Putting the I in IoT: Variables

44-440/640-IoT

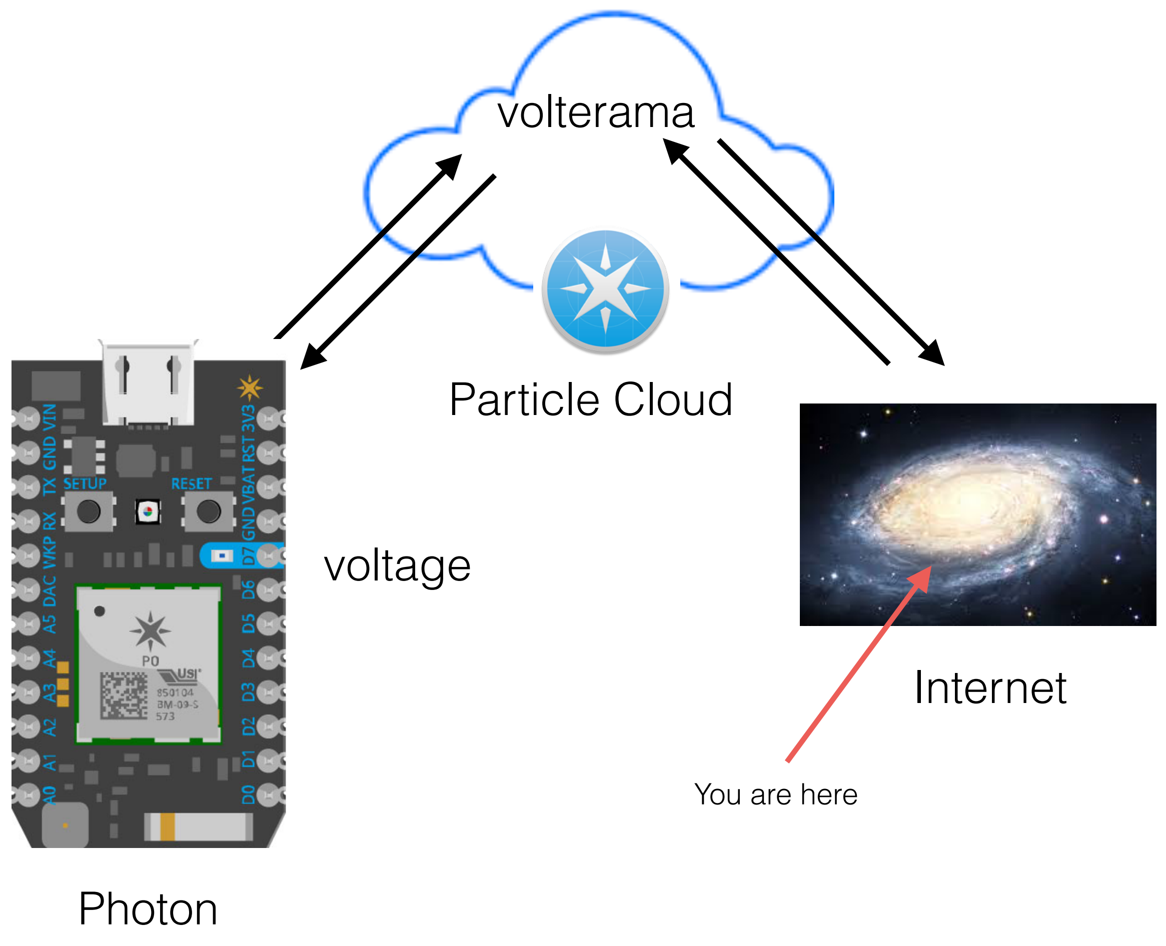
Before class, please construct the circuit corresponding to the code on slide # 11

Objectives

- Students will be able to
  - read firmware variables from the internet
  - explain the difference between Particle and firmware variables
  - explain how GET works in the context of HTTP
  - send GET messages using curl and webpages

The Big Idea

- To borrow a phrase, what happens in Photon stays in Photon: variables defined in code are only accessible on that device, *unless* you specifically expose those variables
- If so, the variables are stored in the cloud, and available over the internet
- The variable name on the device and in the cloud can differ and often do.
- There are now 2 variables in play, so to avoid confusion we will use these terms:
    - **Cloud Variable / Particle variable** - the variable as known to the particle cloud (e.g., volterama)
    - **Firmware Variable / Variable** - the variable as defined in code (e.g., voltage)

