TPP Connectivity Tools Technical Description

By: Jim Schrempp and Bob Glicksman; v1, 10/21/2021

NOTICE: Use of this document is subject to the terms of use described in the document "Terms_of_Use_License_and_Disclaimer" that is included in this release package. This document can be found at:

https://github.com/TeamPracticalProjects/TPP_Connectivity_Tools/blob/master/Terms_of_Use_License_and_Disclaimer.pdf

© 2021 Team Practical Projects, Maker Nexus, Bob Glicksman, and Jim. All rights reserved.

TABLE OF CONTENTS.

© 2021, Team Practical Projects, Jim Schrempp and Bob Glicksman; all rights reserved. *Subject to “Terms of Use, License and Disclosure”*

1. OVERVIEW.

1.1. Background.

We at Team Practical Projects (TPP) have been developing Internet connected projects for a number of years. In the course of these development activities, we have tested and developed a number of solutions for communicating with embedded devices. We have been using Particle (Particle.io) IoT solutions since their early days and we have settled on Particle because of their high level of support for making IoT simple. Particle features include:

- IoT devices that communicate over the Internet using either WiFi or Cellular connections. Particle devices are code-compatible across their product line. One set of firmware works both on WiFi and on Cellular networks, allowing great flexibility in deployment.
- Firmware development in the Arduino programming language.
- Extended firmware device functions for securely communicating over the Internet, using the Particle Cloud.
- Ability to communicate with Particle devices, via the Particle Cloud, using a REST API that is based on secure, industry standards.
- Built-in cloud service integrations, specifically including Webhooks.
- Ability to flash firmware over the Internet, supporting remotely deployed embedded devices.
- Particle's cloud-based Console that supports integration, testing, and health monitoring of remotely deployed Particle devices.
- Good value for the price.

The purpose of this document is to share what we have learned; in particular, our now-standard way of integrating Particle-based projects with self-developed mobile apps and with cloud-based services.

1.2. Integration Scenarios.

This document is organized around several specific integration scenarios. The scenarios are summarized here and are detailed in Section 3 of this document.

© 2021, Team Practical Projects, Jim Schrempp and Bob Glicksman; all rights reserved. *Subject to "Terms of Use, License and Disclosure"*

Most of our projects use a combination of these scenarios. Section 3 provides references to published open-source code and documentation for projects that use one or more of these integration scenarios. The scenarios are:

- Real-Time Communication with a Mobile App. We have developed templates and copy/paste source code for apps developed using MIT App Inventor 2. This makes it easy for anyone to create an app for a smartphone that can exchange information and commands with a Particle device. Our mobile apps read data from embedded Particle devices and call functions to execute code on these embedded devices. Examples include our "Garage Door Controller" (https://github.com/TeamPracticalProjects/Garage_Door_Controller) where our mobile app can be used to status a garage door (open or closed) and to activate opening/closing of the garage door from anywhere in the world that has cellular Internet service.

- Data Logging to Cloud Spreadsheet. We have found that cloud-based Google Spreadsheets are a great place to log remote sensor data. Google sheets can be read from anywhere that has Internet access, can easily be programmed to perform limited processing of the data, and can easily be set up to present sensor data both in tabular and graphical forms.

  We originally used IFTTT (ifttt.com) to perform this logging function (e.g. our SIS project: https://github.com/TeamPracticalProjects/SISProject) . IFTTT is extremely easy to use, but we have found that it has limitations.

  More recently, we bypass this third party solution and write our own web apps (using Google Apps Script) to log the data directly. We believe that this method is more reliable, has much less delay, and allows for the web app to preprocess the data for the best spreadsheet presentation (e.g. timestamping data with local time, including adjustment for daylight savings time). We plan to publish this solution shortly in our "Well System Monitoring" project:

  https://github.com/TeamPracticalProjects/WellSystemMonitor

- Real-Time Alarm/Alert Notification. Many IoT projects require a means to alert a user; i.e. of an alarm condition. Alarm/alerts are best delivered to a mobile device that the user carries with them. SMS texts and mobile push notifications are the two possibilities for delivering alarm/alerts to users. We used push notifications to a Blynk app (blink.io) in our Water Leak Sensor project:

  https://github.com/TeamPracticalProjects/WaterLeakSensor

© 2021, Team Practical Projects, Jim Schrempp and Bob Glicksman; all rights reserved. *Subject to "Terms of Use, License and Disclosure"*

The push notifications worked, but we experienced variable, sometimes very long, delays in getting the notification and the notification was occasionally lost altogether. We solved the latter problem by publishing a second notification, 30 seconds after the first.

In our estimation, SMS texts are delivered faster and more reliably than push notifications. In addition, SMS texts do not require an app to be installed on a user's mobile device (mobile phones come with SMS texting built in). You don't even need a smartphone to receive SMS texts; an old flip phone works just fine.

We have developed a system for delivering SMS texts to one or more mobile phones from our Particle-based project. We plan to publish this solution shortly in our "Well System Monitoring" project:

https://github.com/TeamPracticalProjects/WellSystemMonitor

- Device – Device Secure Wireless Communication. It is sometimes desirable to have one IoT device communicate wirelessly, over the Internet, with another IoT device. Particle provides a secure publish and subscribe mechanism that is ideal for this purpose. In our RFID access control system for Maker Nexus, we needed a way to communicate access control decisions from an RFID card station to an electronic locking mechanism; e.g. to unlock a solenoid controlled door. The card stations need to be out in the open, easily available to the public, whereas the electronics that control the door solenoid must be placed in a physically secure location. Secure communication between RFID stations and electronic lock controllers was easily implemented via Particle's Cloud. For details, see:

  https://github.com/TeamPracticalProjects/MN_ACL

- Integrating a Cloud Database. IoT projects often need to communicate with some sort of cloud-based service. Our RFID access control system for Maker Nexus required RFID card stations to communicate with a commercial, cloud-based membership management system in order to obtain current membership status information. In addition, we needed to log member RFID card tap events to a general purpose cloud database for statistical reporting purposes. Particle Webhooks and the Particle Cloud provide a secure and very easy to use capability to perform these integrations. Particle's Webhooks can be used to communicate with any cloud based service that uses an industry standard REST API. For examples, see:

  https://github.com/TeamPracticalProjects/MN_ACL

© 2021, Team Practical Projects, Jim Schrempp and Bob Glicksman; all rights reserved. *Subject to "Terms of Use, License and Disclosure"*

1.3. Our Criteria.

There are many ways in which developers can implement the scenarios described above. We (Team Practical Projects) have our own criteria for selecting the techniques and services that are described in this document. Our criteria are based upon the fact that we:

- develop projects for use by hobbyists
- feel a need to support our projects long term, even if we are not using them ourselves.

These factors heavily influence our choice of integration products and strategies. *Your circumstances may differ*. If so, our choices may not be the best ones for you! Our criteria are:

a) Free of ongoing costs (completely free if possible). We are not commercial developers and we don't get paid for our development work. Any costs involved must be absorbed by us, with no expectation of reimbursement. The fact that we feel a need to supply long term support means that even low cost, ongoing fees represent a significant financial burden.

b) Minimal use of third party products and services. There are many excellent third party services that make the aforementioned integrations easy for hobbyists. Many of these are entirely free, have a free tier for hobbyists, or at least incur only a one-time purchase cost. Some of the best of these are Blynk (blynk.io), IFTTT (ifttt.com), and ThingSpeak (thingspeak.com). We have used a number of these in the past, and we highly recommend these to amateur developers who need a very quick and very simple integration solution. However, our preferences have evolved toward more robust capabilities from vendors with whom we already have accounts. These include Particle (obviously, if our projects are Particle based), Google (everyone already has a Gmail account; perhaps more than one!) and your cell phone carrier (can't get SMS texts without an active cell phone).

c) Web-based development tools. Because our projects are open source, we expect others to implement and improve upon our designs. The best tools for hobbyists are web-based IDEs, since a hobbyist who wishes to make small changes to some software need not spend a lot of time installing and configuring the tools needed to accomplish this.

