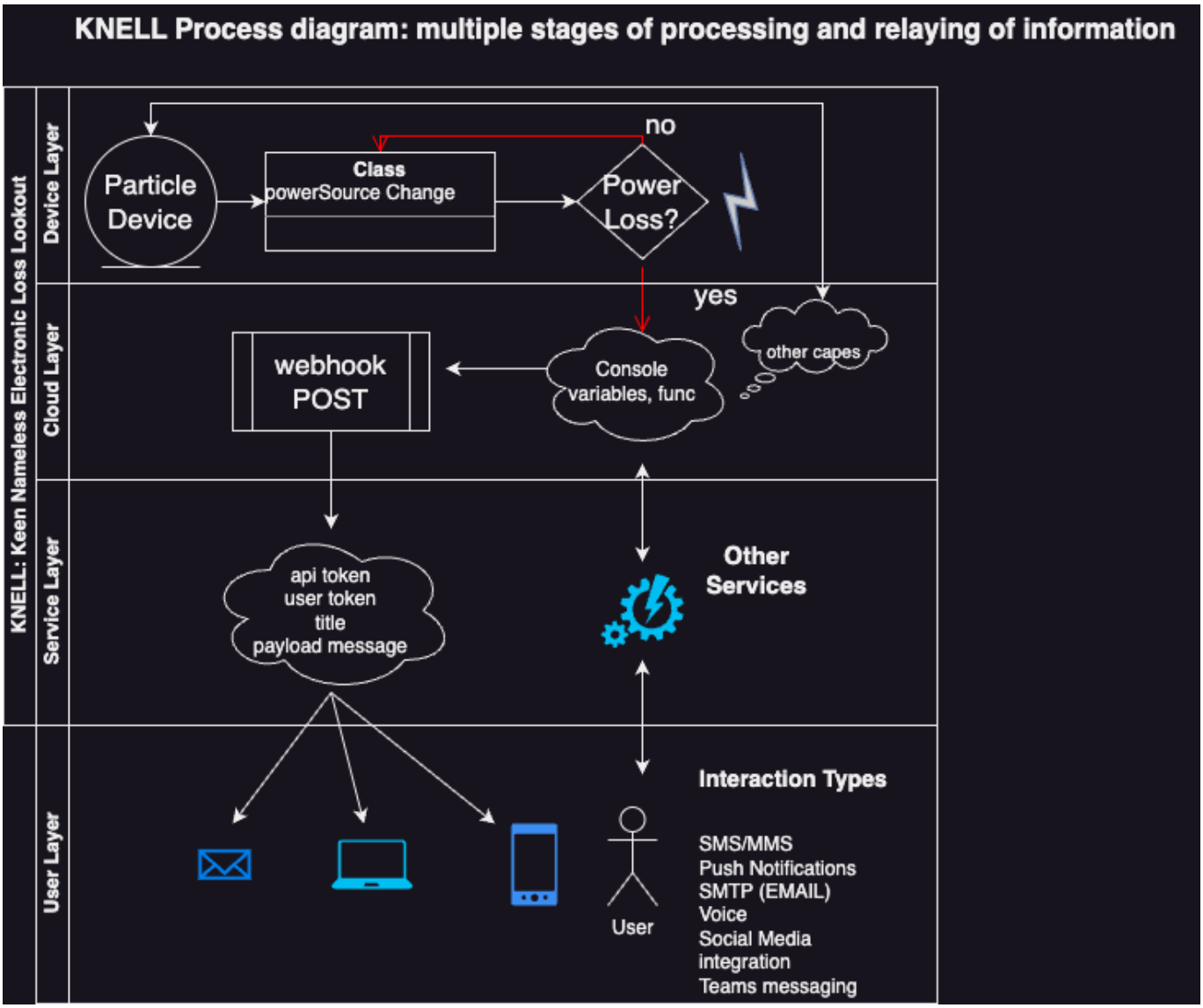
KNELL & KNESL: Nameless Electronic Lookouts for power loss, temperature changes, and more!

A cellular Internet of Things (IOT) device that can detect power loss from wall AC. Designed as a system as a whole to send out notifications to facility managers/ Staff of interest so that memebers can perform graceful shutdown of key Information Technology infrastructure and services within alloted Uninterrupted Power Supply (UPS) alloted time.

