HARDWARE EXAMPLES

Here you will find a bunch of examples to get you started with your new Particle device! The diagrams here show the Photon, but these examples will work with either the Photon or the Core.

These examples are also listed in the online IDE in the Code menu.

To complete all the examples, you will need the following materials:

Materials

- **Hardware**
  - Your Particle device
  - USB to micro USB cable (included with Photon Kit and Maker Kit)
  - Power source for USB cable (such as your computer, USB battery, or power brick)
  - (2) Resistors between 220 Ohms and 1000 Ohms (220 Ohm Resistors included with Photon Kit and Maker Kit)
  - (1) LED, any color (Red LED included with Photon Kit and Maker Kit)
  - (1) Photoresistor or phototransistor (explained below)

Blink an LED

Intro

Blinking an LED is the "Hello World" example of the microcontroller universe. It's a nice way to warm up and start your journey into the land of embedded hardware.

Depending on the kit you have, you may have one or more of the items above. Left to right:

- IR (infrared) LED. It has a blue-ish tint. Don't use that one here, as you can't see the light with the naked eye!
- White LED. It has a rounded top. You can use this instead of the red LED in the examples below.
- Red LED.
- Photo transistor. The top of this is flat and is not an LED, it's a light sensor. You'll use that later.

There may also be smaller red or green LEDs. Those are also fine.

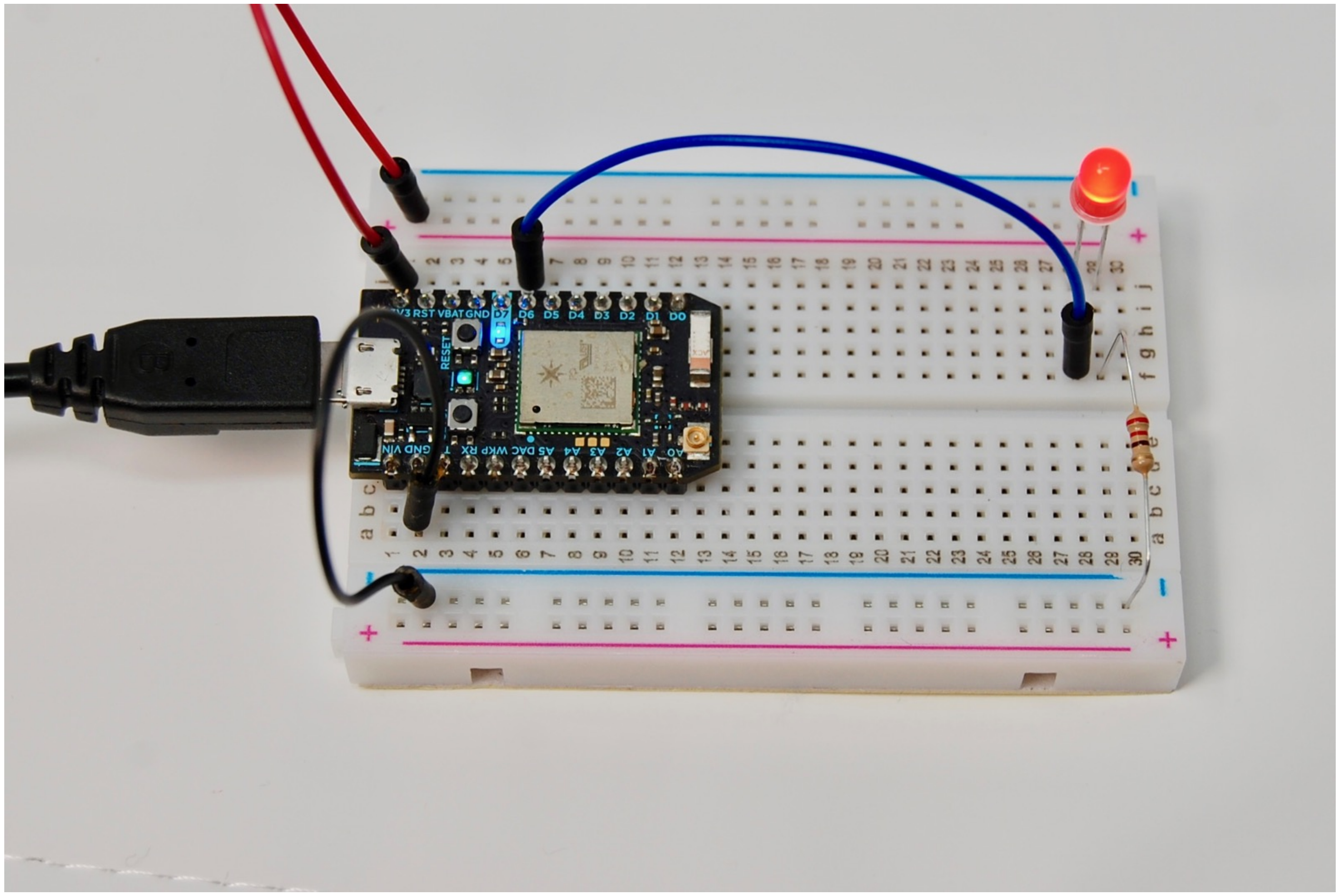
Setup

It's good practice to connect the red (+) bus bar on the top to 3V3 and the blue (-) bus bar on the bottom to ground.

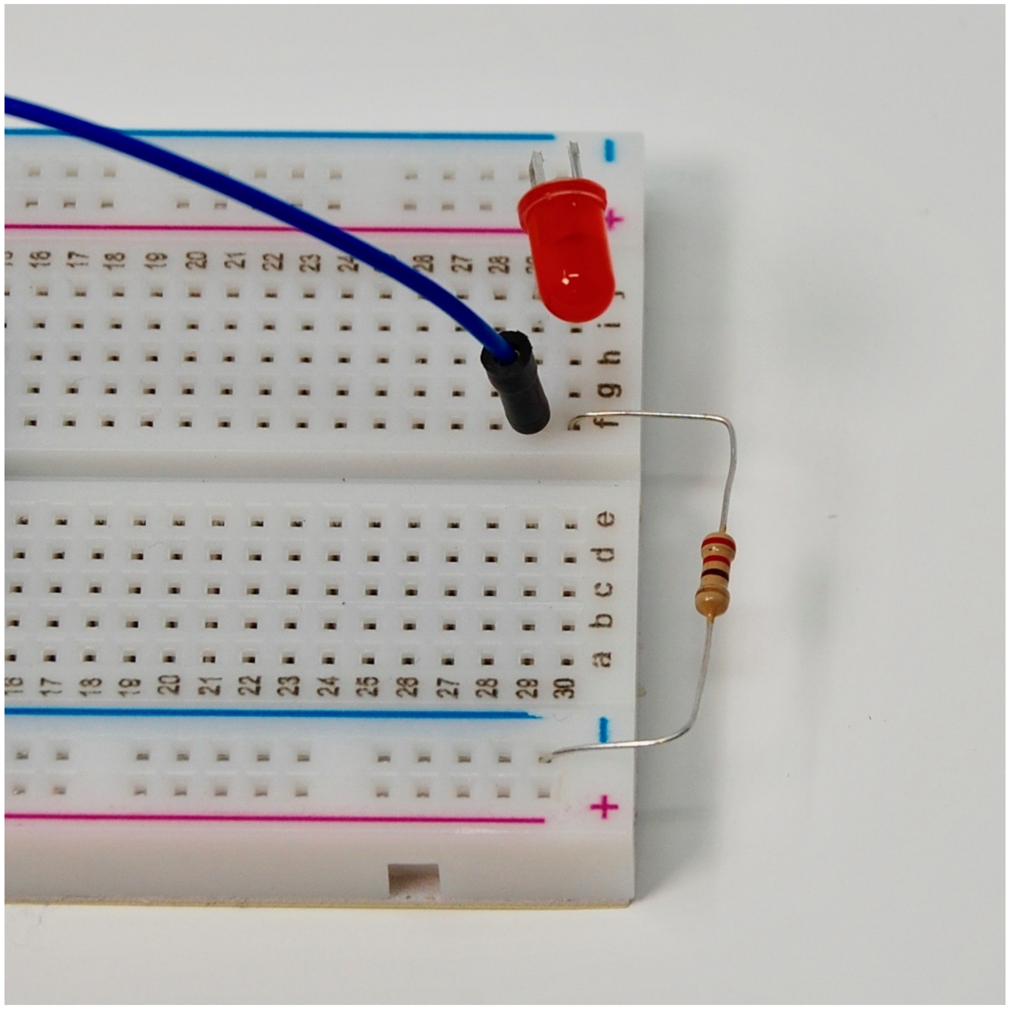
- 3V3 is left-most pin on the top (with the USB connector on the left). It is often connected using a red wire.
- Ground is the second from the left on the bottom. It is typically connected using a black wire.