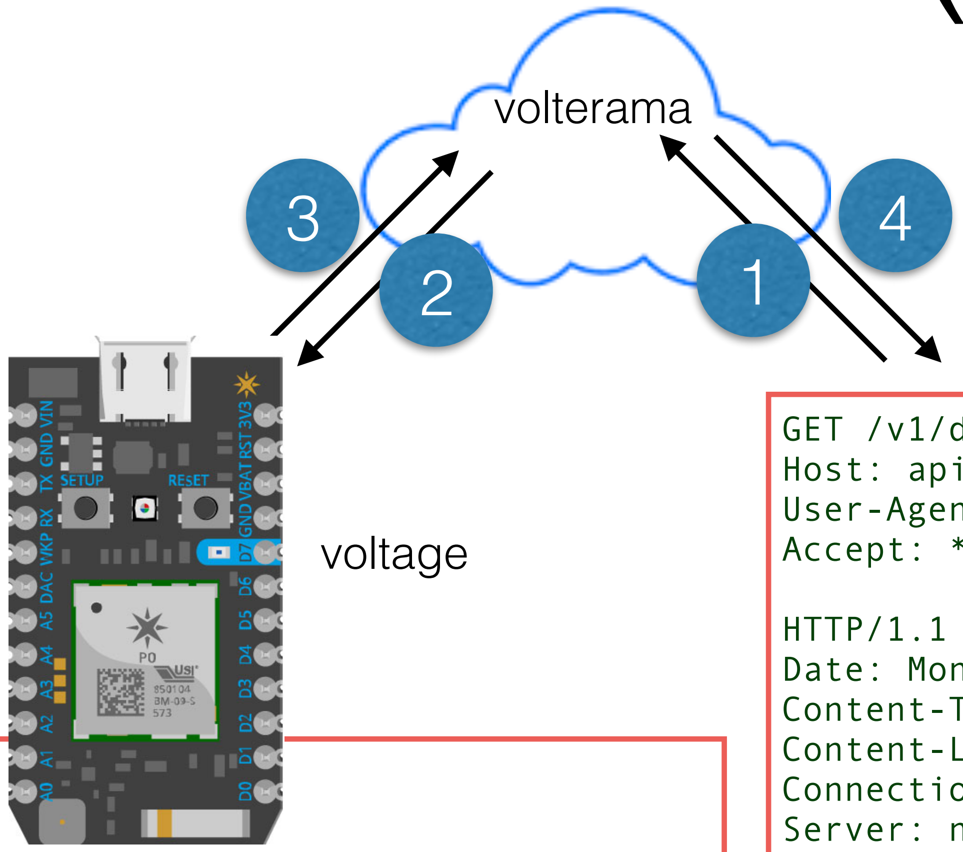


Not to scale

The Magic of Particle.variable()

- **Particle.variable(*particleVariableName, variable*)** exposes firmware variables so that they can be accessed (read) over the internet via a GET message
  - *particleVariableName* is a **string** (≤ 64 chars - letters, numbers, dashes, underscores) provided in the GET URL.
  - *variable* is the firmware variable to be accessed -- a global variable
- It is possible to register up to 20 cloud variables

Particle.variable(), Illustrated



Client

```
GET /v1/devices/xxxxx/volterama?access_token=xxxxx HTTP/1.1
Host: api.particle.io
User-Agent: curl/7.54.0
Accept: */*

HTTP/1.1 200 OK
Date: Mon, 14 Aug 2017 18:44:07 GMT
Content-Type: application/json; charset=utf-8
Content-Length: 249
Connection: keep-alive
Server: nginx
X-Request-Id: f2afd7f3-dc70-4dac-8d4e-e7c015a13a8d
Access-Control-Allow-Origin: *

{"cmd":"VarReturn",
"name":"volterama",
"result":2.250769230769231,
"coreInfo":
{"last_app":"","last_heard":"2017-08-14T18:44:06.990Z","connect
ed":true,"last_handshake_at":"2017-08-14T17:21:24.920Z","device
ID":"xxxxx","product_id":6}
}
```

```
double voltage;

void setup() {
    pinMode(A3,AN_INPUT);
    Serial.begin(9600);
    Particle.variable("volterama", voltage);
}

void loop() {
   voltage = analogRead(A3)/4095.0 * 3.3;
   Serial.println(voltage);
   delay(1000);

}
```

1. Client accesses cloud
2. Cloud accesses Photon
3. Photon returns result to cloud
4. Cloud returns result to Client

Details TBA

Accessing Firmware Variables via a GET Request

- Photon uses GET to access firmware variables, based on this URL:

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

  - **DEVICE_ID** is the Photon's device id, found at console.particle.io, in Particle Dev's status bar, or by clicking on the devices icon in build.particle.io
  - **VARIABLE** is the first argument in Particle.variable()
  - **ACCESS_TOKEN** is available in settings (gear in build.particle.io)
- Variables may be of type **int**, **double**, or **String** (max length of 622 bytes)
- particle.io uses https: so everything other than the host is encrypted

Access Tokens

- An access token is a long string that uniquely identifies you or each of your customers
- It must be provided to access Particle variables or invoke Particle functions (more on this later).
- When a request is sent to the cloud, the access token is looked up to find the list of associated devices
- Access tokens can be reset when necessary. Most expire after 90 days, but not the one in the build.particle.io
- Access tokens are sent encrypted, but take precautions to keep them **private**.

Techy Aside: Particle.variable() Registration

- When calling Particle.variable(), it must happen fairly quickly after connecting to the cloud for the first time (in setup()). This code below, for example, failed if the user didn't actually press a key, and let 30s elapse.