System description

The entire system is composed of a sensor layer, gateway layer, a cloud layer, a service layer, and a user layer. See the following diagram for a visual model: Markup :



Account credentials

Currently, the system operates off of a single particle account (controlled by the primary developer) that is used for device monitoring and control of the cloud events and a single user account through the Pushover.net service: (https://pushover.net/)

There are no future plans to expand the number of accounts directly interfacing with Particle. If the future dictates that end users want more access to the particle cloud a seperate client (under the authentication tab) will be created for them to interact with the data.

Account login details are stored in the 705 CTS government only drive under Facility management/KNELL_Facility_Monitoring_System

Push notification service: Pushover

Markup :

Why pushover? It is dead simple and cheap!

Users install an application onto their smartphone and receive push notifications with alert sounds.

Google play: (https://play.google.com/store/apps/details?id=net.superblock.pushover/)

Apple store: (https://apps.apple.com/us/app/pushover-notifications/id506088175/)

Ultimately, the government sponsoring this development was very risk adverse and unwilling to spend money so we compromised on reoccuring service charges and use a singular, paid, pushover account to deliver push notifications. Thus, when registering or setting up new users, you simply give them the account credentials mentioned above for the facility team and let them configure other settings in the app as it suits them.

Example process:

1. New user requests access to pushover push notifications.
2. User meets with facility management team who instructs them to install application and provides or enters login information into the pushover application.
3. The new user receives the push notifications!

Developer information and documentation

Particle software orginization

`/src` folder:

This is the source folder that contains the firmware files for your project. It should *not* be renamed. Anything that is in this folder when you compile your project will be sent to our compile service and compiled into a firmware binary for the Particle device that you have targeted.

If your application contains multiple files, they should all be included in the `src` folder. If your firmware depends on Particle libraries, those dependencies are specified in the `project.properties` file referenced below.

`.ino` file:

This file is the firmware that will run as the primary application on your Particle device. It contains a `setup()` and `loop()` function, and can be written in Wiring or C/C++. For more information about using the Particle firmware API to create firmware for your Particle device, refer to the Firmware Reference section of the Particle documentation.

`project.properties` file:

This is the file that specifies the name and version number of the libraries that your project depends on. Dependencies are added automatically to your `project.properties` file when you add a library to a project using the `particle library add` command in the CLI or add a library in the Desktop IDE.

`models` folder:

This folder contains all the 3D CAD, OBJ, STL files used in the outdoor enclosure.

Current libraries in use

DiagnosticsHelperRK : a library to access lower level things inside of the Device OS that particle boards run. Specifically, we use it to read the system power source. Credit to rickkas7: https://github.com/rickkas7/DiagnosticsHelperRK MIT license type so can be utilized for proprietery/ commercial use.

Projects with external libraries

TODO: we need to restructure the simple library use in this repo to match the below:

If your project includes a library that has not been registered in the Particle libraries system, you should create a new folder named `/lib/<libraryname>/src` under `/<project dir>` and add the `.h`, `.cpp` & `library.properties` files for your library there. Read the Firmware Libraries guide for more details on how to develop libraries. Note that all contents of the `/lib` folder and subfolders will also be sent to the Cloud for compilation.

Device setup/ Hardware

Hardware required:

BORON 404x LTE (USA) Markup :

STOCK ANTENNA: Markup :

Battery (any 3.7 LIPO): Markup :

Follow the Boron 404x quickstart: https://tools.particle.io/setup Specifically, there may be some issues with the default firmware/ version of device OS that came installed on the device. Please visit the device doctor for diagnosing any problems encountered with the device not connecting to the cloud properly: https://docs.particle.io/tools/doctor/

Once firmware is flashed properly you should have success for connection to the particle cloud. It is recommended you download the mobile app onto your device and log onto the particle cloud console https://console.particle.io/ to see your device connected.

From here you can do any of the LED flash setups/ tests to ensure you are understanding the Particle object etc. part of this code **currently** maintains some test code and explanations on how to use the Particle objects/ cloud.

Future updates to hardware will include reference schematics for easy build out.