Position the LED in the breadboard. The long lead (anode) goes to + (left) and the short lead (cathode) goes to - (right). When using an LED, you must always add a current liming resistor. Normally you'd use a 220 ohm resistor (red-red-brown-gold) for 3.3 volt circuits.

In the picture, the long lead of the LED connects to pin D6 using the blue wire. The short lead of the LED connects to a 220 ohm resistor that connects it to ground. That completes the circuit.


3V3 RST VBAT GND D7 D6 D5 D4 D3 D2 D1 D0
RESET
VIN GND TX RX WKP DAC A5 A4 A3 A2 A1 A0

Here's a close-up of the connections



- The long lead of the LED is on the left. The blue wire connects to this lead.
- The short lead of the LED is on the right. This connects to ground with a 220 ohm resistor.

Next, we're going to load code onto your device.

Code

Click this link to open the source code in the Particle Web IDE, or copy and paste the code below.

Go ahead and save this application, then flash it to your device. You should be able to see that LED blinking away!

```
// ------------
// Blink an LED
// ------------

/*-------------

We've heavily commented this code for you. If you're a pro, feel free to ignore
it.

Comments start with two slashes or are blocked off by a slash and a star.
You can read them, but your device can't.
It's like a secret message just for you.

Every program based on Wiring (programming language used by Arduino, and
Particle devices) has two essential parts:
setup - runs once at the beginning of your program
loop - runs continuously over and over

You'll see how we use these in a second.

This program will blink an led on and off every second.
It blinks the D7 LED on your Particle device. If you have an LED wired to D0, it
will blink that LED as well.

-------------*/
```

```
// First, we're going to make some variables.
// This is our "shorthand" that we'll use throughout the program:

int led1 = D6; // Instead of writing D0 over and over again, we'll write led1
// You'll need to wire an LED to this one to see it blink.

int led2 = D7; // Instead of writing D7 over and over again, we'll write led2
// This one is the little blue LED on your board. On the Photon it is next to
D7, and on the Core it is next to the USB jack.

// Having declared these variables, let's move on to the setup function.
// The setup function is a standard part of any microcontroller program.
// It runs only once when the device boots up or is reset.

void setup() {

    // We are going to tell our device that D0 and D7 (which we named led1 and
led2 respectively) are going to be output
    // (That means that we will be sending voltage to them, rather than
monitoring voltage that comes from them)

    // It's important you do this here, inside the setup() function rather than
outside it or in the loop function.

    pinMode(led1, OUTPUT);
    pinMode(led2, OUTPUT);

}

// Next we have the loop function, the other essential part of a microcontroller
program.
```

```
// This routine gets repeated over and over, as quickly as possible and as many
times as possible, after the setup function is called.
// Note: Code that blocks for too long (like more than 5 seconds), can make
weird things happen (like dropping the network connection).  The built-in delay
function shown below safely interleaves required background activity, so
arbitrarily long delays can safely be done if you need them.

void loop() {
    // To blink the LED, first we'll turn it on...
    digitalWrite(led1, HIGH);
    digitalWrite(led2, HIGH);

    // We'll leave it on for 1 second...
    delay(1000);

    // Then we'll turn it off...
    digitalWrite(led1, LOW);
    digitalWrite(led2, LOW);

    // Wait 1 second...
    delay(1000);

    // And repeat!
}
```

Control LEDs over the 'net

Intro

Now that we know how to blink an LED, how about we control it over the Internet? This is where the fun begins.

We've heavily commented the code below so that you can see what's going on. Basically, we are going to:

- Set up the pins as outputs that have LEDs connected to them
- Create and register a Particle function (this gets called automagically when you make an API request to it)
- Parse the incoming command and take appropriate actions

Setup

Set up the LED as in the previous example.

Code

Click this link to open the source code in the Particle Web IDE, or copy and paste the code below.

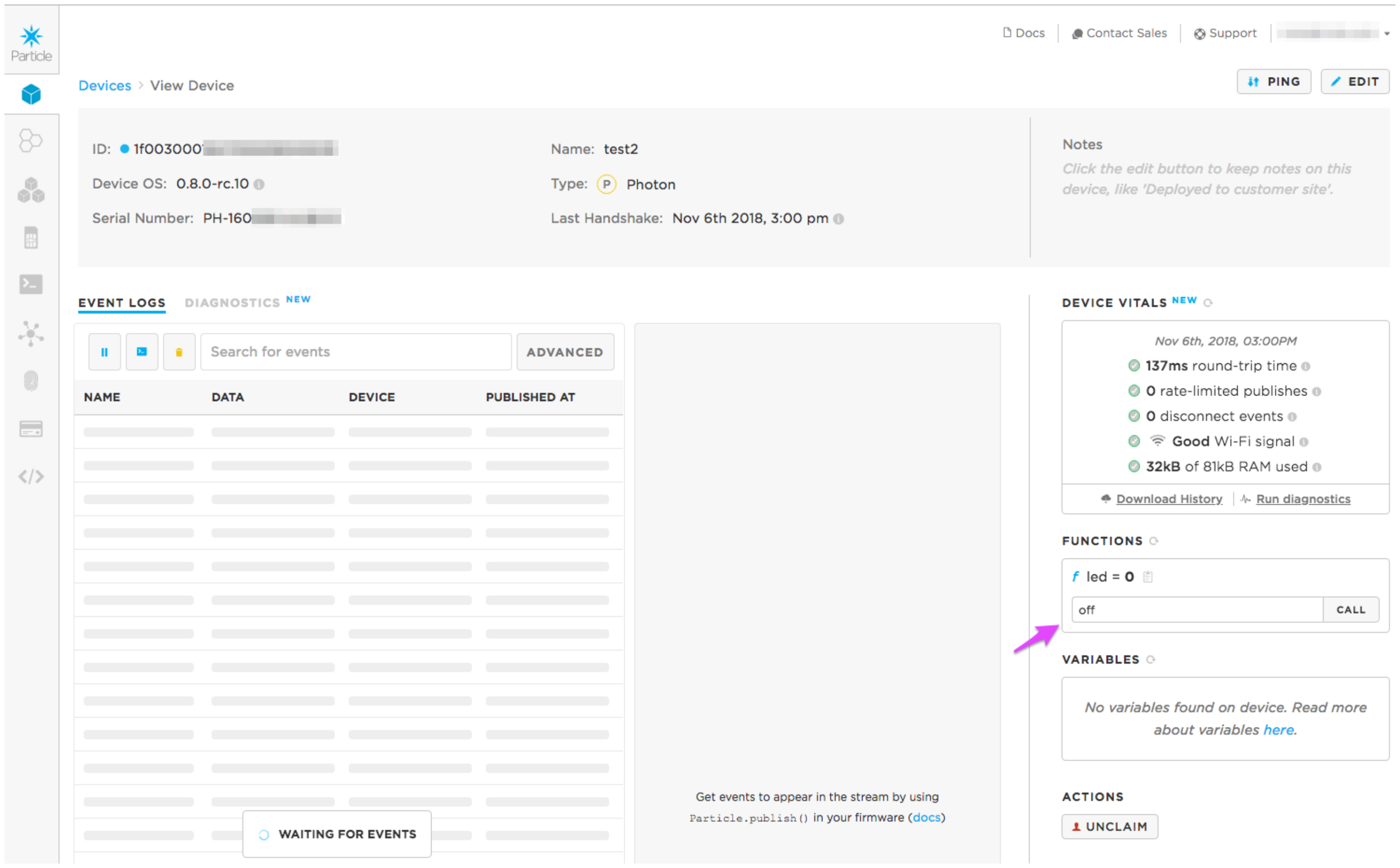
```
// -----------------------------------
// Controlling LEDs over the Internet
// -----------------------------------

/* First, let's create our "shorthand" for the pins
Same as in the Blink an LED example:
led1 is D6, led2 is D7 */
```

```
int led1 = D6;
int led2 = D7;

// Last time, we only needed to declare pins in the setup function.
// This time, we are also going to register our Particle function

void setup()
{

   // Here's the pin configuration, same as last time
   pinMode(led1, OUTPUT);
   pinMode(led2, OUTPUT);

   // We are also going to declare a Particle.function so that we can turn the
LED on and off from the cloud.
   Particle.function("led",ledToggle);
   // This is saying that when we ask the cloud for the function "led", it will
employ the function ledToggle() from this app.

   // For good measure, let's also make sure both LEDs are off when we start:
   digitalWrite(led1, LOW);
   digitalWrite(led2, LOW);

}


/* Last time, we wanted to continously blink the LED on and off
Since we're waiting for input through the cloud this time,
we don't actually need to put anything in the loop */

void loop()
{
   // Nothing to do here
```

```
}

// We're going to have a super cool function now that gets called when a
matching API request is sent
// This is the ledToggle function we registered to the "led" Particle.function
earlier.

int ledToggle(String command) {
    /* Particle.functions always take a string as an argument and return an
integer.
    Since we can pass a string, it means that we can give the program commands
on how the function should be used.
    In this case, telling the function "on" will turn the LED on and telling it
"off" will turn the LED off.
    Then, the function returns a value to us to let us know what happened.
    In this case, it will return 1 for the LEDs turning on, 0 for the LEDs
turning off,
    and -1 if we received a totally bogus command that didn't do anything to the
LEDs.
    */

    if (command=="on") {
        digitalWrite(led1,HIGH);
        digitalWrite(led2,HIGH);
        return 1;
    }
    else if (command=="off") {
        digitalWrite(led1,LOW);
        digitalWrite(led2,LOW);
        return 0;
    }
    else {
        return -1;
        }
}
```

