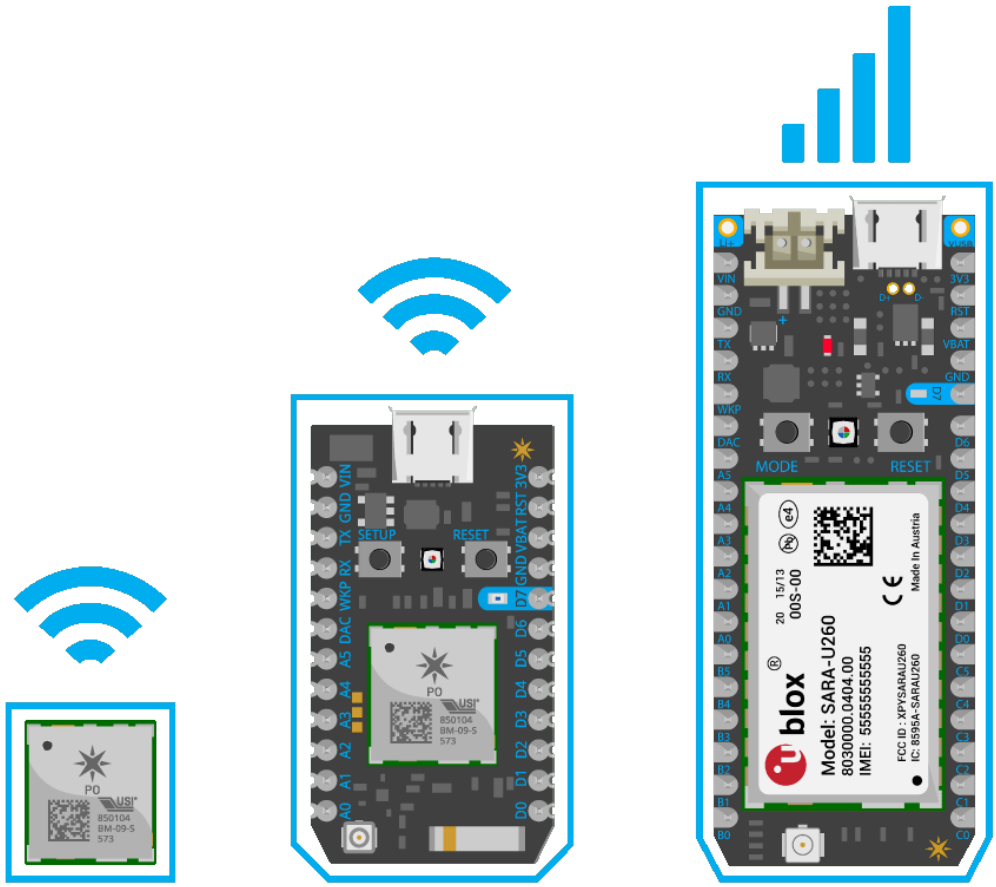
INTRODUCTION TO THE GUIDE



This guide will show you how to use Particle to make a connected device, from the first time you connect to the cloud to the day you manufacture and ship your connected product.

Anxious to connect right away?

SETUP MY PHOTON

The guide is broken down into three parts:

- **Getting Started** goes over how to connect your device and then dives into some examples to get you used to the connected platform. If you've never played with connected hardware before, Getting Started is the section for you! It's filled with small that you can hover over or (on smaller screens and

mobile) click or tap for more info. If you're an expert, you may want to take the first few chapters to connect your device, then move on to Tools and Features.

- **Tools and Features** details the different parts of the Particle platform that come in handy as you build a connected product. It includes using Particle Dev, the Console, the Command Line Interface (CLI) and more.
- **How to Build a Product** is the final section. It reviews everything you'll need to do when taking the connected device you've prototyped into full production and deployment.

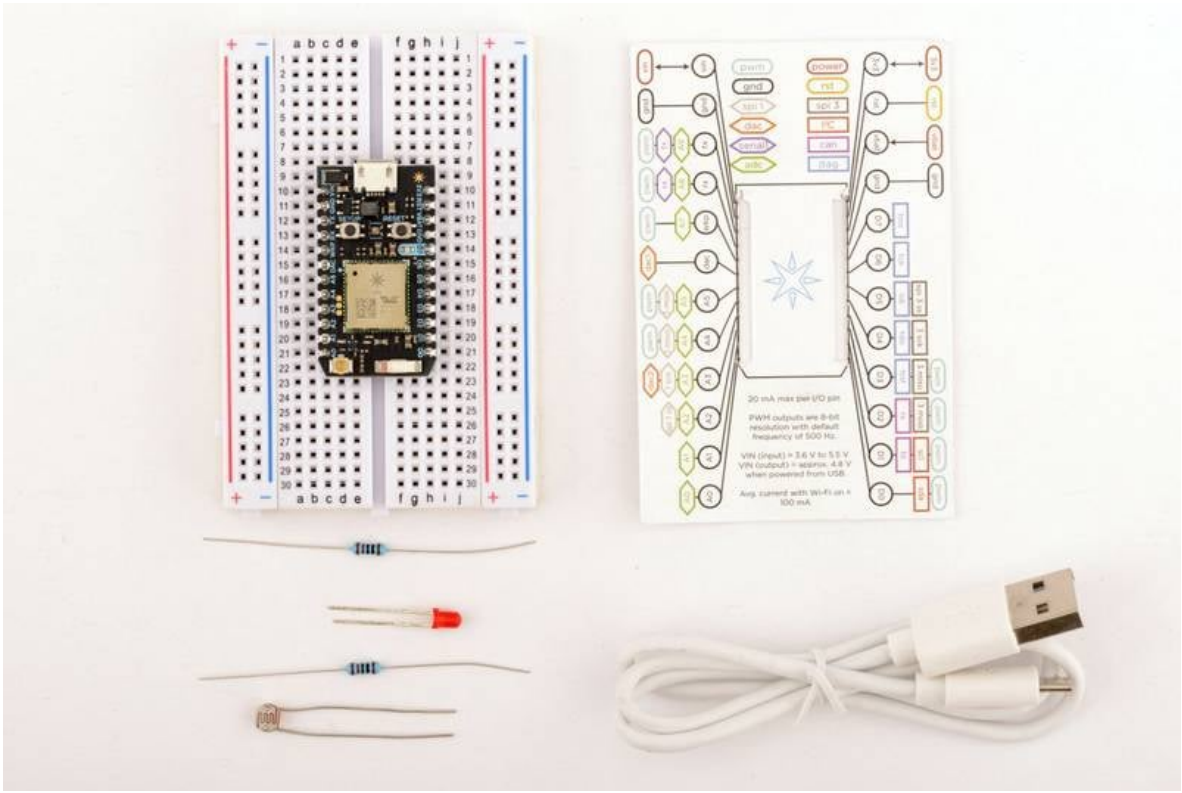
The best way to use the guide is:

- Go through **Getting Started**
- Review **Tools and Features**
- Check out the **Tutorials section** for cool projects you can build
- Search the **Community forums** for advice relevant to your exact project
- Make sure to read **How to Build a Product** if you're going to production!

Ready to get started? Click the arrow to the right of the screen to read on.

GETTING STARTED

What's in the Box?



Your new Photon! Note that many components pictured will only be included if you purchased a Photon Kit.

Congratulations on being the owner of a brand new Particle Device! Go ahead and open the box. You can see the different kit addons and check out the Photon datasheet if you like!

If you have an Internet Button, read through this section to get started and connect your device, then hop over to the Internet Button Guide for more detailed info.

Let's quickly go over what you see.

What's on it?

This is probably why you bought your device-- the Wi-Fi module allows your Photon to communicate with the internet. It connects your device to the internet in the same way that your smartphone might connect to a wifi network. **Do not press down on the Photon's module.** Doing so triggers a reset and is generally not good for the Photon.

The microcontroller is the brain of your device. It runs your software and tells your prototype what to do. Unlike your computer, it can only run one application (often called *firmware* or an *embedded application*). This application can be simple (just a few lines of code), or very complex, depending on what you want to do. The microcontroller interacts with the outside world using pins.

Pins are the input and output parts of the microcontroller that are exposed on the sides of your device. GPIO pins can be hooked to sensors or buttons to listen to the world, or they can be hooked to lights and buzzers to act upon the world. There are also pins to allow you to power your device, or power motors and outputs outside of your device. There are pins for Serial/UART communication, and a pin for resetting your device.

and

There are several awesome buttons and LEDs on your Photon to make it easier to use.

- The `SETUP` button is on the left and the `RESET` button is on the right. You can use these buttons to help you set your device's mode.
- The **RGB LED** is in the center of your Photon, above the module. The color of the RGB LED tells you what mode your Photon is currently in.
- The **D7 LED** is next to the D7 pin on your Photon, on the upper right quadrant. This LED will turn on when the D7 pin is set to `HIGH`.

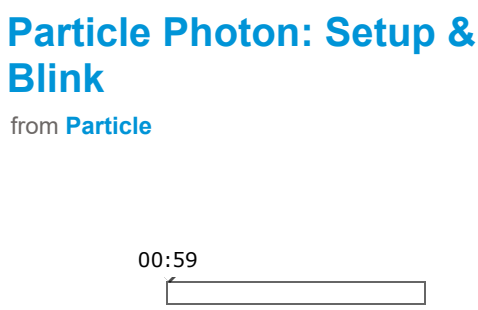
- **Software**
    - Particle Mobile App - iPhone | Android | Windows
    - *Note: We highly recommend using the mobile app for first time setup.*
- **Hardware**
    - Your Particle device, brand new and out of the box!

- 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)
- Your iPhone or Android or Windows smartphone

- **Wi-Fi Settings**
  - 2.4GHz capable router
  - Channels 1-11
  - WPA/WPA2 encryption
  - On a broadcast SSID network
  - Not behind a hard firewall or Enterprise network
  - *Note: We do not recommend using WEP Wi-Fi settings, for security reasons.*
- **Experience**
  - None! This is your first project.

Connect Your Photon

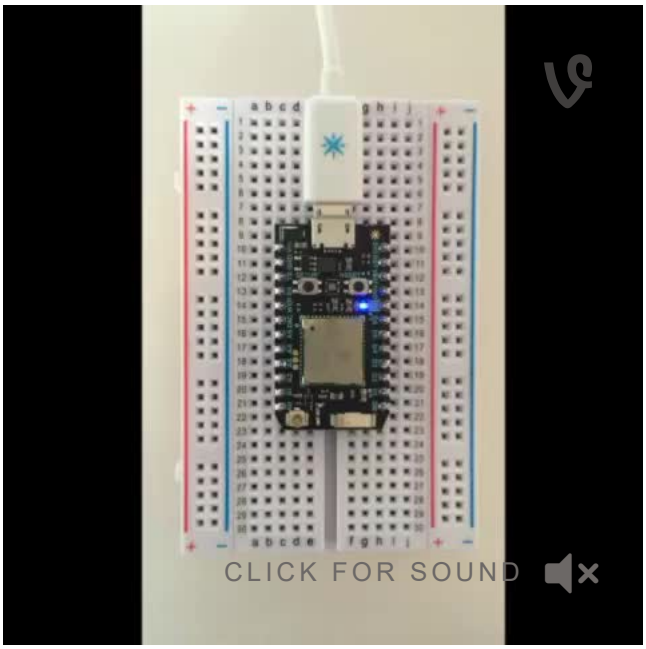
In this example, we will connect your device to the internet for the very first time. Then, we will blink the D7 LED on your device by using your smartphone.



