Well System Monitor

Project and Technical Overview.

Version 1.1; date: 10/16/22

Table of Contents

1. Introduction 4

2. Project Overview 4

3. Technical Overview 5

4. Hardware and Firmware Overview 9

5. Overview of Webhooks and Google Apps Scripts 10

Files 10

Webhooks 11

Scripts 11

Log data to spreadsheet 12

Send an SMS message 14

6. Overview of Logging Spreadsheet 15

Tab 1: Sheet 1 - Raw Data 15

Tab 2: Graphs 16

7. SMS Text Alerts. 17

Sample Alerts 17

Appendix 1 - Software/Firmware Installation. 18

Build the Hardware. 18

Installing the Particle Firmware. 18

Creating the Webhooks. 19

Creating/Deploying Google Apps Scripts. 20

Appendix 2 - Creating the Google Logging Spreadsheet. 25

Set up 25

Tab 3: Historical Data 26

Column J: PP Total When WP Goes Off 26

Column K: PP time for graph 26

Column L: WP last run 26

Column M: days since last WP run 27

Column N: water used 27

Columns O:Q 27

Sample Calculation Columns 27

The Historical Graphs 28

Tab 4: Water Usage 29

1. Introduction

This document provides a complete description of the Well System Monitor (WSM) project.

- Section 2 contains an overview of the project, including a description of the target well system, the need for monitoring the well system, and the requirements for data logging and alerting.
- Section 3 contains a technical overview of the project, including the project's hardware and software components and the flow of information between these components.
- Section 4 contains a more detailed technical description of the hardware and firmware of the Well System Monitor.
- Section 5 provides a technical description of the webhooks and Google Apps Scripts that interface the hardware with the logging spreadsheet and alert system.
- Section 6 provides a description of the logging Google spreadsheet itself, including on-sheet calculations and graphing.
- Section 7 describes how SMS text alerts are generated.
- Finally, the Appendices contain installation and configuration information for webhooks, scripts and the Google cloud spreadsheet.

2. Project Overview

The purpose of this project is to monitor the operation of a water well system and to alert the system owner of problems with the system before they cause system damage. The well system consists of the following components:

Well Pump: the Well Pump is buried and is very expensive to replace. The pump is used to pump water out of the well and into a reservoir. The Well Pump should run infrequently to extend its life. Additionally, the well pump running time should be limited so that the water level in the well does not drop below the level of the pump, decreasing its lifespan.

Reservoir: a large reservoir holds water from the well, in order to limit the pumping on/off cycles of the Well Pump. A float sensor in the reservoir determines when the well pump should turn on and off.

Pressure Tank: a large pressure tank holds water, under pressure, for use within the premises.

Pressure Pump: the Pressure Pump recharges the pressure tank using water from the reservoir.

The Well Pump and the Pressure Pump are run by a control system whose inputs are the reservoir **float sensor** and the Pressure Tank **pressure sensor**. A problem with either of these sensors, or with the Pressure Tank bladder, or other component of the system, can cause the Pressure Pump and/or the Well Pump to run either too long or too frequently or not at all. Long term improper running of these pumps can cause them to fail prematurely, resulting in a very expensive repair. It is much more cost effective to proactively repair the cause of improper pump operation rather than wait for a pump to fail.

The current control system does not provide the owner with information that can alert him/her when the system is running outside of normal operating parameters. This project adds in logging of Well Pump and Pressure Pump activity (to a cloud based spreadsheet) and an alert system that sends SMS texts to the well owner whenever an out-of-spec operation occurs. Out of spec operations that cause alerts are:

- Pressure Pump runs too long.
- Pressure Pump runs too short.
- Well Pump runs too long.
- Well Pump runs too short.
- Well Pump does not come on after accumulated Pressure Pump times reach a threshold.
- Well Pump comes on too soon (not enough previous Pressure Pump time).
- Pressure Pump does not come on at all within an expected period of time (e.g. one day).

The SMS text alerts generated whenever any of these alert events occurs should prompt the well system owner to consult the on-line spreadsheet log of historical pump events. One year of data collection and evaluation has allowed us to develop criteria on the trends of the logged data that are indicative of various system failure conditions that should be repaired in order to preserve the life of the expensive pumps. If this monitor is deployed to another well water system the alert criteria may need to be adjusted to suit that system.

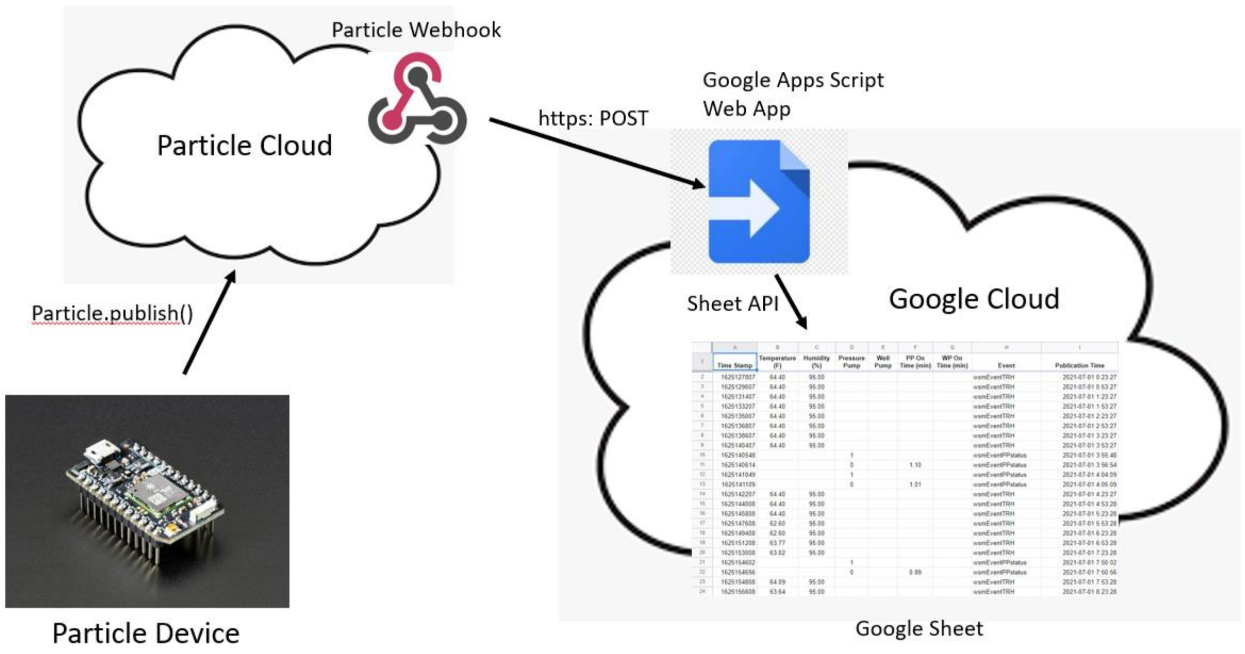
3. Technical Overview

The Well System Monitor uses a microcontroller-based system to monitor the state of the well control system.   A year's worth of data acquisition and analysis has provided us with a detailed understanding of what is the normal range of operation of these pumps, and what abnormalities in pump operation mean. For example, if the pressure pump runs for an abnormally short amount of time, this usually means that there is a leak in the bladder of the pressure tank and the bladder needs to be re-inflated or else the pressure tank needs to be replaced. If the well pump runs for an abnormally long amount of time, this either means that the fill sensor on the reservoir has malfunctioned or else the well pump has failed or the well water level is below the well pump or perhaps there is a leak in the above ground system. In all such abnormal cases, the problem must be investigated and repaired or else one or both of the expensive pumps will exhibit undo wear leading to premature failure.

The hardware used on this project is repurposed from our Water Leak Sensor project (https://github.com/TeamPracticalProjects/WaterLeakSensor). The hardware uses a Particle (https://www.particle.io/) Photon microcontroller with built-in WiFi connectivity. Two general purpose digital inputs are wired to spare poles on the well control system relays to monitor activation and deactivation of the Pressure Pump and the Well Pump. The hardware also contains an inexpensive DHT-11 temperature and humidity sensor to log ambient weather conditions at the well site. Ambient temperature readings, in particular, aid in the analysis of overall water usage information that can be computed from the Pressure Pump running times. The "WSM Build Manual" that is included in this folder provides detailed information about the installation and wiring of the hardware to the well control system relays.

The firmware that runs on the Particle Photon monitors activation and deactivation of the well pump and of the pressure pump as well as ambient temperature and humidity. The firmware debounces the relay contacts and computes pump run times during pump activations. The pump activations and pump run times are published to the Particle Cloud as they occur. The ambient temperature and humidity readings are published to the Particle cloud once every ½ hour. Various alert conditions (abnormal pump operations) are computed in the firmware and are published to the Particle Cloud when they occur. In addition, the current status of the system monitoring can be interrogated at any time using the Particle cloud API. An android app is provided for this purpose (https://github.com/TeamPracticalProjects/WellSystemMonitor/tree/master/TestApp).

Figure 3-1 shows how data that is collected and computed by the hardware/firmware gets logged to a Google Sheet cloud spreadsheet. The spreadsheet is cloud-accessible and serves as a system event log and analysis tool.