Testing

The easiest way to test your new function is from the console. In the devices tab, click on the row for the device you just flashed your code to.



Docs | Contact Sales | Support

Devices > View Device

PING EDIT

ID: 1f003000[redacted]
Device OS: 0.8.0-rc.10
Serial Number: PH-160[redacted]

Name: test2
Type: Photon
Last Handshake: Nov 6th 2018, 3:00 pm

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

EVENT LOGS DIAGNOSTICS NEW

Search for events ADVANCED

NAME	DATA	DEVICE	PUBLISHED AT

WAITING FOR EVENTS

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

DEVICE VITALS NEW

Nov 6th, 2018, 03:00PM
137ms round-trip time
0 rate-limited publishes
0 disconnect events
Good Wi-Fi signal
32kB of 81kB RAM used

Download History | Run diagnostics

FUNCTIONS

f led = 0
off CALL

VARIABLES

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

ACTIONS

UNCLAIM

On the right-hand side of the screen is a box for functions. There should be one labeled "led". If you type "on" (without the quotes") and click the Call button the LED should turn on.

Use

When we register a function or variable, we're basically making a space for it on the internet, similar to the way there's a space for a website you'd navigate to with your browser. Thanks to the REST API, there's a specific address that identifies you and your device. You can send requests, like `GET` and `POST` requests, to this URL just like you would with any webpage in a browser.

Remember the last time you submitted a form online? You may not have known it, but the website probably sent a `POST` request with the info you put in the form over to another URL that would store your data. We can do the same thing to send information to your device, telling it to turn the LED on and off.

Use the following to view your page:

```
/* Paste the code between the dashes below into a .txt file and save it as an
.html file. Replace your-device-ID-goes-here with your actual device ID and
your-access-token-goes-here with your actual access token.

---------------------------
<!-- Replace your-device-ID-goes-here with your actual device ID
and replace your-access-token-goes-here with your actual access token-->
<!DOCTYPE>
<html>
  <body>
  <center>
  <br>
  <br>
  <br>
  <form action="https://api.particle.io/v1/devices/your-device-ID-goes-here/led?
access_token=your-access-token-goes-here" method="POST">
    Tell your device what to do!<br>
    <br>
    <input type="radio" name="arg" value="on">Turn the LED on.
    <br>
    <input type="radio" name="arg" value="off">Turn the LED off.
    <br>
    <br>
    <input type="submit" value="Do it!">
  </form>
  </center>
  </body>
</html>
---------------------------
*/
```

Edit the code in your text file so that "your-device-ID-goes-here" is your actual device ID, and "your-access-token-goes-here" is your actual access token. These things are accessible through your IDE at build.particle.io. Your device ID can be found in your Devices drawer (the crosshairs) when you click on the device you want to use, and your access token can be found in settings (the cogwheel).

Open that `.html` file in a browser. You'll see a very simple HTML form that allows you to select whether you'd like to turn the LED on or off.

When you click "Do it!" you are posting information to the URL `https://api.particle.io/v1/devices/your-device-ID-goes-here/led?access_token=your-access-token-goes-here`. The information you give is the `args`, or argument value, of `on` or `off`. This hooks up to your `Particle.function` that we registered with the cloud in our firmware, which in turn sends info to your device to turn the LED on or off.

You'll get some info back after you submit the page that gives the status of your device and lets you know that it was indeed able to post to the URL. If you want to go back, just click "back" on your browser.

If you are using the command line, you can also turn the LED on and off by typing:

```
particle call device_name led on
```

and

```
particle call device_name led off
```

Remember to replace `device_name` with either your device ID or the nickname you made for your device when you set it up.

This does the same thing as our HTML page, but with a more slick interface.

The API request will look something like this:

```
POST /v1/devices/{DEVICE_ID}/led

# EXAMPLE REQUEST IN TERMINAL
# Core ID is 0123456789abcdef
# Your access token is 123412341234
curl https://api.particle.io/v1/devices/0123456789abcdef/led \
  -d access_token=123412341234 \
  -d arg=on
```

Note that the API endpoint is 'led', not 'ledToggle'. This is because the endpoint is defined by the first argument of Particle.function(), which is a string of characters, rather than the second argument, which is a function.

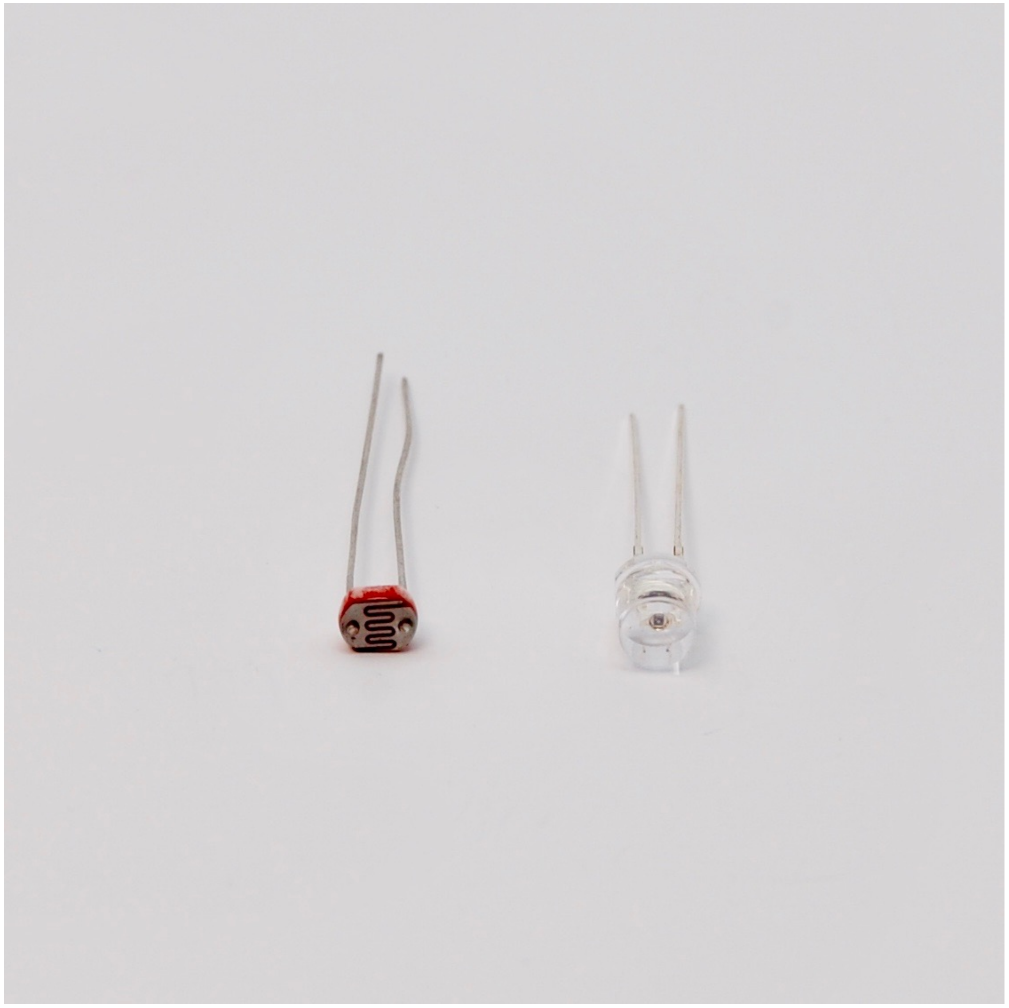
To better understand the concept of making API calls to your device over the cloud checkout the Cloud API reference.

