

EE 1301: Introduction to Computing Systems

IoT Laboratory #3

Internet Connectivity

Flying (not so) High in the Cloud

Created by: John Sartori, David Orser, and Kia Bazargan



Many thanks to the students, teaching assistants, and faculty that work to continually improve this document. Together we make better labs!

Please send comments and suggestions to orser@umn.edu

Copyright 2018



In the previous labs, you only used the wifi connection to program (flash) the microcontroller, after which the device was on its own (that's a lie, BTW). The device only interfaced with sensors and actuators directly connected to it, using the data from sensor to figure out what was going on in its environment (e.g., the room getting too warm), and responding by sending out commands to actuators (such as a fan or an A/C unit to cool down the room).

In this lab, we are going to look into how we can use the internet to connect devices that are physically far apart. For example, you might want your vegetable garden to send you an email or text message when the water level in its water reservoir is getting too low. To fill the reservoir remotely, you could send a command to a web-connected water valve to open for one minute and fill the reservoir.

In this lab, you will learn how to communicate with and control your microcontroller remotely using the cloud. After completing this lab, you will be able to call functions on your microcontroller and read the values of variables in your program using the web. You will also be able to log data gathered by your microcontroller into an online google spreadsheet and use events signalled by your microcontroller to trigger actions online.

Prelab

Reading Material

Read up on cloud functionalities in Particle's reference documentation:

<https://docs.particle.io/reference/firmware/photon/#cloud-functions>.

Specifically you should review the following closely:

- ☐ Particle.variable()
- ☐ Particle.function()
- ☐ Particle.publish()
- ☐ Particle.subscribe()

Lab Procedure

Registering Cloud Variables, Functions, and Events in your Program Code and Accessing them in the Cloud

The internet of things (IoT) is powered by microcontrollers that can be queried and controlled by networked devices like cellphones and computers. In this section, we will learn how to program our microcontrollers to send data to the **cloud**, and how we can call functions on our microcontrollers using a networked device.

For the purpose of this lab, we can define the **cloud** as a location on the web where we can write data or read data that has been written. This space in the cloud is convenient for us

because we can use it to communicate with our microcontroller remotely. For example, we can write data to the cloud with our microcontroller and then access the data with a networked device. Likewise, we can write data to the cloud with a networked device and access the data with our microcontroller.

Cloud Variables

One way our microcontroller sends data to the cloud is with a **cloud variable**. A cloud variable is just like a normal program variable, except that when a cloud variable is updated in the program running on your microcontroller, the microcontroller also updates the value of the variable in the cloud. This allows us to get data from an IoT device from far away. For example, I could query a web-connected thermostat to find out the temperature of my house when I am away on vacation.

Currently, four types of cloud variables are supported -- INT (for fixed point / integer data), DOUBLE (for floating point data), STRING (for character-based data), and BOOLEAN (for true / false data). The example below shows how to declare three variables in a program and register them as cloud variables that will be written and updated in the cloud when the program is running. The usual place to register cloud variables is in the `setup()` function. The `Particle.variable()` function registers a cloud variable; it takes two parameters -- the name of the cloud variable being registered and the name of the program variable the cloud variable is associated with.

```
int integer_value = 0;
double double_value = 0.0;
char string_value[16] = "This is a string";

void setup()
{
    // max length of a cloud variable name is 12 characters
    Particle.variable("intVal", integer_value);
    Particle.variable("doubleVal", double_value);
    Particle.variable("stringVal", string_value);
}
```

NOTE: Currently, only **up to 20 cloud variables** may be defined, and each **cloud variable name is limited to a maximum of 12 characters**. In other words, the space reserved for you in the cloud can only hold 20 variables, and the names of the variables cannot be longer than 12 characters. The name of a cloud variable can either be the same or different than its program variable name, as long as the name is 12 characters or fewer.

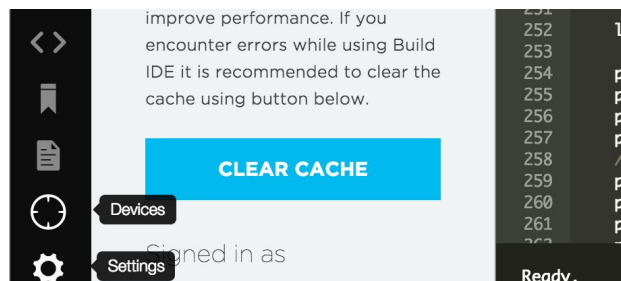
Querying the value of a cloud variable:

Once cloud variables are written to the cloud by the microcontroller, they are associated with a **unique URL** and can be accessed by a networked device (e.g., computer, cellphone, etc.) by entering the URL in a web browser. The following URL queries the value of the cloud variable called `stringValue1` in our example above. After entering the URL into a browser, the browser returns the value of the string, e.g., "This is a string".

```
"https://api.particle.io/v1/devices/0123456789abcdef/stringVal?access_token=123412341234"
```