	Time Stamp	Temperature (F)	Humidity (%)	Pressure Pump	Well Pump	PP On Time (min)	WP On Time (min)	Event	Publication Time
2	1625127807	64.40	95.00					wsmEventTRH	2021-07-01 0:23:27
3	1625129607	64.40	95.00					wsmEventTRH	2021-07-01 0:53:27
4	1625131407	64.40	95.00					wsmEventTRH	2021-07-01 1:23:27
5	1625133207	64.40	95.00					wsmEventTRH	2021-07-01 1:53:27
6	1625135007	64.40	95.00					wsmEventTRH	2021-07-01 2:23:27
7	1625136807	64.40	95.00					wsmEventTRH	2021-07-01 2:53:27
8	1625138607	64.40	95.00					wsmEventTRH	2021-07-01 3:23:27
9	1625140407	64.40	95.00					wsmEventTRH	2021-07-01 3:53:27
10	1625140546			1				wsmEventPPstatus	2021-07-01 3:55:46
11	1625140614			0		1.10		wsmEventPPstatus	2021-07-01 3:56:54
12	1625141049			1				wsmEventPPstatus	2021-07-01 4:04:09
13	1625141109			0		1.01		wsmEventPPstatus	2021-07-01 4:05:09
14	1625142207	64.40	95.00					wsmEventTRH	2021-07-01 4:23:27
15	1625144008	64.40	95.00					wsmEventTRH	2021-07-01 4:53:28
16	1625145808	64.40	95.00					wsmEventTRH	2021-07-01 5:23:28
17	1625147608	62.60	95.00					wsmEventTRH	2021-07-01 5:53:28
18	1625149408	62.60	95.00					wsmEventTRH	2021-07-01 6:23:28
19	1625151208	63.77	95.00					wsmEventTRH	2021-07-01 6:53:28
20	1625153008	63.02	95.00					wsmEventTRH	2021-07-01 7:23:28
21	1625154602			1				wsmEventPPstatus	2021-07-01 7:50:02
22	1625154656			0		0.89		wsmEventPPstatus	2021-07-01 7:50:56
23	1625154808	64.09	95.00					wsmEventTRH	2021-07-01 7:53:28
24	1625156608	63.64	95.00					wsmEventTRH	2021-07-01 8:23:28

Figure 3-1. Data Logging to a Cloud Spreadsheet.

Data is published to the Particle cloud using the Particle.publish() firmware primitive. The event name for this publication is “wsmEventxxx”, where “xxx” is an event extension that describes what event is being published. The data that is published with each such wsmEventxxx is a json formatted string that contains the following information:

- temperature and humidity (event: wsmEventTRH)
- pressure pump activation or deactivation with run time (event: wsmEventPPstatus)
- well pump activation or deactivation with run time (event: wsmEventWPstatus)

A Particle webhook (https://docs.particle.io/reference/cloud-apis/webhooks/) listens for publications that begin with “wsmEvent” and POSTs the event information to a Google Apps Script that is deployed as a web app. The Google Apps Script decodes the full event name and the json formatted data and logs this information to the Google cloud spreadsheet. Section 5 of this document describes the webhook and Google Apps Script in more detail.

Figure 3-2 shows how SMS text alerts are generated. We chose SMS texts as the means to alert the user about abnormal Well System conditions because they are fast, reliable, and work on any type of mobile phone device.

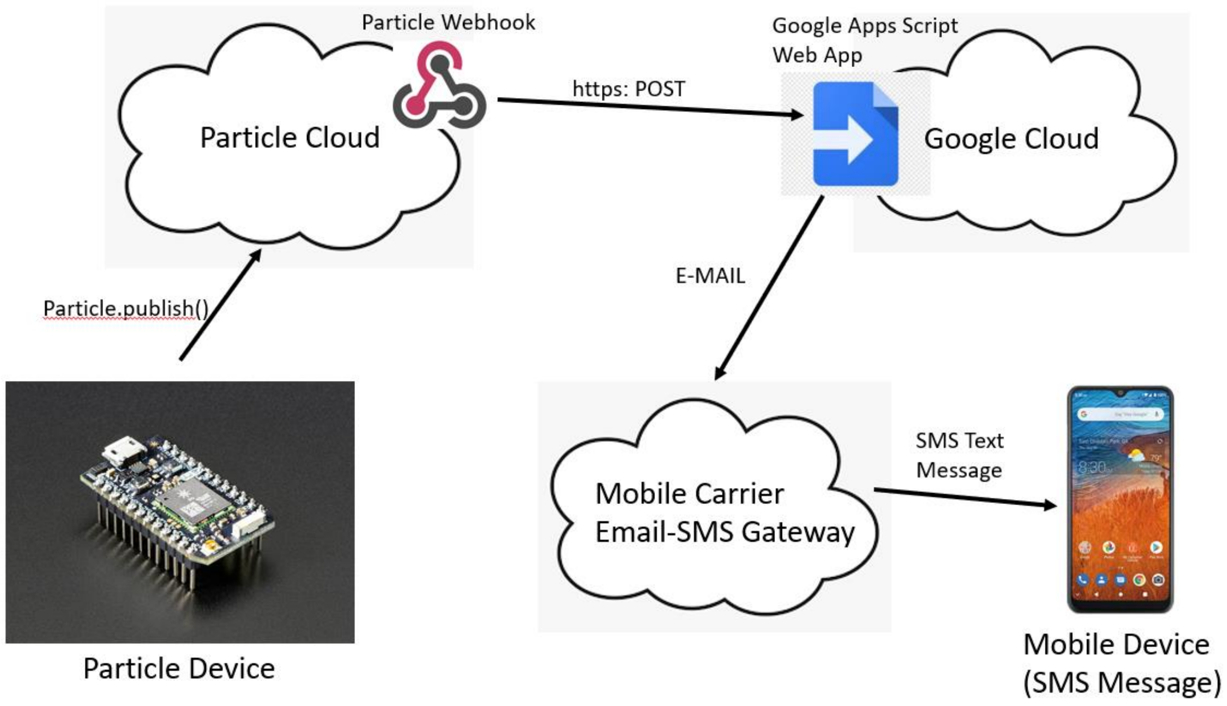


Figure 3-2. SMS Text Alert Generation.

Firmware running on the Particle Photon tracks the operation of the Well Pump and the Pressure Pump and detects 7 specific conditions that alert the user to a potential problem with the Well System. We determined these specific alert conditions during our year-long data acquisition and analysis phase of this project. These seven alert conditions are enumerated in Section 2 of this document.

Alerts are published to the Particle cloud using the Particle.publish() firmware primitive. The event name for this publication is “wsmAlertxxx”, where “xxx” is an event extension that describes what alert is being published. The data that is published with each such wsmAlertxxx is a text string to be presented as the body of the SMS text message.

A Particle webhook (https://docs.particle.io/reference/cloud-apis/webhooks/) listens for publications that begin with “wsmAlert” and POSTs the alert information to a Google Apps Script that is deployed as a web app. The Google Apps Script decodes the full event name and uses this full event name to create a “subject” string for the SMS text message. The Google Apps Script sends the subject and body strings as an e-mail to the SMS gateway of the user’s cellular carrier. This action results in an SMS text being sent to the user’s mobile phone. Section 5 of this document describes the webhook and Google Apps Script in more detail.

4. Hardware and Firmware Overview

The hardware used on this project is repurposed from our Water Leak Sensor project (https://github.com/TeamPracticalProjects/WaterLeakSensor). The hardware uses a Particle (https://www.particle.io/) Photon microcontroller with built-in WiFi connectivity. Two general purpose digital inputs are wired to spare poles on the well control system relays to monitor activation and deactivation of the Pressure Pump and the Well Pump. The hardware also contains an inexpensive DHT-11 temperature and humidity sensor to log ambient weather conditions at the well site. Ambient temperature readings, in particular, aid in the analysis of overall water usage information that can be computed from the Pressure Pump running times. The "WSM Build Manual" that is included in this folder provides detailed information about the installation and wiring of the hardware to the well control system relays. See: https://github.com/TeamPracticalProjects/WellSystemMonitor/blob/master/Documentation/WSM%20Build%20Manual.pdf

Note that the Well System Monitor requires a WiFi connection to the Internet. In this particular installation, the well control system is housed in the "pump house" that is remote from the main house where WiFi is present. We elected to install the hardware in the pump house where we could connect the hardware with spare poles on the pump control relays with only a few feet of cable. The digital inputs to the Photon from the relay contacts use internal (to the Photon) pullup resistors that are relatively weak and therefore susceptible to pickup noise over long cables. The debouncing logic in the firmware helps to filter out spurious noise; however, it is advisable to keep the cabling relatively short. The pump house is far enough from the main house where the WiFi access point is located and the WiFi signal is quite weak. We solved this issue by adding an external 2.4 GHz antenna to the Photon, rather than using its relatively weak internal patch antenna. The firmware allows the Photon to select whichever antenna gives the strongest signal.

The firmware that runs on the Particle Photon monitors activation and deactivation of the well pump and of the pressure pump as well as ambient temperature and humidity. The pump activations and pump run times are published to the Particle Cloud as they occur. The ambient temperature and humidity readings are published to the Particle cloud once every ½ hour. Various alert conditions (abnormal pump operations) are computed in the firmware and are published to the Particle Cloud when they occur. In addition, the current status of the system monitoring can be interrogated at any time using the Particle cloud

API. An android app is provided for this purpose (https://github.com/TeamPracticalProjects/WellSystemMonitor/tree/master/TestApp).

Source code for the firmware that runs on the Photon can be found in this repository at: https://github.com/TeamPracticalProjects/WellSystemMonitor/tree/master/Firmware/WellSystemMonitor/src

The firmware contains the following files (in `/src`):

- WellSystemMonitor.ino: the main code for the project firmware. The main loop is non-blocking. It samples the Well Pump contact status (Photon pin A0) and the pressure pump status (Photon pin A1) and debounces and filters these samples for reliable readings. It also reads the DHT11 temperature and humidity values using a non-blocking library and computes when to publish these, as well as timing intervals for alerts and alert holdoffs. It computes time string values for publication; however these do not automatically adjust for daylight savings time, and the logged times are computed (with DST applied) in the Google Apps Scripts. We also left in some legacy Water Leak Sensor code (to move a servo display, etc.). This code is not used and can be deleted.
- WSMGlobles (.h and .cpp): Defines globals used in this project.
- TPPUtils (.h and .cpp): Some utilities that we reuse frequently in our projects.
- WSMAlertProcessor (.h and .cpp): Supplies a class that processes the alerts.

We also use a third party library called "PietteTech_DHT" which is included in the `/lib/PietteTech_DHT` folder. This is a non-blocking library for the DHT-11 (and similar) temperature and humidity sensor.