Read your photo sensor: Function and Variable

Intro

This example uses the same setup as the LED control example to make a `Particle.function`. This time, though, we're going to add a sensor.

We will get a value from a photo resistor or photo transistor and store it in the cloud.



Depending on the kit you have, there may be a photo resistor (left, thin and flat with the squiggle pattern on the face) or photo transistor (right, clear and round with a flat top). The photo transistor and a clear LED look similar, but the LED has rounded top and the photo transistor has a flat top.

Select the type of photo sensor in your kit:
Transistor [✓ Resistor]

Paste the following code into your IDE, or just access the examples on the left hand menu bar in the online IDE.

Setup

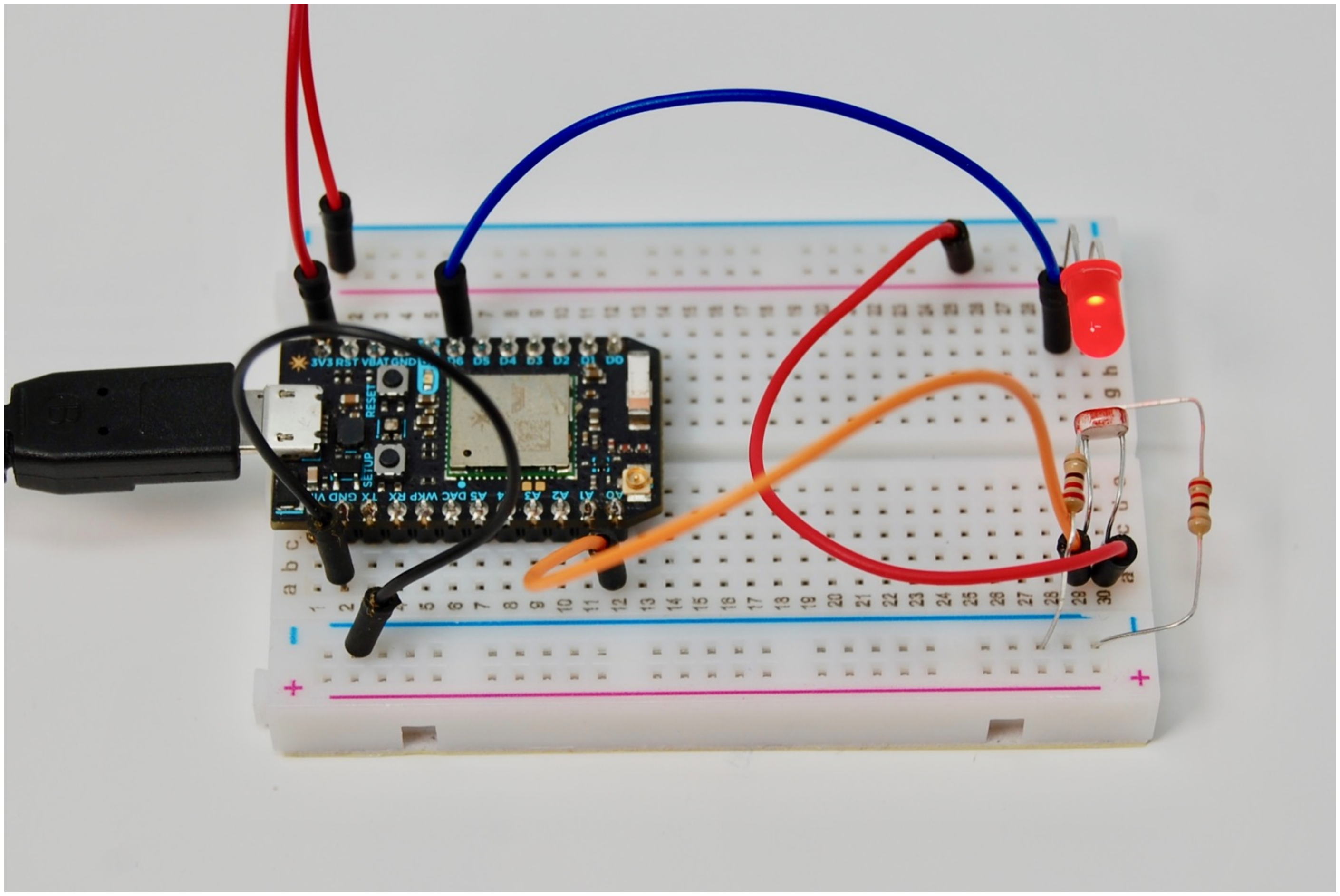
The basic wiring is the same as the previous example:

- The long lead of the LED connects to D6 using the blue wire.
- The short lead of the LED connects to a 220 ohm (red-red-brown-gold) resistor. The other side of the resistor connects to ground.

The photo resistor is connected as follows:

- One lead of the photo resistor connects to 3V3. In the picture, that's the right lead, using the red wire.
- The other lead of the photo resistor connects to another 220 ohm resistor. The other side of the resistor connects to ground.
- The left lead of the photo resistor (connected to the resistor) also connects to pin A0 using the orange wire.

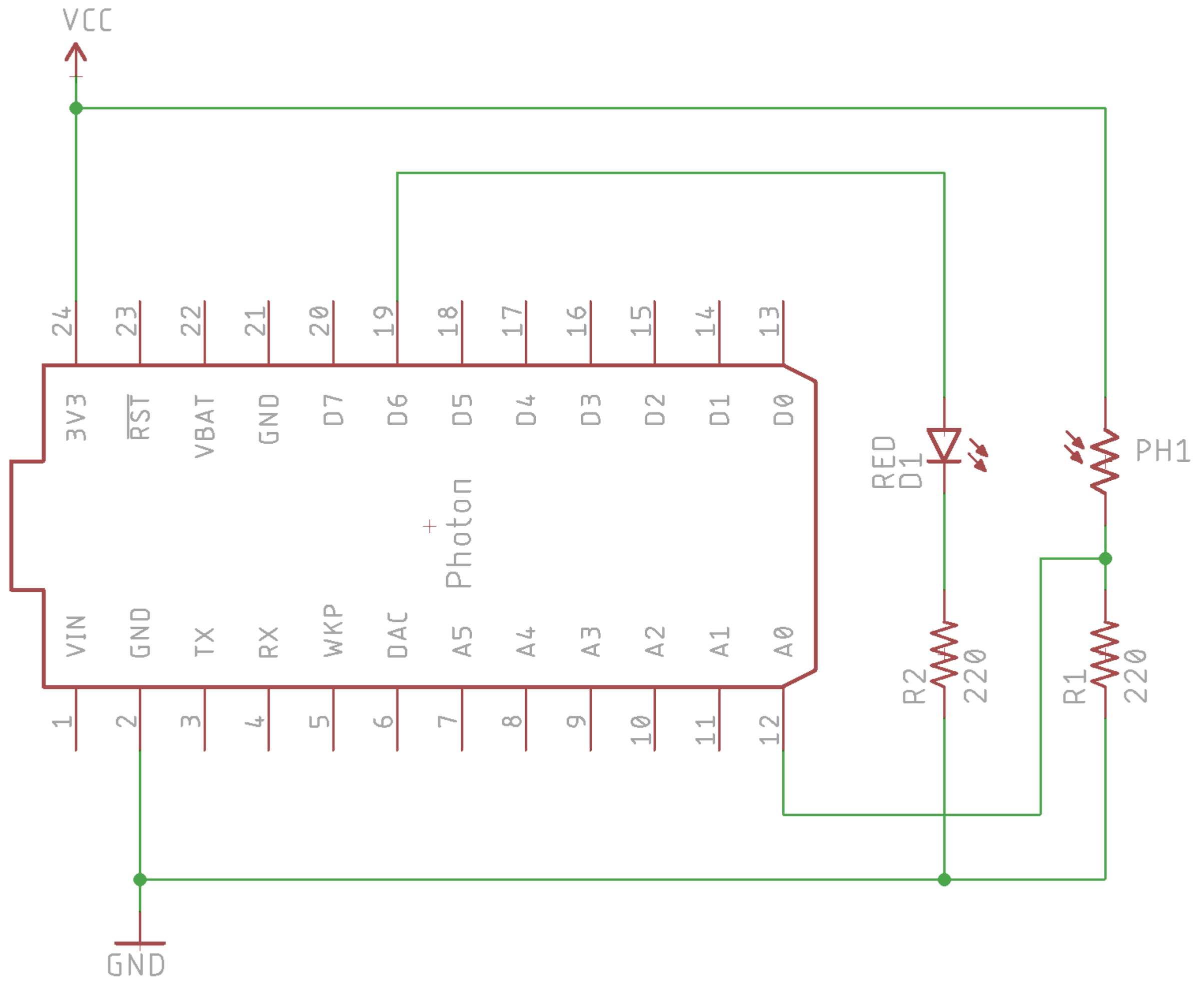
Point the LED and photo resistor at each other.