NOTE that the URL contains **two unique character strings** that are used to securely identify one particular microcontroller, so that only you (or those you grant access to) can access and control your microcontroller through the cloud. The first string is the **Device ID** -- this is the name of your microcontroller. The next string is the **access token** -- this is like a password associated with your device for security purposes. Your microcontroller will only respond to requests made using the correct device ID and access token. In the example above, the device ID is "0123456789abcdef", and the access token is "123412341234". You can find the device ID of your device by clicking on the **"Devices"** tab in Particle Build, and you can find your access token by clicking on the **"Settings"** tab. In general, the URL used to access the value of a cloud variable is formatted as follows.

```
"https://api.particle.io/v1/devices/[DEVICE ID]/[CLOUD VARIABLE  
NAME]?access_token=[ACCESS TOKEN]"
```



EXERCISE 1

- Create a program and corresponding circuit that uses your microcontroller and a TMP36 sensor to take temperature measurements. (HINT: You should have done something very similar in IoT Lab 2.)
- Add a cloud variable in your program that will update the temperature values read by your microcontroller in the cloud.
- Open a browser and type in a URL to read a temperature value that is being sampled by your microcontroller. Refresh the browser a few times to gather additional samples. See if you can make the value change (e.g., by heating up or cooling down the sensor).

Cloud Functions

Just as cloud variables are used to read and write data in the cloud, a **cloud function** can be used to call a function on the microcontroller using a networked device. This allows us to send commands to our IoT devices from far away. For example, I could call a cloud function on a web-connected thermostat telling it to start heating up the house when I am on my way home.

Currently, your reserved space in the cloud allows you to register **up to 15 cloud functions**. The `Particle.function()` function registers a cloud function. It requires two arguments -- first, the name of the cloud function (a string), and second, a C-function in your code. The C-function must take **only one argument, a string**. The length of the string will be **limited to 63 characters**. Although the argument must be passed as a string, it can easily be converted to whatever data type(s) the program requires. The example below shows how to register a cloud function using the function `Particle.function("nameInCloud", nameInProgram)`.

```
enum thermostat_mode_t {
    COOL,
    OFF,
    HEAT,
};
thermostat_mode_t mode = OFF;

// If you are not familiar with the "enum" construct, the above code acts
// exactly like the following three lines:
//
// #define COOL 0
// #define OFF 1
// #define HEAT 2
//
// and defining an int variable "mode" that is initialized to OFF

void setup()
{
    // register the cloud function
    // first argument is the name called from the web (12 chars or less)
    // second argument is the name of the cloud function in the program
    Particle.function("setMode", setModeFromString);
    Serial.begin(9600);
}

void loop() {

    // the loop function is called repeatedly, as long as microcontroller is turned on
}

// this function automatically gets called upon a matching POST request
// Note that the return type must be int and the argument must be a string
int setModeFromString(String inputString)
{
    if (inputString == "Cool") {
        mode = COOL;
        return 1;
    } else if (inputString == "Off" ) {
        mode = OFF;
        return 1;
    }
}
```

```

    } else if (inputString == "Heat") {
        mode = HEAT;
        return 1;
    } else {
        Serial.print("Invalid Mode: ");
        Serial.println(inputString);
        return -1;
    }
}

```

Once a cloud function has been registered, it can be called using a POST HTML request. Unfortunately, POST requests can not be generated with a URL alone (i.e., you can't just type something into the browser address bar.) Instead, we'll have to create a simple webpage of our own to issue a POST request.

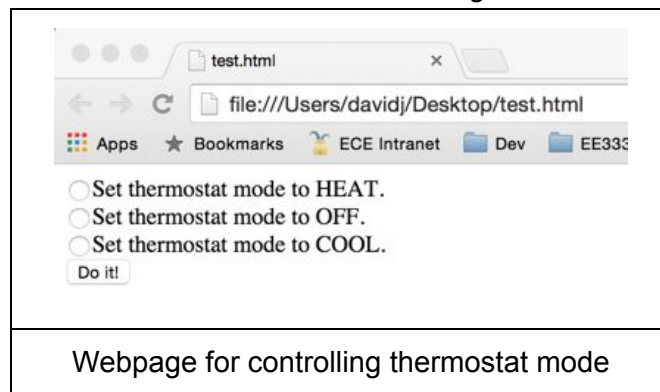
The following is a template to start from:

```

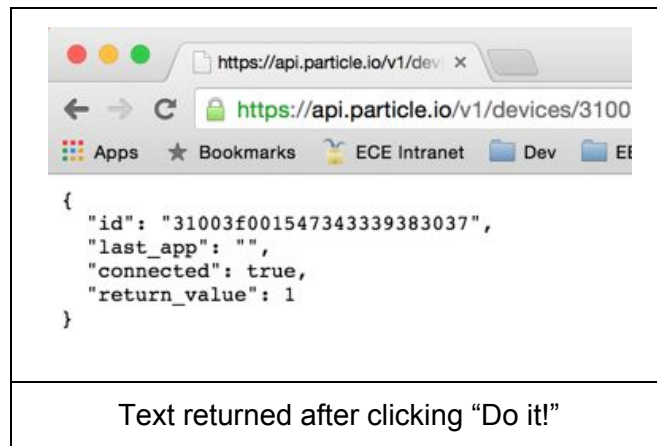
<html>
  <body>
    <form
action="https://api.particle.io/v1/devices/31003f00154f34339383037/setMode?access_token=cd5d28
f314ddbae6724d9efbf703cedc493f3802" method="POST">
      <input type="radio" name="args" value="Heat">Set thermostat mode to HEAT.<br>
      <input type="radio" name="args" value="Off">Set thermostat mode to OFF.<br>
      <input type="radio" name="args" value="Cool">Set thermostat mode to COOL.<br>
      <input type="submit" value="Do it!">
    </form>
  </body>
</html>