5. Overview of Webhooks and Google Apps Scripts

Files

The Particle Webhooks and Google Apps Scripts that are part of this project are contained in the folder called "Google Apps Scripts" in this repository (https://github.com/TeamPracticalProjects/WellSystemMonitor/tree/master/GoogleAppsScripts). The files are:

  - WSM Event Webhook.png: a picture of the webhook configuration in the Particle Console.

    - WSM Alert Webhook.png: a picture of the webhook configuration in the Particle Console.
    - wsmWriteData.txt: the script source code for writing logging data to the Google sheet spreadsheet.
    - WSM_Send_Alert.txt: the script source code for sending sms text alerts.
    - wsmAlertSantaRosaAlerts.txt: NOT USED. This was script code that was used during development only and is not needed for the installed project.

Webhooks

The webhooks are very simple. They can be configured directly from the "integrations" tab in the Particle Console when logged into the Particle account where the Photon device used on the project is claimed. No coding is necessary – see the screenshots for details.

Each webhook has a name and the naming is VERY IMPORTANT. The "Event Name" for the webhook that sends logging data to the logging script must be "`wsmEvent`". Likewise, the name of the webhook for sending alert data for sms alerts is "`wsmAlert`". The names correspond to the first part of the event names in the Particle.publish() calls in the Photon firmware, and they must match exactly!

The "Full URL" for each webhook is the url of the corresponding Google Apps Script. When a Google Apps Script is deployed, Google provides this url, which you copy and then paste into this field in the webhook editor in the Particle Console. The url always ends with "`/exec`". Each time you edit the script and deploy the new version, you get a different url, so be sure to always copy/paste the latest url into the webhook.

The webhook "Request Type" is always "POST". The webhook "Request Format" is always "Web Form". The default "Form" and "Headers" are used. The default form publishes the event name, the event data, the particle device ID, and the publication time. The Google Apps Scripts use the event name and event data. Finally, "Enforce SSL" should be "Yes", for security reasons.

Scripts

Google Apps Scripts that are deployed as "Web Apps" are required to have at least one of the following functions:

- doGet(e)
- doPost(e)

We generally provide both functions, although we only deploy the POST method. Having the doGet() function allows for testing the script using a web browser, which might come in handy. Each of these mandatory functions takes an argument that is the name of an "event object" that contains all of the relevant information about the GET or POST event that triggered the function. This object is used to obtain the event information that the webhook provides.

Log data to spreadsheet

The Google Apps Script for writing logging data to a Google spreadsheet is in the file called "`wsmWriteData.txt`". You can copy this text and paste it into the Google script editor. The following is a description of the source code.

One of the functions doGet() and doPost() are required. We provide both, as noted above. The code for these functions is identical:

```
function doGet(e) {
        var ss = SpreadsheetApp.openByUrl("https://docs.google.com/spreadsheets/d/<url
        of Google spreadsheet>/edit#gid=0");

        var sheet = ss.getSheetByName("Sheet1");
        addUser(e,sheet);
}

function doPost(e) {
        var ss = SpreadsheetApp.openByUrl("https://docs.google.com/spreadsheets/d/<url
        of Google spreadsheet>/edit#gid=0");

         var sheet = ss.getSheetByName("Sheet1");
         addUser(e, sheet);

}
```

The variable "ss" gets a reference to the Google spreadsheet using the spreadsheet's url. The latter is available when the spreadsheet is created. The variable "sheet" gets a reference to the active sheet in the spreadsheet ("sheet 1" by default when the spreadsheet is created). Both doGet() and doPost() then call the function "addUser()" which performs the main function of the script:

```
function addUser(e, sheet) {
	var edata = e.parameter.data;
	var wsmData = JSON.parse(edata);
	var ev = e.parameter.event;
	 var pub = e.parameter.published_at;

	var time = wsmData.etime ;
	var temp = wsmData.temp ;
	var rh = wsmData.rh ;
	var pp = wsmData.pp ;
	var wp = wsmData.wp ;
	var ptm = wsmData.ppon ;
	var wtm = wsmData.wpon ;
	var loctm = wsmData.loctime ;  // produced by the Photon; does not convert for DST
	var tzAdjustedTime = computeLocalTime(time);  // computed actual time string
from photon unix time
	…
	sheet.appendRow([time,temp,rh,pp,wp,ptm,wtm,ev,tzAdjustedTime]);

	cleanUpSheet(sheet);  // keep the number of rows within bounds by deleting the
oldest entries
}
```

The variable "edata" receives the url encoded data that was sent in the GET or POST and recorded in the "e" event object. The url encoded data includes the data (a json encoded string called "data"), the event name "event" and the publication time "published_at", which are extracted into their respective variables (the webhook also publishes the "coreid" but we don't use this). The json string that is created in the Photon firmware is parsed into a json object ("wsmData") from which the individual data items are extracted:

- "temp": the temperature (from the DHT11)
- "rh": the relative humidity (from the DHT11)
- "pp": the pressure pump on or off status
- "wp": the well pump on or off status
- "ppon": the pressure pump on (run) time, if turned off
- "wpon": the well pump on (run) time, if turned off
- "loctm": The time of the event publication by the Photon (unix time, utc).

The time of the event publication by the Photon is then converted to the timezone and daylight savings time conditions of the local time at the well system using the function computeLocalTime(). This required logging data is then appended to the last row of the Google spreadsheet via an array of the data items arranged in the spreadsheet's column order ("sheet.apprendRow()"). Finally, the size of the spreadsheet is kept within 2000 rows by the function "cleanUpSheet()".

The function cleanUpSheet() deletes 200 of the earliest rows on the Google spreadsheet if the number of rows is equal to or greater than 2000. The function "computeLocalTime()" uses a Google utility to format a unix time variable into the local time for the specified timezone ("Americas/Los_Angeles", in our project).

Send an SMS message

The Google Apps Script for sending sms text alerts begins with the necessary doGet(e) and doPost(e) functions. Both of these functions simply call the function "sendAlert(e)".

```
function doGet(e) {
        sendAlert(e);
}

function doPost(e) {
        sendAlert(e);
}

function sendAlert(e) {
         const targetBob = "<cellular sms gateway email address goes here>";
         //const targetTwo = "<cellular sms gateway email address goes here>";
         var ev = e.parameter.event;
         var eventTxt = e.parameter.data;
         var wsmData = JSON.parse(eventTxt);
         var message = wsmData.msg;
         var subject = generateSubjText(ev);
         var uTime = wsmData.etime;
         var tmStamp = computeLocalTime(uTime);
         var body = message + " at: " + tmStamp;
         MailApp.sendEmail(targetBob, subject, body);
        //MailApp.sendEmail(targetTwo, subject, body);
}
```

The function sendAlert() is provided with constants that are the email address of the sms gateway for each recipient's cellular carrier; e.g. "8885551212@txt.att.net" for an AT&T mobile customer with cell phone number 888-555-1212. The event name and message data (the data string produced by the Photon firmware) are extracted from the event object and placed into variables “ev” (webhook event name) and “message”. The function “generateSubjText()” is called with the event name as the argument. This function simply translates each full event name (designating a specific alert) into text for the email “subject” line. A local timestamp string is computed from the unix time that was json encoded into the webhook data by the Photon firmware using the same function as described above. The Google Apps Script utility “MailApp.sendEmail()” is then used to send an email to the SMS text gateway. The arguments are:

- gateway email address for the sms text
- the subject line string
- the message body string

The cellular carrier’s sms gateway then takes care of sending the sms text to the user’s mobile phone.

6. Overview of Logging Spreadsheet

The WSM sends various real time alerts to the user. In addition, the WSM logs data for offline analysis. It does this by sending data via an event to the Particle Cloud. A webhook integration in the Particle Cloud takes the event from the WSM and sends it as a new row to a Google Sheet. This sheet is where a user will look for historical data and graphs. To use this function you will need to set up a Google Sheet.

The repository contains an .odf format spreadsheet that you can upload to the Google account associated with your WSM. (/Firmware/WellSystemMonitor/googleInfo/) Here we will explain each tab in the Google Sheet; you may need to fool with the Sheet after uploading to Google Docs. The screenshots below are from a sample Google Docs Sheet.

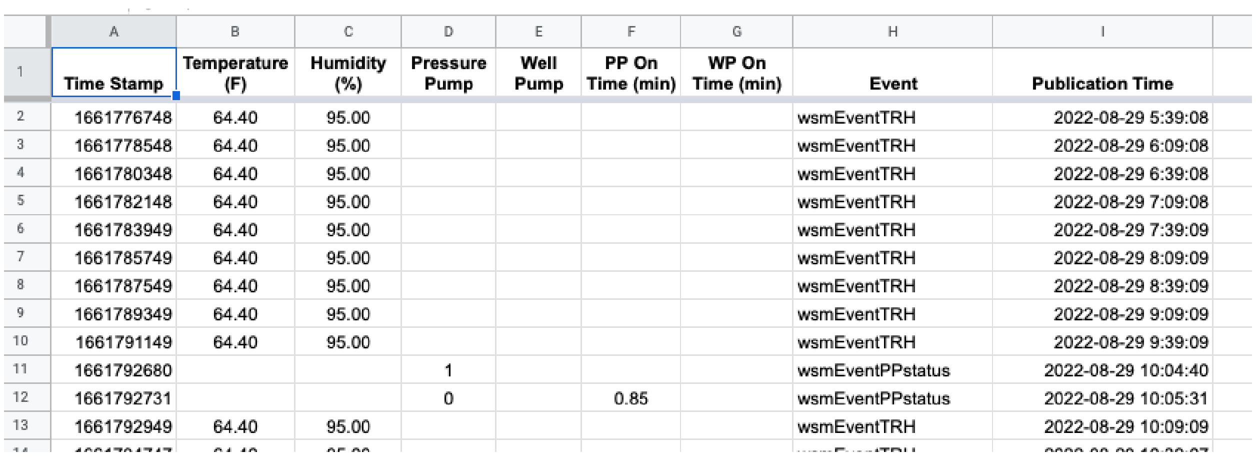
We cover the two primary tabs in this section of the document. In an appendix we cover Tab 3 and Tab 4 which provide some analysis of the data.