Bend the LED and the photo sensor so that they are pointing at each other. (You want the LED, when turned on, to shine its beam of light directly at the photo sensor.)

Schematics

A schematic diagram is often use to document a circuit. You don't need to understand schematic diagrams, so feel free to skip this section, but they can make it easier to understand a circuit once you know the basics.

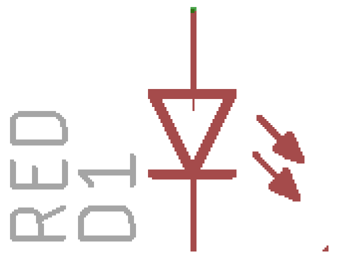


LED

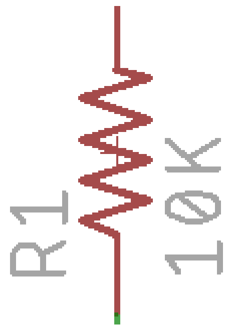
A LED (light emitting diode) produces light.

- The side with bar is the cathode and connects to ground. It's the short lead.
- The side with the triangle is the anode and connects to 3V3/VCC. It's the long lead.

You must always connect an LED using a current-limiting resistor. It can be on either the + or - side.

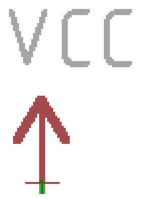


Resistor

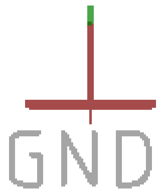


VCC

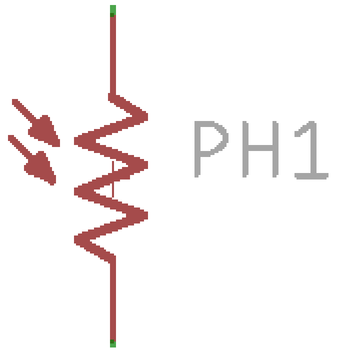
VCC in this diagram is 3V3 (3.3 volts DC).



GND (Ground)



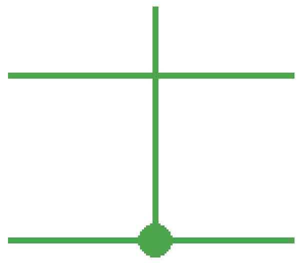
Photoresistor



Lines

The green lines are connections between two pins.

- When the line crosses and there is no dot, then the crossing lines are just passing by each other and are not connected.
- When the line crosses and there's a dot - both lines are connected.



Code

Click this link to open the source code in the Particle Web IDE, or copy and paste the code below.

```
// -----------------------------------------------------------------
// Function and Variable with photo sensors (resistor or transistor)
// -----------------------------------------------------------------
// In this example, we're going to register a Particle.variable() with the cloud
so that we can read brightness levels from the photoresistor or phototransistor.
// We'll also register a Particle.function so that we can turn the LED on and
off remotely.

// We're going to start by declaring which pins everything is plugged into.

int led = D6; // This is where your LED is plugged in. The other side goes to a
resistor connected to GND.

int photosensor = A0; // This is where your photoresistor or phototransistor is
plugged in. The other side goes to the "power" pin (below).

int analogvalue; // Here we are declaring the integer variable analogvalue,
which we will use later to store the value of the photoresistor or
phototransistor.

int ledToggle(String command); // Forward declaration

// Next we go into the setup function.

void setup() {
    // This is here to allow for debugging using the USB serial port
    Serial.begin();
    // First, declare all of our pins. This lets our device know which ones will
be used for outputting voltage, and which ones will read incoming voltage.
    pinMode(led, OUTPUT); // Our LED pin is output (lighting up the LED)
    digitalWrite(led, HIGH);

    // We are going to declare a Particle.variable() here so that we can access
the value of the photosensor from the cloud.
    Particle.variable("analogvalue", &analogvalue, INT);
    // This is saying that when we ask the cloud for "analogvalue", this will
reference the variable analogvalue in this app, which is an integer variable.

    // We are also going to declare a Particle.function so that we can turn the
LED on and off from the cloud.
    Particle.function("led",ledToggle);
    // This is saying that when we ask the cloud for the function "led", it will
employ the function ledToggle() from this app.

}


// Next is the loop function...

void loop() {

    // check to see what the value of the photoresistor or phototransistor is
and store it in the int variable analogvalue
    analogvalue = analogRead(photosensor);

    // This prints the value to the USB debugging serial port (for optional
debugging purposes)
    Serial.printlnf("%d", analogvalue);
    // This delay is just to prevent overflowing the serial buffer, plus we
really don't need to read the sensor more than
    // 10 times per second (100 millisecond delay)
    delay(100);
}


// Finally, we will write out our ledToggle function, which is referenced by the
Particle.function() called "led"

int ledToggle(String command) {

    if (command=="on") {
        digitalWrite(led,HIGH);
        return 1;
    }
    else if (command=="off") {
        digitalWrite(led,LOW);
        return 0;
    }
    else {
        return -1;
    }
}
```

Use

Just like with our earlier example, we can toggle our LED on and off by creating an HTML page:

```
/* Paste the code between the dashes below into a .txt file and save it as an
.html file. Replace your-device-ID-goes-here with your actual device ID and
```

```
your-access-token-goes-here with your actual access token.

---------------------------
<!-- Replace your-device-ID-goes-here with your actual device ID
and replace your-access-token-goes-here with your actual access token-->
<!DOCTYPE>
<html>
  <body>
  <center>
  <br>
  <br>
  <br>
  <form action="https://api.particle.io/v1/devices/your-device-ID-goes-here/led?
access_token=your-access-token-goes-here" method="POST">
    Tell your device what to do!<br>
    <br>
    <input type="radio" name="args" value="on">Turn the LED on.
    <br>
    <input type="radio" name="args" value="off">Turn the LED off.
    <br>
    <br>
    <input type="submit" value="Do it!">
  </form>
  </center>
  </body>
</html>
---------------------------
*/
```

Or we can use the Particle CLI with the command:

```
particle call device_name led on
```

and

```
particle call device_name led off
```

where device_name is your device ID or device name.

