestore Config

import firebase from 'firebase';

import 'firebase/firestore';

const firebaseConfig = {

apiKey: "AIzaSyCHt2P3aR35s7YLS4t-Lk6ewvXpPM-W8gI",

authDomain: "revents-6a702.firebaseapp.com",

databaseURL: "https://revents-6a702.firebaseio.com",

projectId: "revents-6a702",

storageBucket: "revents-6a702.appspot.com",

messagingSenderId: "503344706898"

}

firebase.initializeApp(firebaseConfig);

const firestore = firebase.firestore();

const settings = {

timestampsInSnapshots: true

}

firestore.settings(settings)

export default firebase;

## Firestore-api

export const createNewValue = ( device, task, value) => {

return {

...device,

task,

value: value,

updated: Date.now()

}

}

export const logValue = (device, task, value) => {

var db = firebase.firestore();

db.settings({

timestampsInSnapshots: true

});

let newValue = createNewValue(device, task, value) ;

console.log( {newValue}, "<-- the new Value object") ;

db.collection("tasks").add(newValue)

.then(function(docRef) {

console.log("Document written with ID: ", docRef.id);

})

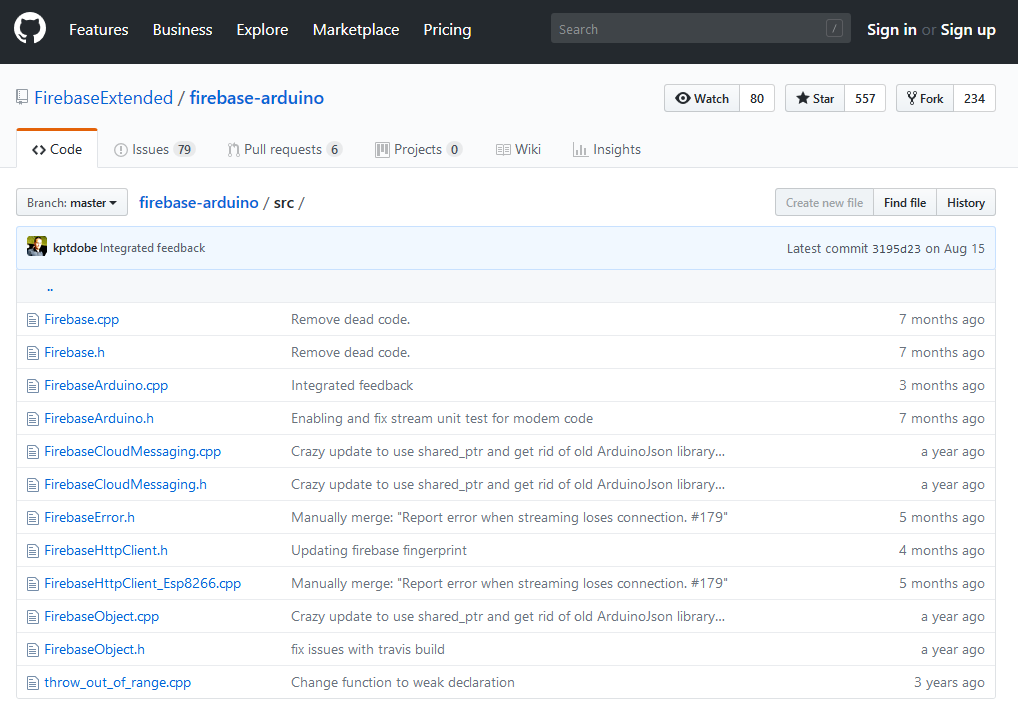
.catch(function(error) {

console.error("Error adding document: ", error);