Once again, if your criteria differ, you will likely find that our solutions are not ideal for you.

© 2021, Team Practical Projects, Jim Schrempp and Bob Glicksman; all rights reserved. *Subject to "Terms of Use, License and Disclosure"*

2. INTEGRATION TOOLSET.

2.1. Particle.

Particle (www.particle.io) is a manufacturer of Internet of Things (IoT) devices. We like Particle because their system architecture makes IoT simple and straightforward. Particle makes a complete line of embedded IoT devices with built-in Internet access and secure communication with Particle's "Cloud". Particle devices communicate over the Internet via WiFi or cellular services. Particle devices are programmed using the Arduino programming language (albeit with Particle's own IDEs – both web-based and desktop). Firmware developed using Particle's development tools are compatible across Particle's hardware devices. Therefore, one set of firmware can be developed that will run equally well across WiFi and cellular Internet access. New firmware can be "flashed" to Particle's devices over the Internet; a key capability for maintaining devices in the field.

We (Team Practical Projects) develop projects aimed at individuals and hobbyists. Therefore, we use Particle's development modules: Photon and Argon for WiFi applications and Boron for cellular applications. Note that the Argon and Boron are fully pin compatible; a printed circuit board can be developed that supports both devices, providing the user complete freedom to choose WiFi or Cellular Internet access.

Particle provides several pathways to scale projects from development to small scale manufacturing up to very large scale manufacturing.

The focus of this section is on Particle's integration services that we use in our projects.

2.1.1. Particle Console.

The Particle Console (console.particle.io) is an essential tool for communicating with embedded Particle devices for the purposes of testing, maintaining, and statusing Particle devices that are deployed in our projects. The Console provides secure communication with Particle devices to (among other things): health check deployed Particle devices, capture publication/subscription communication to/from deployed Particle devices, read data variables from deployed Particle devices, and call functions on deployed Particle devices. The Particle Console is also used to configure cloud integration tools, particularly Webhooks. The Particle Cloud and Webhooks are discussed further below.

2.1.2. Particle OS.

The Particle device OS documentation defines the firmware instruction set for programming Particle devices. This instruction set is compatible with Arduino; however, Particle has extended the standard Arduino programming language (Wiring) with various "Cloud" instructions

© 2021, Team Practical Projects, Jim Schrempp and Bob Glicksman; all rights reserved. *Subject to "Terms of Use, License and Disclosure"*

that support Particle's IoT functionality. These added instructions are called "Cloud Functions" and can be found in Particle's on-line documentation:

https://docs.particle.io/reference/device-os/firmware/#cloud-functions

We have found that four of these Cloud instructions are particularly useful:

- **Particle.variable()**: makes a global variable in the Particle device firmware accessible to external devices (e.g. a mobile app) via Particle's REST API (https: GET). The variable value can be read out at any time.
- **Particle.function()**: makes a function declared within the device's firmware callable from external devices (e.g. a mobile app) via Particle's REST API (https: POST). The function can be called at any time from anywhere in the world, and can also be used as an ordinary function inside the firmware.
- **Particle.publish()**: Publishes data to the Particle Cloud under an event name. Published events can be subscribed to by other Particle devices in the same Particle account and can also fire off Webhooks and other Cloud integrations.
- **Particle.subscribe()**: Particle devices can subscribe to published events for secure device-device communication. Particle devices can also subscribe to Webhook responses and other Cloud integrations.

It must be noted that free Particle accounts are limited to 100,000 "data operations" per month. A data operation is roughly one cloud based message to a variable, function, publication, or subscription. In our experience, 100,000 data operations has not been a limiting factor.

2.1.3. Particle Cloud.

The Particle Cloud is the interface between firmware on Particle devices and the external world of the Internet. Particle devices synchronize their operation with the Particle Cloud. The Particle Cloud provides over the air flashing of new firmware to Particle devices; no physical connection is required, allowing devices in the field to be maintained and updated as needed. The Particle Cloud provides status and health checks of Particle devices in the field. The Particle Cloud further provides the REST API endpoints for the Particle Cloud firmware functions described above. The complete Particle REST API reference is documented here:

https://docs.particle.io/reference/device-cloud/api/

The Particle Cloud REST API uses industry standard OAUTH2 and encryption for authentication and security.

2.1.4. Particle Webhooks.

Particle Webhooks let you connect Particle events to all manner of services on the Internet. We have found Webhooks to be extremely easy to use to perform otherwise complex IoT integrations. Particle Webhook documentation can be found here:

© 2021, Team Practical Projects, Jim Schrempp and Bob Glicksman; all rights reserved. *Subject to "Terms of Use, License and Disclosure"*

https://docs.particle.io/reference/device-cloud/webhooks/

2.2. Google.

Almost everyone has at least one Google account. In addition to Gmail, Google provides cloud-based data storage and a variety of cloud-based apps, including Docs (Word compatible word processing), Sheets (Excel compatible spreadsheet), Slides (PowerPoint compatible presentation), Calendar, etc. Google's apps are entirely scriptable using Google Apps Script. All of this capability is available to the user cost free.

2.2.1. Google Apps Script.

Google Apps Script is a JavaScript-based scripting capability for all of Google's apps. Google Apps Scripts can be deployed in several different ways: bound to a Google document, unbound, and as a web app. A Google Apps Script that is deployed as a Web App can be accessed through a REST API (https: GET, POST, etc.). In particular, a Google Apps Script that is deployed as a Web App can be triggered by a GET or a POST generated by a Particle Webhook. We have used this capability to log data from a Particle device to a Google Sheet and to send an Email containing event information. Google's documentation for Google Apps Script can be found at:

https://developers.google.com/apps-script

2.2.2. Google Sheets.

We have found that Google Sheets is a particularly handy way to log sensor data from a Particle device. Google Sheets are accessible via any web browser from any device that is connected to the Internet. The spreadsheet format is handy for presenting data logs in tabular form. Google Sheets can format the logged data to make it more presentable, can perform calculations on this logged data, and can graph the data in many different formats; all without writing a single line of code! Google's documentation for their Google Sheet API can be found at:

https://developers.google.com/sheets/api

2.3. Mobile App Development.

Mobile apps are often required to communicate with embedded devices. There are many fine low- or no-code app builders available to developers, e.g. IFTTT "Do" and Blynk. However, we

© 2021, Team Practical Projects, Jim Schrempp and Bob Glicksman; all rights reserved. *Subject to "Terms of Use, License and Disclosure"*

have found it useful to be able to develop our own apps. Our self-developed apps operate faster and more reliably than commercial app builders and we can customize our apps with whatever logic and calculations we desire. Particle.variable() and Particle.function(), in conjunction with the Particle Cloud's REST API, make writing our own apps particularly easy. Particle provides some language bindings to their REST API, particularly JavaScript. However, we prefer to develop our own apps using MIT App Inventor 2 (AI2):

https://appinventor.mit.edu/

AI2 is completely free to use and is completely web based (albeit MIT also offers an offline development version). As of this writing, AI2 apps only run on Android devices, however iOS development capability is in final beta testing. AI2 apps are developed using a block-based programming language that is very easy to learn and is essentially foolproof to use (can't make typo's, drag and drop components, etc.).

AI2's "web" component is used to implement the REST API on the mobile app. We have described how to do this (with examples) in the project:

https://github.com/TeamPracticalProjects/MIT-App-Inventor-Particle-Photon-test

In addition, we have developed a "template" for our apps that you could use. The template provides a setup page that allows the user to log in to their Particle account, obtain a list of Particle devices in their account, and select the device to use with the app. The template also provides code that "pings" the selected device whenever the app is opened so that the device's on-line status is immediately evident. For details on the template, see:

https://github.com/TeamPracticalProjects/Particle_App_Template

2.4. SMS Gateway.

Embedded projects often contain sensing logic for out-of-bounds conditions that require the user to be altered immediately. The most obvious alert uses a mobile device, since the user normally carries the device with them. There are two ways to send asynchronous message alerts to mobile devices: (1) push notifications to a registered app, and (2) SMS texts. We have tried both methods and have a clear preference for SMS texts for the following reasons:

- Push notifications require an app to be installed on the mobile device. SMS texting is an inherent capability of all mobile devices, even non-smart ones (e.g. flip phone). Push notifications are generally thought of as a mechanism for marketing and customer engagement by app providers.

