Pushing Data to Google Docs

Story

In this article I will explain how your hardware can **push** data into a Google spreadsheet.

**Push Versus Poll**

In the **poll mechanism**, as described [in my previous article](https://www.hackster.io/gusgonnet/storing-data-in-google-714dde), the Google spreadsheet runs a script that sends a request to fetch data from our hardware at a regular interval.

You can use the poll mechanism when your hardware is online all the time, for instance to capture sensor data that changes slowly over time (example: the temperature of your pool).

In the **push mechanism**, described in the current article, your hardware sends a request with data to a Google server running a script that will, in turn, store that data received in a Google spreadsheet.

The push mechanism is ideal when your hardware might be sleeping from time to time (hence not reachable), to capture a specific event (example: your garage door is opening) or to store a log of what your hardware is doing.

**Note:** I used a Particle Photon in this project, but I think the mechanism can be helpful with other hardware in general, like Arduinos and Raspberry Pies.

**Explanation**

Like [PopQuiz explained nicely in this post](https://community.particle.io/t/pushing-data-from-the-photon-into-google-sheets/26108/23?u=gusgonnet), here's what you will need to make:

* A Google Sheet with labels at the top of each column where your data will go.
* A Google Java script which controls the behavior of the Sheet. The tutorial links to a page which shows how to do this. The script will be deployed as a web app which gets hit by your webhook.
* Particle firmware which publishes JSON strings like this:{variable name : valueother variable: value2}
* A webhook which hits your Google Web App, is web form type, with query parameters like this:{GoogleSheetLabel1: {{variable name}}GoogleSheetLabel2: {{other variable}}}

**Setup the Google Docs Side**

Please follow the instructions on [this site (pasted below in Word Doc).](http://railsrescue.com/blog/2015-05-28-step-by-step-setup-to-send-form-data-to-google-sheets/)In particular, follow ONLY these two sections:

* "The sheet"
* "The script"

**Setup your Hardware to Push Data - Particle's Case**

In the case you are using a Particle, you will need two things:

* configure a webhook
* code a publish command in your firmware to trigger that webhook with the wanted information

How this works:

**STEP 1: the webhook**

Create [a particle webhook](https://docs.particle.io/guide/tools-and-features/webhooks/) with the following information:

**event name:** googleDocs

**full url:** what you get from google docs (example: <https://script.google.com/macros/s/1236278936489127634-2876348/exec)>

**method:**POST

**form** **(one** **way** **to** **look** **at** **it):** key=name value={{my-name}}

**form in JSON** **(another** **way** **to** **look** **at** **it):**

{

"name": "{{my-name}}"

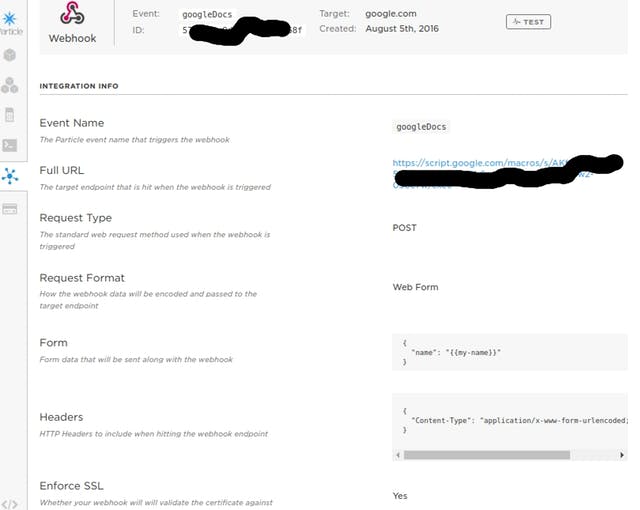
}

**headers:** "Content-Type": "application/x-www-form-urlencoded; charset=UTF-8"

**include defaults:** no

**Enforce SSL:** yes

Here's a screenshot of the webhook:



Webhook to google sheets

**STEP 2: the firmware**

Then in your firmware, add a line like this one:

String tempMessage = "Your garage door is opening";

Particle.publish("googleDocs", "{\"my-name\":\"" + tempMessage + "\"}", 60, PRIVATE);

Note: I'm using a dynamic custom field feature on webhooks that I learned [in this discussion](https://community.particle.io/t/more-events-in-one-webhook-solved/16516/10). You can read [a bit more in this tutorial](https://github.com/rickkas7/particle-webhooks#mustache) (search for mustache since the link seems not to work perfectly).

**STEP 3: verify the console logs**

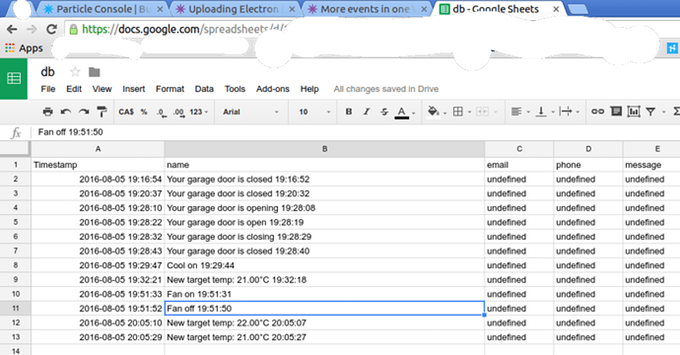
Every time your hardware triggers the webhook you should see something like this in your [Particle console logs](https://console.particle.io/logs):



The webhook getting triggered

**Results**

Here you can see how my hardware is filling up my Google spreadsheet:



My hardware is pushing data in Google Docs

Note: There are limits on how many times per day the Google API can be hit by your hardware. I'm pretty sure you would be able to pay Google for increasing your traffic quota, but I haven't looked into it.