- Moving the Particle.variable() statements before the while loop solved the problem

- This is now described in the documentation (thanks to pestering by the author): it used to be only alluded to in the community discussions

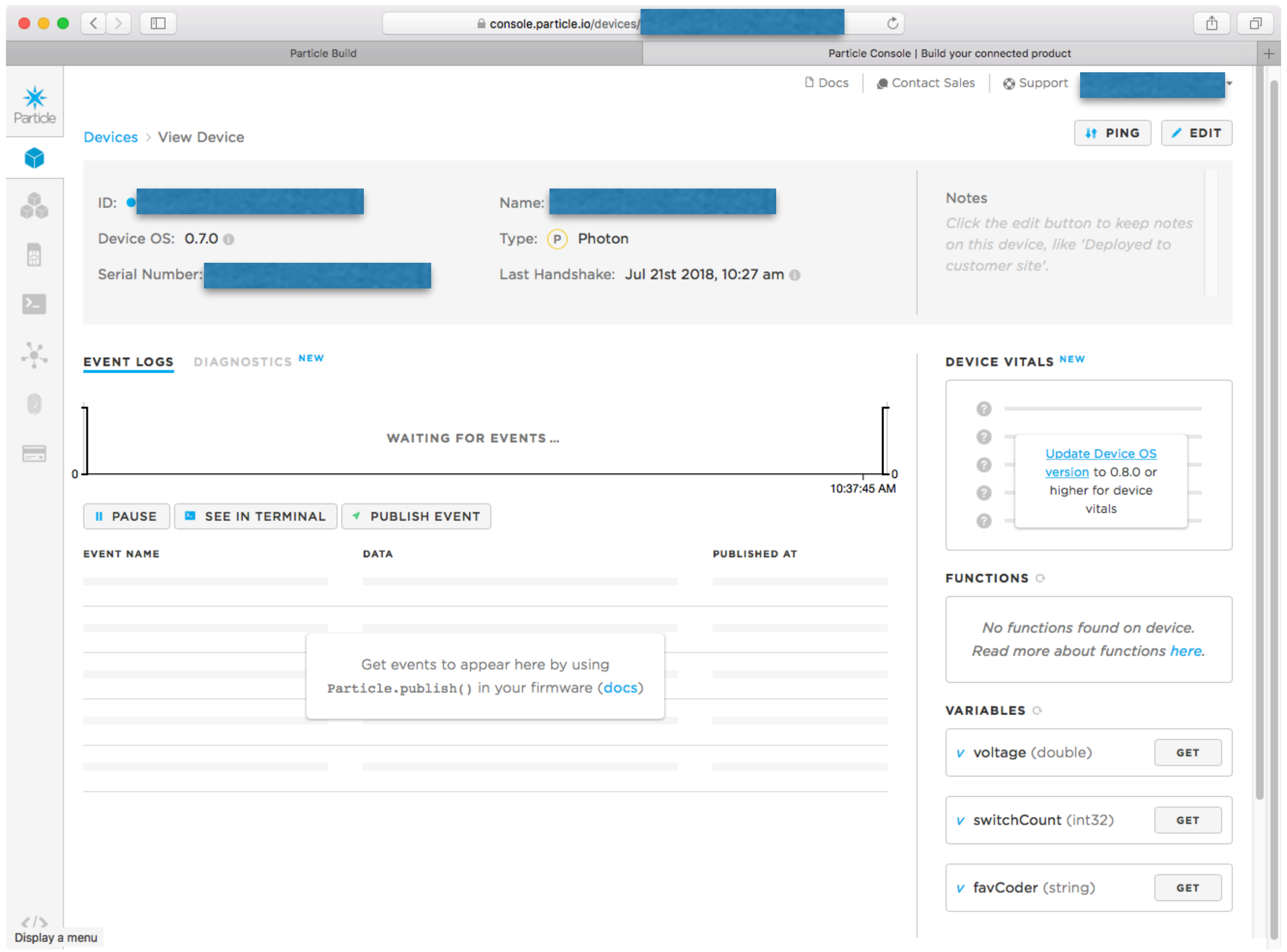
```
void setup(){
 Serial.begin(9600);
  while (!Serial.available() && millis() < 30000) {
    Serial.println("Press any key to start.");
    Particle.process();
    delay(1000);
  }
  Particle.variable("humidity",humidity);
  Particle.variable("temperature",temperature);
}
```

IoT Security: Brought to You By ... HTTPS

- We use **HTTPS** in this class to communicate with the particle cloud -- everything except the hostname is encrypted
- From the particle cloud, if we communicate with an external website, we will use websites that also use HTTPS.

Debugging Particle Variables

- Our main goal is to retrieve variables so we can process them. However, for debugging purposes we can view them on the particle.io console and in Particle Dev



console.particle.io/devices/[redacted]

Particle Build | Particle Console | Build your connected product

Docs | Contact Sales | Support [redacted]

Devices > View Device

PING | EDIT

ID: [redacted]
Name: [redacted]
Device OS: 0.7.0
Type: Photon
Serial Number: [redacted]
Last Handshake: Jul 21st 2018, 10:27 am

Notes
Click the edit button to keep notes on this device, like 'Deployed to customer site'.

EVENT LOGS | DIAGNOSTICS NEW

WAITING FOR EVENTS ...