© 2021, Team Practical Projects, Jim Schrempp and Bob Glicksman; all rights reserved. *Subject to "Terms of Use, License and Disclosure"*

- SMS texting is faster and more reliable than push notifications. SMS texting replaced text based paging and is a service that professional on-call people have relied on for a long time.

Our strategy for sending SMS texts to a user's mobile phone is to send an Email to the SMS gateway that the user's mobile carrier provides. All mobile carriers have an SMS/MMS gateway. The gateway is generally free to use and is a part of the user's mobile account with the carrier[1]. Here is a list of gateways for mobile carriers:

https://jagerpro.com/help-center/sms-and-mms-gateway-list/

For example, to send an SMS text message to an AT&T (formerly Cingular) mobile phone, send an e-mail to:

<mobile phone number>@txt.att.net

Where <mobile phone number> is the 10 digit phone number of the user's mobile phone. The user, of course, already has an account with the mobile carrier, so no additional account is necessary. The resulting SMS text contains the e-mail sender information, the subject (of the e-mail) and the body (text message) of the e-mail.

[1] Receipt of SMS/MMS messages may incur charges to the user. Most modern mobile accounts offer unlimited voice and text, but please check with your carrier to be certain.

© 2021, Team Practical Projects, Jim Schrempp and Bob Glicksman; all rights reserved. *Subject to "Terms of Use, License and Disclosure"*

3. INTEGRATION DETAILS.

3.1. Real-Time Communication via Mobile App.

Mobile Apps provide a way to communicate with embedded Particle-based systems from anywhere that Internet access is available. We develop our apps in MIT App Inventor 2 (AI2), as introduced in section 2.3, above. Figure 3-1 illustrates the integration between the mobile app and the Particle device.

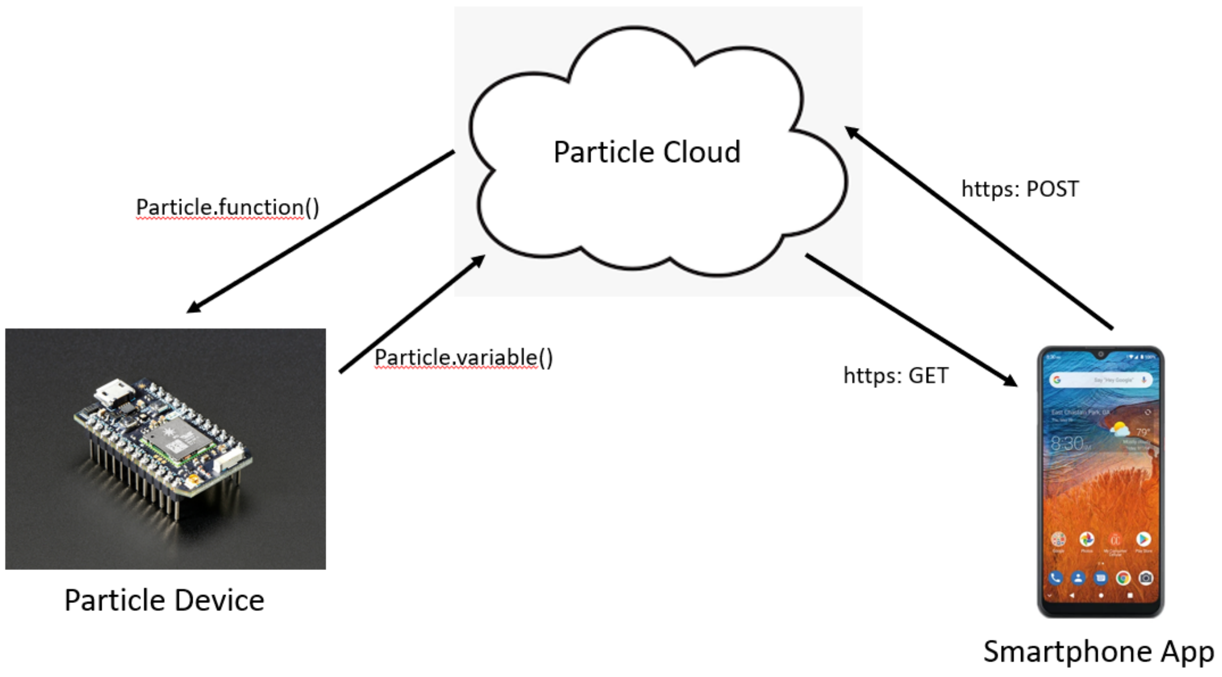


Figure 3-1. Real-time Communication with a Mobile App.

AI2 has a “web” component that provides the app with the ability to issue https: GET and POST commands (any other http command as well). An https: GET is used to fetch the value of a global variable on the Particle device that has been declared cloud accessible with Particle.variable() in the device firmware. Likewise, an https: POST is used to call a function on the Particle device that has been declared cloud accessible with Particle.function() in the device firmware. Complete documentation and an example project is posted at:

https://github.com/TeamPracticalProjects/MIT-App-Inventor-Particle-Photon-test

We frequently develop our AI2 apps within a template:

https://github.com/TeamPracticalProjects/Particle_App_Template

© 2021, Team Practical Projects, Jim Schrempp and Bob Glicksman; all rights reserved. *Subject to “Terms of Use, License and Disclosure”*

The Template provides a setup page wherein the user can login to their Particle account, retrieve the OAUTH2 user access (bearer) token, and select the Particle device within the user's account that will be used by the app. The Template also pings the device whenever the app is opened so that the app user knows that the device is on-line to the Internet.

Some examples of projects that use a mobile app to communicate with an embedded Particle device include:

- Remote door unlocking (https://github.com/TeamPracticalProjects/Particle-Door-Open): The mobile app trips an electric door strike for a pre-set time, allowing someone access through an otherwise locked door.

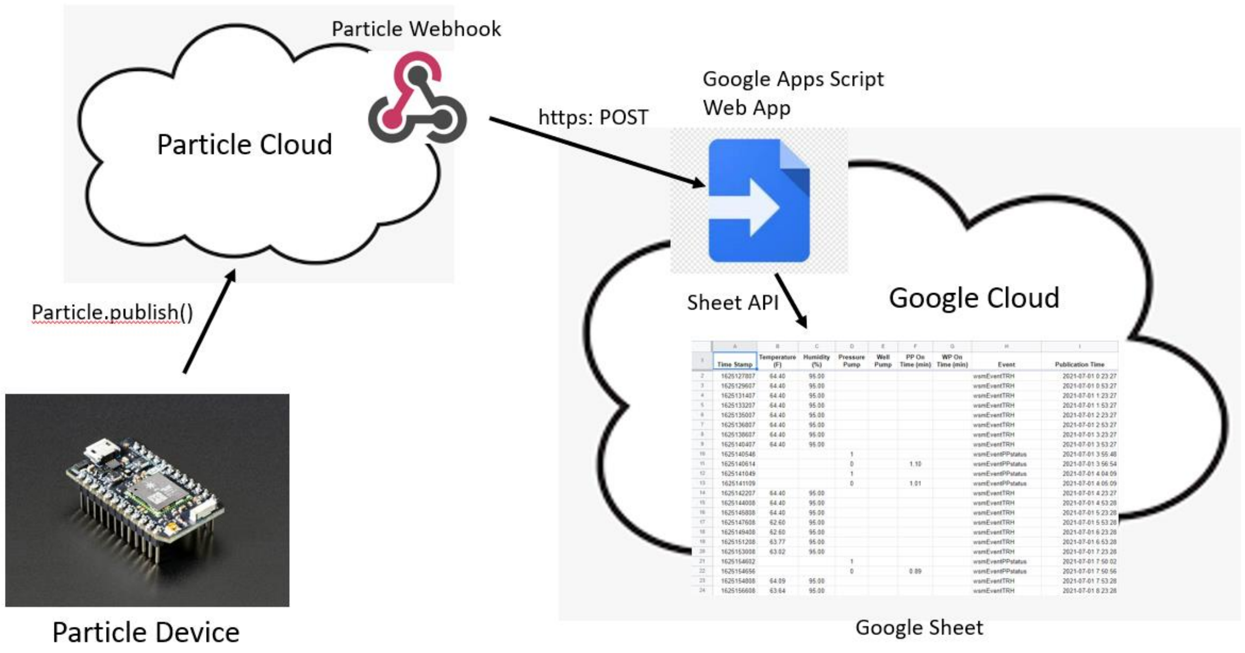
- Remote Garage Door Opening and Status (https://github.com/TeamPracticalProjects/Garage_Door_Controller): Obtain the current status (open or closed) of a garage door and activate the door to open or close.