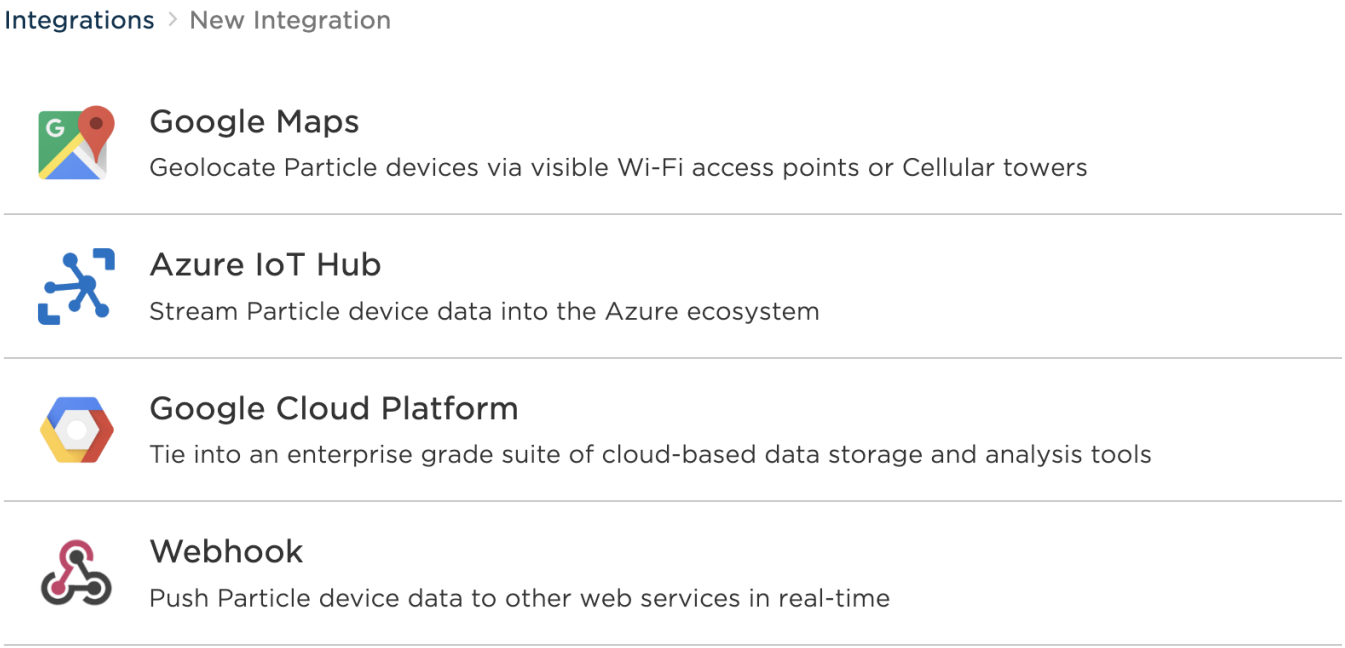
Particle Console setup and webhooks

This sections explains the particle console and how to setup basic webhooks.