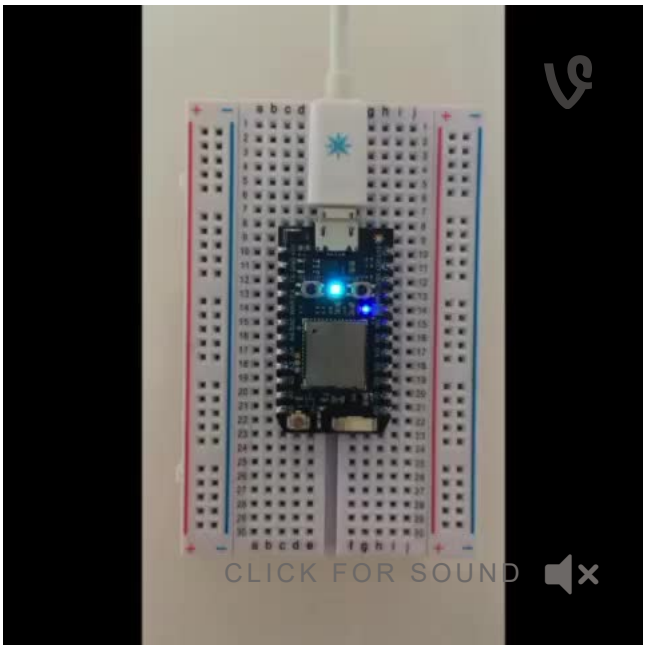
Step 1: Power On Your Device

Plug the USB cable into your power source.

As soon as it is plugged in, the RGB LED on your device should begin



If your device is not blinking blue,



If your device is not blinking at all, or if the LED is burning a dull orange color, it may not be getting enough power. Try changing your power source or USB cable.

Step 2a: Connect your Photon to the Internet using the setup web application

Note: This process only works in Chrome / Firefox / Opera

- **Step 1** Go to setup.particle.io
- **Step 2** Click on `Setup a Photon`
- **Step 3** After clicking on `NEXT`, you should be presented with a file (`photonsetup.html`)
- **Step 4** Open the file

After opening the file:

- **Step 5** Connect your PC to the Photon, by connecting to the network named `PHOTON-...`

- **Step 6** Configure your Wi-Fi credentials

Note: If you mistyped your credentials, the Photon will blink dark blue or green. You have to go through the process again (by refreshing the page or clicking on the retry process part)

- **Step 7** Rename your device. You will also see a confirmation if the device was claimed or not

Note: Make sure your Photon is not part of a product before claiming it

Why a separate file?

We care a lot about security, and we want to make sure that everything you do is safe. Downloading a local file ensures that the credentials are sent directly to the Photon, without any chance of being intercepted.

Step 2b: Connect your Photon to the Internet using your smartphone

Open the app on your phone. Log in or sign up for an account with Particle if you don't have one.

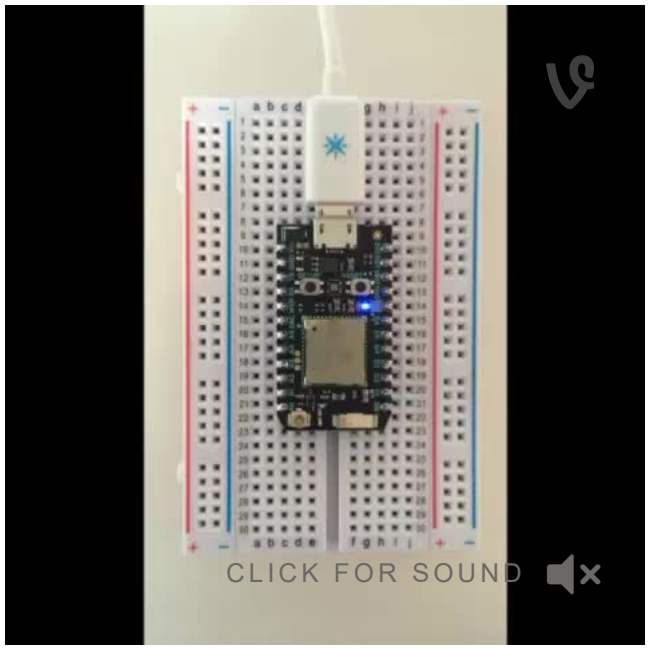
Press the plus icon and select the device you'd like to add. Then follow the instructions on the screen to Remember that to connect the Core, you need the older Spark Core app and to connect the Photon you need the new Particle App.

This may take a little while - but don't worry.

If this is your Photon's first time connecting, it will blink purple for a few minutes as it downloads updates. **This is perfectly normal.** It may take 6-12 minutes for the updates to complete, depending on your internet connection, with the Photon restarting a few times in the process. **Please do not restart or unplug your Photon during this time.** If you do, you may need to follow this guide to fix your device.

If you can't seem to get the Mobile App to connect your device, that's okay! Read over this example quickly, and then check out the next lesson to connect your device using the USB cable.

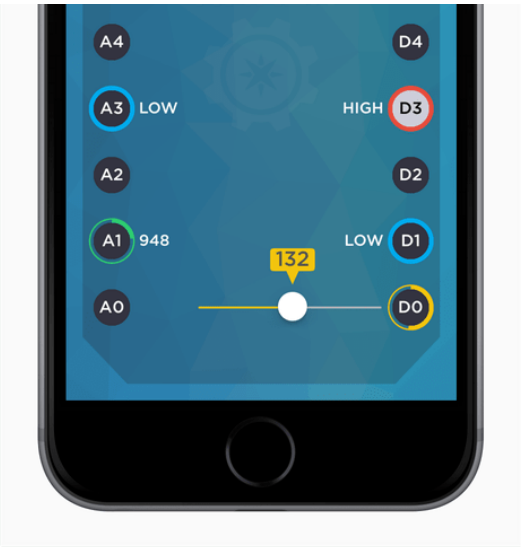
Once you have connected your device, it has learned that network. Your device can store up to five networks. To add a new network after your initial setup, you'd put your device into



again and proceed as above (the claiming part can be skipped). If you feel like your device has too many networks on it, you can wipe your device's memory of any Wi-Fi networks it has learned. You can do so by continuing to hold the SETUP button for 10 seconds until the RGB LED flashes blue quickly, signaling that all profiles have been deleted.

Step 3: Blink an LED!

The Particle App should now be on the screen, as shown below.



As you can see on your smartphone, the circles represent different pins on your device. If you tap on these circles, you can see the Tinker functions available for the associated pins.

We could use Tinker and the smartphone app to talk to any pin on your device. If you had a buzzer, an LED, a sensor, etc., you could interact with it using Tinker on your phone. But since I know you're very eager to get started, let's use an LED already provided on your device.

The D7 pin comes already wired to a small blue LED on the face of your device. When you set the power of the D7 pin to high, this LED turns on. Let's do that now.

Tap `D7` then `digitalWrite` in the popup. Now when you tap the D7 circle the tiny blue LED should turn off or on!

Congratulations, you just blinked an LED over the internet, using your Particle device!

Keep in mind that with Tinker, you can communicate with any of the pins, not just with the D7 LED. You can wire things to the pins to run motors, read sensors, and much more. The real fun part comes when you write your own firmware, of course. We'll go over that in later sections.

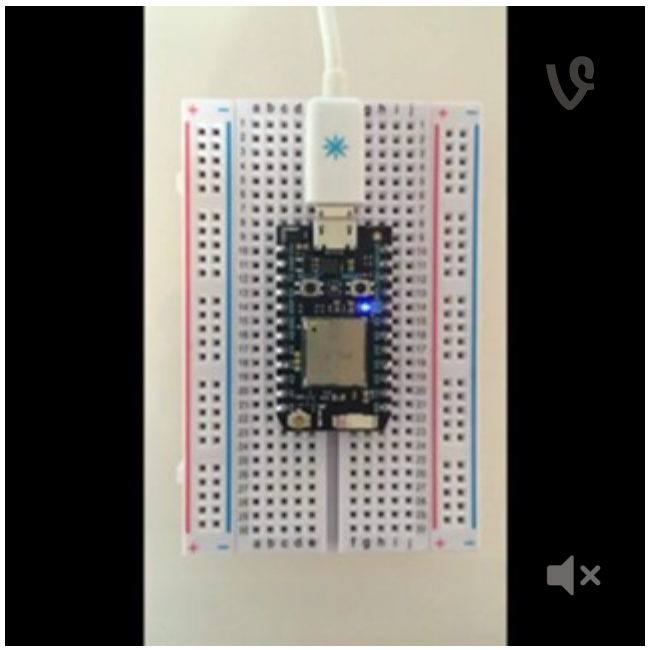
If you don't have your smartphone with you, go ahead and move to the next lesson on connecting over USB.. If you've successfully connected with your smartphone and you'd like to keep playing around with Tinker, skip ahead to learn device modes and then do some Tinker examples.

Otherwise, go to the next section to learn to connect over USB.

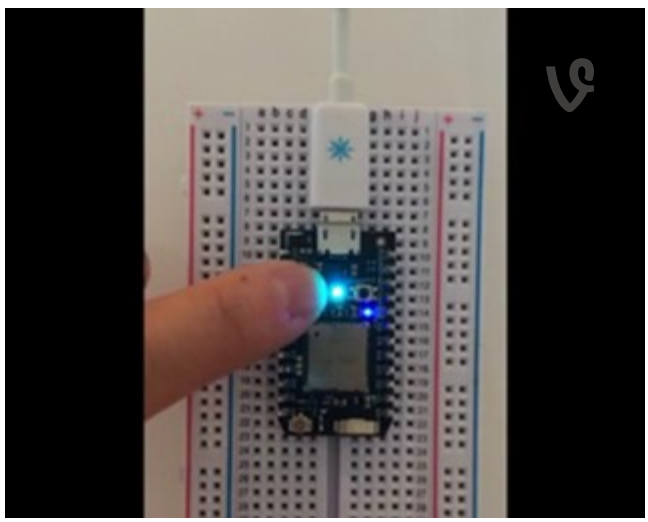
CONNECTING YOUR DEVICE OVER USB

The easiest way to connect your device to Wi-Fi network is using the mobile app as described in the previous lesson. But in case that's not working for you, there are other methods as well.

For all of the following methods, the device must be in Listening Mode, where the RGB LED is



Particle devices boot into listening mode by default, so if your device is brand new, it should go straight into listening mode. If your device is not blinking blue,



Install the Particle CLI

Next we're going to install the Particle CLI on your computer.

Using macOS or Linux

Open Terminal, or preferred terminal program and paste this command to install the Particle CLI:

```
bash <( curl -sL https://particle.io/install-cli )
```

You may need to open a new terminal after the install completes to use the `particle` command.

If you get a message about installing dfu-util for your OS, make sure you have homebrew installed and run the command above again.