- Administrative User Interface to issue RFID cards (https://github.com/TeamPracticalProjects/MN_ACL): An app running on an inexpensive Android tablet provides an RFID card access system administrator with a user interface to select the cardholder, burn an RFID card for that cardholder, determine the owner of a unknown RFID card, and reset an RFID card to its factory fresh condition for future re-use.

3.2. Data Logging to Cloud Spreadsheet.

Google Sheets provide a nice place to log data from embedded systems. Spreadsheets can format data, graph data and make complex calculations on data, all without programming. Google Sheets are free spreadsheets that live in the cloud and can be accessed from any Internet connected device that has a web browser. The basic process is illustrated in figure 3-2:

© 2021, Team Practical Projects, Jim Schrempp and Bob Glicksman; all rights reserved. *Subject to "Terms of Use, License and Disclosure"*



	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	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	1625140548			1				wsmEventPPstatus	2021-07-01 3:55:48
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 3-2. Data Logging to a Google Sheet.

The firmware on a Particle device acquires/computes new data and determines when the data is to be published to the Particle Cloud, using Particle.publish(). The arguments to Particle.publish() include: (a) an "event" name for the publication (e.g. "OutsideTemperature") and (b) the data associated with the event (e.g. "71.2"). By default, the publication is PRIVATE (restricted to the Particle account that the device is claimed in). Note that the published data is not limited to a simple value or string. Complex data can be encoded into strings, e.g. using JSON, XML, CSV, etc. Since the target of the data will ultimately be a Google Apps Script that is coded in Javascript, JSON is usually the preferred format.

A Webhook is created in the user's Particle account via the Particle Console ("Integrations"). The Webhook may be configured via dropdowns and "fill in blanks" forms; no coding is required. In general, the Webhook is set to subscribe to the published event (e.g. "OutsideTemperature") and the target URL is that of a web app that was created using Google Apps Scripting. Https: GET or POST can be used to trigger the Google Apps Script – we usually prefer the POST method, but this is up to the developer (configured under "requestType" in the Webhook definition). When the Webhook is configured this way, several parameters are automatically url-encoded in the GET or POST:

- The Particle Device-ID that published the event.
- The event name that triggered the Webhook.
- The data that was published by the Particle device.
- The publication date-time.

This simple, default Webhook configuration is often all that is needed for the project. However, Particle webhooks are very flexible and can do a lot more than these defaults; see Particle's

© 2021, Team Practical Projects, Jim Schrempp and Bob Glicksman; all rights reserved. *Subject to "Terms of Use, License and Disclosure"*

documentation at:

https://docs.particle.io/reference/device-cloud/webhooks/#overview

In order to create a Google App Script that will respond to the https: GET or POST from the Webhook, the script must be deployed as a “Web App”. Detailed deployment instructions are in the appendix to this document. In order for a Google App Script to respond to GET or POST, the script must contain one of the Javascript functions “doGet(e)” or “doPost(e)”, as appropriate. The argument “e” of these functions is an “event object” that contains, among other things, the url-encoded parameters that were included in the GET or POST (i.e. Particle Device-ID, Event Name, published data, publication date-time).

In order to access a sheet on the Google spreadsheet, one has to select both the spreadsheet (e.g. by its url) and a sheet name. For example, to have a web app access a Google Sheet within a spreadsheet, code:

```
function doPost(e) {
  var ss = SpreadsheetApp.openByUrl("<url of a Google Sheet that you create>");
  var sheet = ss.getSheetByName("Sheet1");  // Sheet 1 is the default name for the first sheet

  …  // other code to process and add the data to the sheet
}
```

The url-encoded data that came from the webhook can be obtained from the event object by the following line of code:

```
  var deviceData = e.parameter.data;
```

If the data was complex and sent as a json encoded string, it can be converted to a Javascript object from which the individual elements can then be accessed, e.g.:

```
  var dataObject = JSON.parse(deviceData);  // parse the json string to an object
  ….
  var outsideTemperature = dataObject.temp;  // gets the value of “temp” from the object
```

Finally, a new row of data can be appended to the Google Sheet using the command:

```
  sheet.appendRow([ ….]); // “sheet” is the variable name from “ss.getsheetByName(), above
```

The argument to the appendRow method is an array of values to post to the corresponding cells of the new row on the Google sheet.

Complete information about Google’s sheet API can be found at:

© 2021, Team Practical Projects, Jim Schrempp and Bob Glicksman; all rights reserved. *Subject to “Terms of Use, License and Disclosure”*

https://developers.google.com/apps-script/reference/spreadsheet

A complete project that uses the techniques of this section will shortly be posted at:

https://github.com/TeamPracticalProjects/WellSystemMonitor

3.3. Real-Time Alarm/Alert Notification.

Embedded systems with sensors often need a way to communicate urgent alarms/alerts to the user. In order for a user to get these alarms/alerts in a timely manner, the alarm/alert needs to be sent to a mobile device (that the user carries with them at all times). The user cannot be presumed to be using an app on their mobile device when the alarm/alert arrives. Asynchronous mobile device notification is required.

There are two basic ways to provide asynchronous messages to modern mobile devices: push notifications and SMS text messages. Our preference is for SMS text messages; see section 2.4, above.

Figure 3-3 illustrates the system that we have developed to send SMS text notifications to mobile devices. The architecture of figure 3-3 looks somewhat “Rube Goldberg” but it is actually quite fast and reliable.