```

For now, copy and paste this text into a new text file called "test.html" on your desktop. Sometimes copying from a pdf file garbles text a bit, so make sure the file looks exactly like the text in the box above. Replace the **Device ID** and **Access Token** in the code with your own. Open the file in your web browser; it should look something like this:



When you click on "Do it!" the cloud will initiate a request to your Photon to run the **setModeFromString** function. When the function completes, the cloud will relay the return value back to your web browser. It should look something like this:



In the last section of this lab manual, you will learn more about how to write a webpage that dynamically displays variables and can be used to call cloud functions interactively.

EXERCISE 2

- Add a cloud function to your temperature measurement program that changes the value of a mode variable.
- Add an RGB LED to your microcontroller circuit that changes color when the mode variable changes. You should have at least three modes with distinct colors.
- Use the HTML form to change the mode on your microcontroller.

Cloud Events

Cloud events are another way that our microcontroller can communicate through the cloud. A cloud event is a way that the microcontroller can signal that something has happened. For example, you could connect a motion sensor to the microcontroller and have it publish an event whenever motion is detected. The microcontroller publishes an event to the cloud using the `Particle.publish()` function. Anyone who subscribes to the event gets notified whenever the event is published.

Cloud events have the following properties:

- **name** (1–63 ASCII characters)
- **public/private** (default public)
- **optional data** (up to 255 bytes)

Anyone may subscribe to public events, but only the owner of a device will be able to subscribe to private events published by the device.

NOTE that a device **cannot** publish events beginning with a case-insensitive match for "spark" or "particle". Such events are reserved for officially curated data originating from the Cloud. **Also**

NOTE that currently, a device can publish events at a maximum rate of about 1 event per second. Trying to publish events faster than that may result in strange behavior from your microcontroller, since the Particle servers will automatically throttle your event publishing rate.

The examples below demonstrate how to publish events with and without data.

```
if(digitalRead(MOTION_DETECTOR_PIN) == HIGH){
    // publish an event to signal that motion has been detected
    Particle.publish("motion-detected");
}

// publish an event with data to indicate where motion was detected
if(digitalRead(FRONT_MOTION_DETECTOR_PIN) == HIGH){
    Particle.publish("motion-detected", "FRONT DOOR");
}
if(digitalRead(BACK_MOTION_DETECTOR_PIN) == HIGH){
    Particle.publish("motion-detected", "BACK DOOR");
}
```

As stated above, anyone (with permission) who subscribes to an event will be notified whenever the event is published. The `Particle.subscribe()` function is called within the `setup()` function to subscribe to a cloud event. The first argument to `Particle.subscribe()` is the name of the event, and the second argument is the name of a handler function that will be called in the subscriber's code when the event is published. A handler function must return `void` and take two arguments, both of which are C strings (`const char *`). The first argument will be the full name of the published event. The second argument (which may be `NULL`) is any data that came along with the event.

```
void motionHandler(const char *event, const char *data)
{
    Serial.print("Motion was detected at the ");
    Serial.print(data);
    Serial.println(" door");
}

void setup()
{
    // with this subscription, the motionHandler function will be called
    // whenever the motion-detected event is published
    Particle.subscribe("motion-detected", motionHandler);
    Serial.begin(9600);

    // only listen to events published by your devices: use the MY_DEVICES argument
    Particle.subscribe("event_name", motionHandler, MY_DEVICES);

    // subscribe to events from a single device by specifying the device's ID.
    Particle.subscribe("event_name", motionHandler, "55ff70064989495339432587");
}
```


NOTE: A device can register up to 4 event handlers. This means you can call `Particle.subscribe()` a maximum of 4 times in your code; after that it will return false.