Using Windows

Download the Windows CLI Installer and run it to install the Particle CLI, the device drivers and the dependencies that the CLI needs.

You'll need to open the command prompt for this next part. You can also use Powershell or a similar command line tool if that is what you are used to.

To open the command prompt: 1) Mouse over the upper right hand corner of the screen and select "Search" 2) Search for `cmd` in the search box 3) Click on Command Prompt

Now your Command Prompt, is open for use.

Connecting Your Device

Make sure your device is plugged in via USB and in Listening Mode (blinking blue). Open the terminal and type: `particle setup`

Log in with your Particle account and follow the prompts to set up your device.

If you have already claimed your device and you want to connect it to Wi-Fi, type `particle serial wifi` instead of `particle setup`. This will set up your device on the current Wi-Fi.

Wait! What is an SSID? What kind of security does my Wi-Fi have?

- **The SSID** is the name of your network. When you connect on your computer, it is the name that you select when you connect your computer to Wi-Fi.
- **The Security** of your Wi-Fi is often set up by the administrator. Typically this is WPA2 if a password is needed, or unsecured if no password is needed. Contact your network administrator if you can't get this step to work, and find out exactly what kind of Wi-Fi you have.

If your device is not connecting, try troubleshooting here.

More info on the CLI and steps to install it manually are available.

Once you've finished connecting your device, head over to the next section to learn about the different modes for your device.

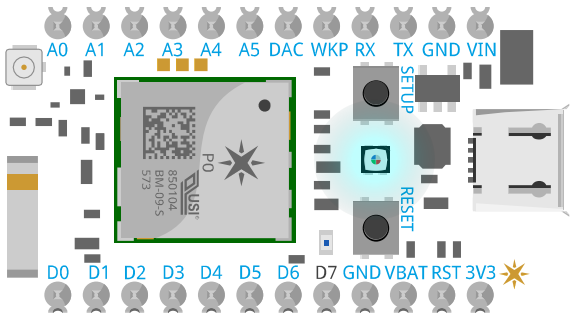
DEVICE MODES

Now that we've gone over connecting your device, we're going to review the different modes for your Photon. We suggest that you work through this section, putting your device in the different listed modes to familiarize yourself with them.

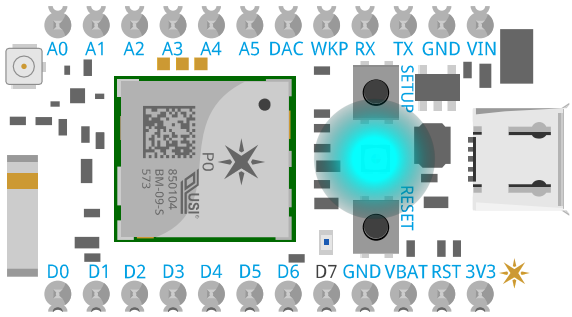
Standard Modes

These modes are the typical behaviors you will see from your Photon on a regular basis. They are the light patterns of a healthy Photon.

Here's the typical pattern of a Photon after power up.

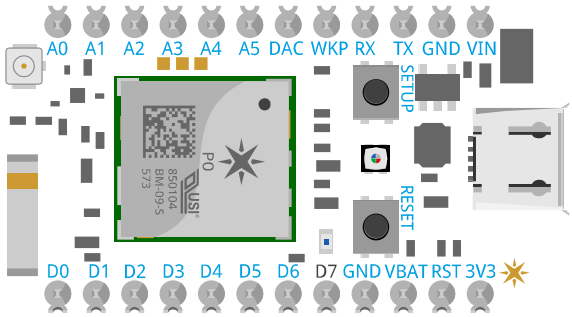


Connected



When it is breathing cyan, your Photon is happily connected to the Internet. When it is in this mode, you can call functions and flash code.

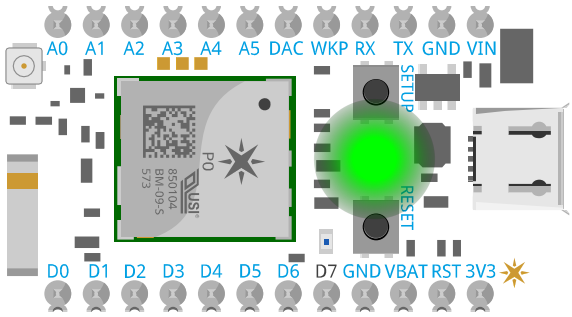
OTA Firmware Update



If your Photon is blinking magenta, it is currently loading an app or updating its firmware. This state is triggered by a firmware update or by flashing code from the Web IDE or Desktop IDE. You might see this mode when you connect your Photon to the cloud for the first time.

Note that, if you enter this mode by holding **SETUP** on boot, blinking magenta indicates that letting go of the **SETUP** button will enter safe mode to connect to the cloud and not run application firmware.

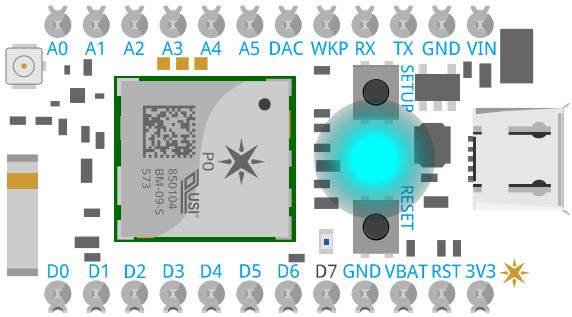
Looking For Internet



If your Photon is blinking green, it is trying to connect to the internet. If you already setup the Wi-Fi connection, give your device a few seconds to connect and start breathing cyan.

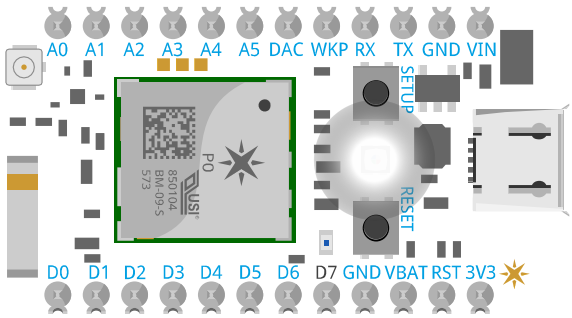
If you haven't yet connected your Photon to Wi-Fi, then set your device to Listening Mode.

Connecting to the Cloud



When the Photon is in the process of connecting to the cloud, it will rapidly blink cyan. You often see this mode when you first connect your Photon to a network, after it has just blinked green.

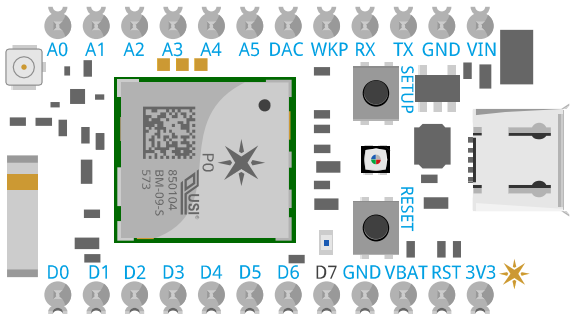
Wi-Fi Off



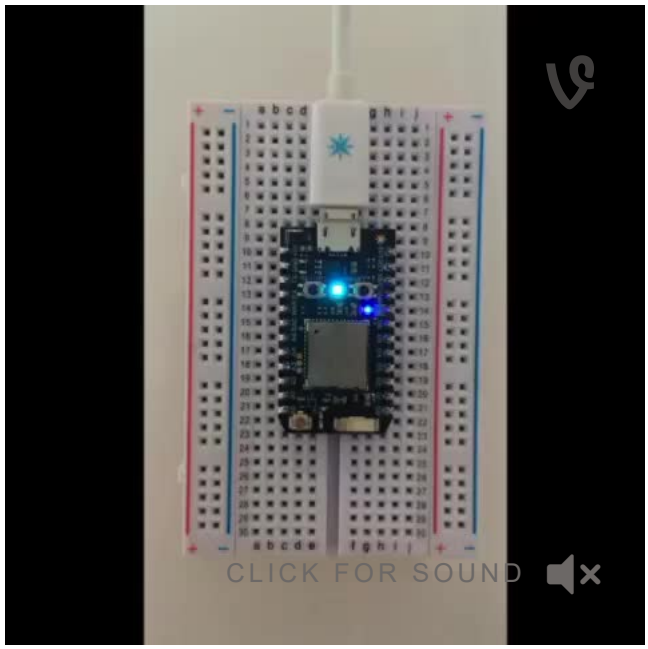
If your Photon is breathing white, the Wi-Fi module is off. You might see this mode if:

- You have set your module to `MANUAL` or `SEMI_AUTOMATIC` in your user firmware
- You have called `WiFi.off()` in your user firmware

Listening Mode



When your Photon is in Listening Mode, it is waiting for your input to connect to Wi-Fi. Your Photon needs to be in Listening Mode in order to begin connecting with the Mobile App or over USB.



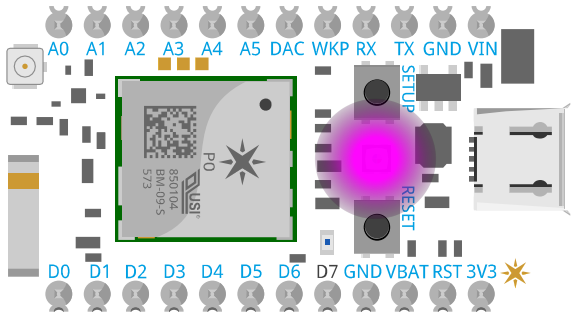
To put your Photon in Listening Mode, hold the `SETUP` button for three seconds, until the RGB LED begins blinking blue.

Wi-Fi Network Reset

To erase the stored Wi-Fi networks on your Photon, hold the SETUP button for about ten seconds, until the RGB LED blinks blue rapidly.

You can also reset the Wi-Fi networks by holding the SETUP button and tapping RESET, then continuing to hold SETUP until the light on the Photon turns white. (This differs from the Core. Doing this action on the Core will result in a factory reset.)

Safe Mode



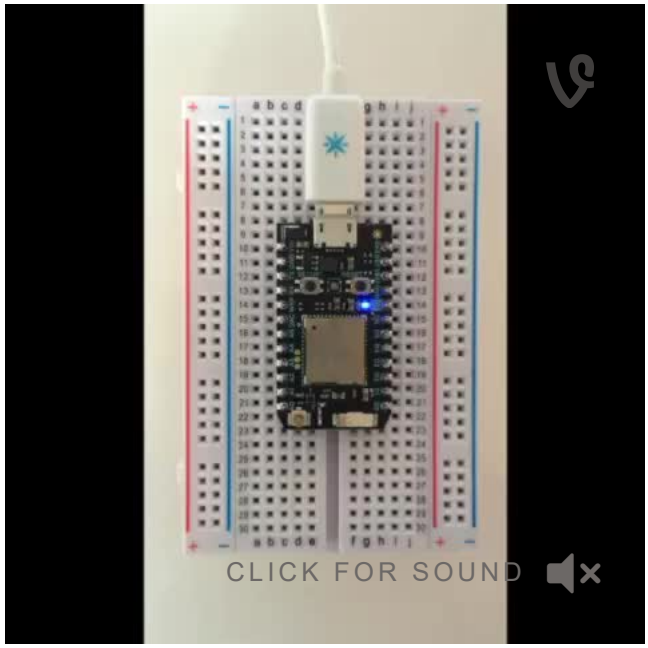
Safe mode connects the Photon to the cloud, but does not run any application firmware. This mode is one of the most useful for development or for troubleshooting. If something goes wrong with the app you loaded onto your device, you can set your device to Safe Mode. This runs the device's system firmware but doesn't execute any application code, which can be useful if the application code contains bugs that stop the device from connecting to the cloud.