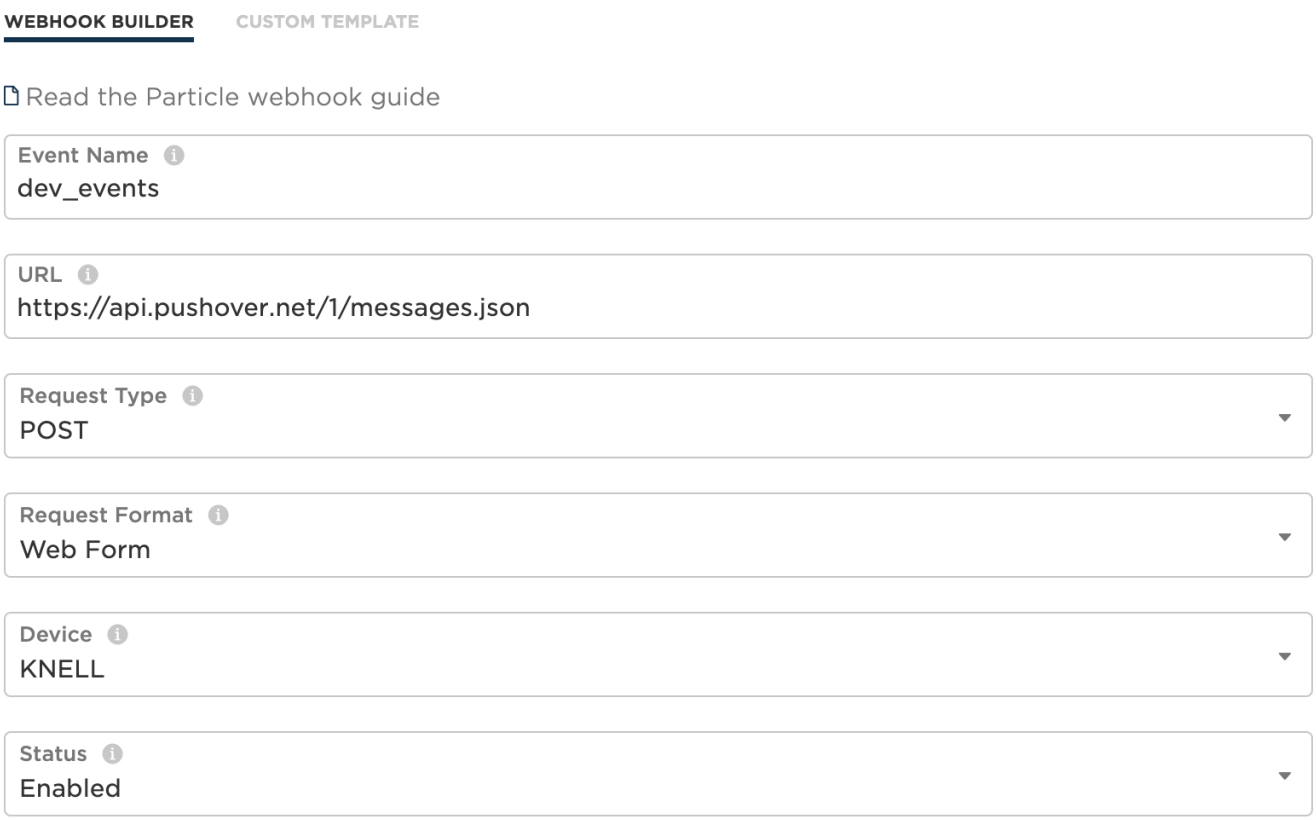
Login with the developer credentials to https://console.particle.io/devices Ensure that the device is registered properly and is active. This should have also been done in the Device setup/ Hardware section.

Navigate to the *Integrations* section. Markup : picture alt

Click new integration to create a new webhook or integrate another service: Markup :



Click webhook and you will enter into the webhook builder. This is where you will create the POST or GET form request to interface into your application. For pusher, we create a POST request and model it off of our basic DEV event one: Markup :



Copy the information in the picture above and adjust the event name and target device or device groups as needed. If you are doing something custom you can copy the format but will definately diverge in the API URL and second part of the webhook builder!

NOTE on event name This must correspond to the event name that your device firmware is pushing! For example, our event name is dev_events in our particle cloud and in the device firmware! Markup :

```
      You, last month • working event detection for power loss
void dev_tests() {
    String devStr = "HELLO WORLD";
    String test_status = String::format("{\"DEV\":\"%s\"}", devStr.c_str());

    bool tst_success = Particle.publish("dev_events", test_status, PRIVATE, WITH_ACK);
    while(!tst_success) {
        // get here if event publish did not work, reattempt
        tst_success = Particle.publish("dev_events", test_status, PRIVATE, WITH_ACK);
      }

}
```

Now that you are in the second part or "advanced" tab in the webhook builder you will create the form POST request for Pushover. Have your Particle API key and Pushover user key (details above in this readme and details on how to generate a new Pushover API key in next section) and add them as the first two fields with exactly the same field names as above. The title can be whatever title message you want.