NOTE: You can monitor cloud events at: <https://console.particle.io/events>

An introduction to If This Then That (IFTTT)

As we've seen in this course, a conditional statement is a very powerful tool in programming. IF This Then That (IFTTT) is a website that specializes in creating conditional statements for various internet-connected API's. It does this in a very user-friendly and intuitive way, allowing the user to control a huge array of online services without detailed knowledge of how those online services communicate. A few useful examples of online services you can control with IFTTT are:

- Gmail
- Google Drive
- Particle (Photon)
- Instagram

IFTTT also supports certain functions embedded in your iOS or Android phone. For example, it can use GPS to detect when your phone enters a certain region. These advanced features require that your phone be running the IFTTT app and have certain background features enabled (like location services and notification).

Setting up an account at IFTTT.com

- 1.) Go to IFTTT.com
- 2.) Click "Sign up"
- 3.) Enter your email and choose a password

That's it, you're done! Well, it's not quite that simple; IFTTT is just a middleman. We will need to set up links to your accounts at other services to fully utilize IFTTT.

1. Creating your first applet

- 1.) Go to "My Applets" and choose to create a new applet. You should see something like the following.



- 2.) Click "this"
- 3.) Search for the "weather underground" channel

We'll now connect to the weather channel. This channel is relatively simple to connect because it only requires you to enter a fixed location (no personal or account information). Fill out the forms as requested.

- 4.) Fill out the information necessary to connect the Weather channel and click continue.
- 5.) You will be prompted to “Choose a Trigger”; click “Tomorrow’s weather report”
- 6.) Select a “Time of day” that is after you wake up but before you’ve left the house.
- 7.) Click “Create Trigger”

Your applet should look something like this:



Click “that”, search for “email”, and select

- 8.) Click “Send me an email”

You should end up on a screen that looks like the one on the right:

- 9.) If desired, you can edit the message to be sent. You can insert “ingredients” into the recipe. This allows you to add data from the “this” element into the email. Ingredients are context-sensitive and fairly intuitive.
- 10.) Click “Create Action”, then “Finish”. You should be returned to your “My Applets” page and should see a description of the applet you just created.
- 11.) Now, say we don’t want to wait until tomorrow at the chosen time to test our applet. Click on the gear icon and edit your applet to trigger a few minutes from now. Save your changes.
- 12.) Continue on with the lab below, but shortly, you should receive an email from IFTTT looking something like this:

Send me an email

This Action will send you an HTML based email. Images and links are supported.

Subject

TomorrowsCondition
tomorrow!

Add ingredient

Body

```

<br>
<br>
With a high of
HighTempFahrenheit F and a
low of LowTempFahrenheit F.
<br>
<br>
via ForecastUrl <br>
<br>
TomorrowsDate

```

Add ingredient

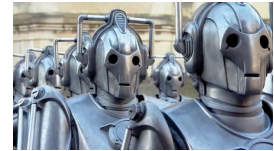
Create action

▼ Unread	1-8 of 8 ▼
<input type="checkbox"/> <input type="star"/> <input type="checkbox"/> IFTTT Action	Mostly Sunny tomorrow! - With a high of 74F and a low of 58
	1:10 pm

Now would be a good time to poke around IFTTT (by creating a couple new applets) and look into the various services with which you can interact. Some of the more interesting channels relate to your cell phone (both iOS and Android are supported).

Connecting your Photon to IFTTT

- 1.) Create a new applet, select “this”, search for “Particle”, and select it.
- 2.) Click “Connect”.
- 3.) Enter your Particle account information.
- 4.) It will prompt you to allow IFTTT to take control all the Cybermen in range of your Photon, don’t worry, and click OK.
- 5.) Click “Monitor your device status”.
- 6.) Fill out the form with your device name and “Online”.
- 7.) Click “Create Trigger”.
- 8.) Click “that”.
- 9.) Choose a channel of your choice.
 - a.) An email is easy
 - b.) If you’re feeling adventurous, install the iOS or Android app from Particle called “IFTTT”, then select either the iOS Notifications or Android SMS Channel.
- 10.) Click “Create Action” and “Create Recipe”
- 11.) Now, plug in (or reset) your Photon
- 12.) If you click on your applet, you can see when the applet last ran and even choose to “Check now” for any activity.



In this section, we explored setting up IFTTT and several IFTTT channels. IFTTT can form a strong component in your upcoming project. It is well worth your time to poke around and explore what IFTTT is capable of, especially the cell phone features available through the app “IFTTT” (available in both the iOS and Android app stores). Good luck, and happy hacking.

EXERCISE 3

- Add a mode to your program called REDALERT.
- Modify your program so that an event called REDALERT is published every time your program enters REDALERT mode.
- Use IFTTT to send yourself an email whenever the REDALERT event is published.
- Use the cloud function you created earlier to put your microcontroller in REDALERT mode, and observe that you receive an email notification.

REMINDER: You can monitor cloud events in the console: <https://console.particle.io/events>

Data Logging with Google Drive and IFTTT

Logging data is one of the most useful features of internet-connected devices. Not only does it provide a record of what has happened with a device (a great debugging tool) it expands the capabilities of the entire Internet of Things, allowing objects completely disconnected from your microcontroller to take action based on your microcontroller’s observations.