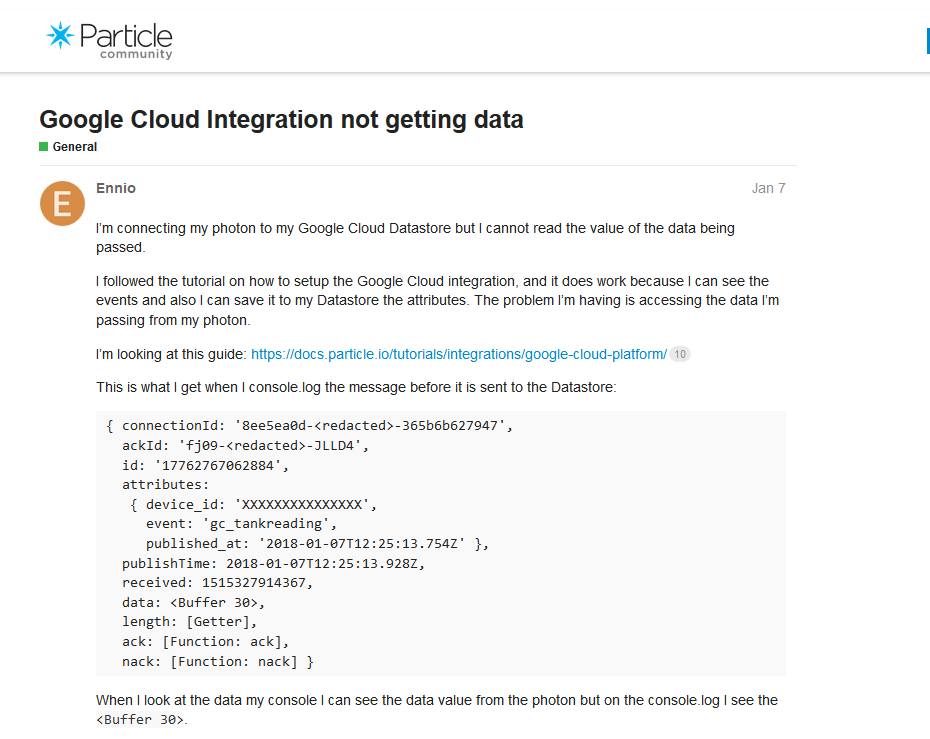
});

};

Maybe take a look at this (bookmark : <https://github.com/FirebaseExtended/firebase-arduino/tree/master/src>)



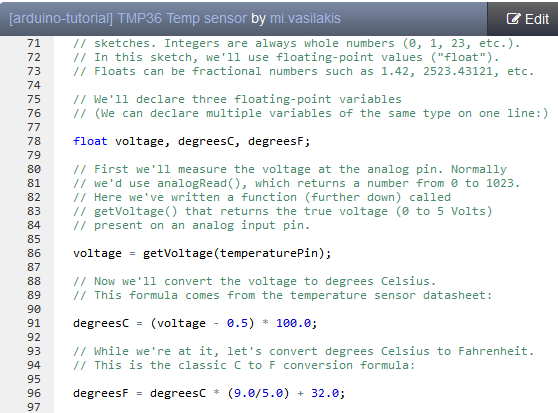
<https://community.particle.io/t/google-cloud-integration-not-getting-data/38553>