10:37:45 AM

PAUSE | SEE IN TERMINAL | PUBLISH EVENT

EVENT NAME	DATA	PUBLISHED AT

Get events to appear here by using `Particle.publish()` in your firmware (docs)

DEVICE VITALS NEW

Update Device OS version to 0.8.0 or higher for device vitals

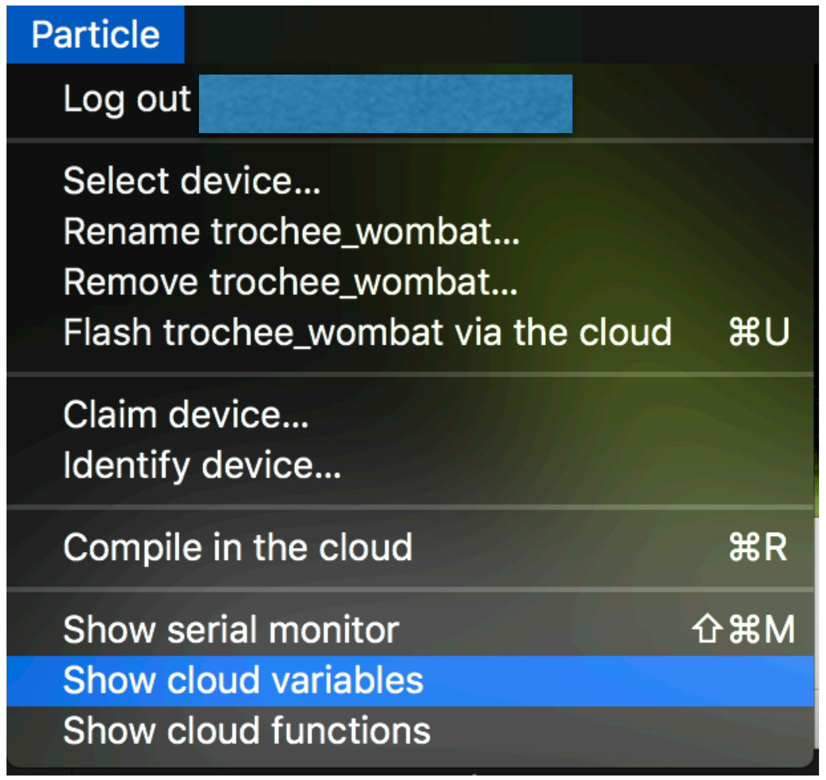
FUNCTIONS

No functions found on device. Read more about functions here.

VARIABLES

voltage (double)	GET
switchCount (int32)	GET
favCoder (string)	GET

Display a menu



Particle

Log out [redacted]

Select device...
Rename trochee_wombat...
Remove trochee_wombat...
Flash trochee_wombat via the cloud ⌘U

Claim device...
Identify device...

Compile in the cloud ⌘R

Show serial monitor ⇧⌘M
Show cloud variables
Show cloud functions

ICE: GETting ints, doubles & Strings

```
// A program to demo cloud variables
// This requires a circuit with a switch on A0, voltage divider on A1, and (yellow) LED on D0
double voltage;
int switchCount;
String favoriteProgrammer;

const int NUM_PROGRAMMERS = 6;
String programmers[NUM_PROGRAMMERS] = {"Ada Lovelace","Niklaus Wirth", "Guido van Rossum",
            "Donald Knuth", "Dennis Ritchie", "Grace Hopper"};

const int switchy = A0;
const int voltagePin = A1;
const int yellowLED = D0;
const int delayTime = 250;

void setup() {
     pinMode(switchy, INPUT_PULLUP);
     pinMode(voltagePin,AN_INPUT);
     pinMode(yellowLED,OUTPUT);
     Serial.begin(9600);
     Particle.variable("voltage", voltage);
     Particle.variable("switchCount", switchCount);
     Particle.variable("favCoder", favoriteProgrammer);
}

void loop() {
   if(digitalRead(switchy)==LOW){
      switchCount++;
      voltage = analogRead(voltagePin)/4095.0 * 3.3;
      favoriteProgrammer = programmers[random(NUM_PROGRAMMERS)];
      Serial.println(String::format("switchCount: %d, voltage: %2.f, favoriteProgrammer: %s",
                              switchCount,voltage, favoriteProgrammer.c_str()));
      flashLED(yellowLED, delayTime);
   }
}
```

1. Study this code
2. Flash it to your device
3. Inspect the cloud variables on the console or Particle Dev, verify that they change as you press the button

```
void flashLED(int led, int time){
  digitalWrite(led,HIGH);
  delay(time);
  digitalWrite(led,LOW);
  delay(time);
}
```

Sending a GET Message

GET can be sent via:

1. an ordinary web browser:

```
https://api.particle.io/v1/devices/xxxx/favCoder?access_token=xxxx
```

2. curl (command url) -- to see details of the transaction

3. code (e.g., JavaScript)

4. Postman

ICE: Sending a GET Message via a Browser

1. Open up a web browser

2. Go to the URL below

Replace the xxxx's with your device ID and access token

```
https://api.particle.io/v1/devices/xxxx/favCoder?access_token=xxxx
```

3. Press the button and refresh the browser several times, to verify that GET is working

ICE: Sending a GET Message via a Browser - Solution

https://api.particle.io/v1/devices/xxxx/favCoder?access_token=xxxx

Request

```
GET /v1/devices/xxxxx/favCoder?access_token=xxxxx HTTP/1.1
Host: api.particle.io
User-Agent: curl/7.54.0
Accept: */*
```

Response

```
HTTP/1.1 200 OK
Date: Mon, 14 Aug 2017 19:23:26 GMT
Content-Type: application/json; charset=utf-8
Content-Length: 246
Connection: keep-alive
Server: nginx
X-Request-Id: 21efc5c7-b245-46f7-bc3c-2b0e7896bc4d
Access-Control-Allow-Origin: *

{"cmd":"VarReturn","name":"favCoder","result":"Niklaus Wirth","coreInfo":
{"last_app":"","last_heard":"2017-08-14T19:23:25.869Z","connected":true,"last_handshake_at
":"2017-08-14T19:19:25.630Z","deviceID":"xxxxx","product_id":6}}
```