In this section, we'll walk through setting up your microcontroller to dump temperature data to a Google Sheets spreadsheet. The focus will be on both the code needed in your microcontroller application and the Recipe required on IFTTT. You are encouraged to supplement the temperature data in the example below with the data available from a sensor of your choice. Modify the example below to suit your needs.

Creating a Cloud Event with Data

We will use a cloud event to publish our data to the cloud. Include the following `Particle.publish()` call somewhere inside your `loop()` function.

```
// change the name of the event to something unique
Particle.publish("orserTemp", sTemp);
```

"orserTemp" is the name of the event. Since this a "public" event, anyone can subscribe to it. Thus, it must be named uniquely. Replace "orser" with your x.500 ID to give your event a unique name.

The "data" field of the event must be a string. "sTemp" is a string describing the temperature. I defined it as follows:

```
sTemp = String( ( data1 - 620 ) / 12.4 );
```

Make sure your event is published at most once every ten seconds (i.e., add a `Timer` or `delay()` of at least 10000ms). We don't want to upset anyone by spamming the servers. If you publish an event more than 4 times per second, you will be timed out for 4 seconds. A maximum long-term average rate of one event per second is supported by Particle.


Compile and Flash the code you created to your Photon. It should result in event(s) you can see being published online at:

<https://console.particle.io/events>


Creating an IFTTT Recipe to Store the Data

Once you have a cloud event occurring on a regular basis, an IFTTT recipe can be created to add a row to a spreadsheet for every event that occurs. Follow the instructions below.

- 1.) Go to the IFTTT website and choose to create a new applet.
- 2.) Click on "this"
- 3.) Search for "particle" and select it.
- 4.) Click on "New event published"
- 5.) Fill out the form with your information, similar to:



Complete Trigger Fields
step 3 of 7
back

New event published


 **If (Event Name)**

orserTemp

Fill in your published event name; ex: monitoring a washing machine? Event Name = Wash_Status

 **is (Event Contents)**

The contents of the published event, "Data"; ex: monitoring a washing machine? Event Contents = Done


 **Device Name or ID**

ORSER-PHOTON ▾

Create Trigger


IFTTT: Completed “this” form for “Particle” → Create an Event

- 6.) Click “Create Trigger” and then “that”
- 7.) Search for “Google Sheets” and select it.
 - a.) At this stage, you may be required to link your Google account to your IFTTT account; please follow the instructions provided. (NOTE: Be warned that mobile browsers might have issues with this step.)
- 8.) Select “Add row to spreadsheet”
- 9.) The defaults for this form will work fine, but we suggest you change the “Drive folder path” to a more descriptive name than “events”. Example below:


Spreadsheet name

EventName


will create a new spreadsheet if one with this title doesn't exist


Formatted row

EventName ||| EventContents ||| DeviceName |||

CreatedAt

use “|||” between cells


Drive folder path

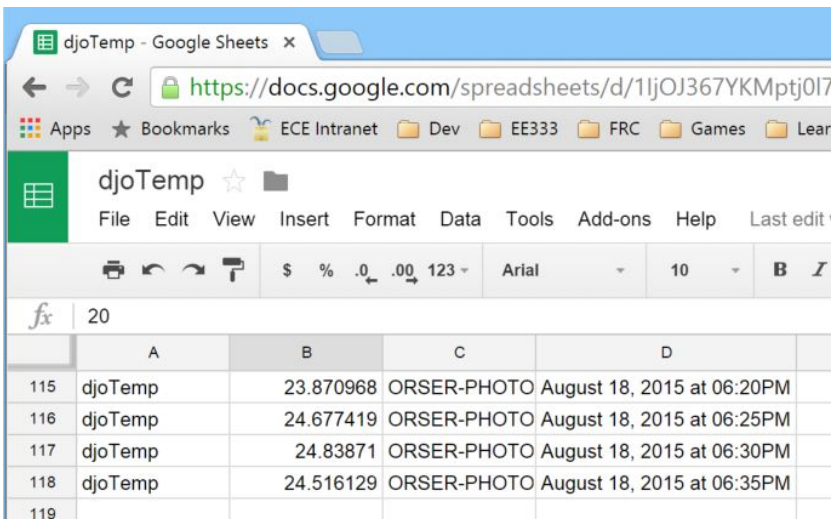
Particle/ORSERPHOTON/events

Format: some/folder/path (defaults to “IFTTT”)

IFTTT: Completed “that” form for
“Google Sheets” → “Add row to spreadsheet”

NOTE: There are many useful “ingredients” available for this form; explore them!