Tab 1: Sheet 1 - Raw Data

This is the sheet that accepts incoming data from your Particle webhook. Columns A:G are data sent from the WSM firmware. Columns H:I are from the Particle cloud and the logging script.

Time Stamp is local time as known by the WSM. The other columns are self explanatory.

The update process for this sheet limits the number of rows to 2,000; about one month's data. We do this to keep the sheet responsive. When the number of rows exceeds 2,000 we delete the first 200 rows.



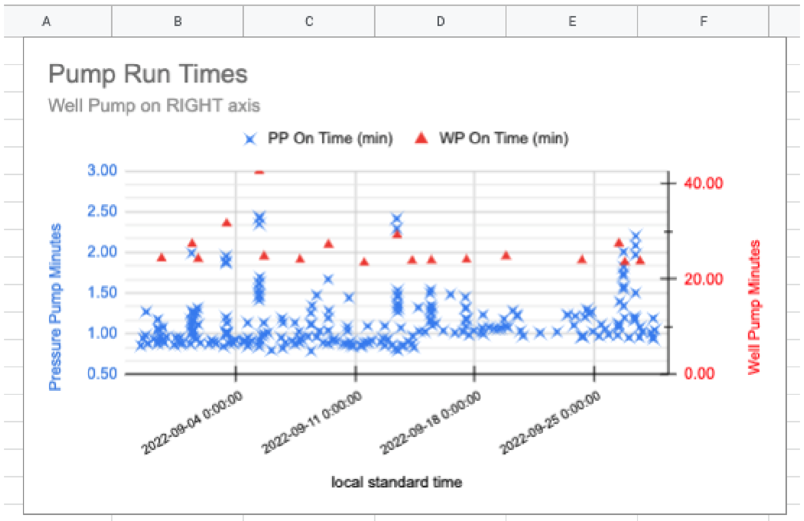
	A	B	C	D	E	F	G	H	I
1	Time Stamp	Temperature (F)	Humidity (%)	Pressure Pump	Well Pump	PP On Time (min)	WP On Time (min)	Event	Publication Time
2	1661776748	64.40	95.00					wsmEventTRH	2022-08-29 5:39:08
3	1661778548	64.40	95.00					wsmEventTRH	2022-08-29 6:09:08
4	1661780348	64.40	95.00					wsmEventTRH	2022-08-29 6:39:08
5	1661782148	64.40	95.00					wsmEventTRH	2022-08-29 7:09:08
6	1661783949	64.40	95.00					wsmEventTRH	2022-08-29 7:39:09
7	1661785749	64.40	95.00					wsmEventTRH	2022-08-29 8:09:09
8	1661787549	64.40	95.00					wsmEventTRH	2022-08-29 8:39:09
9	1661789349	64.40	95.00					wsmEventTRH	2022-08-29 9:09:09
10	1661791149	64.40	95.00					wsmEventTRH	2022-08-29 9:39:09
11	1661792680			1				wsmEventPPstatus	2022-08-29 10:04:40
12	1661792731			0		0.85		wsmEventPPstatus	2022-08-29 10:05:31
13	1661792949	64.40	95.00					wsmEventTRH	2022-08-29 10:09:09

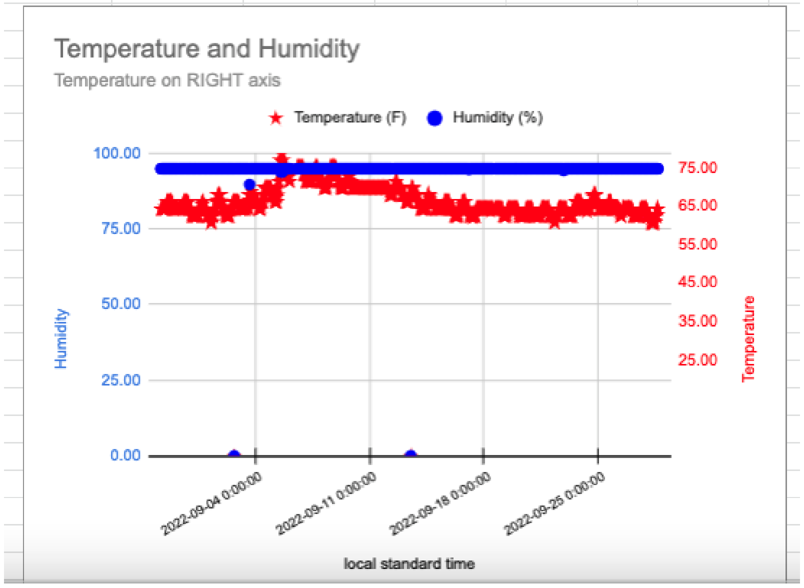
Tab 2: Graphs

This sheet contains 2 graphs of the raw data.

The first scatter graph shows Pressure Pump run times on the Y-axis and Well Pump run times on the Second Y-axis. Both use Publication Time as the X-axis.

The second scatter graph shows the reported Temperature on the Y-axis and Relative Humidity on the Second Y-axis as reported by the WSM. Note that in this sample data the RHS probe has likely malfunctioned (they are not very reliable).





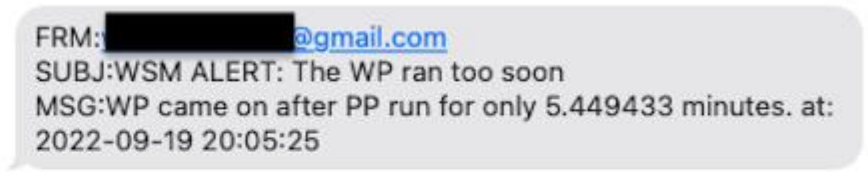
7. SMS Text Alerts.

- o

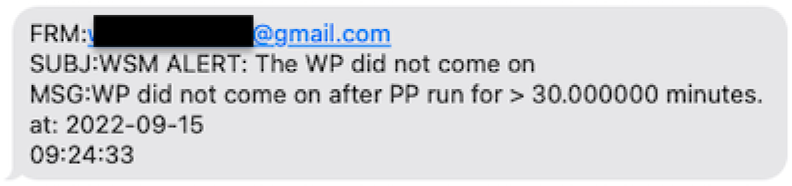
Sample Alerts

When an alert condition is met, the WSM will send an SMS message with the subject "WSM ALERT". Each SMS message will describe the alert condition. Two sample alerts are shown below. A description of each alert, possible causes, and actions to consider taking can be found here:

https://github.com/TeamPracticalProjects/WellSystemMonitor/wiki/Alerts-Explained



FRM:[redacted]@gmail.com
SUBJ:WSM ALERT: The WP ran too soon
MSG:WP came on after PP run for only 5.449433 minutes. at: 2022-09-19 20:05:25



FRM:[redacted]@gmail.com
SUBJ:WSM ALERT: The WP did not come on
MSG:WP did not come on after PP run for > 30.000000 minutes. at: 2022-09-15 09:24:33

Appendix 1 - Software/Firmware Installation.

Build the Hardware.

Complete instructions for building and installing the Well System Monitor hardware can be found in the “WSM Build manual” at: https://github.com/TeamPracticalProjects/WellSystemMonitor/blob/master/Documentation/WSM%20Build%20Manual.pdf

Installing the Particle Firmware.

Source code for the Particle Photon firmware is located in this repository in the /Firmware/WellSystemMonitor folder at: https://github.com/TeamPracticalProjects/WellSystemMonitor/tree/master/Firmware/WellSystemMonitor

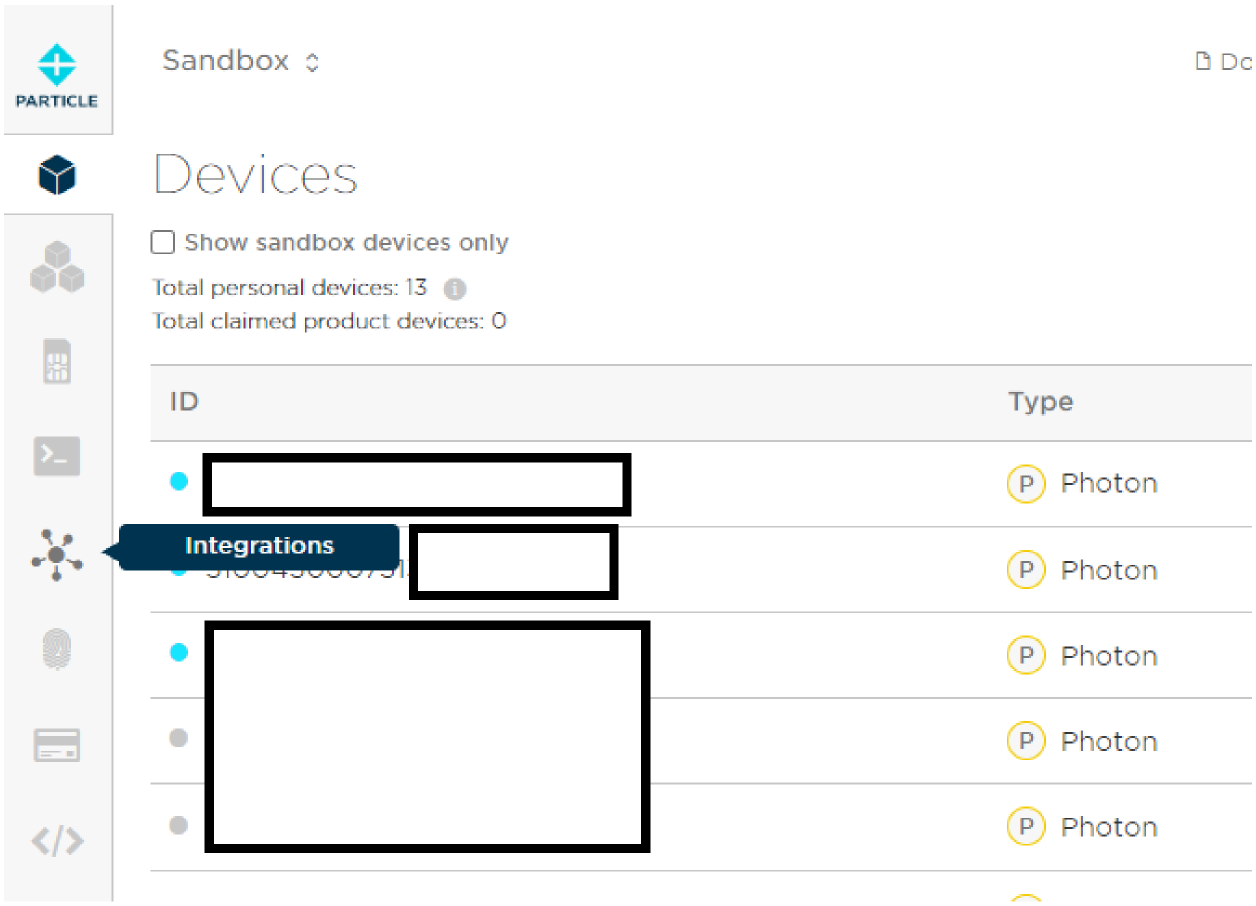
This folder is all set up to edit and build the firmware using Particle’s Workbench (VSCode based desktop IDE). The suggested approach is to use the Particle Workbench to build the code and flash it to your Photon. You can use Particle’s web based IDE, but you will have to set up additional tabs for the PietteTech_DHT library and for the various .cpp and .h files that are part of this project.