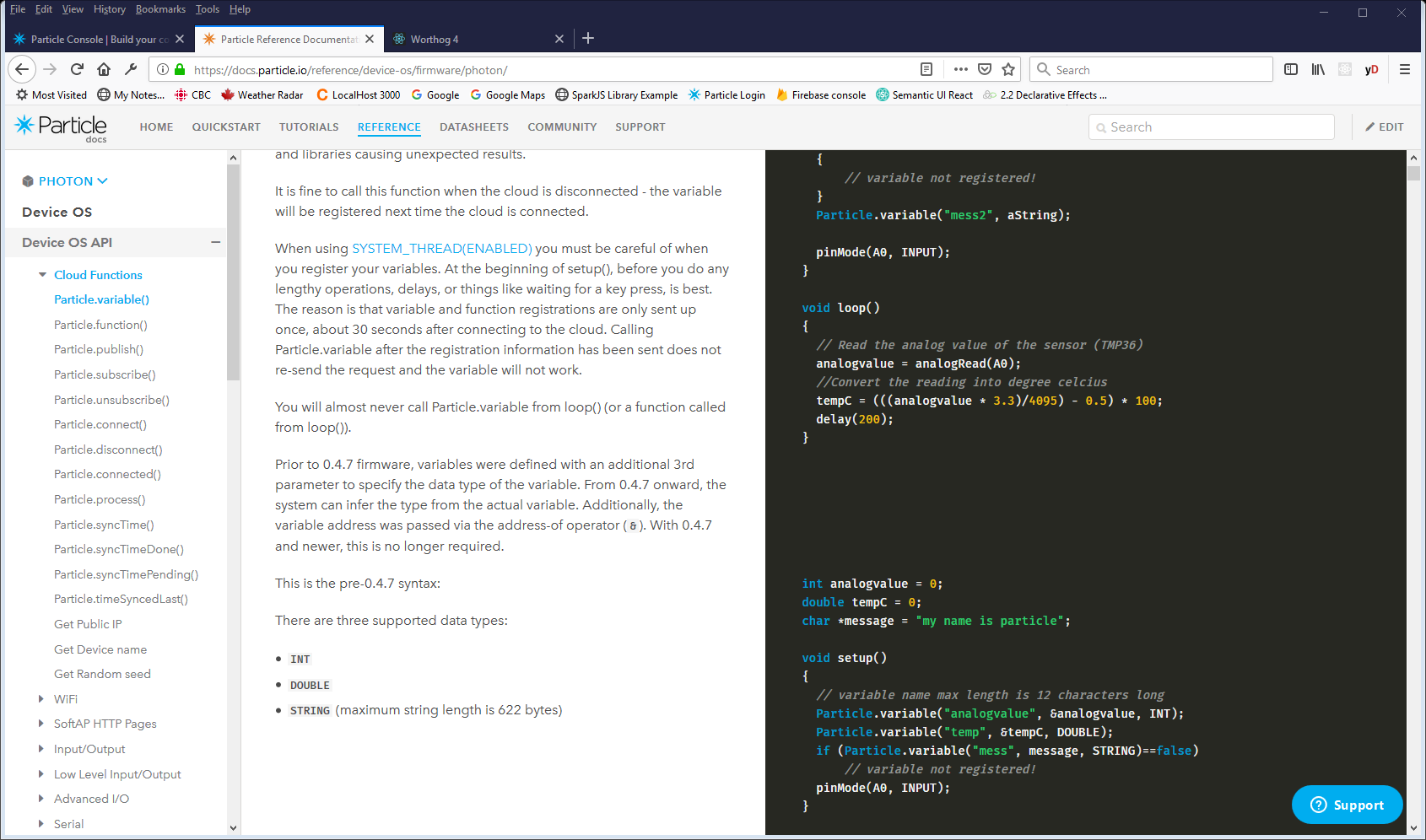


Review this tutorial

<https://docs.particle.io/reference/device-os/firmware/photon/>

Temp Sensor :





void loop()

{

// Read the analog value of the sensor (TMP36)

**analogvalue = analogRead(A0);**

//Convert the reading into degree celcius

tempC = (((analogvalue \* 3.3)/4095) - 0.5) \* 100;

delay(200);

}

## NOTE : WEBHOOKS

**Sending Complex Data**

Create a webhook that posts JSON data using several fields from the event.

WEBHOOK

{

"event": "Elevation",

"url": "https://example.com/123456789",

"json": {

"coordinates": {

"lat": "{{{lat}}}",

"lng": "{{{lng}}}"

}

},

"noDefaults": true

}

FIRMWARE

float lat = 39.73915360;

float lng = -104.98470340;

String data = String::format(

"{\"lat\":%f, \"lng\":%f}",

lat, lng);

Particle.publish("Elevation", data, PRIVATE);

REQUEST

POST /123456789 HTTP/1.1

User-Agent: ParticleBot/1.1 (https://docs.particle.io/webhooks)

host: example.com

accept: application/json

content-type: application/json

content-length: 55

{"coordinates":{"lng":"-104.984703","lat":"39.739155"}}

If you have to send the data as JSON numbers instead of JSON strings, the current workaround is to use the [body](https://docs.particle.io/reference/device-cloud/webhooks/#body) property.