- 10.) Click “Create Action” and then review the applet and click “Finish”.
- 11.) Wait about 20 seconds and then go find the new spreadsheet in your Google Drive!
It should look something like this:



The screenshot shows a Google Sheet titled "djoTemp" with the following data:

	A	B	C	D
115	djoTemp	23.870968	ORSER-PHOTO	August 18, 2015 at 06:20PM
116	djoTemp	24.677419	ORSER-PHOTO	August 18, 2015 at 06:25PM
117	djoTemp	24.83871	ORSER-PHOTO	August 18, 2015 at 06:30PM
118	djoTemp	24.516129	ORSER-PHOTO	August 18, 2015 at 06:35PM
119				

Automagic Google Drive Sheet Example

Your completed Photon code may look something like the following.



periodic-events.ino

```
1  int data0;
2  int data1;
3  int Temp;
4  String sTemp;
5  unsigned Timer1;
6  unsigned Timer2;
7  bool HeartBeatState = FALSE;
8
9  void setup()
10 {
11     Timer1 = 0;
12     Timer2 = 0;
13     Serial.begin(9600);
14     pinMode(D7, OUTPUT);
15     digitalWrite(D7, HeartBeatState);
16 }
17
18 void loop() {
19     if (Timer1<=millis()) {
20         Timer1 = millis()+2000; // Once every 2 seconds read our sensors
21         HeartBeatState = !HeartBeatState;
22         digitalWrite(D7, HeartBeatState);
23         data0 = analogRead(A0);
24         data1 = analogRead(A1);
25         sTemp = String( ( data1 - 620 ) / 12.4 );
26         Serial.print(data0);
27         Serial.print(",");
28         Serial.print(data1);
29         Serial.print(",");
30         Serial.print(sTemp);
31         Serial.println(";");
32     }
33     if (Timer2<=millis()) {
34         Timer2 = millis()+5*60*1000; // Once every 5 minutes
35         Spark.publish("djoTemp", sTemp);
36     }
37 }
38
39
```

Full Example Code for Event Creation

NOTE: This code uses software “timers” as described in the **Quick Lesson - Programming Constructs**. They are not required, but extremely useful, please review that document if you have questions.

Dynamic Webpages - View and control your Photon online!

Creating a webpage that interfaces with your microcontroller project allows you to control the microcontroller and read the values of program variables from anywhere you can access the web. Let's learn how to write a webpage that can read the value of a cloud variable and call a cloud function.

The basic language of the web is called **hypertext markup language (HTML)**. HTML is made up of **tags** (text inside pointy-brackets) and **content**. Tags typically structure the document and describe how to display the content. For example, the content could be text and a tag could specify that the text should be displayed in **bold**.

Every HTML document starts with `<HTML>` and ends with `</HTML>`. In general, for every tag `<TAG>`, there is a matching closing tag `</TAG>`. There are two basic fields inside an HTML document. The `<HEAD>` (header), which is *optional*, defines things like the title bar (`<TITLE>`) and metadata for search engines. The `<BODY>` contains the contents of the document. Many HTML elements exist to make webpage contents look nice, but for this tutorial we will ignore most of these and stick to the basics. You can find links at the bottom of the document for further reading on this topic, if you want to improve the appearance of your webpages.

HTML was originally intended to describe the contents of a webpage and provide links to other webpages. HTML in its natural state doesn't deal well with information that changes (also referred to as dynamic webpages). One way to improve a webpage's ability to process data and change its output is by using a **scripting language** called Javascript. Anything inside `<script>` and `</script>` tags is processed by the web browser as a script rather than an HTML document. Javascript is a programming language, like C++, and a script is a type of program, like the ones you write in C++.

The description of our webpage will start with the following code.

```
<!DOCTYPE HTML>
<html>
  <script src="http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.min.js"
  type="text/javascript" charset="utf-8"></script>
<body>
```

The `<script src="http://...">` above simply loads a library of functions for use in this webpage (like a `#include` statement). You can simply copy and paste this html code at the top of the file for your webpage.

Let's learn how to display the value of Javascript variables in our webpage. The `` tag creates a blank field in a webpage that can be populated with text and updated dynamically. The text around the `` tags describes the text surrounding the field. For example, the first

statement below will display “Current Temperature ____°F”, where the blank can be filled in by updating the value of the `current_temp` variable.

```
Current Temperature:<span id="current_temp"></span>&deg; F<br>
Desired Temperature:<span id="desired_temp"></span>&deg; F<br>
```

The `
` tag at the end of each line creates a line break. This behaves just like the new line character (`\n`). The spans will eventually be accessed in Javascript by their IDs (`current_temp` and `desired_temp`).