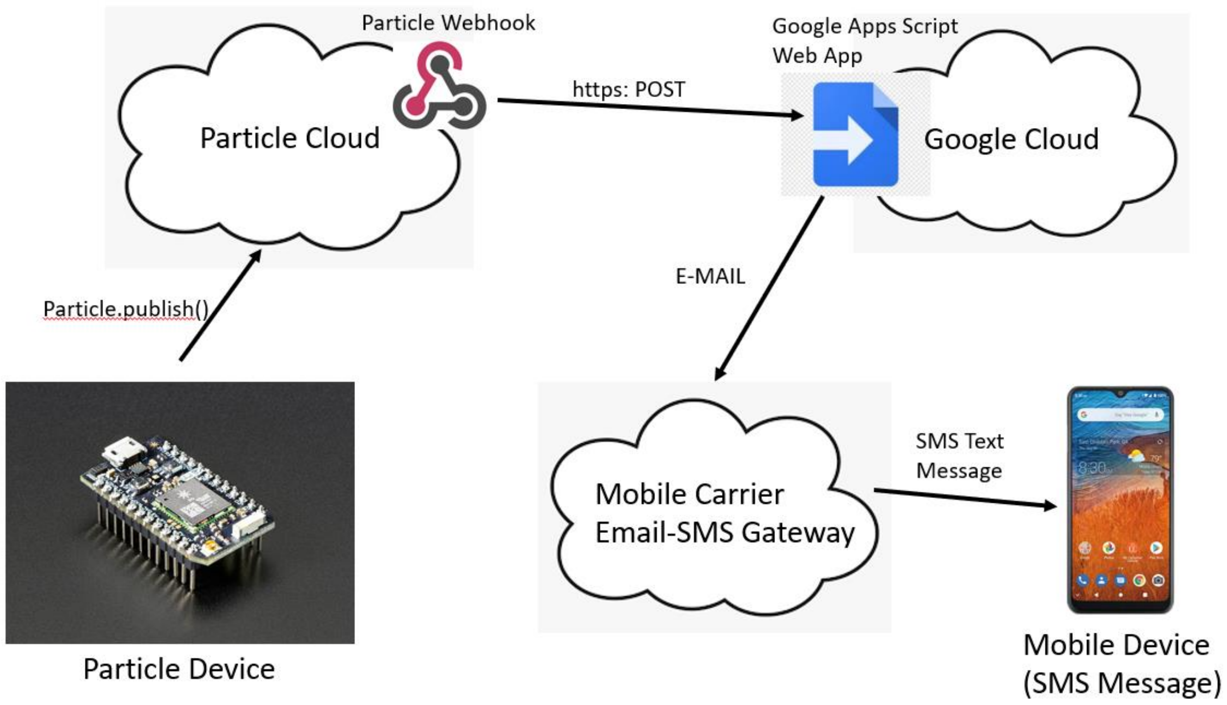


Figure 3-3. Real-Time Alarm/Alert Notification Architecture.

© 2021, Team Practical Projects, Jim Schrempp and Bob Glicksman; all rights reserved. *Subject to “Terms of Use, License and Disclosure”*

The essential element of our alarm/alert notification integration is the mobile carrier e-mail/SMS gateway. All mobile carriers have such a gateway; see section 2.4, above, for a link to the list. The problem then reduces to having the embedded Particle device send an e-mail to the user's mobile carrier gateway. This can be accomplished using a Google App Script that is deployed as a web app, similar to spreadsheet logging of section 3.2, above.

Google App Scripts have a utility object called "MailApp". An e-mail can be sent (to any valid e-mail address) using the function call:

```
MailApp.sendMail(gateway e-mail address, subject, messageText);
```

All of the arguments are strings:

- *gateway e-mail address*: the e-mail address of the user's mobile carrier SMS gateway, e.g. <10 digit mobile phone number>@txt.att.net

- *subject*: the subject line of the e-mail; listed as Subject on the resulting SMS text.

- *messageText*: the text body of the e-mail; listed as Message on the resulting SMS text.

We use Particle.publish() on the Particle device to fire off a Webhook that is configured in the user's Particle account. The Webhook target is the url of the web app deployed from the Google App script. Generally speaking, we use canned text for the Subject line, since the Webhook is triggered by some event on the Particle device. We encode whatever data needs to be sent to the user in the "data" part of the Particle.publish() and format up the specific message text in the Google App Script. For example:

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

function tppNotify(e) {
  const targetNumber = "<mobile phone number>@txt.att.net";
  const subj = "Well Pump Trigger Notification";
  var eventTxt = e.parameter.data;  // the data sent in the Particle.publish()
           ….

  var wsmData = JSON.parse(eventTxt);
           …

  var wp = wsmData.wp ;
           …
```

© 2021, Team Practical Projects, Jim Schrempp and Bob Glicksman; all rights reserved. *Subject to "Terms of Use, License and Disclosure"*

```
  var wtm = wsmData.wpon ;
          …

  if(wp == 0) {             // well pump has turned off
    var messageTxt = "Well pump ran for " + wtm;
    messageTxt += " minutes";
  }
  else {            // well pump has turned on
    var messageTxt = "Well pump turned on at " + loctm;
  }

  MailApp.sendEmail(targetNumber, subj, messageTxt);