You can find information about Particle's IDEs and how to compile, build and flash firmware to your Particle device on Particle's web site, at: https://docs.particle.io/getting-started/getting-started/

NOTE: This project was developed and tested using Particle OS version 3.3.0 and the PietteTech_DHT library version 0.0.12. We strongly recommend that you use these specific versions for your WSM project.

Creating the Webhooks.

Particle webhooks are created in the Particle Console (console.particle.io). Log into the Console using the Particle account name and password that you claimed your Particle Photon into. You will see the Console screen:



Devices

Show sandbox devices only

Total personal devices: 13

Total claimed product devices: 0

ID	Type
[redacted]	Photon
[redacted]	Photon
[redacted]	Photon
[redacted]	Photon
[redacted]	Photon

Click on "Integrations, as shown in the photo. Then click on "NEW INTEGRATION" and Webhook". Simply fill in the webhook building form:

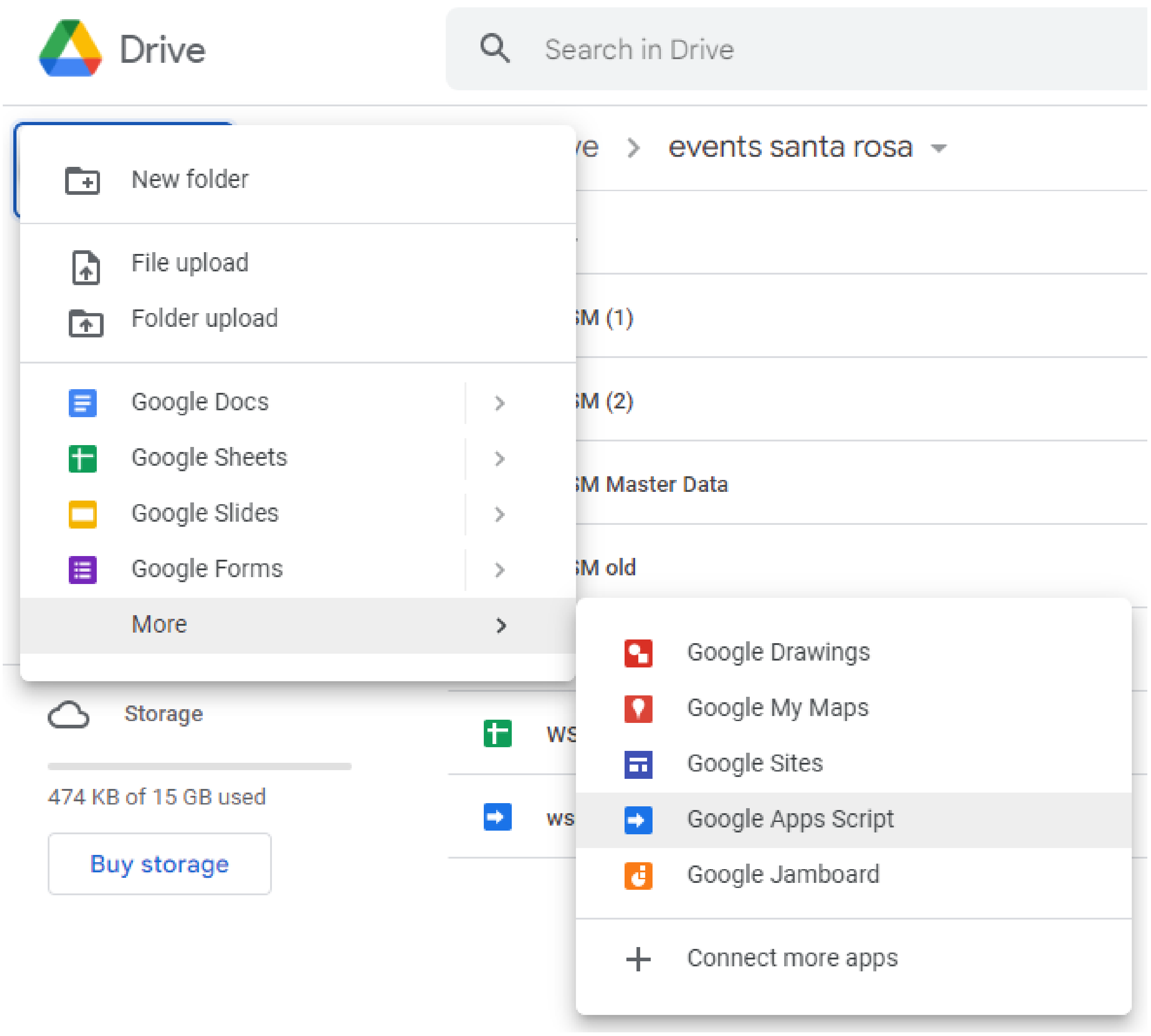
- For the wsmEvent webhook, see https://github.com/TeamPracticalProjects/WellSystemMonitor/blob/master/GoogleAppsScripts/WSM%20Event%20Webhook.png
- For the wsmAlert webhook, see https://github.com/TeamPracticalProjects/WellSystemMonitor/blob/master/GoogleAppsScripts/WSM%20Alert%20Webhook.png

Note that you will need the URL for the associated Google Apps Script in order to create these webhooks, so you will need to create the scripts first. See the section below for details.

Creating/Deploying Google Apps Scripts.

NOTE: *Before you can create and deploy the logging script, you will need to create the Google spreadsheet that you will log data into. See Appendix 2 below for detailed information. Specifically, you will need the URL of this spreadsheet for the "wsmWriteData" script.*

You create, modify and deploy Google Apps Scripts using Google's web-based script editor. NOTE: you may need to open an incognito window on your web browser in order for this process to work properly. In the browser window, go to "drive.google.com" and log into the same account that you used for the Google spreadsheet. Once in Google Drive in the proper account, you open a new Google Apps script, as a "Web App":



NOTE: *if you don't see the option for creating a Google Apps Script in your browser, then click on the "+ Connect more apps" and add the Google Apps Script to your Google account.*

The Script editor window will appear after clicking on Google Apps Script, as above. Give your script a name (your choice) and delete the default function and everything else in the script editor window. Then copy and paste the script code into the editor window, making sure that you copy the entire script.

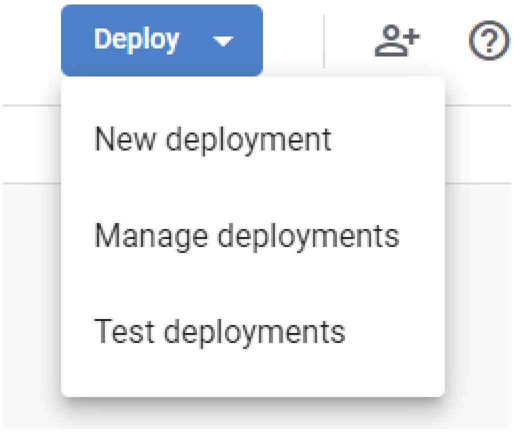
The script for logging data to the Google spreadsheet is called "wsmWriteData.txt" and is located at:
https://github.com/TeamPracticalProjects/WellSystemMonitor/blob/master/GoogleAppsScripts/wsmWriteData.txt

In both of the functions "doGet()" and "doPost()", locate the line:

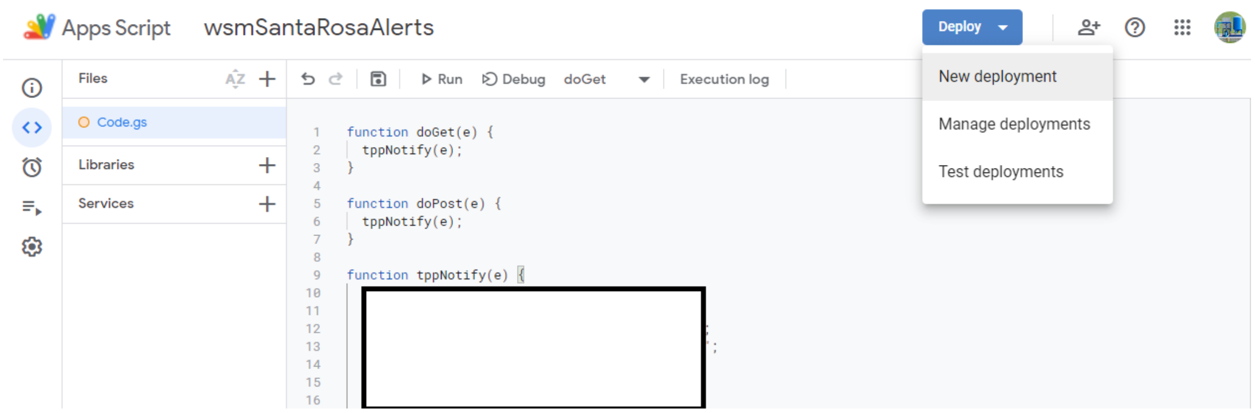
```
var ss = SpreadsheetApp.openByUrl("https://docs.google.com/spreadsheets/d/<url of Google
spreadsheet>/edit#gid=0");
```

and replace "<url of Google spreadsheet>" with the url that you copied when you created the Google logging spreadsheet. Make sure to paste the url into the right place in BOTH of the functions.