WEBHOOK

{

"event": "Elevation",

"url": "https://example.com/123456789",

"headers": {

"content-type": "application/json"

},

**"body": "{ \"coordinates\": { \"lat\": {{{lat}}}, \"lng\": {{{lng}}} } }",**

"noDefaults": true

}

FIRMWARE

float lat = 39.73915360;

float lng = -104.98470340;

String data = String::format(

"{\"lat\":%f, \"lng\":%f}",

lat, lng);

Particle.publish("Elevation", data, PRIVATE);

REQUEST

POST /123456789 HTTP/1.1

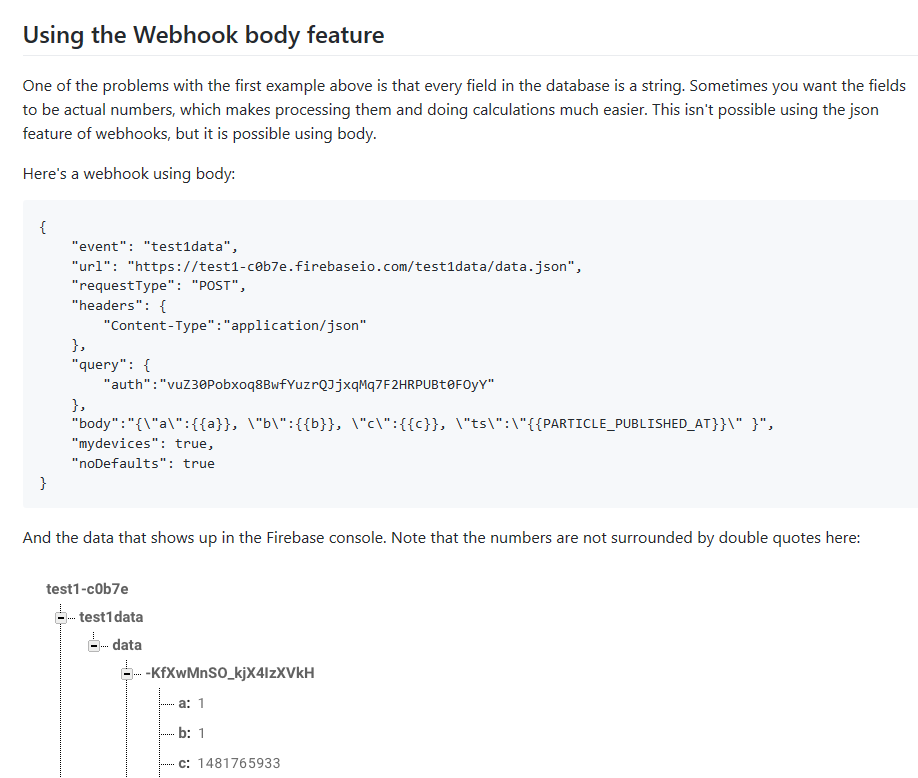
content-type: application/json

User-Agent: ParticleBot/1.1 (https://docs.particle.io/webhooks)