}
```

This Google App Script fragment (from our Well System Monitor project) receives JSON encoded data produced on a Particle device into the string variable “eventTxt”. The JSON-encoded “eventTxt” is parsed and the individual data items are stored into their own variables. This webhook event is actually used for logging well system data as well as for text message alerts. The “if(wp==0)” conditional determines if a new well pump status was sent from the Particle device to the webhook and, if so, which message to send (“well pump activated” message or “well time run-for time message”).

3.4. Device – Device Secure Wireless Communication.

Particle.publish() and Particle.subscribe() can also be paired on Particle devices. Devices can publish results to the Particle Cloud and devices can subscribe to these publications. The devices must be in the same Particle account. The process is shown in figure 3-4.

© 2021, Team Practical Projects, Jim Schrempp and Bob Glicksman; all rights reserved. *Subject to “Terms of Use, License and Disclosure”*

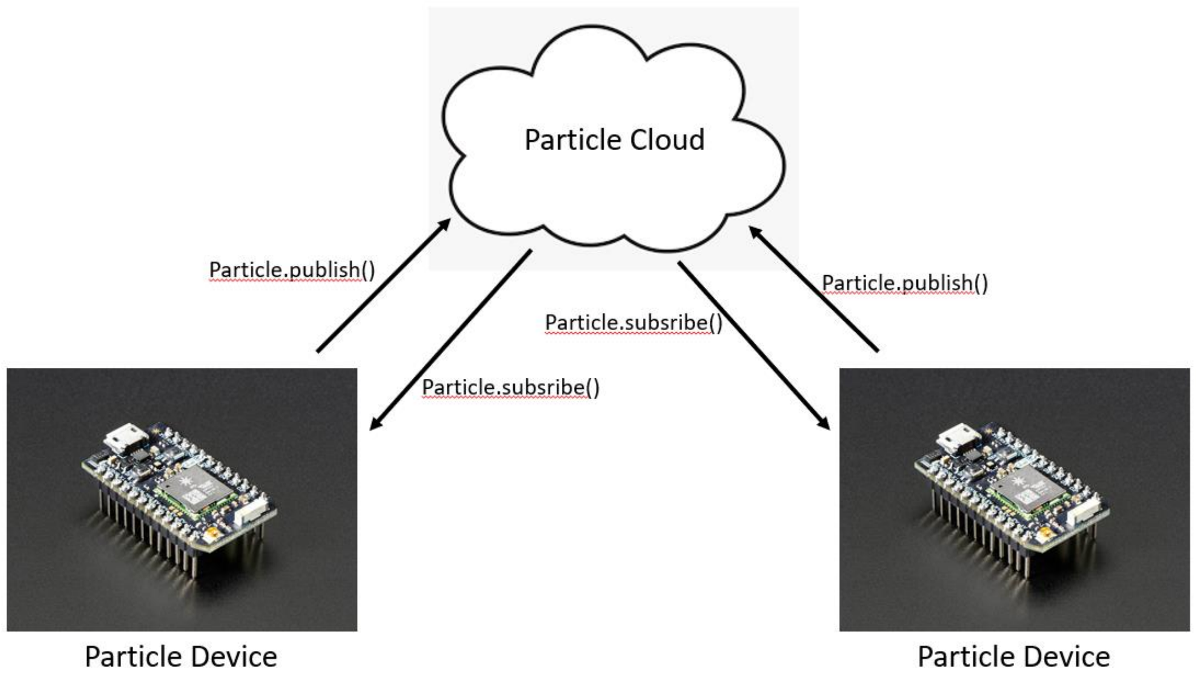


Figure 3-4. Device to Device Secure Wireless Communication.

The point of this integration is the simple and straightforward way to create wireless, secure communication between Particle devices that are physically located anywhere in the world where Internet access exists. An example of using this capability is our RFID card access system that we developed for Maker Nexus; see:

https://github.com/TeamPracticalProjects/MN_ACL

This project provides various types of RFID access stations throughout the Maker Nexus facility. Members are provided with RFID cards that are used to check them into and out of the facility and are also used to provide members with access to various locations within the facility (based upon their training credentials). One RFID station is located at the front door and, if the user's card is verified, unlocks the door using a solenoid embedded on the door's push bar.

Since the RFID Station must be easily accessible to users (to tap their cards), it would be insecure to have the Particle device in the station activate the door solenoid via some wired connection (e.g. a relay). It would be too easy for a hacker to access the Station hardware and wire in their own solenoid control circuit in parallel with the Station's hardware.

The solution to this security problem is to have the RFID Station publish a message stating that front door access has been granted. An RFID Lock control board, located in a secured IT room, subscribes to these events and triggers the front door solenoid accordingly. Because a cryptographically secure Internet connection exists between publisher and subscriber, only someone with access to the system's Particle account (or to a Particle device claimed into this account) can trigger the front door to open.

© 2021, Team Practical Projects, Jim Schrempp and Bob Glicksman; all rights reserved. *Subject to "Terms of Use, License and Disclosure"*

All RFID Stations at Maker Nexus contain Particle devices that are claimed into the Maker Nexus access control account. It is difficult, but perhaps possible, for a hacker to reprogram one of these devices to publish the activation message. Over-the-air flashing of new firmware to a Particle device requires access to the secure Particle account, but perhaps a hacker with physical access to a Particle device (that is claimed into the secure Particle account) could force their own firmware onto the device and thus publish an activation message. To protect against this (unlikely) scenario, we added a secret key to the publication message. The RFID lock, responding to the unlock door event, checks for the secret key before unlocking the door. The hacker cannot learn the secret key (published with the unlock event) because the publication itself is encrypted. In this manner, wireless communication between the RFID Lock device and the RFID Station device is secured.

3.5. Integrating a Cloud Database.

Sometimes we need to communicate with a third party cloud service or, in fact, with our own database hosted in the cloud. Our RFID project for Maker Nexus contains examples of both of these integrations; see:

https://github.com/TeamPracticalProjects/MN_ACL

Integrating with a third party cloud service is relatively easy if the service in question offers a REST API. Particle Webhooks can be used to perform the necessary http GET, POST, PUT and other such operations. The Webhooks are triggered by Particle.publish() calls from the Particle device. In addition, devices can Particle.subscribe() to Webhook responses if the external cloud service returns data needed by the device. All that is needed to respond to a Webhook response is a callback function that processes the hook response. The callback function is just like any other function on the Particle device and the function argument is a String that can contain simple data or more complex, encoded data (json, XML, CSV, etc).

The details for third party cloud service integration depends upon the third party service, of course. The process is best illustrated here by integration with our own database hosted in the cloud. This process is illustrated in figure 3-5, below.

© 2021, Team Practical Projects, Jim Schrempp and Bob Glicksman; all rights reserved. *Subject to "Terms of Use, License and Disclosure"*

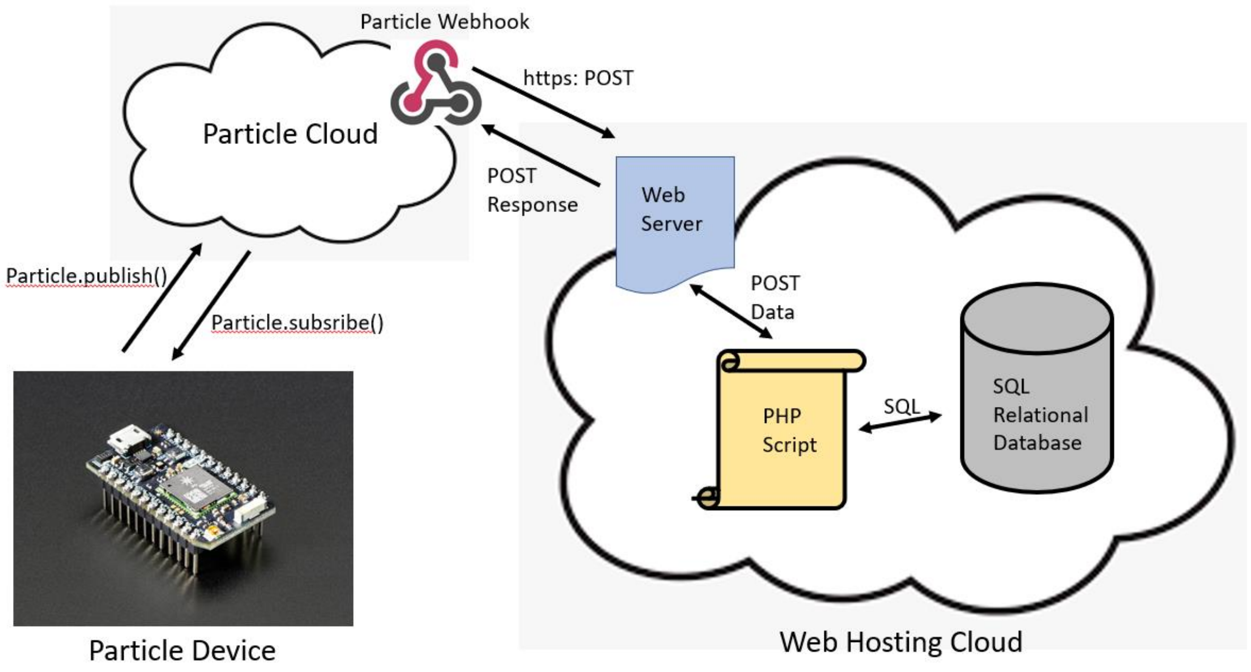


Figure 3-5. Communicating with a Cloud Database.

The process in figure 3-5 is similar to logging to a Google cloud spreadsheet, as previously described in section 3.2. of this document. In lieu of a Google App Script, POSTs from the Webhook target a PHP script running on the cloud hosting service. The PHP script performs the necessary queries on the database.

We use tools such as cURL and Postman to test third party APIs and our own PHP scripts prior to integration with the Particle Cloud. Once we know how to work with the external REST API calls, we then integrate with the Webhooks that we create in the Particle Cloud.

© 2021, Team Practical Projects, Jim Schrempp and Bob Glicksman; all rights reserved. *Subject to "Terms of Use, License and Disclosure"*

APPENDIX – PUBLISHING GOOGLE APPS SCRIPT AS A WEB APP.

In order to create a new Google App Script, go to Google Docs in your account and select “More > Google Apps Script”; see figure A-1. If you don’t have Google Apps Script in your Drive, you can add it with “+ Connect more apps”.

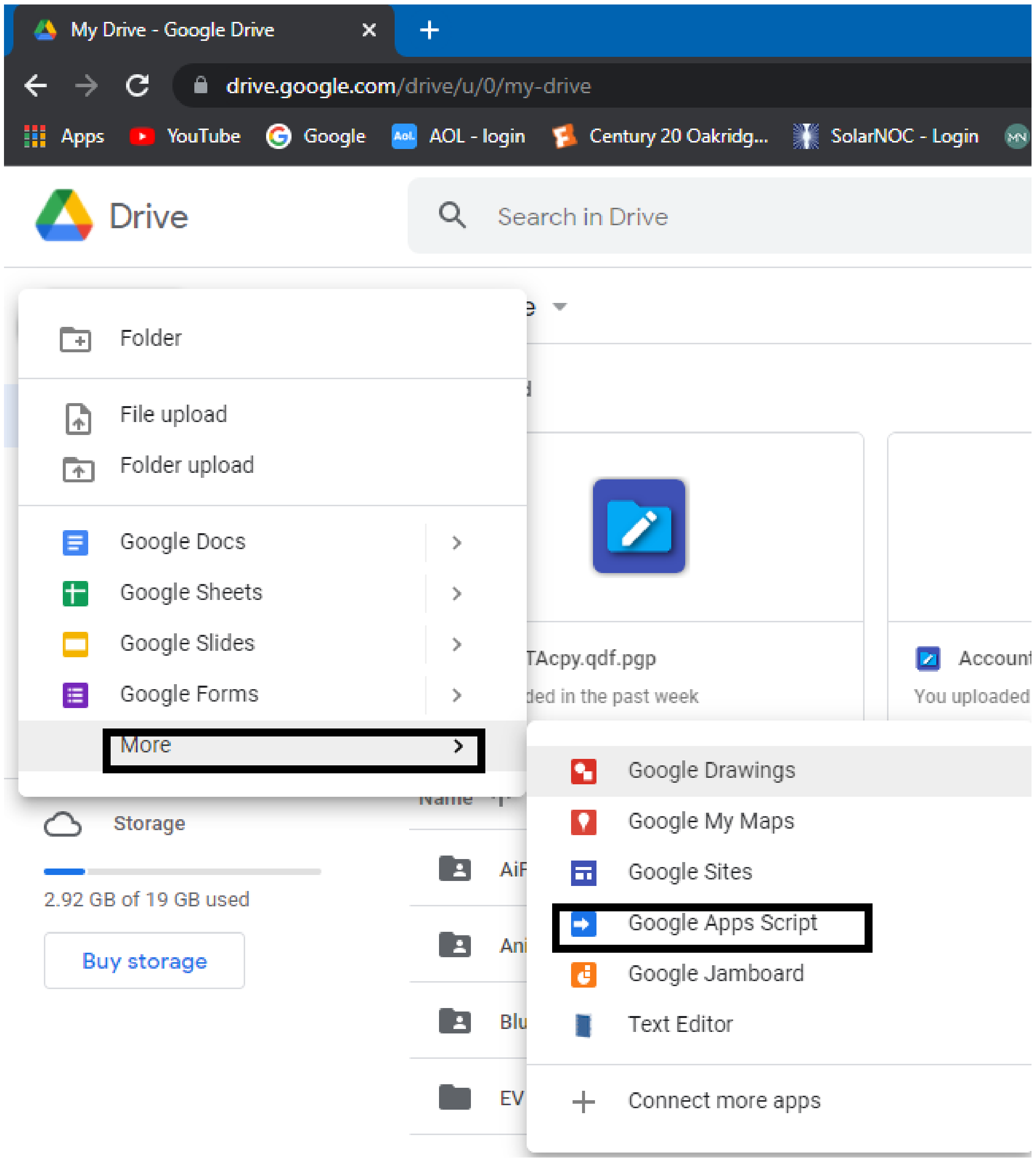
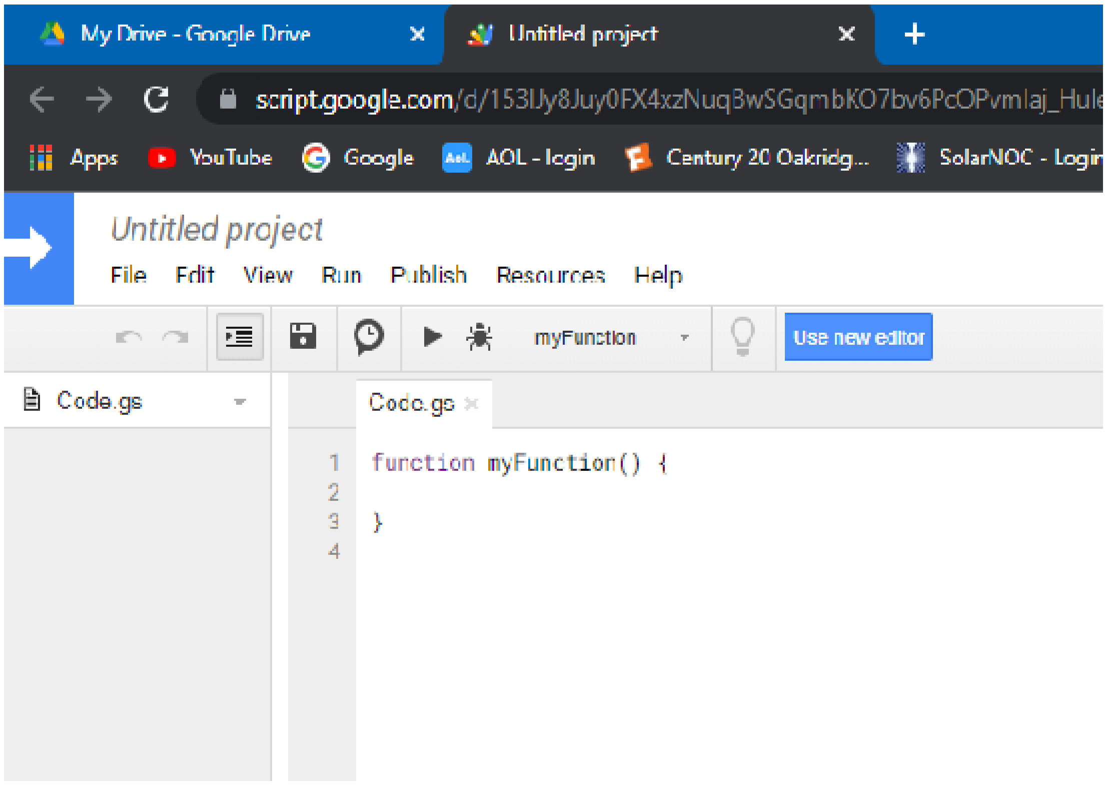


Figure A-1. Opening a New Google App Script.

After selecting “Google Apps Script”, you will get an opening screen similar to figure A-2, below.

© 2021, Team Practical Projects, Jim Schrempp and Bob Glicksman; all rights reserved. *Subject to “Terms of Use, License and Disclosure”*



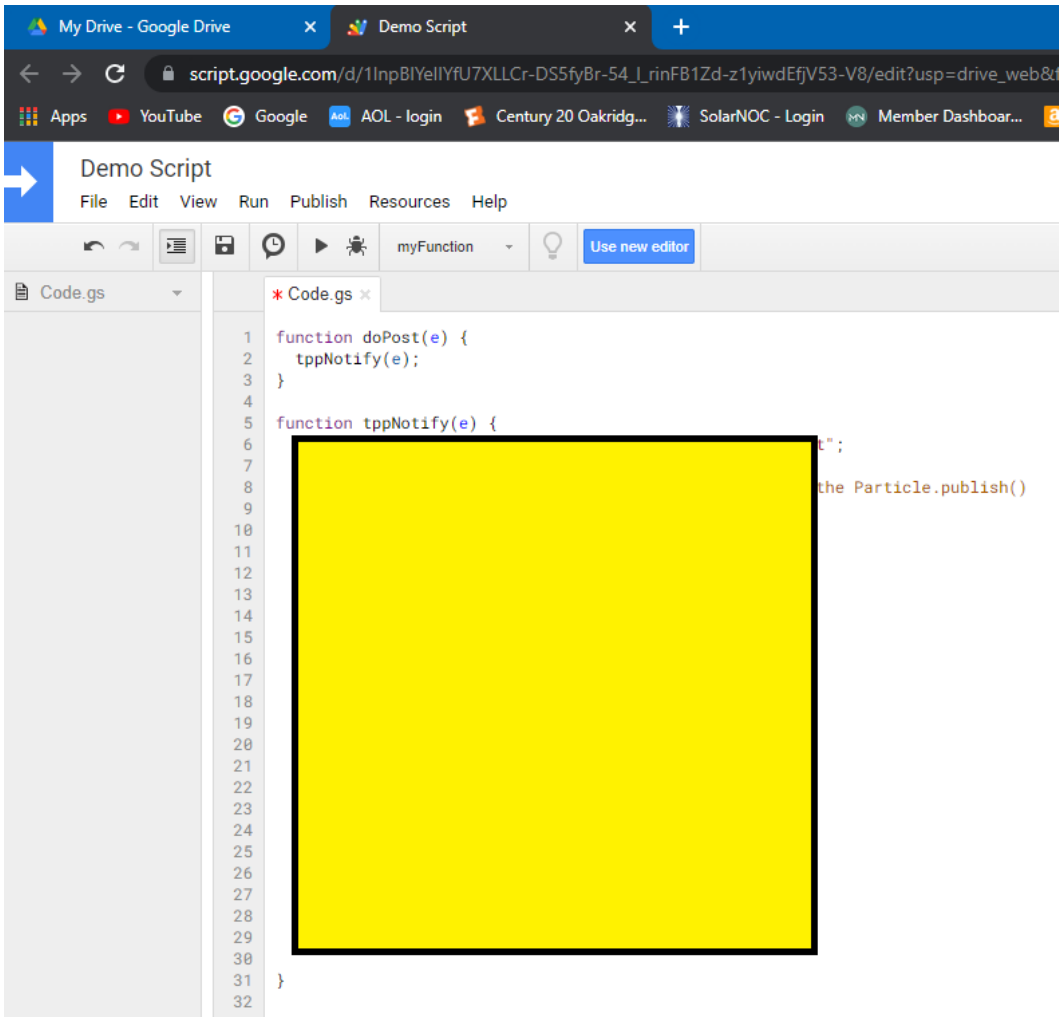
```
function myFunction() {