Once you have checked that all of the script code has been copied into the script editor, click on "Deploy":



Click on "New deployment". The fill in the information below:

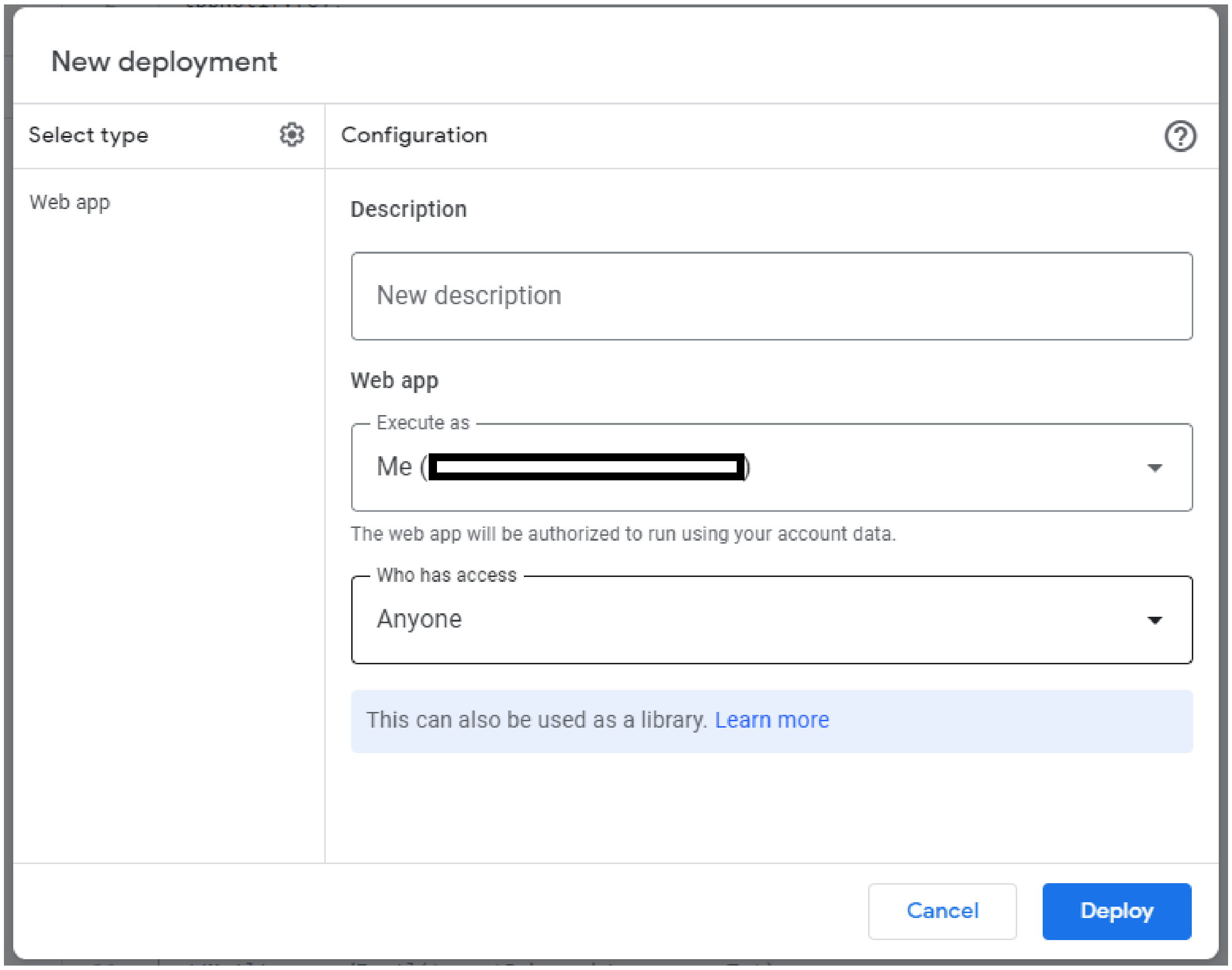


```
function doGet(e) {
  tppNotify(e);
}

function doPost(e) {
  tppNotify(e);
}

function tppNotify(e) {
```

Fill in the New deployment form as follows:


New deployment

Select type

Web app

Configuration

Description

New description

Web app

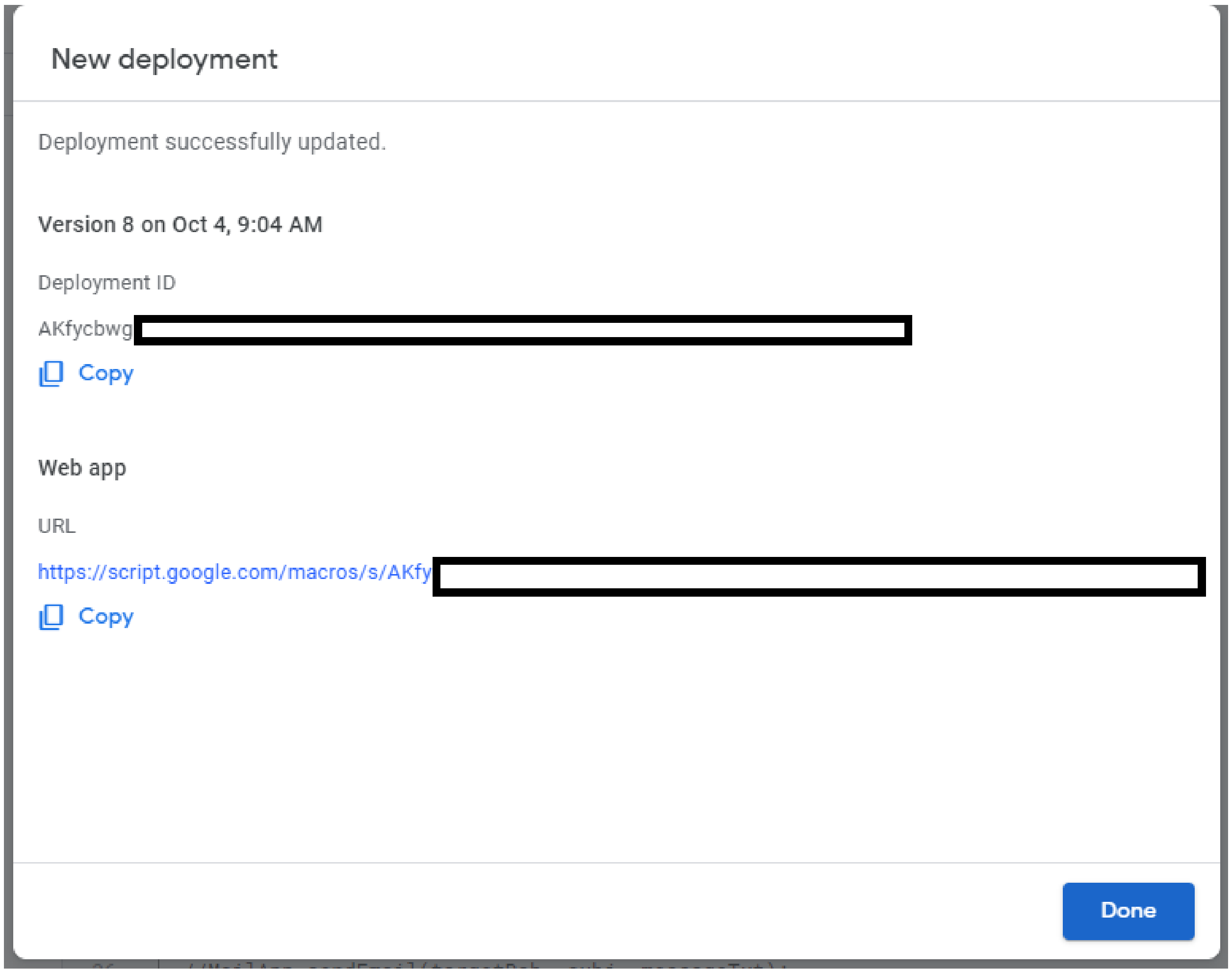
Execute as: Me ([redacted])

The web app will be authorized to run using your account data.

Who has access: Anyone

This can also be used as a library. Learn more

- The type of the deployment must be "Web app" (under "Select type").
- Replace "New description" under "Description" with whatever information you want to help you later identify this script version. We suggest something like "initial version" or "version 1.0".
- Under "Web app – Execute as" select "Me (<your email will appear here>)".
- Under "Who has access", select "Anyone".
- Then, click on "Deploy". You will be presented with a screen that shows the "Deployment ID" and the Web App URL. use the "Copy" button under the Web App URL to copy the url to your computer's clipboard. You will need to paste this url into the webhook form for the "wsmEvent" webhook.



New deployment

Deployment successfully updated.