As for your Particle.variable, the API request will look something like this:

```
GET /v1/devices/{DEVICE_ID}/analogvalue

# EXAMPLE REQUEST IN TERMINAL
# Core ID is 0123456789abcdef
# Your access token is 123412341234
curl -G https://api.particle.io/v1/devices/0123456789abcdef/analogvalue \
  -d access_token=123412341234
```

You can see a JSON output of your Particle.variable() call by going to:

https://api.particle.io/v1/devices/your-device-ID-goes-here/analogvalue?access_token=your-access-token-goes-here

(Be sure to replace `your-device-ID-goes-here` with your actual device ID and `your-access-token-goes-here` with your actual access token!)

You can also check out this value by using the command line. Type:

```
particle variable get device_name analogvalue
```

and make sure you replace `device_name` with either your device ID or the casual nickname you made for your device when you set it up.

Now you can turn your LED on and off and see the values at A0 change based on the phototransistor or photoresistor!

Make a Motion Detector: Publish and the Console

Intro

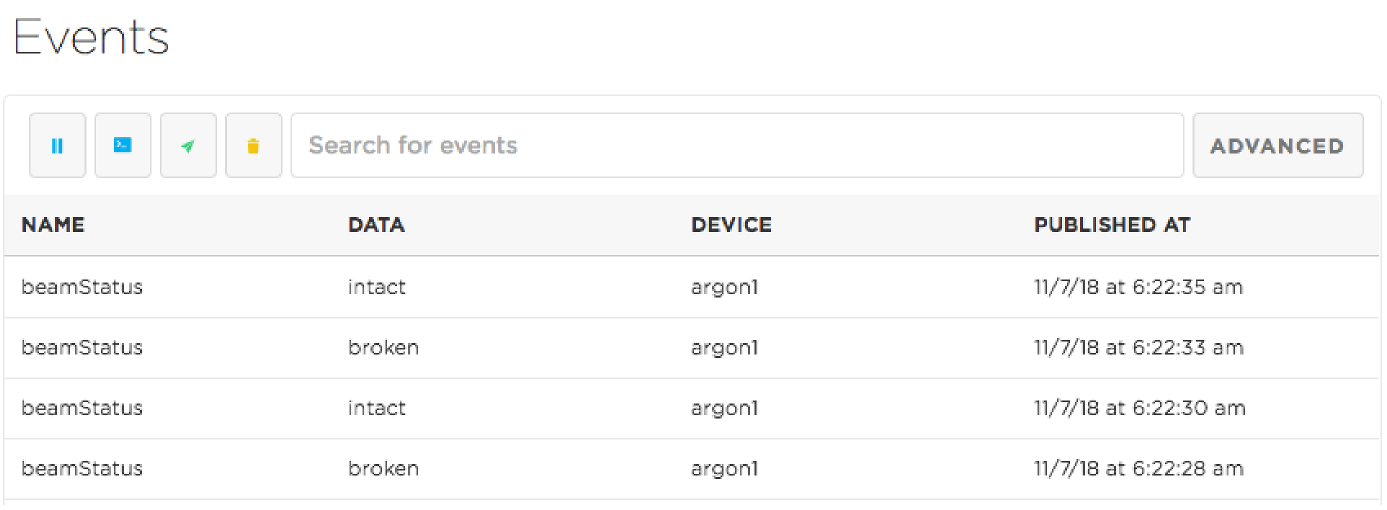
What if we simply want to know that something has happened, without all the information of a variable or all the action of a fuction? We might have a security system that tells us, "motion was detected!" or a smart washing machine that tells us "your laundry is done!" In that case, we might want to use `Particle.publish`.

`Particle.publish` sends a message to the cloud saying that some event has occured. We're allowed to name that event, set the privacy of that event, and add a little bit of info to go along with the event.

In this example, we've created a system where you turn your LED and photo sensor to face each other, making a beam of light that can be broken by the motion of your finger. Every time the beam is broken or reconnected, your device will send a `Particle.publish` to the cloud letting it know the state of the beam. Basically, a tripwire!

For your convenience, we've set up a little calibrate function so that your device will work no matter how bright your LED is, or how bright the ambient light may be. Put your finger in the beam when the D7 LED goes on, and hold it in the beam until you see two flashes from the D7 LED. Then take your finger out of the beam. If you mess up, don't worry-- you can just hit "reset" on your device and do it again!

You can check out the results on your console at console.particle.io. As you put your finger in front of the beam, you'll see an event appear that says the beam was broken. When you remove your finger, the event says that the beam is now intact.



NAME	DATA	DEVICE	PUBLISHED AT
beamStatus	intact	argon1	11/7/18 at 6:22:35 am
beamStatus	broken	argon1	11/7/18 at 6:22:33 am
beamStatus	intact	argon1	11/7/18 at 6:22:30 am
beamStatus	broken	argon1	11/7/18 at 6:22:28 am

You can also hook up publishes to IFTTT! More info here.

Setup

The setup for your breadboard is the same as in the last example.

Code

Click this link to open the source code in the Particle Web IDE, or copy and paste the code below.

```
// ------------------------------------------------------------
// Publish and Subscribe with photoresistors or phototransistors
/* ------------------------------------------------------------

Go find a buddy who also has a Particle device.
Each of you will pick a unique event name
   (make it weird so that no one else will have it)
   (no more that 63 ASCII characters, and no spaces)

In the following code, replace "your_unique_event_name" with your chosen name.
Replace "buddy_unique_event_name" with your buddy's chosen name.

Have your buddy do the same on his or her IDE.

Then, each of you should flash the code to your device.

Breaking the beam on one device will turn on the D7 LED on the second device.

But how does this magic work? Through the miracle of publish and subscribe.

We are going to Particle.publish a public event to the cloud.
That means that everyone can see you event and anyone can subscribe to it.
You and your buddy will both publish an event, and listen for each others
events.

------------------------------------------*/


int led = D6;
int boardLed = D7;
int photosensor = A0;

int intactValue;
int brokenValue;
int beamThreshold;

bool beamBroken = false;

void myHandler(const char *event, const char *data); // forward declaration

// We start with the setup function.

void setup() {
    // This part is mostly the same:
    pinMode(led,OUTPUT); // Our LED pin is output (lighting up the LED)
    pinMode(boardLed,OUTPUT); // Our on-board LED is output as well

    // Here we are going to subscribe to your buddy's event using
Particle.subscribe
    Particle.subscribe("buddy_unique_event_name", myHandler);
    // Subscribe will listen for the event buddy_unique_event_name and, when it
finds it, will run the function myHandler()
    // (Remember to replace buddy_unique_event_name with your buddy's actual
unique event name that they have in their firmware.)
    // myHandler() is declared later in this app.

    // Since everyone sets up their LEDs differently, we are also going to start
by calibrating our photosensor.
    // This one is going to require some input from the user!

    // Calibrate:
    // First, the D7 LED will go on to tell you to put your hand in front of the
beam.
```

```
digitalWrite(boardLed,HIGH);
delay(2000);

// Then, the D7 LED will go off and the LED will turn on.
digitalWrite(boardLed,LOW);
digitalWrite(led,HIGH);
delay(500);

// Now we'll take some readings...
int off_1 = analogRead(photosensor); // read photosensor
delay(200); // wait 200 milliseconds
int off_2 = analogRead(photosensor); // read photosensor
delay(1000); // wait 1 second

// Now flash to let us know that you've taken the readings...
digitalWrite(boardLed,HIGH);
delay(100);
digitalWrite(boardLed,LOW);
delay(100);
digitalWrite(boardLed,HIGH);
delay(100);
digitalWrite(boardLed,LOW);
delay(100);

// Now the D7 LED will go on to tell you to remove your hand...
digitalWrite(boardLed,HIGH);
delay(2000);

// The D7 LED will turn off...
digitalWrite(boardLed,LOW);

// ...And we will take two more readings.
int on_1 = analogRead(photosensor); // read photosensor
```

```
    delay(200); // wait 200 milliseconds
    int on_2 = analogRead(photosensor); // read photosensor
    delay(300); // wait 300 milliseconds

    // Now flash the D7 LED on and off three times to let us know that we're
ready to go!
    digitalWrite(boardLed,HIGH);
    delay(100);
    digitalWrite(boardLed,LOW);
    delay(100);
    digitalWrite(boardLed,HIGH);
    delay(100);
    digitalWrite(boardLed,LOW);
    delay(100);
    digitalWrite(boardLed,HIGH);
    delay(100);
    digitalWrite(boardLed,LOW);

    intactValue = (on_1+on_2)/2;
    brokenValue = (off_1+off_2)/2;
    beamThreshold = (intactValue+brokenValue)/2;

}


void loop() {
    // This loop sends a publish when the beam is broken.
    if (analogRead(photosensor)>beamThreshold) {
        if (beamBroken==true) {
            Particle.publish("your_unique_event_name", "intact", PUBLIC);
            // publish this public event
            // rename your_unique_event_name with your actual unique event name.
No spaces, 63 ASCII characters.
```

```
            // give your event name to your buddy and have them put it in their
app.

            // Set the flag to reflect the current status of the beam.
            beamBroken=false;
        }
    }

    else {
        if (beamBroken==false) {
            // Same deal as before...
            Particle.publish("your_unique_event_name", "broken", PUBLIC);
            beamBroken=true;
        }
    }
}


// Now for the myHandler function, which is called when the cloud tells us that
our buddy's event is published.
void myHandler(const char *event, const char *data)
{
    /* Particle.subscribe handlers are void functions, which means they don't
return anything.
      They take two variables-- the name of your event, and any data that goes
along with your event.
      In this case, the event will be "buddy_unique_event_name" and the data
will be "intact" or "broken"

      Since the input here is a char, we can't do
         data=="intact"
        or
         data=="broken"
      chars just don't play that way. Instead we're going to strcmp(), which
compares two chars.
      If they are the same, strcmp will return 0.
     */

    if (strcmp(data,"intact")==0) {
        // if your buddy's beam is intact, then turn your board LED off
        digitalWrite(boardLed,LOW);
    }
    else if (strcmp(data,"broken")==0) {
        // if your buddy's beam is broken, turn your board LED on
        digitalWrite(boardLed,HIGH);
    }
    else {
        // if the data is something else, don't do anything.
        // Really the data shouldn't be anything but those two listed above.
    }
}
```

