Putting the I in IoT: Functions

44-440/640-IoT

Objectives

- Students will be able to
  - invoke firmware functions from a web page
  - explain the difference between Particle and firmware functions
  - explain how POST works in the context of HTTP
  - send POST messages using curl and webpages

Overview

Physically, there are a myriad ways to make the device-internet connection, e.g., an Arduino + wi-fi shield; an Arduino Yún, Raspberry Pi, Tessel, or, ... Photon! The nice thing about the Photon is that it makes the network connection very, very easy. We will gladly use that power, but later on in the course we will develop our own connectivity solutions, bypassing the Photon cloud.

- We now know how to retrieve information *from* the Photon, thanks to Particle variables and GET. Now we are going to learn how to *securely control* the Photon from the internet (bwahahahaha)

- The key to all this magic is another cloud function:

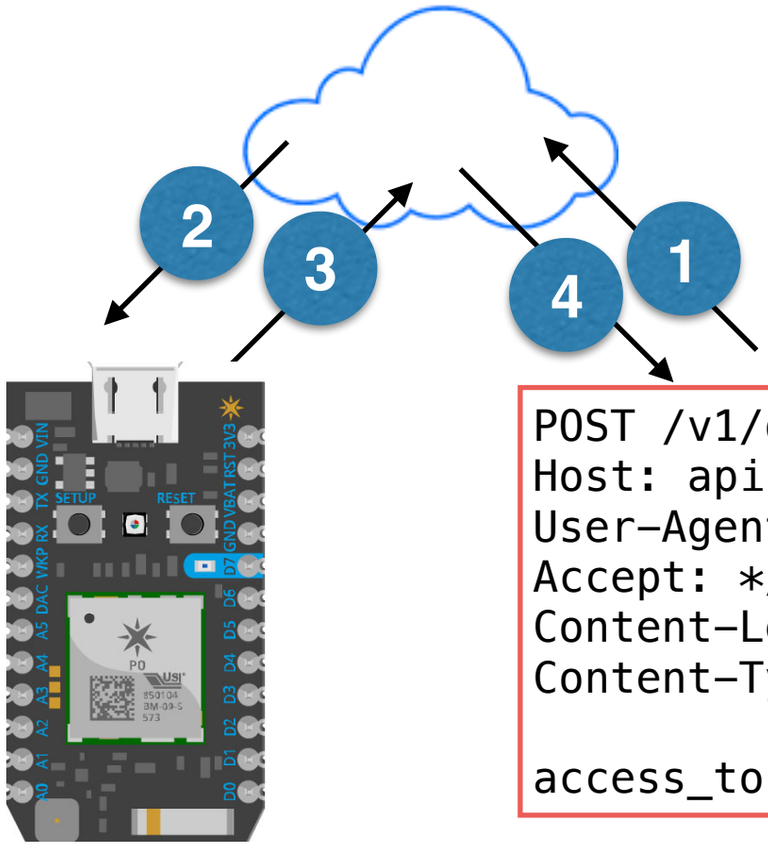
  - **Particle.function()** exposes firmware functions so that they can be invoked over the internet via a POST message

The Magic of Particle.function()

- **Particle.function(*particleFunctionName, function*)** exposes firmware functions so that they can be invoked over the internet via a POST message
    - *particleFunctionName* is a string (≤ 64 chars) used in the POST URL
    - *function* is the firmware function to be accessed
- It is possible to register up to 15 cloud functions
- **Cloud function / Particle function** - the name by which the firmware function is known on the Particle cloud (e.g., "on")
- **Firmware Function / Function** - the function as defined in code (e.g, turnOn(string))

Particle.function(), Illustrated

```
int redLED = D0;

void setup() {
    pinMode(redLED,OUTPUT);
    Particle.function("fireItUp",turnOn);
    Particle.function("shutItDown",turnOff);
}

void loop() {}

int turnOn(String message){
    digitalWrite(redLED,HIGH);
    return 0;
}

int turnOff(String message){
    digitalWrite(redLED,LOW);
    return 0;
}
```



1. POST sent to Cloud
2. Cloud relays to Photon
3. Photon performs action & returns result
4. result is forwarded to the POSTER

```
POST /v1/devices/ddddd/fireItUp HTTP/1.1
Host: api.particle.io
User-Agent: curl/7.43.0
Accept: */*
Content-Length: 53
Content-Type: application/x-www-form-urlencoded

access_token=aaaaa&args=just%20ignore%20me%
```

- Particle.function() takes two Strings - a particle function name that will be included in a POST message, and the name of the firmware function (an **int** function with a **String** parameter).