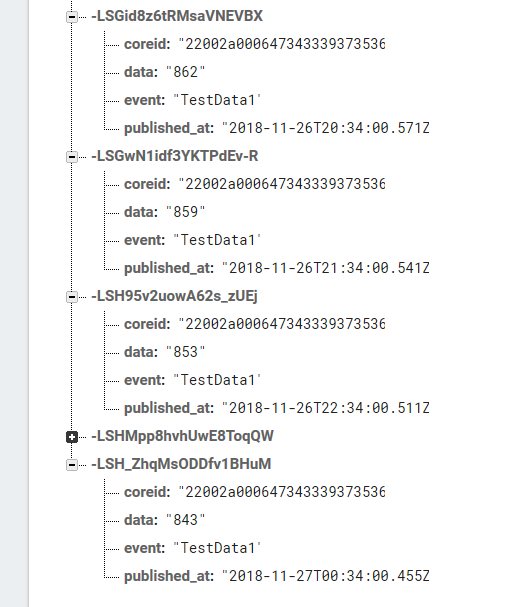
host: example.com

content-length: 59

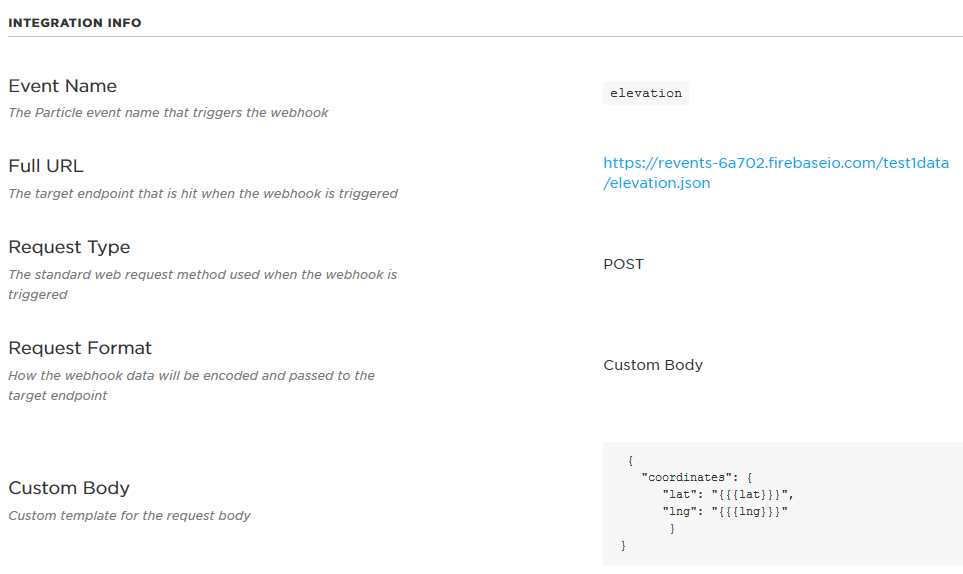
{ "coordinates": { "lat": 39.739155, "lng": -104.984703 } }



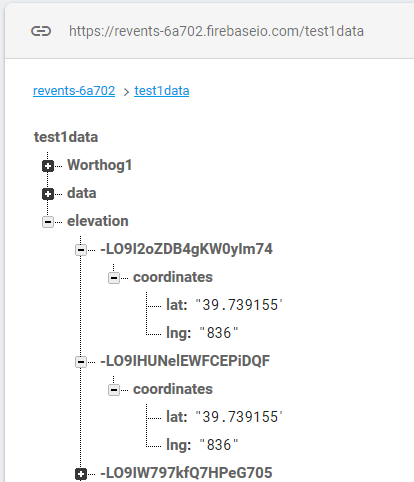
<https://github.com/rickkas7/particle-webhooks>



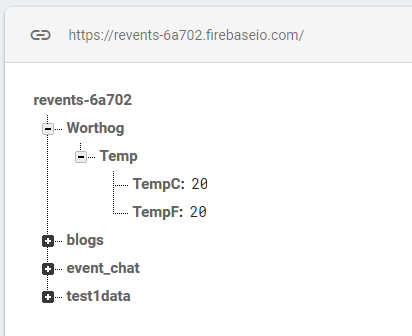
Data is comng through as a string :