The variable is returned as part of a JSON object.
Might we do something with that?🧐 Of course!😍

ICE: Sending a GET Message with curl

- curl is a command line utility that allows you to transfer data to and from a server using one of numerous protocols, including HTTPS
- It allows you to send POST and GET messages to api.particle.io
- **-v** means verbose: it shows response and request headers, not just the response body
- **-H** sends information in a header, rather than in a query string
- **-d** (not used here) causes data to be sent as a POST message (the default is GET). We'll use this factoid in the next presentation.
- As before, replace the ddddd's with your device's ID and aaaa's with your access token

```
curl -v https://api.particle.io/v1/devices/ddddd/favCoder?access_token=aaaaa
```

```
curl -v -H "Authorization: Bearer xxxxxxx" https://api.particle.io/v1/devices/xxxxx/favCoder
```

ICE: Sending a GET Message with curl - Solution

```
curl -v -H "Authorization: Bearer aaaaa" https://api.particle.io/v1/devices/xxxxx/favCoder
```

```
curl -v https://api.particle.io/v1/devices/xxxx/favCoder?access_token=aaaaa
```

Response

```
{
   "cmd": "VarReturn",
   "name": "favCoder",
   "result": "Niklaus Wirth",
   "coreInfo": {
      "last_app": "",
      "last_heard": "2019-08-08T17:07:12.892Z",
      "connected": true,
      "last_handshake_at": "2019-08-08T16:05:48.467Z",
      "deviceID": "3e0028000447373336323230",
      "product_id": 1519
   }
}
```

Requests

```
GET /v1/devices/ddddd/favCoder HTTP/1.1
Host: api.particle.io
Authorization: Bearer aaaaa
cache-control: no-cache
```

```
GET /v1/devices/ddddddd/favCoder?access_token=aaaaa HTTP/1.1
Host: api.particle.io
User-Agent: PostmanRuntime/7.15.2
Accept: */*
Host: api.particle.io
Accept-Encoding: gzip, deflate
Connection: keep-alive
cache-control: no-cache
```

JSON, ASAP

- The result returned from GET is a JSON object, so it behooves us to spend just a little time discussing it.
- According to RFC 7159, JSON is a lightweight[1], text-based[2], language-independent[3] data interchange format[4], i.e., it is

    1. relatively simple to implement, and fast
    2. human-readable, is ordinary text
    3. compatible with any programming language
    4. a protocol for organizing data so that it can be exchanged between two entities

- Typically, JSON is used to **store** data (e.g., preferences on a computer are often stored in JSON) or **transmit** data on the web
- JSON is short for JavaScript Object Notation, because it uses the same syntax as JavaScript does for representing objects (but it remains language independent)

JSON, ASAP

- JSON is composed of two fundamental entities:
- **Objects:** a { } collection of comma-delimited name:value pairs, in which the name is a string, and the value can be a string, number, object, array, true, false, or null. Strings are enclosed in " "
- **Arrays:** a [ ] collection of comma-delimited values

```
{"name":"Trudeau", "age": 47, "inOffice":true}

{"id":1538, "grades":[93, 87, 100]}
```

```
[ {"name":"Afghanistan", "popn": 36373176, "inNATO":false},
  {"name":"Albania", "popn": 2876591, "inNATO":false},
  {"name":"Algeria", "popn": 42200000, "inNATO":false} ]
```

HTML and JavaScript, ASAP

- HTML, another invention from Tim Berners-Lee
- JavaScript, sorta, kinda like Java, but not really
- We will talk about HTML <now/>, and JavaScript in detail later()

Optional ICE: Sending a GET Message In Code

fetchIt.js

```
let url = "https://api.particle.io/v1/devices/ddddd/voltage?access_token=aaaaa"

function fetchVoltage(){
  // construct the URL so we don't have to hardcode the ID and access token
  let deviceID = prompt("Enter device ID")n // JS uses let to declare a variable, and does *not* specify its type
  let accessToken = prompt("Enter access token")
  url = url.replace("ddddd",deviceID) // substitutes deviceID where the ddddd was previously
  url = url.replace("aaaaa",accessToken)
  // make the request
  let xhttp = new XMLHttpRequest() // instantiates an XMLHttpRequest, which can send HTTP requests
  xhttp.open("GET", url, true)
  xhttp.send()
  url = ""
  xhttp.onreadystatechange = function() { // The xhttp object will call this code several times
    if (this.readyState == 4 && this.status == 200) {
      let resp = JSON.parse(this.responseText)
      document.getElementById("demo").innerHTML = resp.result
    }
  }
}
```