- When a POST message is sent with the particle function name, the corresponding firmware function will be triggered. It's magic!

What Just Happened

- Photon sends a POST message to invoke a firmware function, based on this URL:

https://api.particle.io/v1/devices/**DEVICE_ID**/**FUNCTION**

  - **DEVICE_ID** is the Photon's device id
  - **FUNCTION** is the first argument in Particle.function()

- The message body (or authorization header) must define an **access_token** and may *optionally* define **args**, which is supplied as an argument to FUNCTION.

This sounds important …

Particle Variables v. Functions: The Joys of Consistency

GET

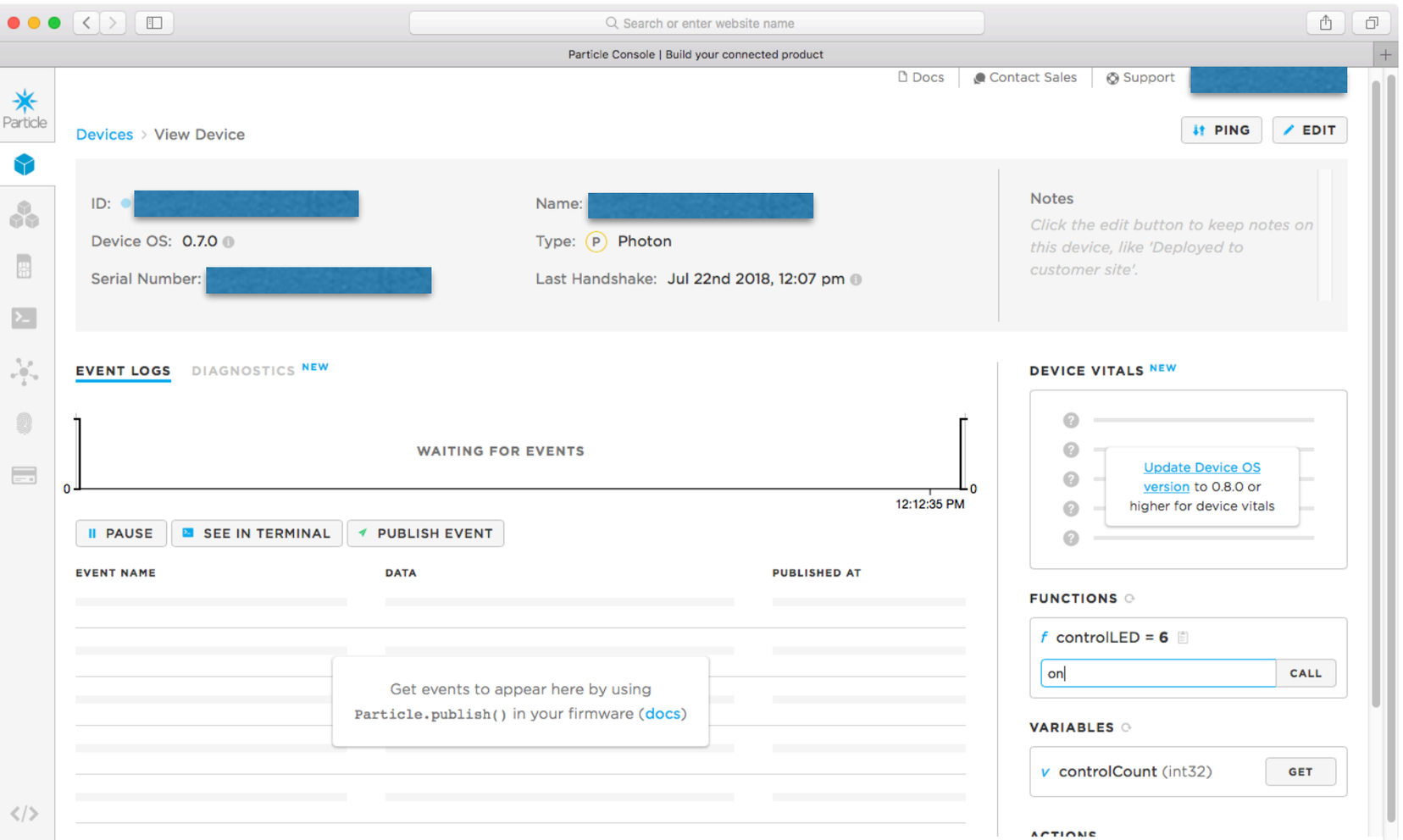
https://api.particle.io/v1/devices/**DEVICE_ID/VARIABLE**?access_token=**ACCESS_TOKEN**