The Photon indicates that it is in Safe Mode with the LED breathing magenta.

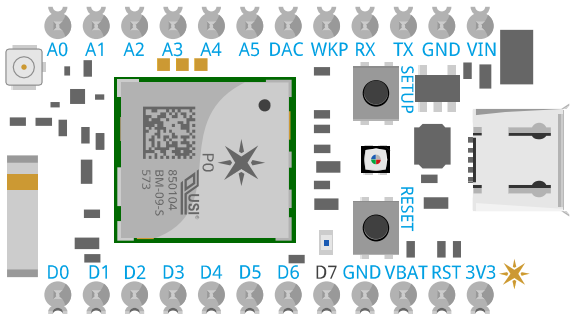
To put your device in Safe Mode:

1. Hold down BOTH buttons
2. Release only the RESET button, while holding down the SETUP button.
3. Wait for the LED to start blinking magenta
4. Release the SETUP button

The device will itself automatically enter safe mode if there is no application code flashed to the device or when the application is not valid.



DFU Mode (Device Firmware Upgrade)



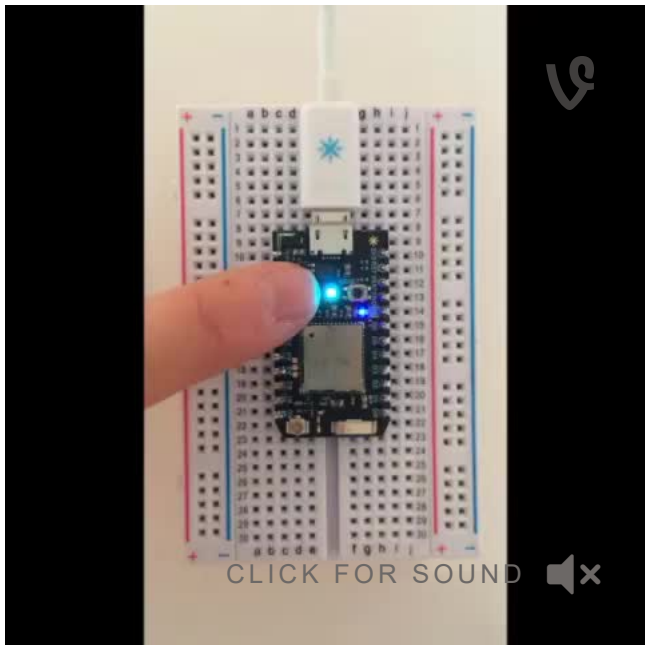
If you wish to program your Photon with a custom firmware via USB, you'll need to use this mode. This mode triggers the on-board bootloader that accepts firmware binary files via the dfu-utility. (Note: Some users reported issues with dfu-util on a USB3.0 ports on Windows. Use a USB2.0 port if the USB3.0 port doesn't work.)

Installation tutorial can be found here.

And a usage guide here.

To enter DFU Mode:

1. Hold down BOTH buttons
2. Release only the **RESET** button, while holding down the **SETUP** button.
3. Wait for the LED to start flashing yellow (it will flash magenta first)
4. Release the **SETUP** button



The Photon now is in the DFU mode.

Firmware Reset

Firmware reset is not available on the Photon, but not to worry! If you are experiencing problems with your application firmware, you can use Safe Mode to recover.

Factory Reset

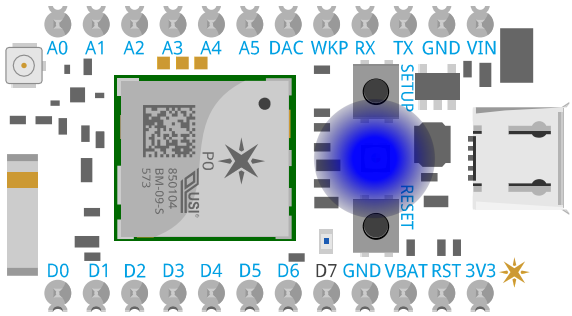
Factory reset is not available on the Photon, but not to worry! If you are experiencing problems with your application firmware, you can use Safe Mode to recover.

You can reset Wi-Fi credentials by performing a Wi-Fi Network Reset.

Troubleshooting Modes

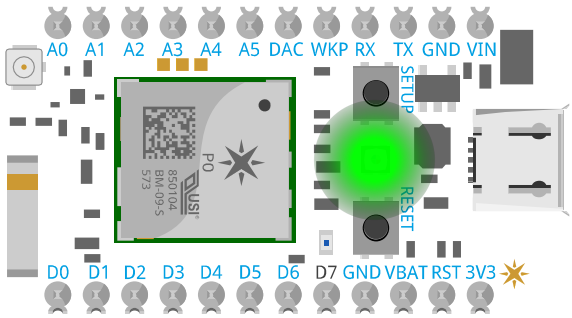
These modes let you know about more atypical issues your Photon might be exhibiting. Use this section to troubleshoot strange colors you might see from your Photon.

Wi-Fi Module Not Connected



If the Wi-Fi module is on but not connected to a network, your Photon will be breathing blue. Note that this will be dark blue and not cyan.

Cloud Not Connected



When the Photon is connected to a Wi-Fi network but not to the cloud, it will be breathing green.

This can be caused by the currently running application firmware which may interfere with the cloud maintenance tasks which are usually executed between iterations of `loop()` or via an explicit call of `Particle.process()`. That commonly happens when the code blocks for more than 10 seconds. In addition to regularly allowing for cloud maintenance (via dropping out of `loop()` and/or calling `Particle.process()`) you can take manual control of the connection, choose a better suited `SYSTEM_MODE` and/or opt for `SYSTEM_THREAD(ENABLED)`. To correct the "offending" firmware you may need to flash new firmware either via USB or Safe Mode.

Bad Public Key

When the server public key is bad, the Photon will blink alternately cyan and red.

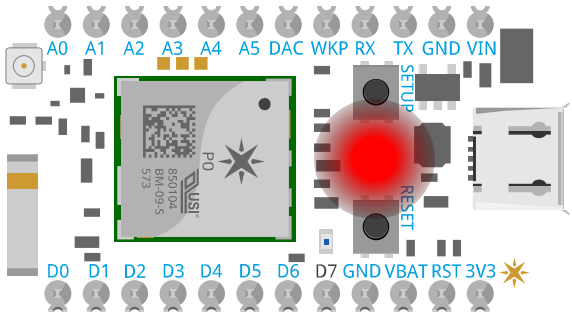
Red Blink Basic Errors

Blinking red indicates various errors.

- 2 red blinks: Could not reach the internet.
- 3 red blinks: Connected to the internet, but could not reach the Particle Cloud.

- Blinking "orange": This sometimes is seen as yellow or red and indicates bad server keys. To fix this issue, use the Particle CLI to restore the server keys using `particle keys server` in your terminal window, while having the device in DFU mode.

Red Flash SOS



Is your Photon blinking red? Oh no!

A pattern of more than 10 red blinks is caused by the firmware crashing. The pattern is 3 short blinks, 3 long blinks, 3 short blinks (SOS pattern), followed by a number of blinks that depend on the error, then the SOS pattern again.

Enter safe mode, tweak your firmware and try again!

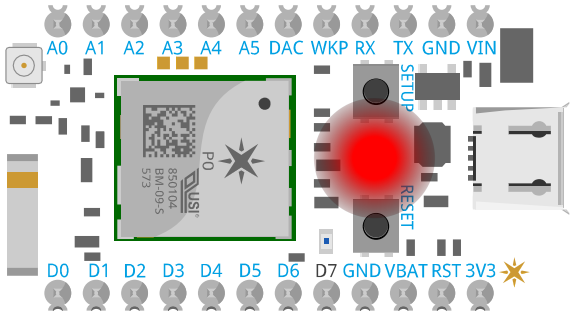
There are a number of other red blink codes that may be expressed after the SOS blinks:

1. Hard fault
2. Non-maskable interrupt fault
3. Memory Manager fault
4. Bus fault
5. Usage fault
6. Invalid length
7. Exit
8. Out of heap memory
9. SPI over-run

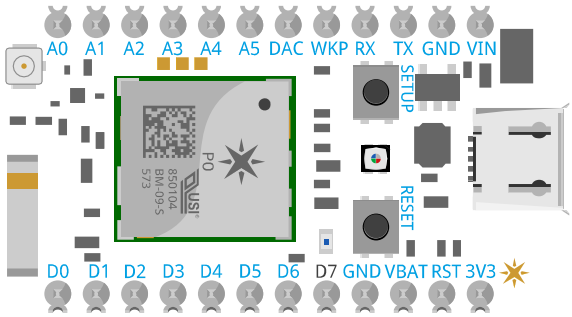
10. Assertion failure
11. Invalid case
12. Pure virtual call
13. Stack overflow

The two most common ones are:

Hard Fault (1 blink between 2 SOS patterns)



Out of heap memory (8 blinks between 2 SOS patterns)



If your Photon crashes repeatedly with an SOS code, first try recovering with Safe Mode and flashing Tinker with the CLI to see if it was something recently added in your user application.

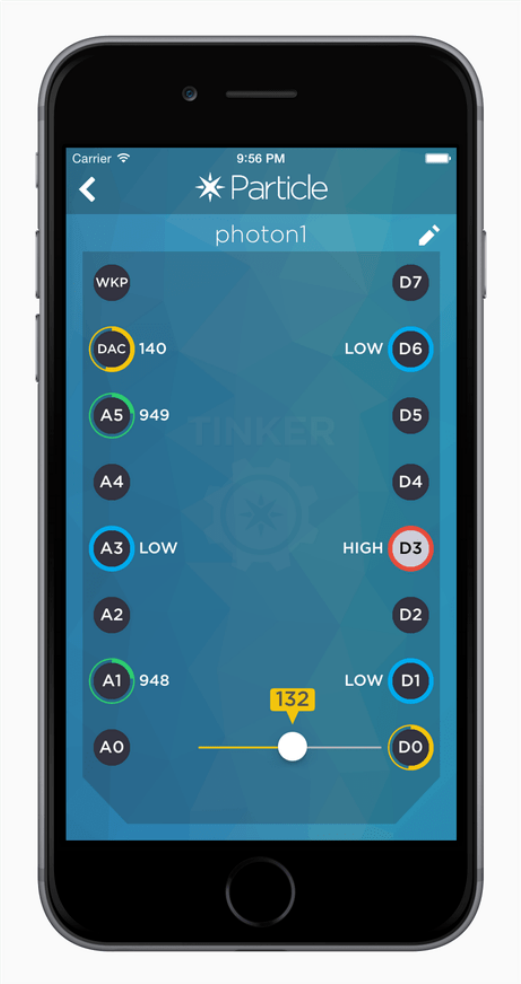
```
particle flash <mydevice> tinker
```

If it's not possible to enter Safe Mode, your system firmware may be corrupted. Use the Device Doctor feature of the CLI to put your Photon into a healthy state.

```
particle device doctor
```

Don't forget that the community forum is always there to help.