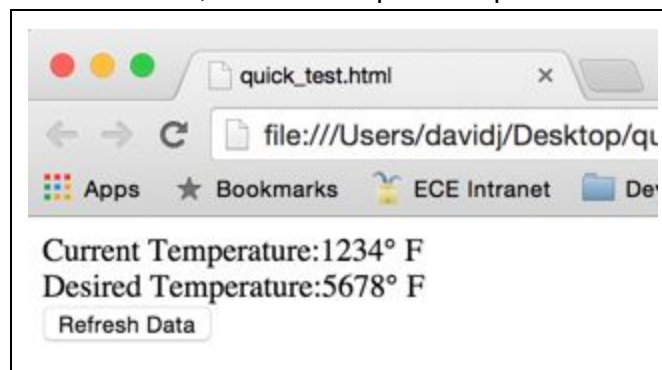
The next item we will place in our webpage is a button. When the button is clicked, it will run a function called **start()**. The text on the button will read **Refresh Data**.

```
<button id="connectbutton" onclick="start()">Refresh Data</button>
```

Now, let’s see how to write data into our spans. We do this by using the **jquery** Javascript library (initialized above), **document** class, and the **innerHTML** function. Here’s a simplified example of how **innerHTML** works:

```
<script type="text/javascript">
document.getElementById("current_temp").innerHTML = "1234";
document.getElementById("desired_temp").innerHTML = "5678";
</script>
```

When embedded in the HTML above, this Javascript would produce the following webpage.



However, we don’t want the temp to be a static number (like 1234 or 5678), but a dynamic number retrieved from our Photon. To do this, we need to replace the `<script>` above with one that does more work.

Now let’s write the `start()` function that gets called when we push the button. Inside this function, we will issue **get requests** to get the values of our cloud variables from the microcontroller. Remember that each cloud variable has a unique URL, and we can get the value of the variable by accessing the URL. To do this, `current_temp` and `desired_temp` must be declared as cloud variables in the microcontroller code. After issuing get requests for

the variables, we will take the values returned from the cloud and display them in our spans. As discussed in **Cloud Variables**, you can get the value of a cloud variable by accessing the URL where the variable is stored in the cloud. Our get requests will simply create the appropriate URL and use the URL to get the variable's value. Don't forget, every microcontroller has its own **device ID** and **access token** that must be used in the URL. The code below appends several text fields together to create a URL called **requestURL** that contains the device ID, the access token, and the requested cloud variable name. Then, the URL is queried using the **getJSON** function. When the variable value is returned, it is placed into the appropriate span by the **innerHTML**.

```
<script type="text/javascript">
function start(objButton) {
    var deviceId = "YOUR_DEVICE_ID_GOES_HERE";
    var accessToken = "YOUR_ACCESS_TOKEN_GOES_HERE";
    var baseUrl = "https://api.particle.io/v1/devices/"

    var varName = "current_temp";
    requestURL = baseUrl + deviceId + "/" + varName + "?access_token=" + accessToken;
    $.getJSON(requestURL, function(json) {
        document.getElementById("current_temp").innerHTML = json.result;
    });

    var varName = "desired_temp";
    requestURL = baseUrl + deviceId + "/" + varName + "?access_token=" + accessToken;
    $.getJSON(requestURL, function(json) {
        document.getElementById("desired_temp").innerHTML = json.result;
    });
}
</script>
```

Notice the **</script>** tag at the end of the HTML code above. This ends the javascript script.


Now that we know how to get the values of cloud variables and put them in our webpage, let's see how to set the value of a variable on the Photon by calling a cloud function. We will do this using a simple HTML form that makes a **post request**. The post request calls a cloud function called **set_temp** and passes the form's input value (**args**) as the string argument of the function. Initially, the form's input field will show the text **(50-90)**, reminding the user that the input temperature for the set_temp function should be between 50-90 degrees. The text on the button will read **Set Temperature**. Since our webpage is now done, we can use the **</body>** tag to end the body of the webpage and the **</html>** tag to end the webpage.

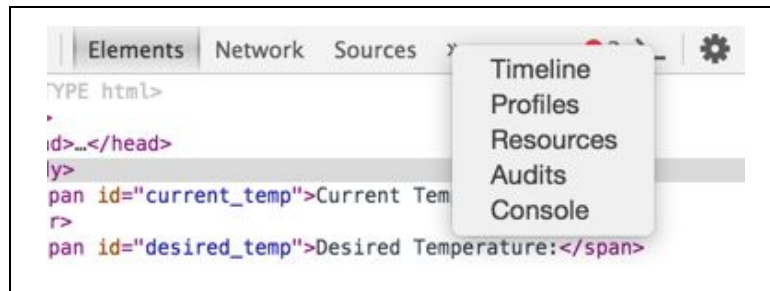
```
<form
action="https://api.particle.io/v1/devices/YOUR\_DEVICE\_ID\_GOES\_HERE/set\_temp?access\_token=YOUR\_ACCESS\_TOKEN\_GOES\_HERE" method="POST">
<input type="text" name="args" value="(50-90)"><br>
<input type="submit" value="Set Temperature">
```

```
</form>
```