The Buddy System: Publish and Subscribe

Intro

In the previous example, we sent a private publish. This publish went to you alone; it was just for you and your own apps, programs, integrations, and devices. We can also send a public publish, though, which allows anyone anywhere to see and subscribe to our event in the cloud. All they need is our event name.

Go find a buddy who has a Core, Photon, or Electron. Your buddy does not have to be next to you--she or he can be across the world if you like! All you need is a connection.

Connect your device and copy the firmware on the right into a new app. Pick a weird name for your event. If your device were named `starfish` for example, you could make your event name something like `starfish_special_event_20150515`. Have your buddy pick a name for their event as well. No more than 63 ASCII characters, and no spaces!

Now replace `your_unique_event_name` with your actual event name and `buddy_unique_event_name` with your buddy's unique event name.

Have your buddy do the same thing, only with their event name and yours (swap 'em).

Flash the firmware to your devices. Calibrate your device when it comes online (same as in the previous example).

When the beam is broken on your device, the D7 LED on your buddy's device will light up! Now you can send little messages to each other in morse code... though if one of you is using an Electron, you should be restrained.

Setup

The setup for your breadboard is the same as in the last example.

Code

Click this link to open the source code in the Particle Web IDE, or copy and paste the code below.

```
// ------------------------------------------------------------
// Publish and Subscribe with photoresistors or phototransistors
/* ------------------------------------------------------------

Go find a buddy who also has a Particle device.
Each of you will pick a unique event name
   (make it weird so that no one else will have it)
   (no more that 63 ASCII characters, and no spaces)

In the following code, replace "your_unique_event_name" with your chosen name.
Replace "buddy_unique_event_name" with your buddy's chosen name.

Have your buddy do the same on his or her IDE.

Then, each of you should flash the code to your device.

Breaking the beam on one device will turn on the D7 LED on the second device.

But how does this magic work? Through the miracle of publish and subscribe.

We are going to Particle.publish a public event to the cloud.
That means that everyone can see you event and anyone can subscribe to it.
You and your buddy will both publish an event, and listen for each others
events.

------------------------------------------*/


int led = D6;
int boardLed = D7;
int photosensor = A0;
int intactValue;
int brokenValue;
int beamThreshold;

bool beamBroken = false;

// We start with the setup function.

void setup() {
    // This part is mostly the same:
    pinMode(led,OUTPUT); // Our LED pin is output (lighting up the LED)
    pinMode(boardLed,OUTPUT); // Our on-board LED is output as well

    // Here we are going to subscribe to your buddy's event using
Particle.subscribe
    Particle.subscribe("buddy_unique_event_name", myHandler);
    // Subscribe will listen for the event buddy_unique_event_name and, when it
finds it, will run the function myHandler()
    // (Remember to replace buddy_unique_event_name with your buddy's actual
unique event name that they have in their firmware.)
    // myHandler() is declared later in this app.

    // Since everyone sets up their LEDs differently, we are also going to start
by calibrating our photosensor.
    // This one is going to require some input from the user!

    // Calibrate:
    // First, the D7 LED will go on to tell you to put your hand in front of the
beam.
    digitalWrite(boardLed,HIGH);
    delay(2000);
```

```
  // Then, the D7 LED will go off and the LED will turn on.
  digitalWrite(boardLed,LOW);
  digitalWrite(led,HIGH);
  delay(500);

  // Now we'll take some readings...
  int off_1 = analogRead(photosensor); // read photosensor
  delay(200); // wait 200 milliseconds
  int off_2 = analogRead(photosensor); // read photosensor
  delay(1000); // wait 1 second

  // Now flash to let us know that you've taken the readings...
  digitalWrite(boardLed,HIGH);
  delay(100);
  digitalWrite(boardLed,LOW);
  delay(100);
  digitalWrite(boardLed,HIGH);
  delay(100);
  digitalWrite(boardLed,LOW);
  delay(100);

  // Now the D7 LED will go on to tell you to remove your hand...
  digitalWrite(boardLed,HIGH);
  delay(2000);

  // The D7 LED will turn off...
  digitalWrite(boardLed,LOW);

  // ...And we will take two more readings.
  int on_1 = analogRead(photosensor); // read photosensor
  delay(200); // wait 200 milliseconds
  int on_2 = analogRead(photosensor); // read photosensor
  delay(300); // wait 300 milliseconds
```

```
    // Now flash the D7 LED on and off three times to let us know that we're
ready to go!
    digitalWrite(boardLed,HIGH);
    delay(100);
    digitalWrite(boardLed,LOW);
    delay(100);
    digitalWrite(boardLed,HIGH);
    delay(100);
    digitalWrite(boardLed,LOW);
    delay(100);
    digitalWrite(boardLed,HIGH);
    delay(100);
    digitalWrite(boardLed,LOW);

    intactValue = (on_1+on_2)/2;
    brokenValue = (off_1+off_2)/2;
    beamThreshold = (intactValue+brokenValue)/2;

}


void loop() {
    // This loop sends a publish when the beam is broken.
    if (analogRead(photosensor)>beamThreshold) {
        if (beamBroken==true) {
            Particle.publish("your_unique_event_name","intact");
            // publish this public event
            // rename your_unique_event_name with your actual unique event name.
No spaces, 63 ASCII characters.
            // give your event name to your buddy and have them put it in their
app.
```

```
            // Set the flag to reflect the current status of the beam.
            beamBroken=false;
        }
    }

    else {
        if (beamBroken==false) {
            // Same deal as before...
            Particle.publish("your_unique_event_name","broken");
            beamBroken=true;
        }
    }
}


// Now for the myHandler function, which is called when the cloud tells us that
our buddy's event is published.
void myHandler(const char *event, const char *data)
{
    /* Particle.subscribe handlers are void functions, which means they don't
return anything.
      They take two variables-- the name of your event, and any data that goes
along with your event.
      In this case, the event will be "buddy_unique_event_name" and the data
will be "intact" or "broken"

      Since the input here is a char, we can't do
         data=="intact"
        or
         data=="broken"

      chars just don't play that way. Instead we're going to strcmp(), which
compares two chars.
```

```
       If they are the same, strcmp will return 0.
      */

    if (strcmp(data,"intact")==0) {
        // if your buddy's beam is intact, then turn your board LED off
        digitalWrite(boardLed,LOW);
    }
    else if (strcmp(data,"broken")==0) {
        // if your buddy's beam is broken, turn your board LED on
        digitalWrite(boardLed,HIGH);
    }
    else {
        // if the data is something else, don't do anything.
        // Really the data shouldn't be anything but those two listed above.
    }
}
```

Tinker

Remember back when we were blinking lights and reading sensors with Tinker on the mobile app?