TINKERING WITH "TINKER"



The Tinker section of the Particle mobile app makes it very easy to start playing with your Particle device without writing any code. It's great for early development, learning, and prototyping. We'll learn to use it in the next few examples.

To get started, you'll need the following things:

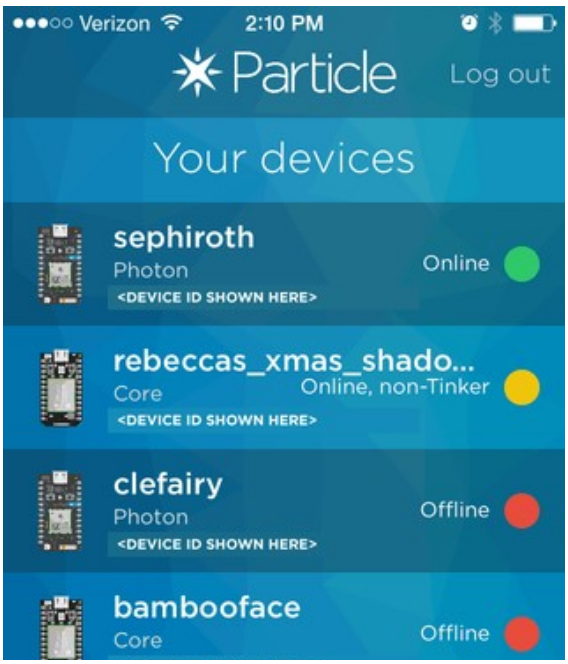
Materials

- **Hardware**
    - Your Particle device, brand new and out of the box!

    - 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)
    - Your iPhone, Windows, or Android smartphone
    - (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 other resistance-based sensor, such as a Thermistor or Force-Sensitive Resistor (Photoresistor included with Photon Kit and Maker Kit)
- **Software**
    - Particle Mobile App - iPhone | Android (requires Android 4.0.3 or higher) | Windows
- **Experience**
    - Connecting your Device with your smartphone or over USB

Step One: Select Your Device

You've already connected your device, either with your smartphone or over USB. This means that when you open the Particle App, you'll see a screen like this:

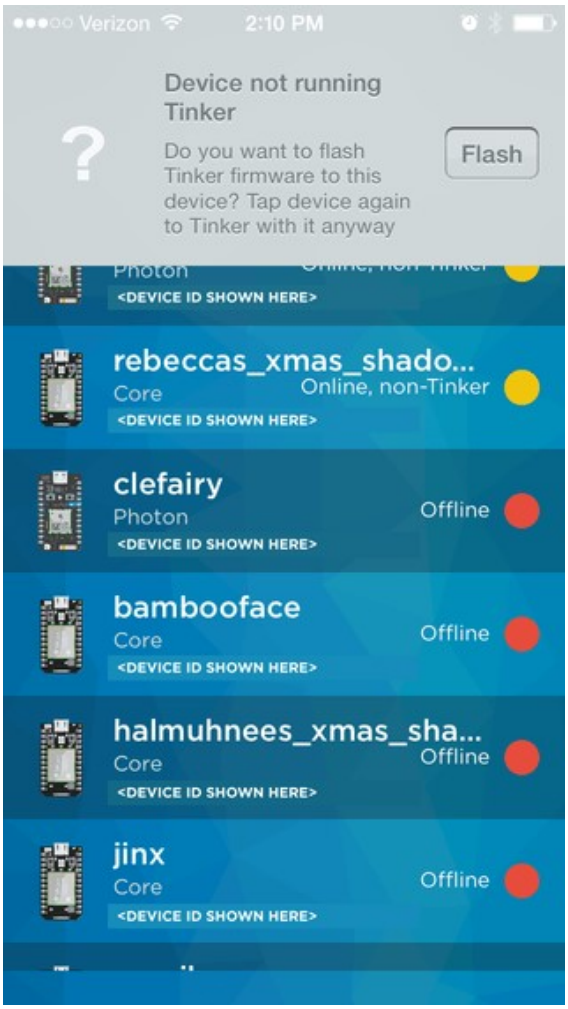


This person has a lot of devices, so the list is pretty crowded. As you can see, you can give your devices unique names to help you remember which is which.

Select the device you would like to Tinker around with by tapping on it.

> **Troubleshooting Note: I have a device online, but it doesn't have Tinker firmware on it.**
>
> *If there is a little yellow circle next to the device you want to use, it is online but does not have Tinker firmware running on it. (You probably replaced it with your user firmware for a different project or example.)*

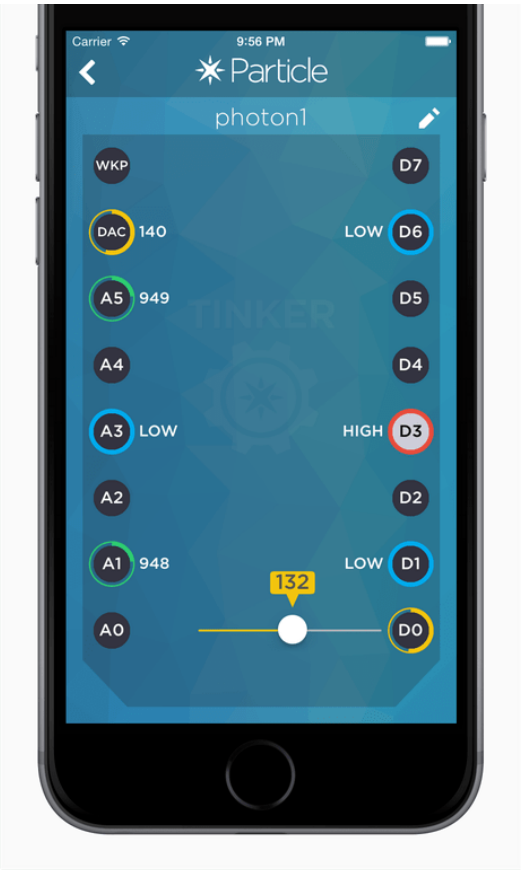
When you tap on one of these devices, it will give you the option to reflash the Tinker firmware.



To do the following examples, you'll want to reflash the Tinker firmware. Go ahead and do this.

Step Two: Explore the Tinker App

The app consists of 16 pins in vertical rows - 8 analog pins on the left, 8 digital pins on the right. These pins represent the 16 GPIO (General Purpose Input and Output) pins on your device.



To begin, tap any of the pins. A menu will pop up showing the functions that pin has available, tap a function name to select the pin functionality. You can reset the pin function by long-pressing it. Each pin can have up to :

- **digitalWrite**: Sets the pin to HIGH or LOW, which either connects it to 3.3V (the maximum voltage of the system) or to GND (ground). Pin D7 is connected to an on-board LED; if you set pin D7 to HIGH, the LED will turn on, and if you set it to LOW, it will turn off.
- **analogWrite**: Sets the pin to a value between 0 and 255, where 0 is the same as LOW and 255 is the same as HIGH. This is sort of like sending a voltage between 0 and 3.3V, but since this is a digital system, it uses a mechanism called Pulse Width Modulation, or PWM. You could use *analogWrite* to dim an LED, as an example. The Photon has two DACs (Digital to Analog converters) onboard connected to pin DAC (A6) and A3, when you select **analogWrite** on those two pins you can set a value between 0 to 4095 (12bit resolution) and continous analog voltage will be applied at the pin output (not PWM), you can use it for controlling electronic devices that require precision analog voltage setting. Those two pins will be

marked in orange color when activated in analogWrite mode (instead of yellow color for the rest of the PWM-enabled pins).

- **digitalRead**: This will read the digital value of a pin, which can be read as either HIGH or LOW. If you were to connect the pin to 3.3V, it would read HIGH; if you connect it to GND, it would read LOW. Anywhere in between, it'll probably read whichever one it's closer to, but it gets dicey in the middle.
- **analogRead**: This will read the analog value of a pin, which is a value from 0 to 4095, where 0 is LOW (GND) and 4095 is HIGH (3.3V). All of the analog pins (A0 to A7) can handle this. *analogRead* is great for reading data from sensors.

In other words, Tinker lets you control and read the voltage of the pins on your device. You can choose to send a lot of voltage to an LED to light it up, or read the voltage at a sensor.

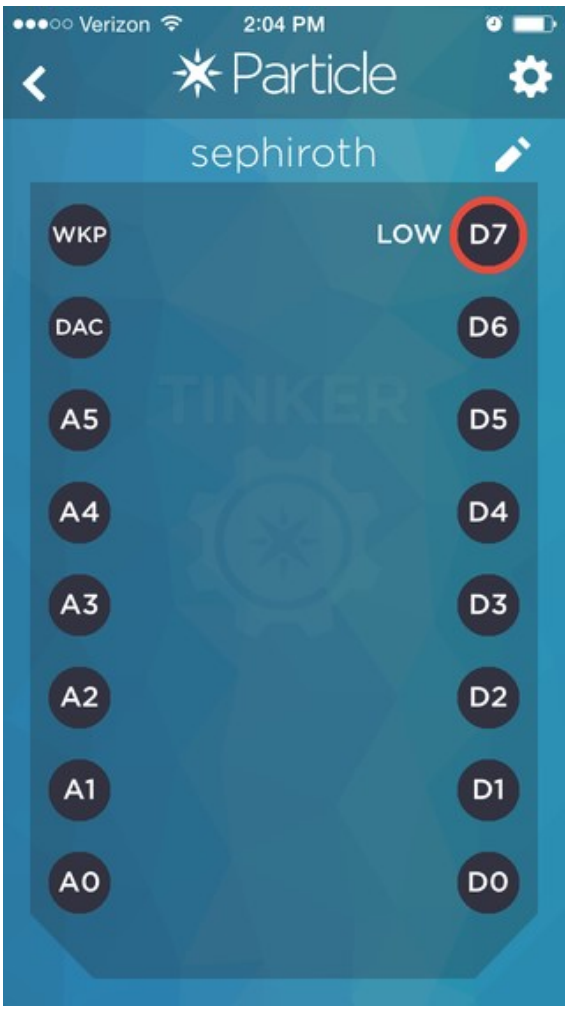
Now for some examples!