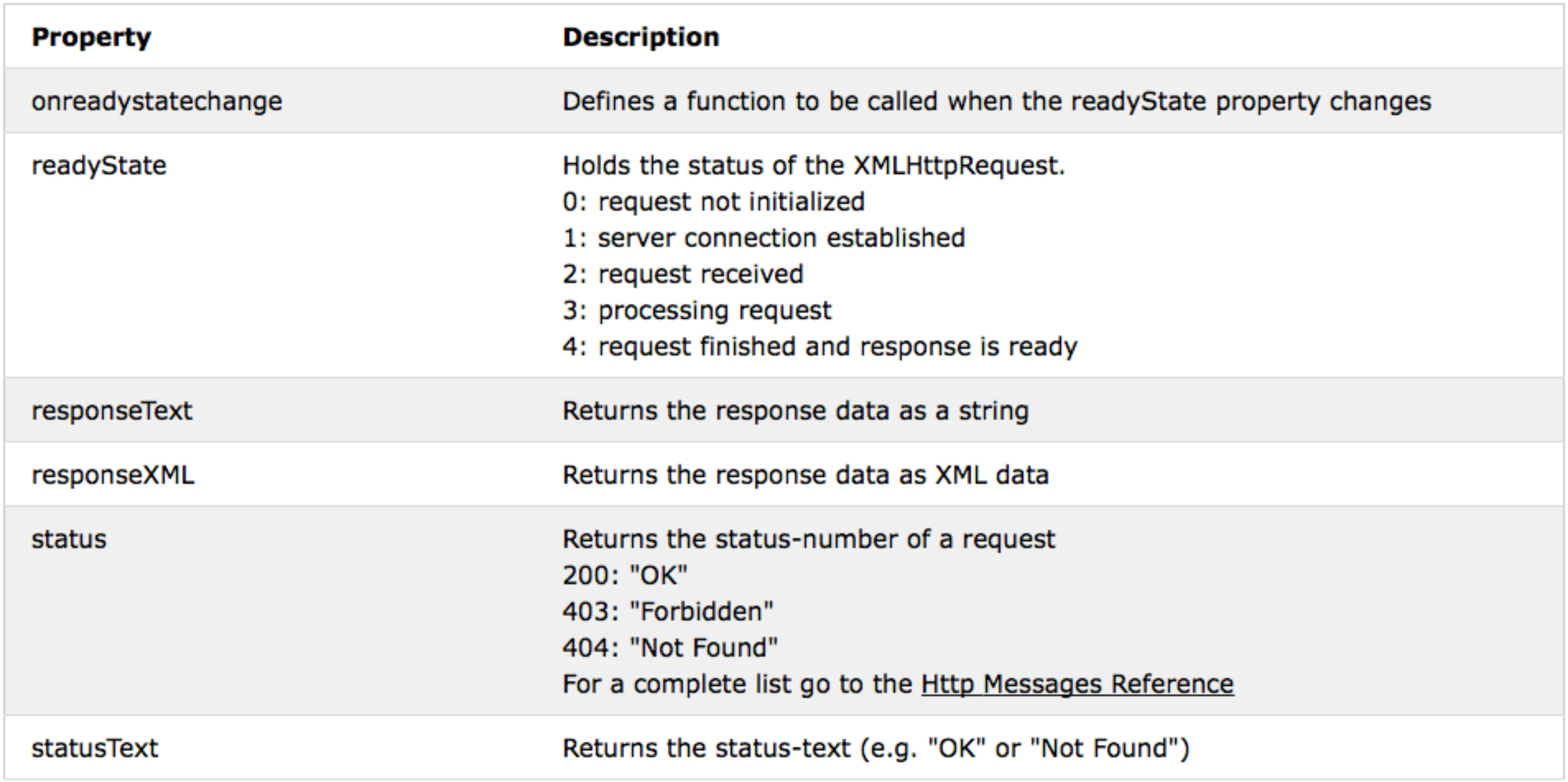
```
<!DOCTYPE html>
<html lang="en-us">
  <head><meta charset="utf-8">
    <title>Hello...</title>
  </head>

  <body>
    <div id="mainContent">
      <h1>The Amazing Voltage Fetcher</h1>
      <p>This will fetch voltage from your Photon!</p>
      <button onclick="fetchVoltage()">Get Voltage</button>
      <p>Voltage: <span id="demo">N/A</span></p>
    </div>
    <script src="fetchIt.js"> </script>
   </body>
</html>
```