Version 8 on Oct 4, 9:04 AM

Deployment ID

AKfycbwg[redacted]

Web app

URL

https://script.google.com/macros/s/AKfy[redacted]

The script for sending SMS text alerts to the user is called “WSM_Send_Alert” and is located at:
https://github.com/TeamPracticalProjects/WellSystemMonitor/blob/master/GoogleAppsScripts/WSM_Send_Alert.txt

Open a new Google Apps Script and give it a name that you will recognise. Follow the instructions above for creating and deploying this script. After pasting the script code into the script, locate the line:

```
const targetBob = "<cellular sms gateway email address goes here>";
```

Change the constant name to your name and change <cellular sms gateway email address goes here> to the sms gateway address for your mobile device and your cellular carrier. You may add as many recipients here as you like. For each recipient that you add, make sure that you have a line of code that replaces

```
MailApp.sendEmail(targetBob, subject, body);
```

with the name that you chose replacing “targetBob” at the top of the script. Leave the other arguments of MailApp.sendEmail() as “subject” and “body”. Then deploy the script as described above. Note that after deployment, you will need to copy the url of the deployed script for the “wsmAlert” webhook.

A NOTE ABOUT THE “DEPLOY” OPTIONS. When you click on “Deploy”, you get three options:

- New deployment: Creates a new deployment. This is the option that you will usually use.
- Manage deployments. This option will give you a list of current and past script deployments, by the name that you gave to the deployment (which is why you should think about appropriate names). You can use this option to obtain the URL of any previous deployment that you wish.
- Test deployments. This option allows you to deploy an altered script *without changing the url*. This can be handy when developing and testing new script code, because you won’t need to change the url in the webhook to test your altered script version. However, you must be sure to perform a “New deployment” when you are ready to deploy the latest version of your script.

Appendix 2 - Creating the Google Logging Spreadsheet.

In the earlier part of this document we described Tab 1 and Tab 2. There are two additional tabs that provide some excellent analysis of the data, but they need to be tweaked by hand once a month. This section describes these tabs and how to work them.

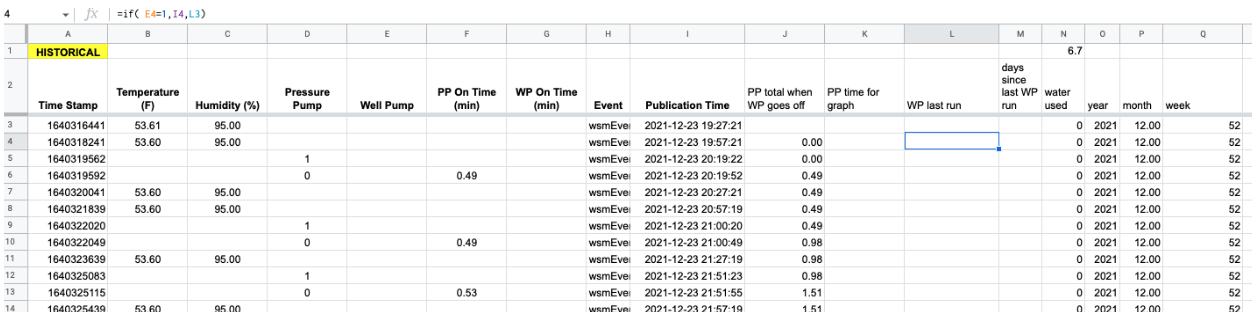
Set up

The repository contains an .odf format spreadsheet that you can upload to the Google account associated with your WSM. (/Firmware/WellSystemMonitor/googleInfo/); you may need to fool with the Sheet after uploading to Google Docs. Here we will explain Tab 3 and Tab 4 in the Google Sheet. The screenshots below are from a sample Google Docs Sheet.

Tab 3: Historical Data

This tab is optional, but valuable. Since the first tab of raw data is limited to 2,000 rows we need a place to store older data - we do this in Tab 3. We also add some additional columns to provide the data needed to generate some valuable graphs.

At least once a month a human scrolls to the bottom of Tab 3 to note the last publication time in Tab 3. The human then goes to Tab 1, finds that exact row, and selects all the rows below that row. Returning to Tab 3, the human pastes those rows into the bottom of Tab 3. Thus columns A:I are exact historical copies of data from Tab 1.



```
4  fx  =if( E4=1,I4,L3)
```

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
1	HISTORICAL													6.7			
2	Time Stamp	Temperature (F)	Humidity (%)	Pressure Pump	Well Pump	PP On Time (min)	WP On Time (min)	Event	Publication Time	PP total when WP goes off	PP time for graph	WP last run	days since last WP run	water used	year	month	week
3	1640316441	53.61	95.00					wsmEve	2021-12-23 19:27:21					0	2021	12.00	52
4	1640318241	53.60	95.00					wsmEve	2021-12-23 19:57:21	0.00				0	2021	12.00	52
5	1640319562			1				wsmEve	2021-12-23 20:19:22	0.00				0	2021	12.00	52
6	1640319592			0		0.49		wsmEve	2021-12-23 20:19:52	0.49				0	2021	12.00	52
7	1640320041	53.60	95.00					wsmEve	2021-12-23 20:27:21	0.49				0	2021	12.00	52
8	1640321839	53.60	95.00					wsmEve	2021-12-23 20:57:19	0.49				0	2021	12.00	52
9	1640322020			1				wsmEve	2021-12-23 21:00:20	0.49				0	2021	12.00	52
10	1640322049			0		0.49		wsmEve	2021-12-23 21:00:49	0.98				0	2021	12.00	52
11	1640323639	53.60	95.00					wsmEve	2021-12-23 21:27:19	0.98				0	2021	12.00	52
12	1640325083			1				wsmEve	2021-12-23 21:51:23	0.98				0	2021	12.00	52
13	1640325115			0		0.53		wsmEve	2021-12-23 21:51:55	1.51				0	2021	12.00	52
14	1640325439	53.60	95.00					wsmEve	2021-12-23 21:57:19	1.51				0	2021	12.00	52

Column J: PP Total When WP Goes Off

The first row in this column is blank. Other rows have the formula:

```
=if( and(isnumber(E4), E4=0),  0,  J3 + F4)
```

This column accumulates the Pressure Pump on times between runs of the Well Pump. You'll see that the values increase until the Well Pump runs; at that point the column value is set back to 0.

Column K: PP time for graph

This column is the value we use to graph. When the Well Pump runs, this column is filled with the accumulated PP time from Column J. Rows have the formula:

```
=if( and(isnumber(E89), E89=0),  J88,  "")
```

Column L: WP last run

This column is used to carry forward the Publication Time of the last event when the Well Pump ran. It is used by the formula in Column M. The first row in this column is blank. The other rows have this formula:

```
=if( E4=1,I4,L3)
```

Column M: days since last WP run

This column is used for graphing. It shows us how long it has been between Well Pump turn on events. The first row in this column is blank. The other rows have this formula:

```
=if(E4=1,I4-L3,"")
```

Column N: water used

This column is used for graphing. It uses the value in N1 as the number of gallons the Well Pump will supply per minute of run time. Each row has this formula:

```
=$N$1*G5
```

Columns O:Q

These columns are all calculated from the Publication Time column. They are used for pivot tables to report on water use by time period.

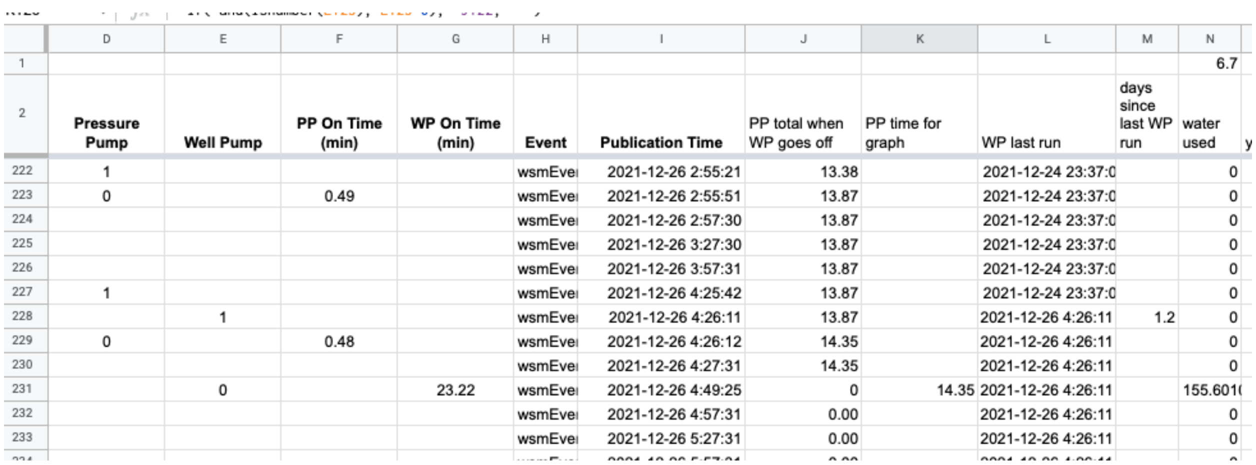
O: `=year(I3)`

P: `=month(I3)`

Q: `=weeknum(I3)`

Sample Calculation Columns

Note how column J continues to grow until row 231 when the WP runs and the value is reset to 0. Then see that column K reports the cumulative PP run time (14.35 minutes) that corresponds to the WP on time in row 231. Note that column L holds the last time the WP ran until at row 228 when the WP turns on, and see that column M has the length of time between the previous WP turn on and this current time (1.2 days). Lastly, see that column N has the amount of water used by multiplying the WP on time (23.22 minutes) by N1 (6.7) to give 155.6 gallons.