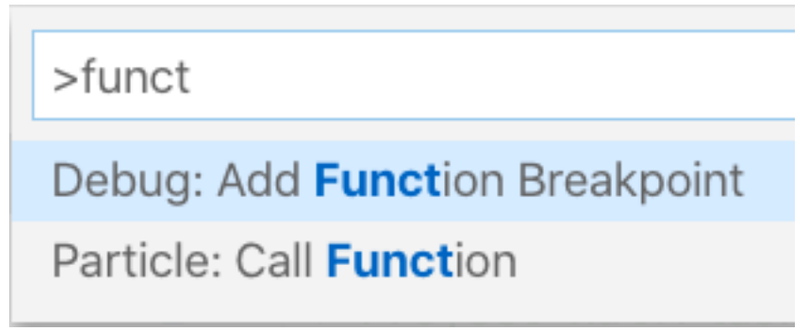
POST

https://api.particle.io/v1/devices/**DEVICE_ID/FUNCTION**

Debugging Particle Functions

- For debugging purposes we can trigger functions via the particle.io console and in Particle Workbench





Sending a POST Message

- We can send POST messages using one of 4 strategies:

  1.HTML forms

  2.curl (command url)

  3.code (e.g., JavaScript)

  4.Postman

- Students are more likely to have seen forms, but they have 2 fatal flaws -- the user will be able to see the access token, and the result from the POST message replaces the form, and the end-user may be startled by the sudden appearance of an Ugly JSON Object (UJO)™

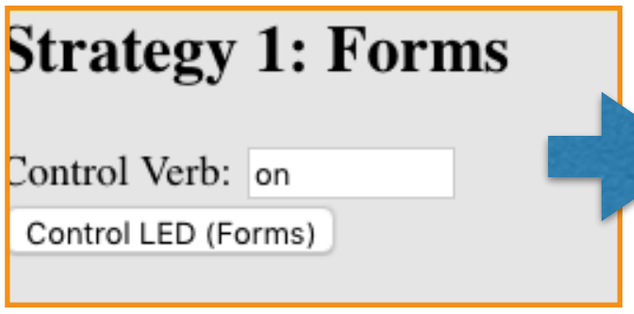
- JavaScript uses an XMLHttpRequest object to send a POST message, and can then parse the result to change an html element on the page (e.g., to show output), causing less distress to the end-user.

Clean this up starting with this slide ...

Hey, kids, follow along, it's fun for the whole family!

Can you spot the security hole?