}
```

Figure A-2. Opening Screen.

If your screen looks different, you are probably in Google's new editor. In order to follow these instructions, select "Use legacy editor" in the menu. You will next want to give your project a name by clicking on "Untitled project" and entering the new project name in the pop-up dialog box, then clicking "OK".

In order to create a web app, you will need to delete the default "function myFunction() { }" and enter your own function as "function doGet(e) { … }" or "function doPost(e) { … }" (or both), as described in section 3.2. You then write in your code; something like figure A-3, below and "File | Save" it. See figure A-3 for example.

© 2021, Team Practical Projects, Jim Schrempp and Bob Glicksman; all rights reserved. *Subject to "Terms of Use, License and Disclosure"*

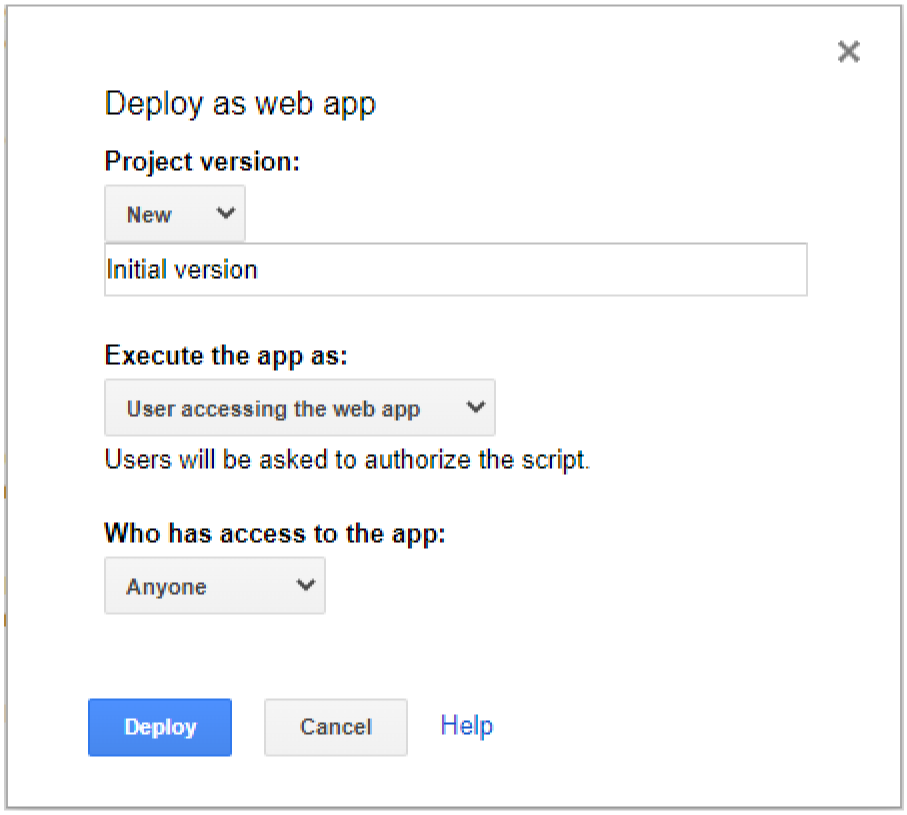