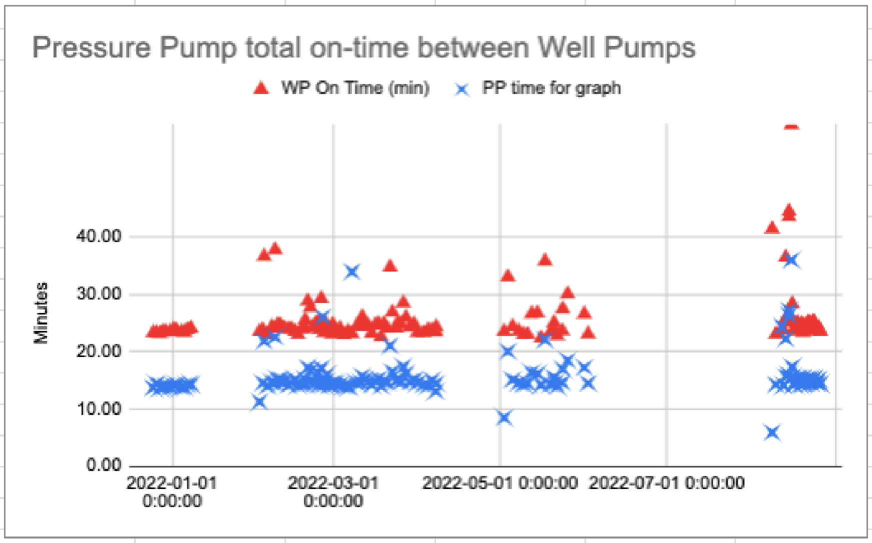
	D	E	F	G	H	I	J	K	L	M	N
1											6.7
2	**Pressure Pump**	**Well Pump**	**PP On Time (min)**	**WP On Time (min)**	**Event**	**Publication Time**	PP total when WP goes off	PP time for graph	WP last run	days since last WP run	water used
222	1				wsmEve	2021-12-26 2:55:21	13.38		2021-12-24 23:37:0		0
223	0		0.49		wsmEve	2021-12-26 2:55:51	13.87		2021-12-24 23:37:0		0
224					wsmEve	2021-12-26 2:57:30	13.87		2021-12-24 23:37:0		0
225					wsmEve	2021-12-26 3:27:30	13.87		2021-12-24 23:37:0		0
226					wsmEve	2021-12-26 3:57:31	13.87		2021-12-24 23:37:0		0
227	1				wsmEve	2021-12-26 4:25:42	13.87		2021-12-24 23:37:0		0
228		1			wsmEve	2021-12-26 4:26:11	13.87		2021-12-26 4:26:11	1.2	0
229	0		0.48		wsmEve	2021-12-26 4:26:12	14.35		2021-12-26 4:26:11		0
230					wsmEve	2021-12-26 4:27:31	14.35		2021-12-26 4:26:11		0
231		0		23.22	wsmEve	2021-12-26 4:49:25	0	14.35	2021-12-26 4:26:11		155.601
232					wsmEve	2021-12-26 4:57:31	0.00		2021-12-26 4:26:11		0
233					wsmEve	2021-12-26 5:27:31	0.00		2021-12-26 4:26:11		0

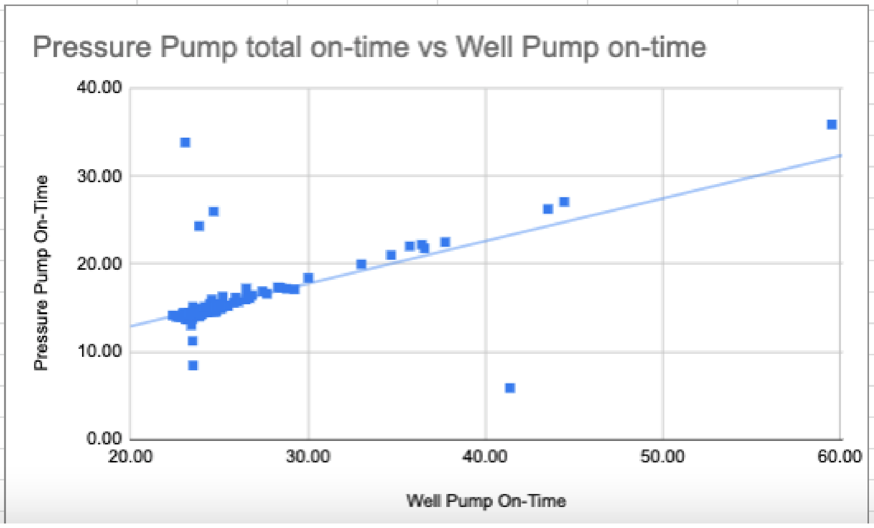
The Historical Graphs

After all that spreadsheet calculation we finally get the payoff, the graphs.

The first graph in Tab 3 shows the cumulative PP time plotted with the WP time. The PP goes on / off in response to water being used by the property. The WP comes on to refill the storage tank when it is low. We should see a correlation that the longer the PP has run, the longer the WP will run.

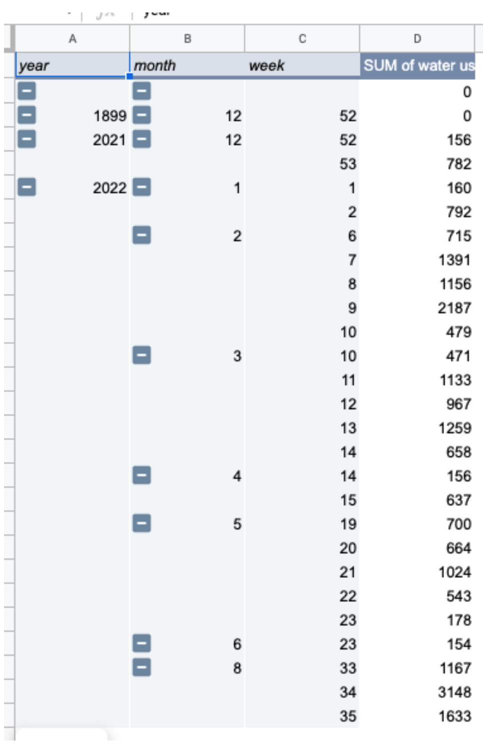
The second graph in Tab 3 is the really valuable graph as it plots these two variables against each other. In a well running system we expect that there will be a linear correlation between the cumulative PP run time and the WP run time. The graph shows that the correlation is strong. When there is an outlier it is a stimulus for an investigation of the system. If the point is above the normal line, then the PP has been running longer than expected - could it be that the PP is failing to pressurize the tank? If the point is below the normal line then the WP has come on for longer than expected - could the well be running dry?



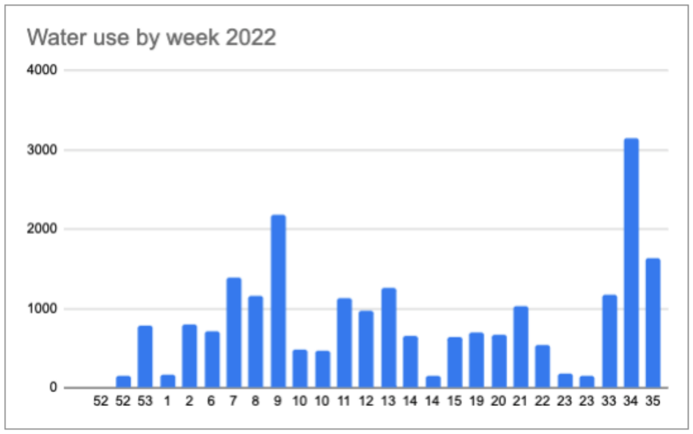


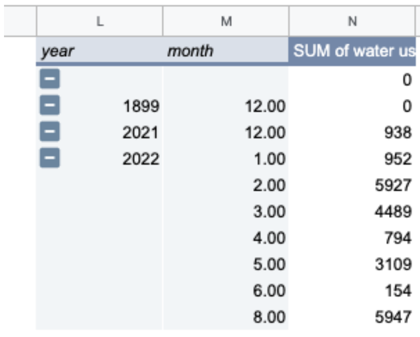
Tab 4: Water Usage

This sheet has two pivot tables that use the date component columns from Tab 3. One pivots water use by Week of Year and the other by Month. These are relatively straight forward if the Tab 3 sheet has been set up correctly. Samples are shown below.

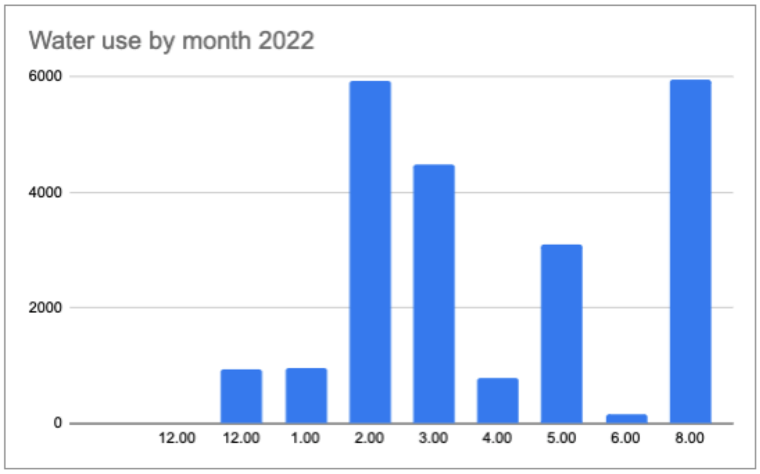


year	month	week	SUM of water us
⊟	⊟		0
⊟ 1899	⊟ 12	52	0
⊟ 2021	⊟ 12	52	156
		53	782
⊟ 2022	⊟ 1	1	160
		2	792
	⊟ 2	6	715
		7	1391
		8	1156
		9	2187
		10	479
	⊟ 3	10	471
		11	1133
		12	967
		13	1259
		14	658
	⊟ 4	14	156
		15	637
	⊟ 5	19	700
		20	664
		21	1024
		22	543
		23	178
	⊟ 6	23	154
	⊟ 8	33	1167
		34	3148
		35	1633





year	month	SUM of water us
⊟		0
⊟ 1899	12.00	0
⊟ 2021	12.00	938
⊟ 2022	1.00	952
	2.00	5927
	3.00	4489
	4.00	794
	5.00	3109
	6.00	154
	8.00	5947


Water use by month 2022