When you tap a pin on the mobile app, it sends a message up to the cloud. Your device is always listening to the cloud and waiting for instructions-- like "write D7 HIGH" or "read the voltage at A0".

Your device already knew how to communicate with the mobile app because of the firmware loaded onto your device as a default. We call this the Tinker firmware. It's just like the user firmware you've been loading onto your device in these examples. It's just that with the Tinker firmware, we've specified special `Particle.function`s that the mobile app knows and understands.

If your device is new, it already has the Tinker firmware on it. It's the default firmware stored on your device right from the factory. When you put your own user firmware on your device, you'll rewrite the Tinker firmware. (That means that your device will no longer understand commands from the Particle mobile app.) However, you can always get the Tinker firmware back on your device by putting it in factory reset mode, or by re-flashing your device with Tinker in the Particle app.

To reflash Tinker from within the app:

- **iOS Users**: Tap the list button at the top left. Then tap the arrow next to your desired device and tap the "Re-flash Tinker" button in the pop out menu.
- **Android Users**: With your desired device selected, tap the options button in the upper right and tap the "Reflash Tinker" option in the drop down menu.

The Tinker app is a great example of how to build a very powerful application with not all that much code. If you're a technical person, you can have a look at the latest release here.

I know what you're thinking: this is amazing, but I really want to use Tinker *while* my code is running so I can see what's happening! Now you can.

Combine your code with this framework, flash it to your device, and Tinker away. You can also access Tinker code by clicking on the last example in the online IDE's code menu.

```
/* Function prototypes -------------------------------------------------------*/
int tinkerDigitalRead(String pin);
int tinkerDigitalWrite(String command);
int tinkerAnalogRead(String pin);
int tinkerAnalogWrite(String command);

SYSTEM_MODE(AUTOMATIC);

/* This function is called once at start up ----------------------------------*/
void setup()
{
    //Setup the Tinker application here

    //Register all the Tinker functions
    Particle.function("digitalread", tinkerDigitalRead);
    Particle.function("digitalwrite", tinkerDigitalWrite);

    Particle.function("analogread", tinkerAnalogRead);
    Particle.function("analogwrite", tinkerAnalogWrite);
}

/* This function loops forever --------------------------------------------*/
void loop()
{
    //This will run in a loop
}

/*******************************************************************************
 * Function Name  : tinkerDigitalRead
 * Description    : Reads the digital value of a given pin
 * Input          : Pin
 * Output         : None.
 * Return         : Value of the pin (0 or 1) in INT type
                    Returns a negative number on failure

*******************************************************************************/
int tinkerDigitalRead(String pin)
```

```
{
    //convert ASCII to integer
    int pinNumber = pin.charAt(1) - '0';
    //Sanity check to see if the pin numbers are within limits
    if (pinNumber < 0 || pinNumber > 7) return -1;

    if(pin.startsWith("D"))
    {
        pinMode(pinNumber, INPUT_PULLDOWN);
        return digitalRead(pinNumber);
    }
    else if (pin.startsWith("A"))
    {
        pinMode(pinNumber+10, INPUT_PULLDOWN);
        return digitalRead(pinNumber+10);
    }
#if Wiring_Cellular
    else if (pin.startsWith("B"))
    {
        if (pinNumber > 5) return -3;
        pinMode(pinNumber+24, INPUT_PULLDOWN);
        return digitalRead(pinNumber+24);
    }
    else if (pin.startsWith("C"))
    {
        if (pinNumber > 5) return -4;
        pinMode(pinNumber+30, INPUT_PULLDOWN);
        return digitalRead(pinNumber+30);
    }
#endif
    return -2;
}
```

```
/*******************************************************************************
 * Function Name  : tinkerDigitalWrite
 * Description    : Sets the specified pin HIGH or LOW
 * Input          : Pin and value
 * Output         : None.
 * Return         : 1 on success and a negative number on failure

*******************************************************************************/
int tinkerDigitalWrite(String command)
{
    bool value = 0;
    //convert ASCII to integer
    int pinNumber = command.charAt(1) - '0';
    //Sanity check to see if the pin numbers are within limits
    if (pinNumber < 0 || pinNumber > 7) return -1;

    if(command.substring(3,7) == "HIGH") value = 1;
    else if(command.substring(3,6) == "LOW") value = 0;
    else return -2;

    if(command.startsWith("D"))
    {
        pinMode(pinNumber, OUTPUT);
        digitalWrite(pinNumber, value);
        return 1;
    }
    else if(command.startsWith("A"))
    {
        pinMode(pinNumber+10, OUTPUT);
        digitalWrite(pinNumber+10, value);
        return 1;
    }
#if Wiring_Cellular
```

```
    else if(command.startsWith("B"))
    {
        if (pinNumber > 5) return -4;
        pinMode(pinNumber+24, OUTPUT);
        digitalWrite(pinNumber+24, value);
        return 1;
    }
    else if(command.startsWith("C"))
    {
        if (pinNumber > 5) return -5;
        pinMode(pinNumber+30, OUTPUT);
        digitalWrite(pinNumber+30, value);
        return 1;
    }
#endif
    else return -3;
}

/*******************************************************************************
 * Function Name  : tinkerAnalogRead
 * Description    : Reads the analog value of a pin
 * Input          : Pin
 * Output         : None.
 * Return         : Returns the analog value in INT type (0 to 4095)
                    Returns a negative number on failure

 *******************************************************************************/
int tinkerAnalogRead(String pin)
{
    //convert ASCII to integer
    int pinNumber = pin.charAt(1) - '0';
    //Sanity check to see if the pin numbers are within limits
    if (pinNumber < 0 || pinNumber > 7) return -1;
    if(pin.startsWith("D"))
    {
        return -3;
    }
    else if (pin.startsWith("A"))
    {
        return analogRead(pinNumber+10);
    }
#if Wiring_Cellular
    else if (pin.startsWith("B"))
    {
        if (pinNumber < 2 || pinNumber > 5) return -3;
        return analogRead(pinNumber+24);
    }
#endif
    return -2;
}

/*******************************************************************************
 * Function Name  : tinkerAnalogWrite
 * Description    : Writes an analog value (PWM) to the specified pin
 * Input          : Pin and Value (0 to 255)
 * Output         : None.
 * Return         : 1 on success and a negative number on failure

*******************************************************************************/
int tinkerAnalogWrite(String command)
{
    String value = command.substring(3);

    if(command.substring(0,2) == "TX")
    {
        pinMode(TX, OUTPUT);
        analogWrite(TX, value.toInt());
        return 1;
    }
    else if(command.substring(0,2) == "RX")
    {
        pinMode(RX, OUTPUT);
        analogWrite(RX, value.toInt());
        return 1;
    }

    //convert ASCII to integer
    int pinNumber = command.charAt(1) - '0';
    //Sanity check to see if the pin numbers are within limits

    if (pinNumber < 0 || pinNumber > 7) return -1;

    if(command.startsWith("D"))
    {
        pinMode(pinNumber, OUTPUT);
        analogWrite(pinNumber, value.toInt());
        return 1;
    }
    else if(command.startsWith("A"))
    {
        pinMode(pinNumber+10, OUTPUT);
        analogWrite(pinNumber+10, value.toInt());
        return 1;
    }
    else if(command.substring(0,2) == "TX")
    {
        pinMode(TX, OUTPUT);
        analogWrite(TX, value.toInt());
        return 1;
    }
    else if(command.substring(0,2) == "RX")
    {
        pinMode(RX, OUTPUT);
        analogWrite(RX, value.toInt());
        return 1;
    }
#if Wiring_Cellular
    else if (command.startsWith("B"))
    {
        if (pinNumber > 3) return -3;
        pinMode(pinNumber+24, OUTPUT);
        analogWrite(pinNumber+24, value.toInt());
        return 1;
    }
    else if (command.startsWith("C"))
    {
        if (pinNumber < 4 || pinNumber > 5) return -4;
        pinMode(pinNumber+30, OUTPUT);
        analogWrite(pinNumber+30, value.toInt());
        return 1;
    }
#endif
    else return -2;
}
```