**Improvements**

Ultimately, one would want to change the default name of headers for both the message body and the spreadsheet (name, email, phone and message) to something more meaningful. Once (and if) I get to this I will post the updates here.

In the meantime, you can start enjoying today the convenience of storing your data in Google Docs.

**Troubleshooting**

If things don't seem to work, you may check [some Particle community posts here](https://community.particle.io/t/pushing-data-from-the-photon-into-google-sheets/26108), specially [this one from awardblvr](https://community.particle.io/t/pushing-data-from-the-photon-into-google-sheets/26108/29?u=gusgonnet).

Feel free to post your questions below this write-up as well.

**Conclusion**

Hope you see value in this tip. Feel free to respect it and/or check [my other projects here.](https://www.hackster.io/gusgonnet/projects)

Gustavo.

**Need help?**

If you require professional help with your projects, don't hesitate to write me a line about your needs at gusgonnet@gmail.com. Thank you!

**The Sheet**

Navigate to drive.google.com and click on NEW > Google Sheets to create a new Sheet. Give it a name, perhaps “Form Google Sheets”. Put the following names into the first row of the first five columns:

Timestamp name email phone message

**The Script**

Click on Tools > Script Editor…, which should open a new window and a dialog called 'Google Apps Script’. Click on Create script for > Custom Functions in Sheets. This will create one script called 'Code.gs’ containing functions such as SAY\_HELLO.

Click on 'Untitled Project’ at the top and give this project a name: 'Form Script’.

Highlight all of this script (we are going to replace it) and paste in the following:

// 1. Enter sheet name where data is to be written below

var SHEET\_NAME = "Sheet1";

// 2. Run > setup

//

// 3. Publish > Deploy as web app

// - enter Project Version name and click 'Save New Version'

// - set security level and enable service (most likely execute as 'me' and access 'anyone, even anonymously)

//

// 4. Copy the 'Current web app URL' and post this in your form/script action

//

// 5. Insert column names on your destination sheet matching the parameter names of the data you are passing in (exactly matching case)

var SCRIPT\_PROP = PropertiesService.getScriptProperties(); // new property service

// If you don't want to expose either GET or POST methods you can comment out the appropriate function

function doGet(e){

return handleResponse(e);

}

function doPost(e){

return handleResponse(e);

}

function handleResponse(e) {

// shortly after my original solution Google announced the LockService[1]

// this prevents concurrent access overwritting data

// [1] http://googleappsdeveloper.blogspot.co.uk/2011/10/concurrency-and-google-apps-script.html

// we want a public lock, one that locks for all invocations

var lock = LockService.getPublicLock();

lock.waitLock(30000); // wait 30 seconds before conceding defeat.

try {

// next set where we write the data - you could write to multiple/alternate destinations

var doc = SpreadsheetApp.openById(SCRIPT\_PROP.getProperty("key"));

var sheet = doc.getSheetByName(SHEET\_NAME);

// we'll assume header is in row 1 but you can override with header\_row in GET/POST data

var headRow = e.parameter.header\_row || 1;

var headers = sheet.getRange(1, 1, 1, sheet.getLastColumn()).getValues()[0];

var nextRow = sheet.getLastRow()+1; // get next row

var row = [];

// loop through the header columns

for (i in headers){

if (headers[i] == "Timestamp"){ // special case if you include a 'Timestamp' column

row.push(new Date());

} else { // else use header name to get data

row.push(e.parameter[headers[i]]);

}

}

// more efficient to set values as [][] array than individually

sheet.getRange(nextRow, 1, 1, row.length).setValues([row]);

// return json success results

return ContentService

.createTextOutput(JSON.stringify({"result":"success", "row": nextRow}))

.setMimeType(ContentService.MimeType.JSON);

} catch(e){

// if error return this

return ContentService

.createTextOutput(JSON.stringify({"result":"error", "error": e}))

.setMimeType(ContentService.MimeType.JSON);

} finally { //release lock

lock.releaseLock();

}

}

function setup() {

var doc = SpreadsheetApp.getActiveSpreadsheet();

SCRIPT\_PROP.setProperty("key", doc.getId());

}

Click on the Save icon. Set the dropdown in the nav bar to 'setup’ and click on the right-pointing triangle to its left to run this function. It should show 'Running function setup’ and then put up a dialog 'Authorization Required’. Click on Continue. In the next dialog 'Request for permission - Formscript would like to’ click on Accept.

In the menus click on File > Manage Versions… We must save a version of the script for it to be called. In the box labeled 'Describe what has changed’ type 'Initial version’ and click on 'Save New Version’, then on 'OK’.

Back to the menus: click on Resources > Current Project’s triggers. In this dialog click on 'No triggers set up. Click here to add one now’. In the dropdowns select 'doPost’, 'From spreadsheet’, and 'On form submit’, then click on 'Save’.

Back to the menus: click on Publish > Deploy as web app… For 'Who has access to the app:’ select 'Anyone, even anonymous’. Leave 'Execute the app as:’ set to 'Me’ and Project Version to '1’. Click the 'Deploy’ button.

A dialog should appear announcing 'This project is now deployed as a web app’. Copy the Current web app URL from the dialog; it should look something like:

https://script.google.com/macros/s/AKfycbw6RTOxn5OT\_BIw9Nl\_3KoFSXEQEbiKSZCLyombb1YqkGfRKUSz/exec

Click OK.