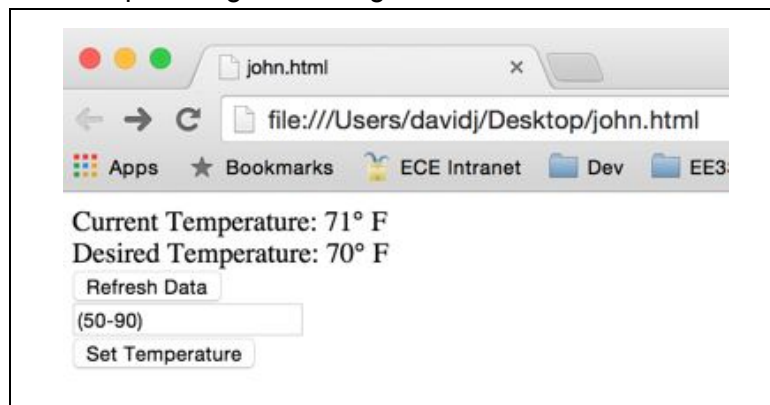
```
</body>
```

```
</html>
```

HINT: You can view Javascript errors on the “Console” in Google Chrome by clicking  → More Tools → Developer Tools, then clicking on the >> and selecting Console. See the picture below:



Our webpage should end up looking something like this:



And the code looks like this:



```
1 <!DOCTYPE HTML>
2 <html>
3   <script src="http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.min.js" type="text/javascript" charset="utf-8"></script>
4   <body>
5     Current Temperature:<span id="current_temp"></span>&deg; F<br>
6     Desired Temperature:<span id="desired_temp"></span>&deg; F<br>
7
8     <button id="connectbutton" onclick="start()">Refresh Data</button>
9
10    <script type="text/javascript">
11      function start(objButton) {
12        var deviceId = "31003f001547343339383037";
13        var accessToken = "cd5d28f314ddb6e6724d9e3bf703cedc493f3802";
14        var baseUrl = "https://api.particle.io/v1/devices/"
15
16        var varName = "current_temp";
17        requestURL = baseUrl + deviceId + "/" + varName + "?access_token=" + accessToken;
18        $.getJSON(requestURL, function(json) {
19          document.getElementById("current_temp").innerHTML = + json.result;
20        });
21
22        var varName = "desired_temp";
23        requestURL = baseUrl + deviceId + "/" + varName + "?access_token=" + accessToken;
24        $.getJSON(requestURL, function(json) {
25          document.getElementById("desired_temp").innerHTML = json.result;
26        });
27      }
28    </script>
29  <br>
30  <form action="https://api.particle.io/v1/devices/YOUR_DEVICE_ID_GOES_HERE/set_temp?access_token=YOUR_ACCESS_TOKEN_GOES_HERE" method="POST">
31    <input type="text" name="args" value="(50-90)"><br>
32    <input type="submit" value="Set Temperature">
33  </form>
34
35 </body>
36 </html>
```

While you may not understand all the details of HTML and Javascript, the good news is that it is relatively simple to copy existing HTML code, like the examples above, and modify it to suit your needs. Instead of writing the code from scratch, copy and paste existing code and modify it to work for the particular cloud variables and cloud functions in your program.

EXERCISE 4

- Create a Particle App and HTML webpage (with Javascript) that does the following:
 - Displays a temperature value sampled by your microcontroller every time you request a sample on the webpage.
 - Has a webpage input that allows you to change the mode on your microcontroller.
 - Indicates the mode of the microcontroller with an LED
- Show your TA the functionality of your IoT Thermostat. Sample some temperature measurements and change modes on your microcontroller using your webpage.

NOTE: You do not need to change modes automatically (i.e., it doesn't have to change modes based on the difference in temperature,) unless you're really excited to do so. ;^)

HINT: You can either use a text input form or a radio button form to select and set the mode. The following webpage gives an example of how to create a radio button form.

http://www.w3schools.com/html/tryit.asp?filename=tryhtml_form_radio

Lab Report

Put together a brief report on the your work in this lab. It should include the following:

- **Intro:** A one paragraph introduction stating what was accomplished in the lab
- **Section 1 (programming):** Describe the code you created in this lab. Add a table to show five temperature values samples from the cloud. Along with each measurement, include a note to describe the conditions under which the measurement was sampled (e.g., room temperature, blowing on sensor, etc.).
- **Section 2 (IFTTT):** Explain how you used IFTTT to receive email notifications when your microcontroller enters REDALERT mode. Also explain how you used IFTTT to log data to a spreadsheet. Include a table that shows the first five rows of the spreadsheet created by your microcontroller.
- **Section 3:** Show a screenshot of the webpage you created to read temperature values and change modes on your microcontroller. Briefly describe the functionality of your webpage.

NOTE: When “describing code”, it is important to break the code into sections and describe how and why you chose the method of implementation you used.

Further Reading

Basic information on HTML tags:

http://www.99lime.com/_bak/topics/you-only-need-10-tags/

Lots of additional information on HTML:

<http://www.w3schools.com/html/default.asp>

An interactive testbed for HTML Forms:

http://www.w3schools.com/html/tryit.asp?filename=tryhtml_form_radio

Javascript Tutorials:

http://www.webmonkey.com/2010/02/get_started_with_jquery/

https://developer.mozilla.org/en-US/Learn/Getting_started_with_the_web/JavaScript_basics