Strategy 1: HTML Forms



When the user taps the button ...

```
{"id":"ddddd",
"last_app":"",
"connected":true,
"return_value":1}
```

... this is what they will see. Yuck!

```
<h2>Strategy 1: Forms</h2>
<form method="POST" action="https://api.particle.io/v1/devices/ddddd/controlLED">
Control Verb: <input type="text" size="10" name="args"/><br>
<input type="submit" value="Control LED (Forms)"/>
<input type="hidden" name="access_token" value="aaaaa"/>
</form>
```

1

```
POST /v1/devices/ddddd/controlLED HTTP/1.1
Host: api.particle.io
Content-Type: application/x-www-form-urlencoded
Origin: file://
Content-Length: 63
Connection: keep-alive
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
User-Agent: Mozilla/5.0
Accept-Language: en-us
DNT: 1
Accept-Encoding: gzip, deflate

args=on&access_token=aaaaa      // this gets sent to the server;
                                // on is passed to controlLED(), in messages
```

2

3

4

5

Today's Fun Fact! while the APIs use args, any name will work. 🤔

```
const int internalLED = D7;
bool flashLED = false;
int controlCount = 0;

void setup(){
    pinMode(internalLED, OUTPUT);
    Particle.function("controlLED", controlLED);
    Particle.variable("controlCount",controlCount);
    Serial.println(9600);
}

void loop(){
    if(flashLED){
        digitalWrite(internalLED,HIGH);
        delay(500);
        digitalWrite(internalLED, LOW);
        delay(500);
    } else {
        digitalWrite(internalLED,LOW);
    }
}

int controlLED(String message){
    controlCount++;
    Serial.println(message);
    if(message == "on")
        flashLED = true;
    else if(message == "off")
        flashLED = false;
    else
        flashLED = !flashLED;
    return controlCount;
}
```

The parameter could also be const char * -- see this discussion for the nitty-gritty details

Strategy 1: How It Works

1. The user clicks on a button, generating a POST request

2. The request is sent to the Particle Cloud

3. The Particle Cloud, based on the particle function, invokes the function on the device

4. The function returns a value, which is sent back to the cloud

5. The result of the entire operation is sent back to the user (and the contents of the form get replaced by an Ugly JSON Object 😕)

Strategy 1 Remarks

- When using an html form, all the inputs are bundled into the body of the POST message as ampersand-delimited name=value pairs*

- Because we have two of them:

```
<input type="text" size="10" name="args" value="on"/>
<input type="hidden" name="access_token" value="xx"/>
```

- args=on&access_token=xx will be POSTed to the server

- If the user enters something else in the text field, that will appear instead of the default value, on

- We hide the access_token, but a) it still gets sent to the server b) the user will be able to see it if they select View Source, so **do not use this on a real web site** and (even unpublished on Thimble), **change your access token after you've run it**.

*content-type:application/x-www-form-urlencoded

Strategy 1 - POST Request and Response

```
POST /v1/devices/xxxxxxx/controlLED HTTP/1.1
Host: api.particle.io
User-Agent: curl/7.52.1
Accept: */*
Content-Length: 65
Content-Type: application/x-www-form-urlencoded

args=on&access_token=xx


HTTP/1.1 200 OK
Date: Tue, 26 Sep 2017 00:52:45 GMT
Content-Type: application/json; charset=utf-8
Content-Length: 81
Connection: keep-alive
Server: nginx
X-Request-Id: f21c8ae9-3aa3-400b-ac78-e0290198acc0
Access-Control-Allow-Origin: *


{"id":"xxxxxxxx","last_app":"","connected":true,"return_value":0}
```

POST Request

POST Response

Strategy 2 - curl

```
curl -d "access_token=aaaaa&args=on" https://
api.particle.io/v1/devices/ddddd/fireItUp
```

- It allows you to send POST and GET messages to api.particle.io

- The -d option causes data to be sent as a POST message (the default is GET)

- Replace the aaaaa's and ddddd's with youraccess_token and device ID, respectively