Step Three: digitalWrite

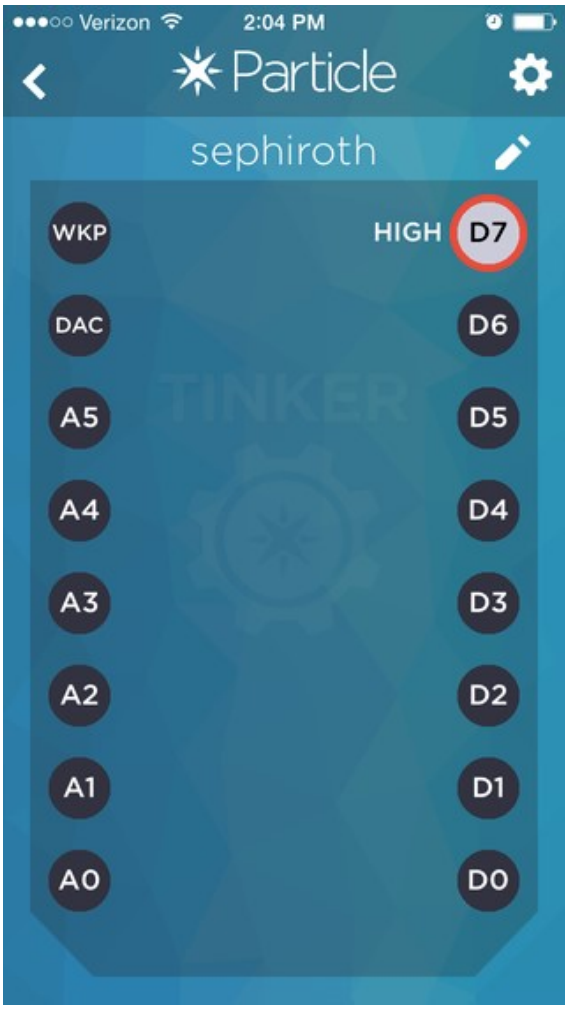
The simplest thing we can do with Tinker is to turn the D7 LED on and off.

To turn on the LED, tap the D7 pin on the mobile app.

Then tap `digitalWrite` to let it know that you want to send high or low voltage to that pin.



You'll see a screen like the one above. Try tapping the D7 pin on the screen again.



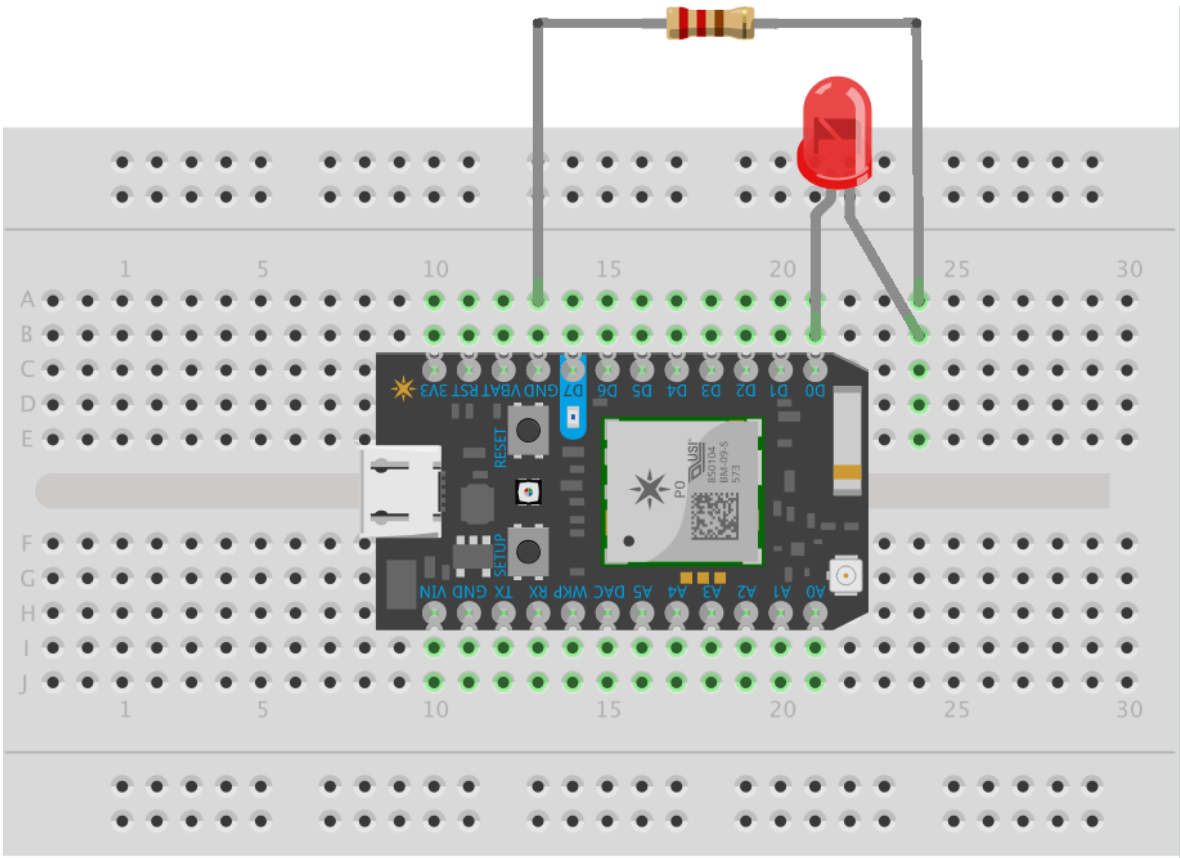
It will change its status to `HIGH` and your device's D7 LED will turn on. Tapping it again will change the status to `LOW` and the LED will turn off.

`digitalWrite` only has two options: `HIGH` and `LOW`. When we speak to our pins digitally, we can only send the maximum voltage or no voltage. That's great for when you only need two settings-- like if you had a light switch that could only go on and off, or locks that could only be open or closed. For everything in between, we use `analogWrite`.

Step Four: analogWrite

In this example, we'll plug an LED into D0 and change its brightness with analogWrite. (D0 is a

Wire up your LED with one of your resistors as shown below. Connect the longer (anode) leg of the LED to pin D0 and the shorter (cathode) leg to GND via a resistor.



Then, pull up your mobile app and select D0 this time. Instead of `digitalWrite`, select `analogWrite`.

You should now be able to set the D0 pin to any number between 0 and 255.

It basically divides the maximum voltage by 255 and allows us to set the slider to any fraction of the voltage between minimum and maximum. Pretty cool, right?

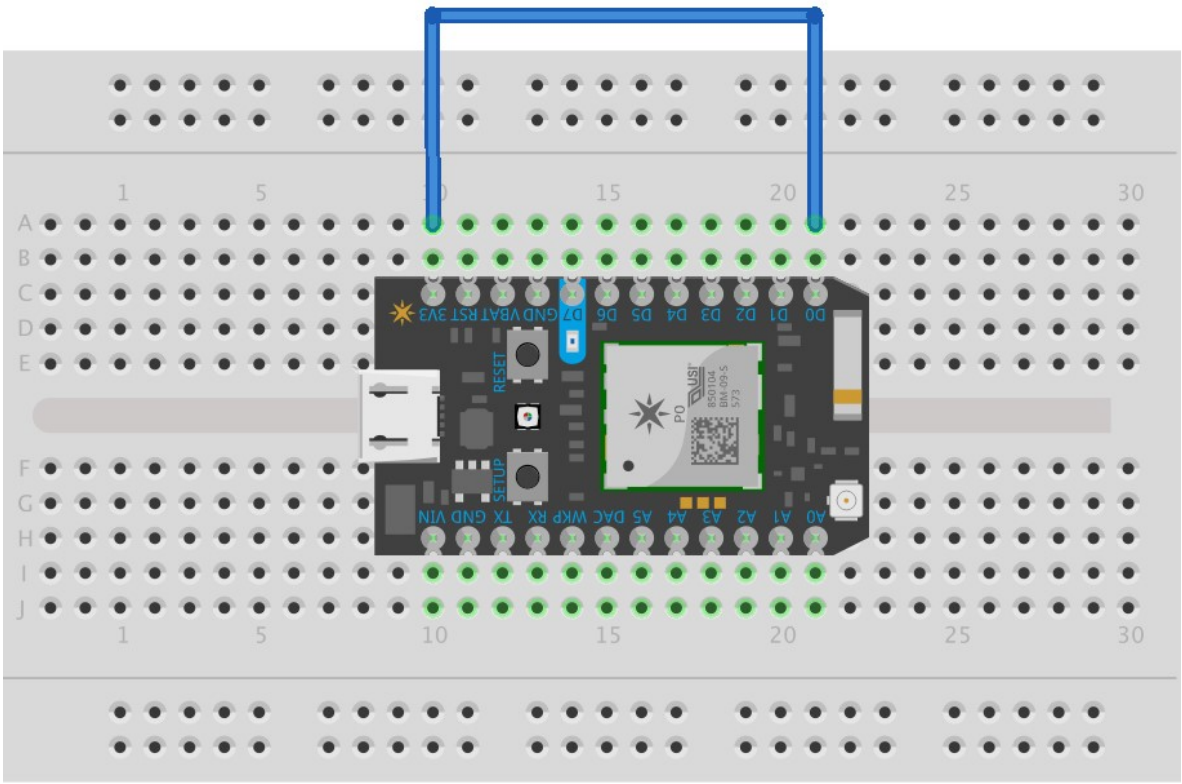
By sliding and releasing the slider, you should be able to see the LED dim and glow at different levels.

If we wanted to, we could also switch modes and `digitalWrite` this LED to turn it on or off. To change the function of the pin, simply tap and hold on the pin, and the function select menu will come back up.

Step Five: digitalRead

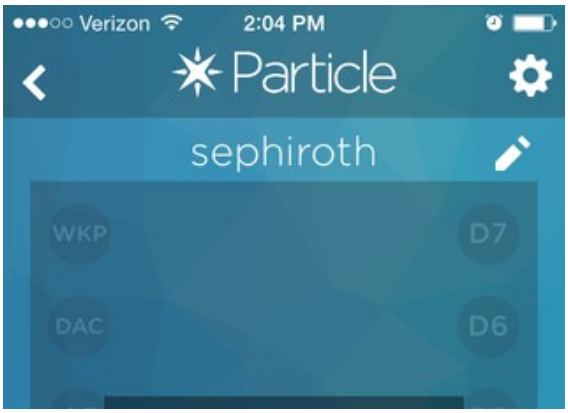
We can also use Tinker to check to see if a pin is on or off. `digitalRead` is great for checking things that only have two states-- switches and buttons for example.

In this case, we're going to do the simplest thing possible and simply use one wire to change the voltage of the D0 pin. Plug a wire or resistor to connect `D0` and `3V3`, as shown below.



As you can see, one side of the wire is plugged into `3V3` and the other is plugged into `D0`. `3V3` sends the maximum voltage out. We've plugged it into `D0`, so `D0` is receiving the maximum voltage.

Let's read that. Go into your mobile app. Press and hold `D0` to reset it. After it goes blank, tap it again and select `digitalRead`.