```
function doPost(e) {
  tppNotify(e);
}

function tppNotify(e) {
[redacted]t";

the Particle.publish()

}
```

Figure A-3. Demo Function in Script Editor.

When you have finished editing and saving your script, click on “Publish | Deploy as web app …” in the menu bar. A “Deploy as web app” window will appear; see figure A-4.

© 2021, Team Practical Projects, Jim Schrempp and Bob Glicksman; all rights reserved. *Subject to “Terms of Use, License and Disclosure”*



Deploy as web app

Project version:

New

Initial version

Execute the app as:

User accessing the web app

Users will be asked to authorize the script.

Who has access to the app:

Anyone

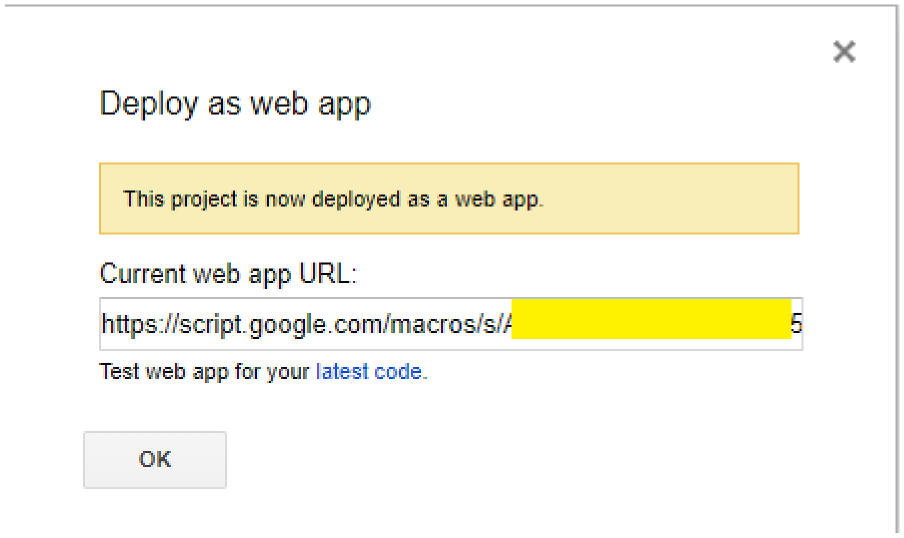
Figure A-4. Deployment Window.

Referring to figure A-4:

- Under "Project version:" type in a description of the version, such as "Initial version" shown in the figure. You can leave the initial version number as "New". However, if you revise your script after initial testing, be sure to bump the version to a new number (1, 2, etc.).
- Under "Execute the app as:" select "User accessing the web app". This will ensure that your Webhook will have access to the app.
- Under "Who has access to the app:" select "Anyone". This means that anyone who has the app's url can access it, so keep the app url private.
- Click on "Deploy".

The next popup to appear will be similar to figure A-5.

© 2021, Team Practical Projects, Jim Schrempp and Bob Glicksman; all rights reserved. *Subject to "Terms of Use, License and Disclosure"*


Deploy as web app

This project is now deployed as a web app.

Current web app URL:

https://script.google.com/macros/s/A[redacted]5

Test web app for your latest code.

Figure A-5. Deployed App URL.

Your Google App Script is now deployed. You will see it in your Google Drive. Referring to figure A-5, copy out the app's url from "Current web app URL:" and save it in a safe place. This is the url that you will place into your Particle Webhook as the Webhook target url.

Click on OK when you are done copying the url. The Google App Script is now ready to use.

© 2021, Team Practical Projects, Jim Schrempp and Bob Glicksman; all rights reserved. *Subject to "Terms of Use, License and Disclosure"*