1. Study this project
2. Run it in Glitch

There is a *lot* happening here, but don't panic: we will explain it in class

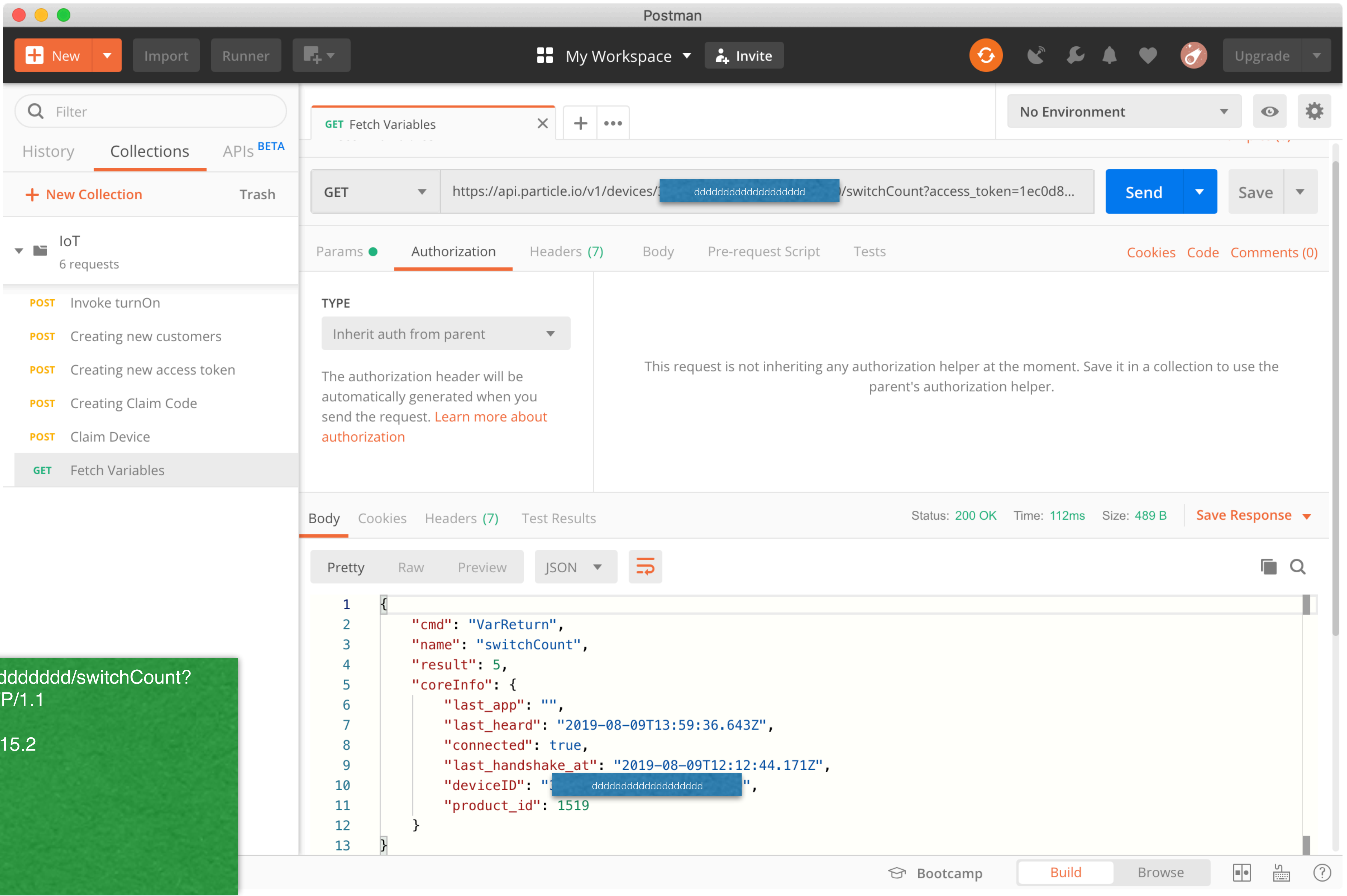
<script> is needed ... otherwise nothing works.

Optional: XMLHttpRequest Object Properties



Property	Description
onreadystatechange	Defines a function to be called when the readyState property changes
readyState	Holds the status of the XMLHttpRequest. 0: request not initialized 1: server connection established 2: request received 3: processing request 4: request finished and response is ready
responseText	Returns the response data as a string
responseXML	Returns the response data as XML data
status	Returns the status-number of a request 200: "OK" 403: "Forbidden" 404: "Not Found" For a complete list go to the Http Messages Reference
statusText	Returns the status-text (e.g. "OK" or "Not Found")

Optional ICE: Sending a GET Message Using Postman


Postman

New | Import | Runner | My Workspace | Invite | Upgrade

History | Collections | APIs BETA

New Collection | Trash

IoT
6 requests

- POST Invoke turnOn
- POST Creating new customers
- POST Creating new access token
- POST Creating Claim Code
- POST Claim Device
- GET Fetch Variables

GET Fetch Variables

No Environment

GET https://api.particle.io/v1/devices/dddddddddddddddddd/switchCount?access_token=1ec0d8...

Send | Save

Params | Authorization | Headers (7) | Body | Pre-request Script | Tests | Cookies | Code | Comments (0)

TYPE
Inherit auth from parent

The authorization header will be automatically generated when you send the request. Learn more about authorization

This request is not inheriting any authorization helper at the moment. Save it in a collection to use the parent's authorization helper.

Body | Cookies | Headers (7) | Test Results — Status: 200 OK Time: 112ms Size: 489 B — Save Response