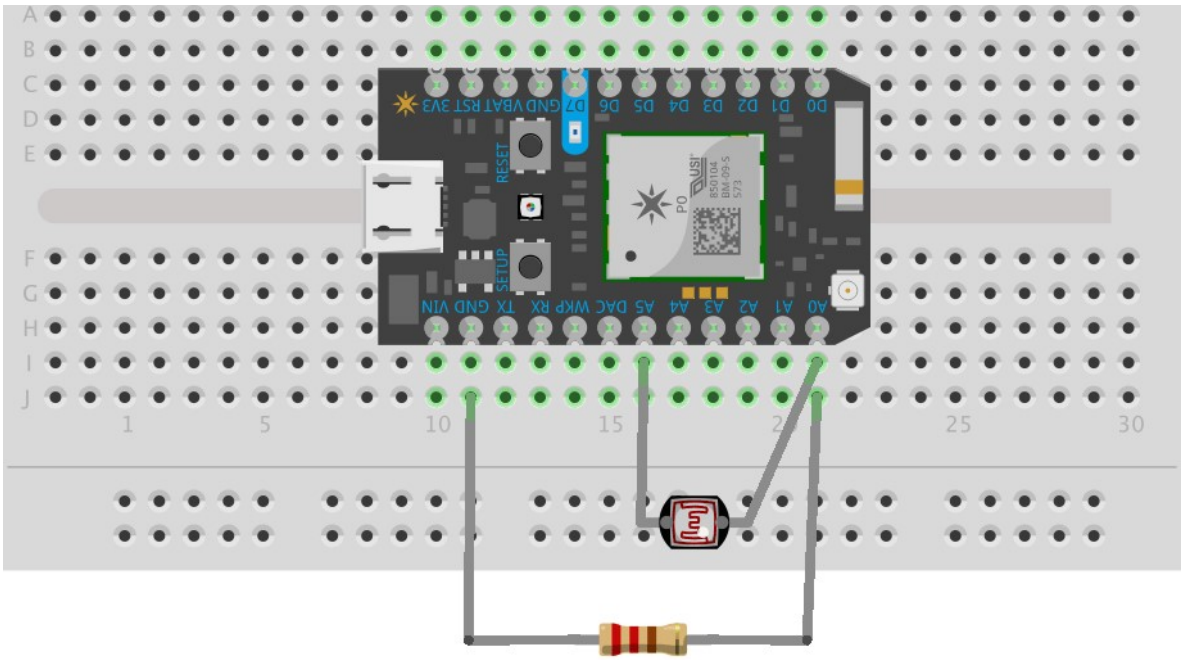
If your wire is plugged in, you'll see the word `HIGH` next to the `D0` pin. Now unplug the wire and tap the `D0` pin on the mobile app once more.

Step Six: analogRead

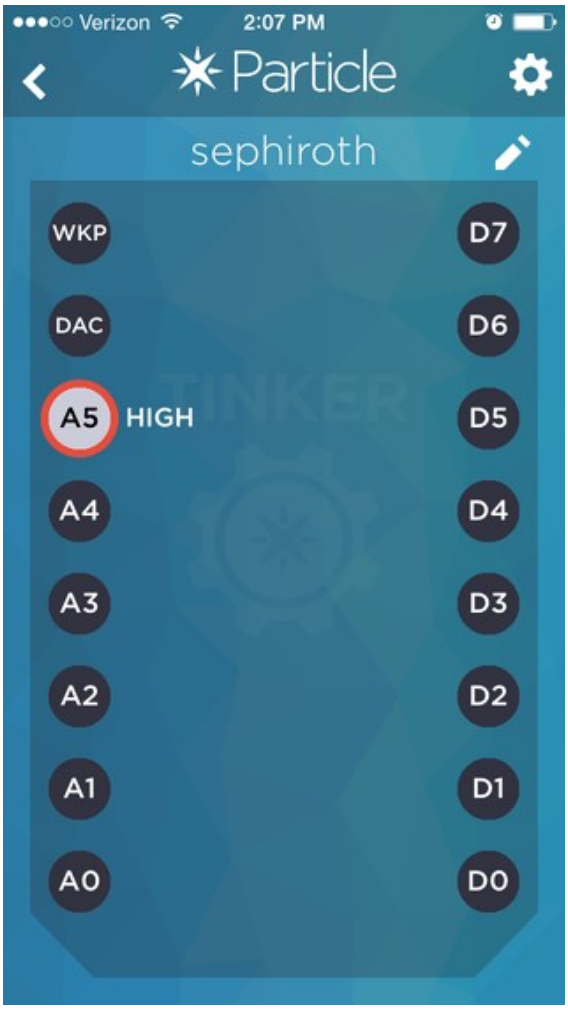
If we want to read a sensor, like a temperature or light sensor, we will need our device to give us more details than than just "It's on!" or "It's off!" When you want to read a value between `LOW` and `HIGH`, use `analogRead`.

Plug in a sensor. In this example, we'll use a photoresistor.

Wire it up as pictured below. You can use any resistor for this; a larger resistor (like 10K Ohms) will give you a wider range of values whereas a smaller resistor (like 330 Ohms) will give you lower range of values.

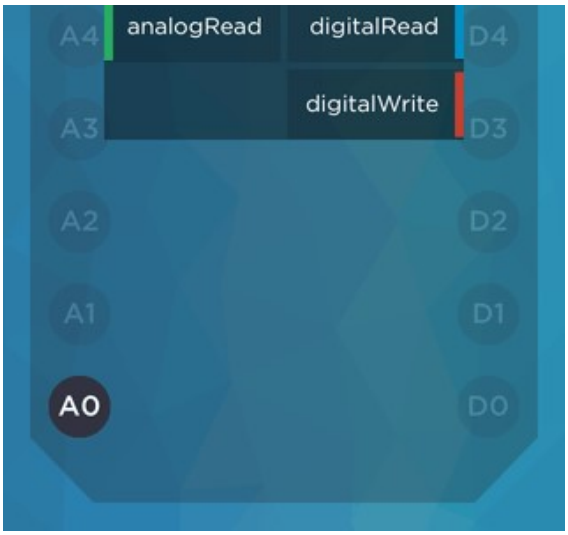


Tap the A5 pin and set it to `digitalWrite` and `HIGH`.

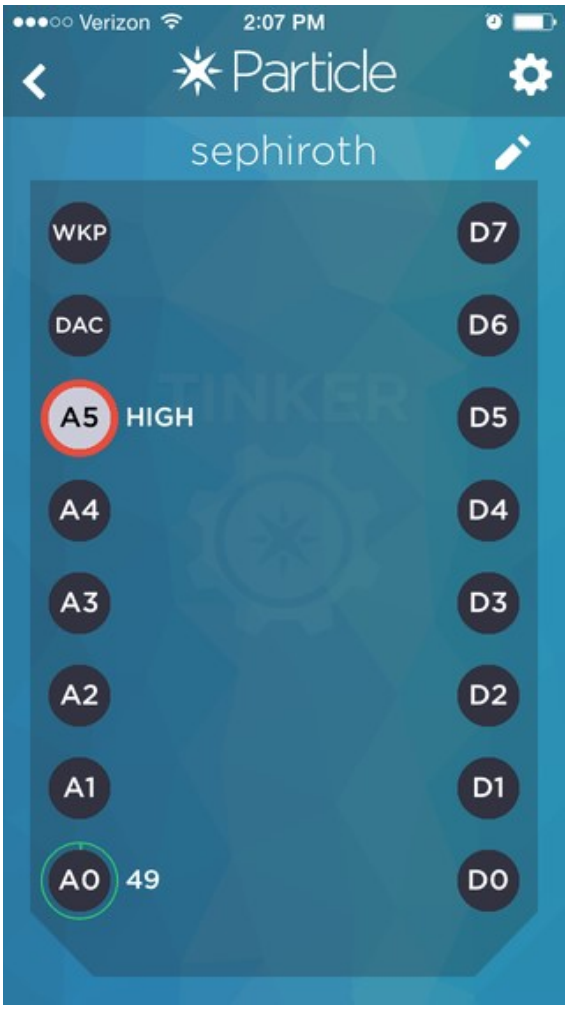


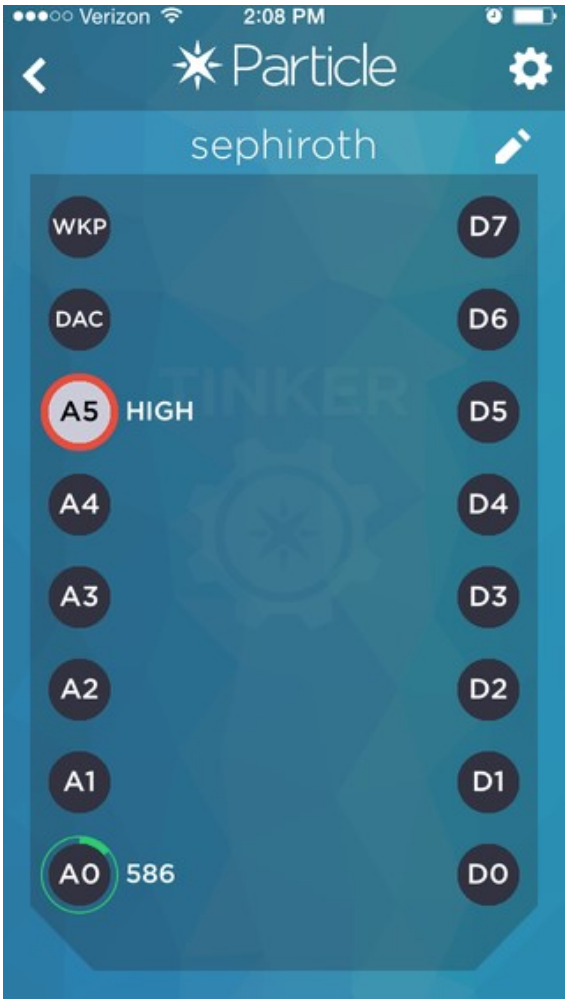
This essentially gives us a consistent power source from A5 that will go to our photoresistor. (We are doing this because sometimes an on-board power source like 3V3 has small fluctuations in power that could affect our photoresistor readings.)

Now tap A0 and set it to `analogRead`.



Hold your breadboard with the photoresistor on it up to a light source and tap A0 again to get the reading of the photoresistor. Now cover the photoresistor and tap A0 again. See the difference?