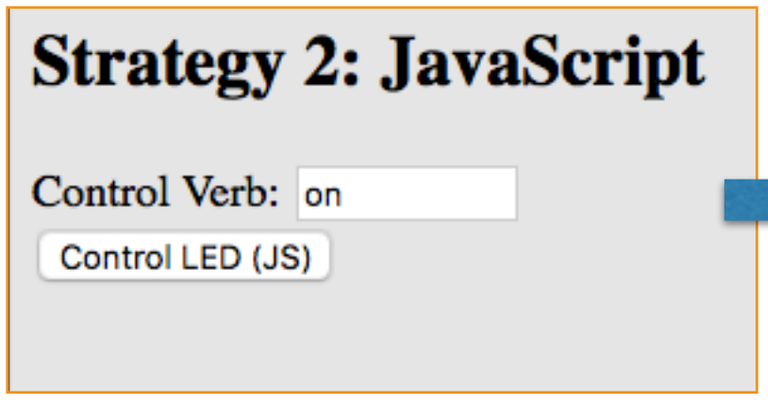
Strategy 3 (Avoiding the Ugly JSON Object): Posting with JavaScript

```
let url = "https://api.particle.io/v1/devices/ddddd/controlLED"

function controlLED(){

  let deviceID = prompt("Enter device ID")
  let accessToken = prompt("Enter access token")
  url = url.replace("ddddd",deviceID)

  let controlVerb = document.getElementById("controlVerb").value
  let xhttp = new XMLHttpRequest()
  xhttp.open("POST", url, true)
  xhttp.setRequestHeader("Content-type", "application/x-www-form-urlencoded");
  xhttp.send("access_token=" + accessToken + "&args=" + controlVerb)
  xhttp.onreadystatechange = function() {
    if (this.readyState == 4 && this.status == 200) {
      let resp = JSON.parse(this.responseText)
      document.getElementById("demo").innerHTML = resp.return_value
    }
  }
}
```

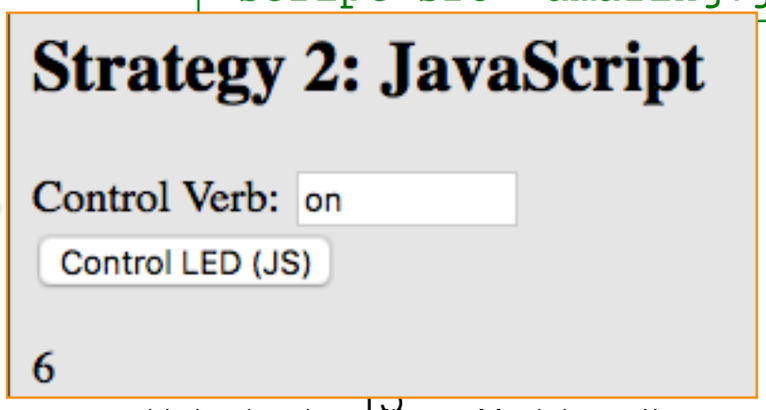
send() causes its argument to appear in the body of the POST message

```
{"id":"xxxxxxx","last_app":"","connected":true,"return_value":0}
```

```
<h2>Strategy 2: JavaScript</h2>
Control Verb: <input type="text" size="10" name="args"
                value="on" id="controlVerb"/><br>
<button onclick="controlLED()">Control LED (JS)</button>
<p id="demo"></p>
<script src="amazing.js"></script>
```



When the user taps the button ...



... this is what they will see. Much better!!

<script> is needed ... otherwise nothing works.

Strategy 3 Remarks

- XMLHttpRequest is a JS object that can send POST (or GET) messages

- send() is used to send something in the body of the message (POST) or as a query string (GET)

- It uses a callback function, stored in its onreadystatechange property, that will be called as the request is processed.

- When the process is complete, the result from the server will be stored in responseText -- we can convert it into a JSON object with JSON.parse(), and then access its fields using dot notation

ICE

- Controlling 4 LEDs from the internet
- Controlling a 7-segment display from the internet
- Sending Morse code from the internet with the Piezo electric buzzer (but don't use ... D1?)

Resources

- http://markup.su/highlighter/
- https://www.charlesproxy.com/documentation/proxying/ssl-proxying/
- https://www.charlesproxy.com/documentation/configuration/browser-and-system-configuration/
- https://tools.ietf.org/html/rfc7230 [HTTP Standard]
- https://community.particle.io/t/when-posting-does-the-argument-need-to-be-called-args/23798
- https://requestb.in