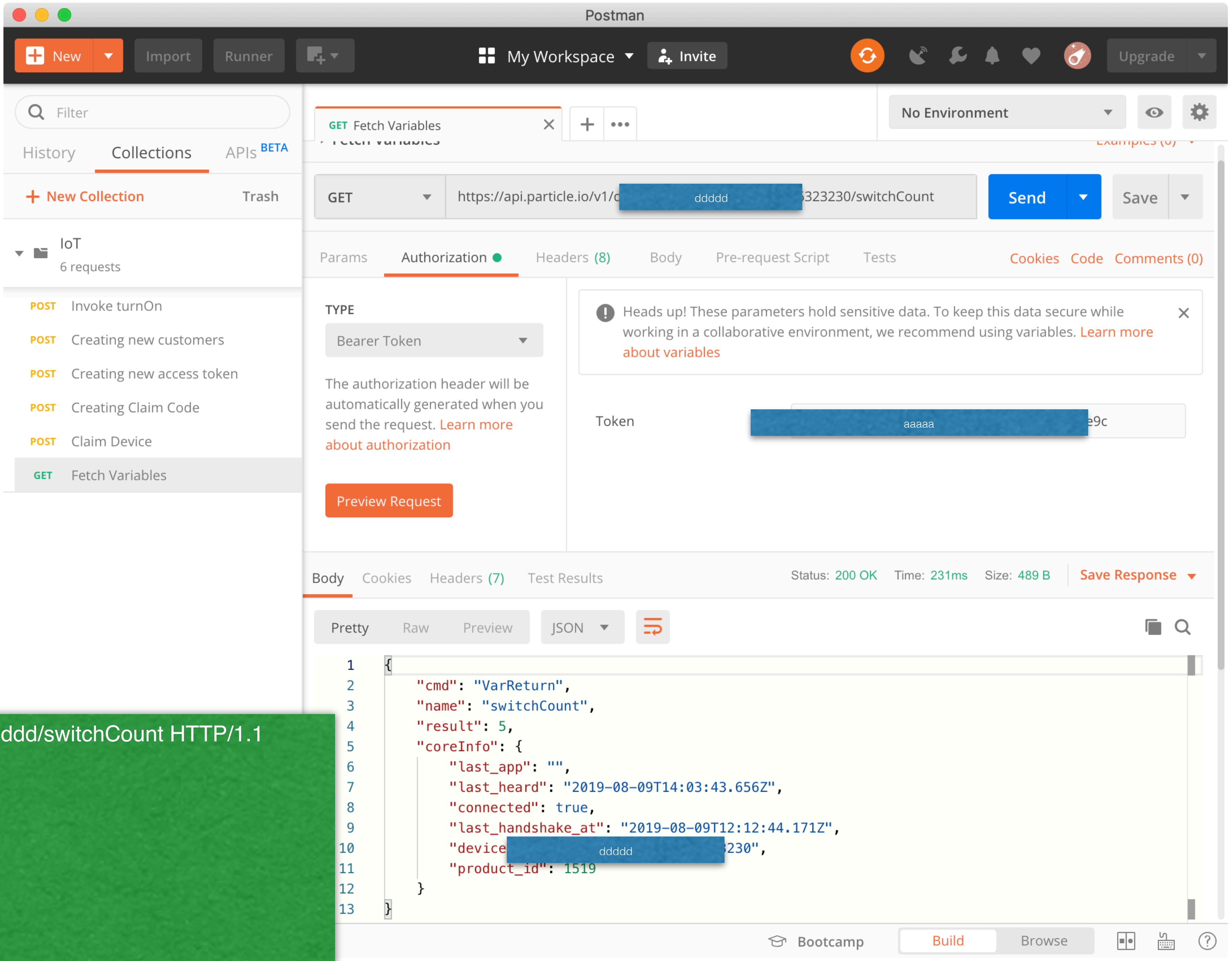
Pretty | Raw | Preview | JSON

```
{
    "cmd": "VarReturn",
    "name": "switchCount",
    "result": 5,
    "coreInfo": {
        "last_app": "",
        "last_heard": "2019-08-09T13:59:36.643Z",
        "connected": true,
        "last_handshake_at": "2019-08-09T12:12:44.171Z",
        "deviceID": "dddddddddddddddddd",
        "product_id": 1519
    }
}
```

Bootcamp | Build | Browse

```
GET /v1/devices/dddddddddddddddddd/switchCount?
access_token=aaaaaaaaaa HTTP/1.1
Host: api.particle.io
User-Agent: PostmanRuntime/7.15.2
Accept: */*
Cache-Control: no-cache
Host: api.particle.io
Accept-Encoding: gzip, deflate
Connection: keep-alive
cache-control: no-cache
```

Optional ICE: Sending a GET Message Using Postman (with access token in authorization header)



GET https://api.particle.io/v1/d[redacted]323230/switchCount

TYPE: Bearer Token

The authorization header will be automatically generated when you send the request. Learn more about authorization

Heads up! These parameters hold sensitive data. To keep this data secure while working in a collaborative environment, we recommend using variables. Learn more about variables

Token: [redacted]e9c

Status: 200 OK Time: 231ms Size: 489 B

```
{
    "cmd": "VarReturn",
    "name": "switchCount",
    "result": 5,
    "coreInfo": {
        "last_app": "",
        "last_heard": "2019-08-09T14:03:43.656Z",
        "connected": true,
        "last_handshake_at": "2019-08-09T12:12:44.171Z",
        "device[redacted]230",
        "product_id": 1519
    }
}
```

```
GET /v1/devices/dddddddddddddddddd/switchCount HTTP/1.1
Host: api.particle.io
Authorization: Bearer aaaaa
User-Agent: PostmanRuntime/7.15.2
Accept: */*
Cache-Control: no-cache
Host: api.particle.io
Accept-Encoding: gzip, deflate
Connection: keep-alive
cache-control: no-cache
```

ICE: Security

- Analyze the security risks in the code we just ran.

Exercises

- Explain the purpose of Particle.variable()
- How long can a Particle variable name be?
- How long can a C++ variable be?
- Name the 3 ways of sending GET messages
- Explain what XhttpRequest does

Resources

- http://markup.su/highlighter/
- https://community.particle.io/t/reading-spark-variables-with-your-own-html-file/4148
- http://www.json.org
- www.postman.com (for HTTP -- although curl works better)
- https://docs.particle.io/reference/api/#how-to-send-your-access-token
- https://https.cio.gov/faq/
- https://www.w3schools.com/xml/ajax_xmlhttprequest_create.asp
- https://javascriptobfuscator.herokuapp.com