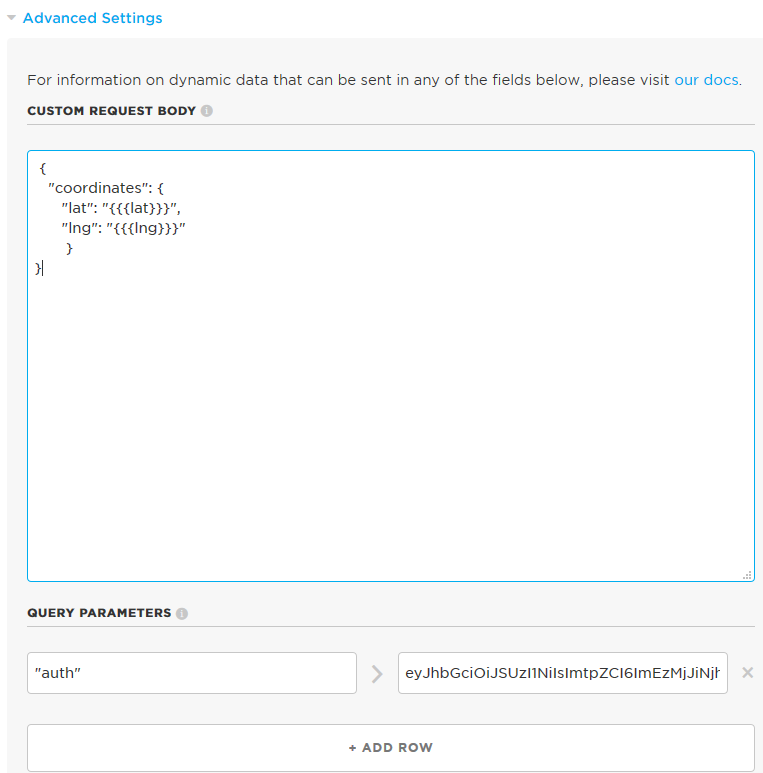
<https://revents-6a702.firebaseio.com/>



New table set up

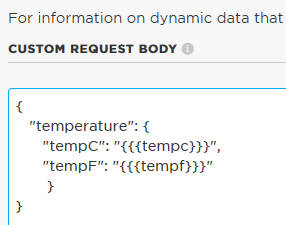


<https://revents-6a702.firebaseio.com/Worthog/Temp.json>



"auth"

eyJhbGciOiJSUzI1NiIsImtpZCI6ImEzMjJiNjhiY2U0MzExZTg2OTYzOTUzM2QzYTFhMjU1MWQ1ZTc0YzYiLCJ0eXAiOiJKV1QifQ.eyJpc3MiOiJodHRwczovL3NlY3VyZXRva2VuLmdvb2dsZS5jb20vcmV2ZW50cy02YTcwMiIsInBpY3R1cmUiOiJodHRwczovL2ZpcmViYXNlc3RvcmFnZS5nb29nbGVhcGlzLmNvbS92MC9iL3JldmVudHMtNmE3MDIuYXBwc3BvdC5jb20vby9XZ0lVMlE4Nmt6TzR5R3RwVzlmOFFLdHpPcUQzJTJGdXNlcl9pbWFnZXMlMkZjamx6ZHkyd2UwMDAwMjg2Mmp5cG4weWhyP2FsdD1tZWRpYSZ0b2tlbj1iNTQ5NmI0OC02M2M1LTQwMTItYTQwOS0zZjZlYWZlOWY1MDIiLCJhdWQiOiJyZXZlbnRzLTZhNzAyIiwiYXV0aF90…lZCI6ZmFsc2UsImZpcmViYXNlIjp7ImlkZW50aXRpZXMiOnsiZW1haWwiOlsiYWxhbkB3b3J0aG9nLmNhIl19LCJzaWduX2luX3Byb3ZpZGVyIjoicGFzc3dvcmQifX0.ac44PyoA-jKujF2iD3hCgjqM9IDBpHKgrW\_0-uBgpEEbKXBxXpSPhkEyPq3SZMrUvrYItKUl05uy5yPf8pjXIJpnCfo5SGnHnpFBUUOKoUWuOBgjcfDxL9zUUF2B2\_y5ab7aw\_pT0YXi4yrMU9advO2FJ0q5cyfxZhXfm7SEbB\_8i4t590kCGfSHmnQcUT4s3Ro1-JrdYvLtPDIK0rnWRDJ4lLgBF2-mouXlF50NSWk59-NiV6P4g0cU9Xd4ipTy7LDfbf80bjbKO4ZOtHNr7ixlVgpnWyuf9Bs7Kdns-HBAbSistTmcg00TLb34WeO7eqhZkBu30ZWNMauZ5754Ag



After the first test



void loop() {

// Get some data

String data = String(10);