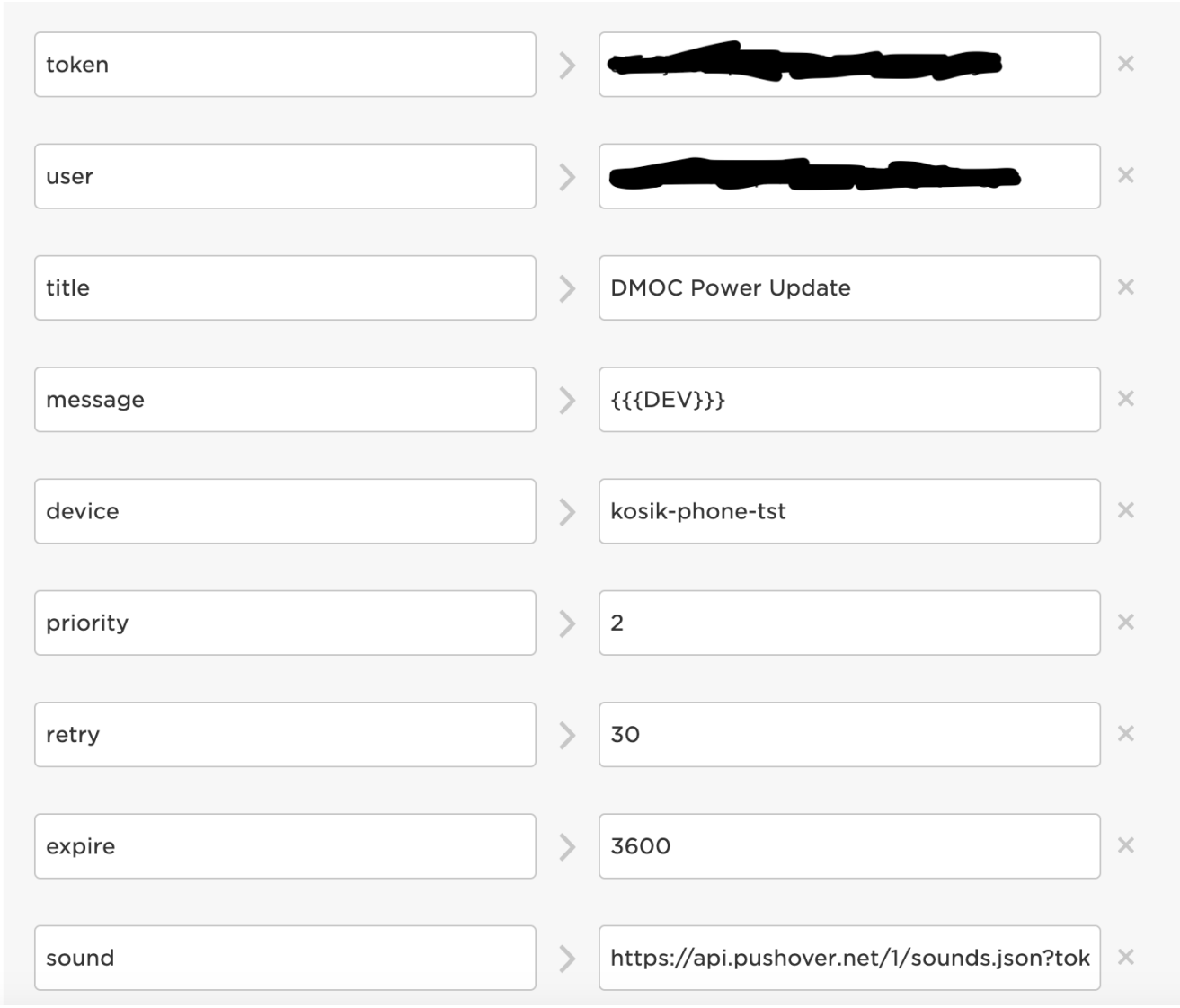
The Interesting part Message: this is where the payload information from the device comes in. You will need to get the information from the Javascript Object Notation, or JSON, format the device sends up! To do this we must unwrap the JSON using the curly brackets shown around the String payload message we have defined "DEV": Markup :


```
      You, last month • working event detection for power loss
void dev_tests() {
    String devStr = "HELLO WORLD";
    String test_status = String::format("{\"DEV\":\"%s\"}", devStr.c_str());

    bool tst_success = Particle.publish("dev_events", test_status, PRIVATE, WITH_ACK);
    while(!tst_success) {
        // get here if event publish did not work, reattempt
        tst_success = Particle.publish("dev_events", test_status, PRIVATE, WITH_ACK);
      }

}
```

Markup :



Other very important notes: The extra fields shown in the example above can be edited as needed or ommitted. In our example we specify only one device to be tied to the cloud webhook. Additional paramaters specify timeout and retry attempts. To send to all devices, leave it blank!

Pushover.net configuration

Most of the time you will not need to configure Pushover.net. The single API key should allow new events to be added on without concern.

API key generation in Pushover.net

1. Login to (https://Pushover.net/)
2. Scroll down and click "generate an Application/API Token"
3. Fill out the name and description
4. Store the infomation in the government only drive or appropriate location!

Future work

Honestly quite a bit. Comprehensively KNELL & KNESL should agree of UUID's and integrations into other services. See each components README for a list/sublist of todo's. None of this incldues rewrites/ making it

more readable etc 😃