// Trigger the integration

Particle.publish("temp", data, PRIVATE);

// Wait 60 seconds

delay(60000);

}

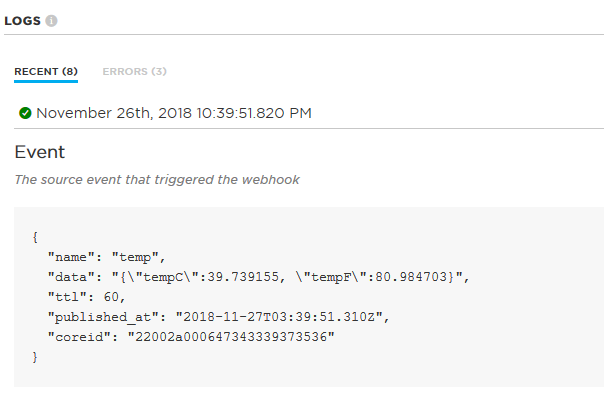
float lat = 39.73915360;

float lng = -104.98470340;

char data[256];

snprintf(data, sizeof(data), "{\"lat\":%f, \"lng\":%f}", lat, lng);

Particle.publish("getElevation", data, PRIVATE);



float tempc = 39.7391536;

float tempf = 80.984703;

char data[256];

snprintf(data, sizeof(data), "{\"tempC\":%f, \"tempF\":%f}", tempc, tempf);

Particle.publish("temp", data, PRIVATE);

// Wait 60 seconds

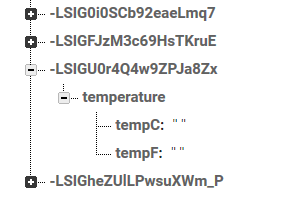
delay(60000);

// Wait 60 Minutes

// delay(3600000);

}

In Firebase the data is showing up BLANK ????



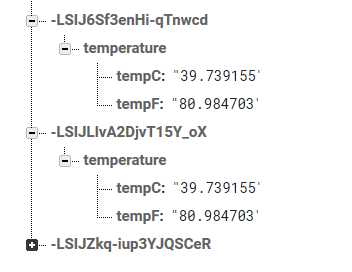
Changed the firmware (alll lowercase)

char data[256];

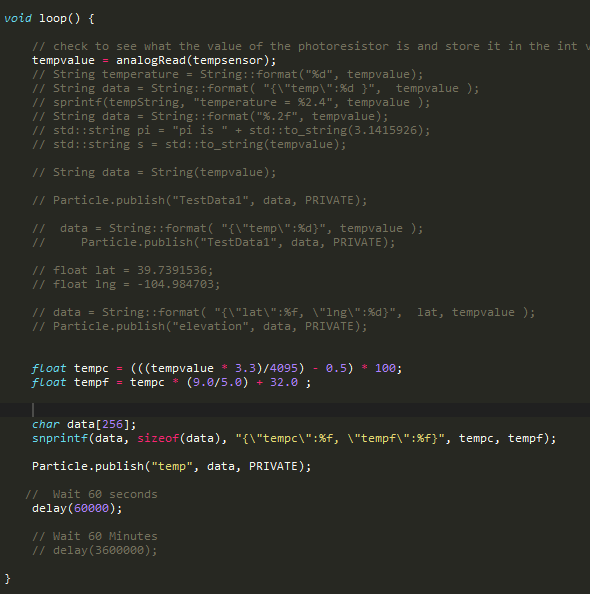
**snprintf(data, sizeof(data), "{\"tempc\":%f, \"tempf\":%f}", tempc, tempf);**

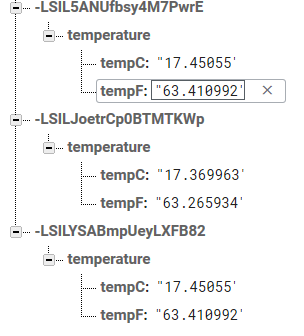
Particle.publish("temp", data, PRIVATE);

**Now it works.**

****

**Added the conversion to the firmware, now we are getting reeal value.**

****

****