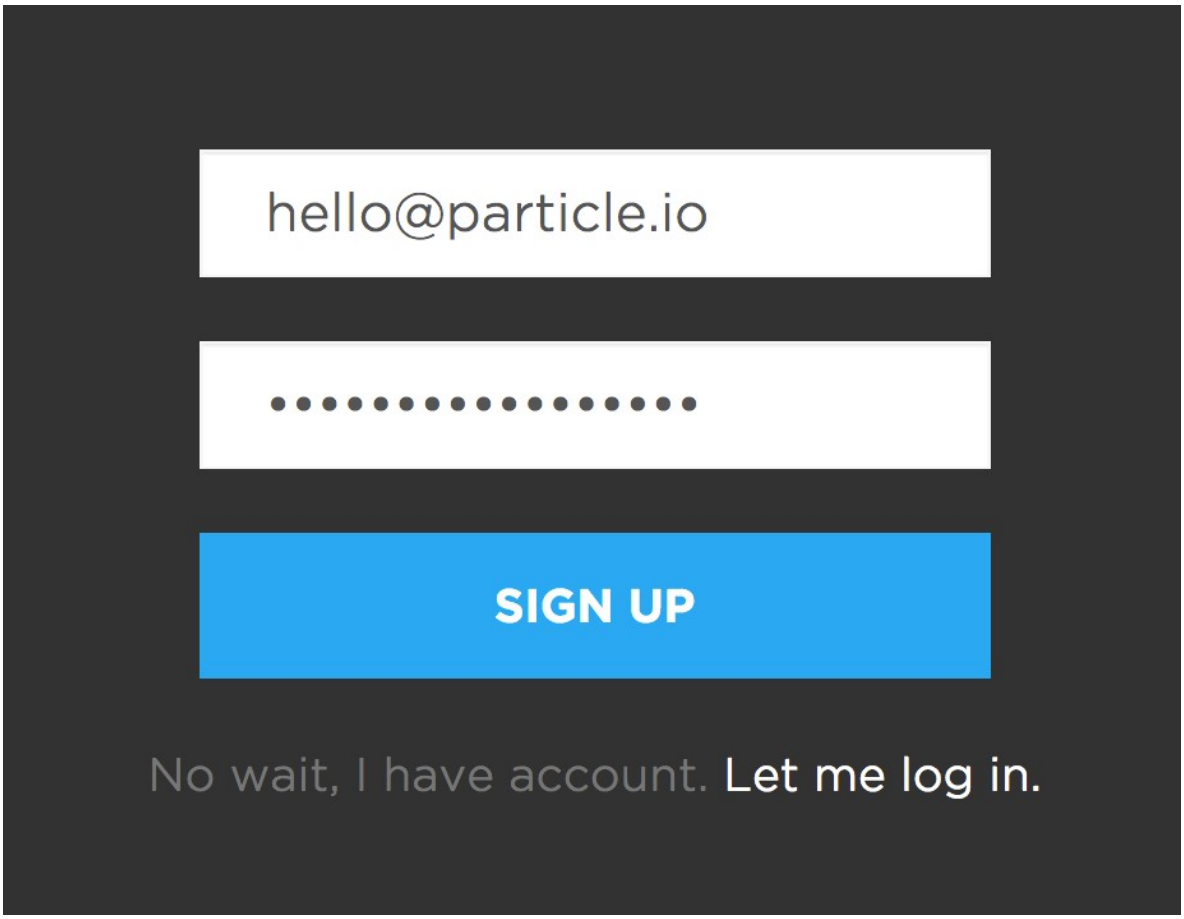


You can try testing different kinds of light, or you can even swap out your photoresistor for another kind of fluctuating resistor like a thermistor or a force sensitive resistor.

When you're ready, let's move on to putting your own firmware on your Particle device using the web IDE.

FLASH APPS WITH PARTICLE BUILD

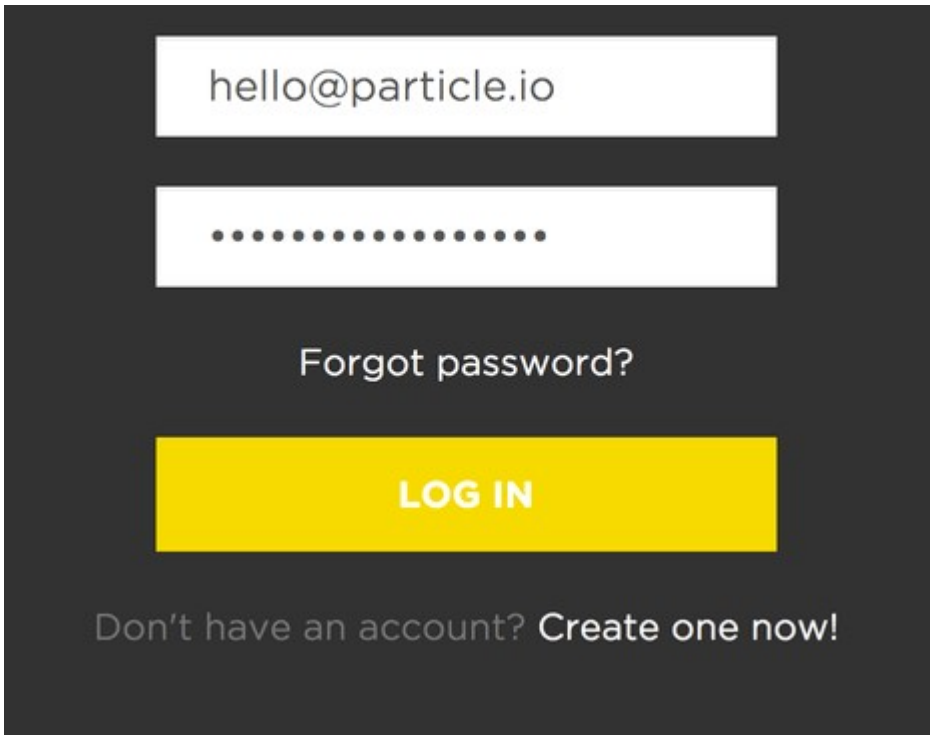
Logging In



When you're ready to reprogram your device, head over to our IDE:

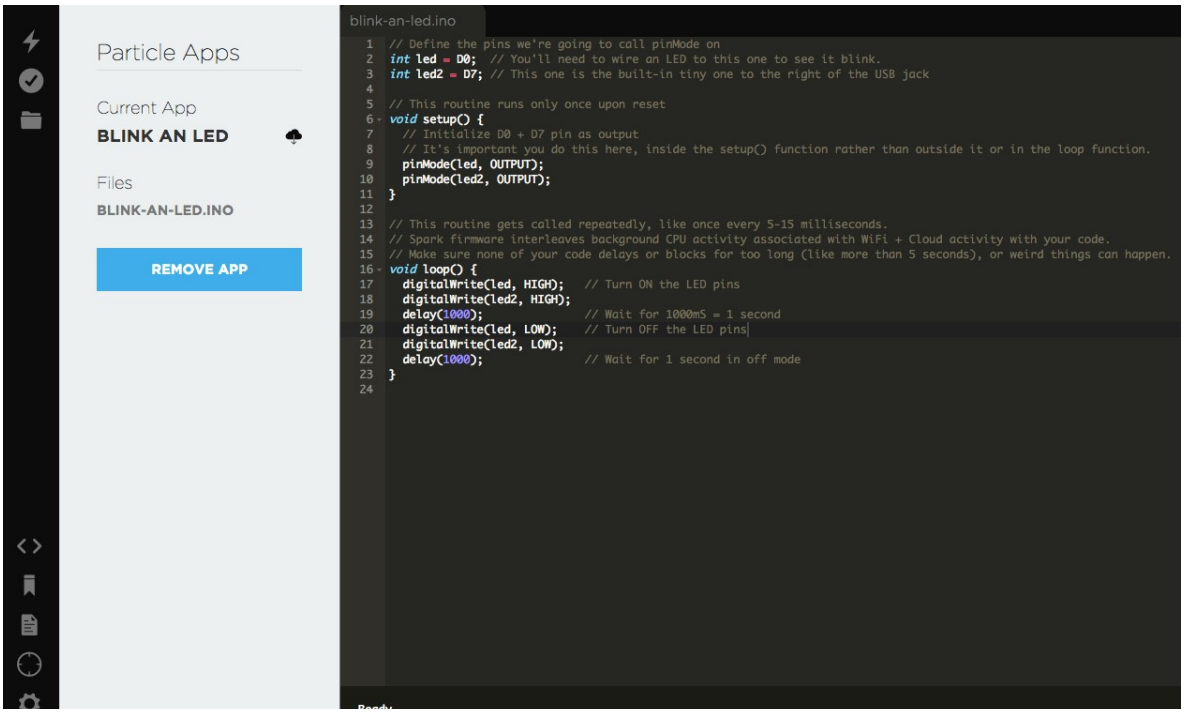
Particle Build >

Creating an account is a simple one-step process. When presented with the login screen, click the "create account" text and fill out the form including your email address (careful!) and desired account password. That's it!



If you haven't logged into Particle Build before, click the "create account" text beneath the Log In button, and you'll be presented with a signup for existing users.

Web IDE



Particle Build is an Integrated Development Environment, or IDE; that means that you can do software development in an easy-to-use application, which just so happens to run in your web browser.

Particle Build starts with the navigation bar on the left. On the top, there are three buttons, which serve important functions:

- **Flash**: Flashes the current code to the device. This initiates an *over-the-air firmware update* and loads the new software onto your device.
- **Verify**: This compiles your code without actually flashing it to the device; if there are any errors in your code, they will be shown in the debug console on the bottom of the screen.
- **Save**: Saves any changes you've made to your code.

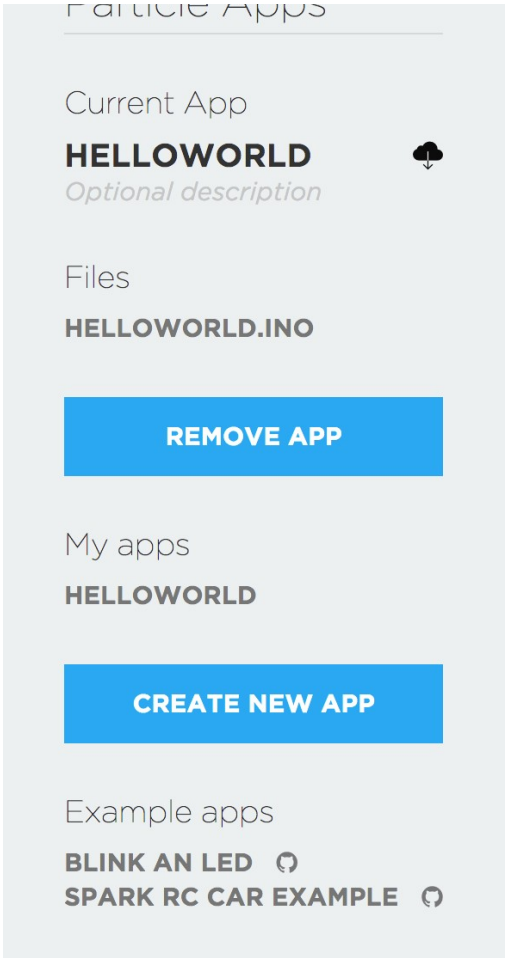
At the bottom, there are four more buttons to navigate through the IDE:

- **Code**: Shows a list of your firmware applications and lets you select which one to edit/flash.
- **Library**: Explore libraries submitted by other users, and develop your own.
- **Docs**: Brings you to the documentation for Particle.
- **Devices**: Shows a list of your devices, so you can choose which to flash, and get more information on each device.
- **Settings**: Change your password, log out, or get your access token for API calls.

Keyboard Shortcuts

Missing your keyboard shortcuts? This cheatsheet will help.

Particle Apps and Libraries



The heart of Particle Build is the "Particle Apps" section, which displays the name of the current app in your editor, as well as a list of your other applications and community-supported example apps.

The application you've got open in the editor is displayed under the "Current App" header. You'll notice that this empty application has only one file, but firmware with associated libraries/multiple files are fully supported.

From this pane, you've got a lot of buttons and actions available to you that can help you grow and manage your library of applications:

- **Create**: You can create a new application by clicking the "Create New App" button. Give it a descriptive name and press enter! Your app is now saved to your account and ready for editing.
- **Delete**: Click the "Remove App" button to remove it forever from your Particle library.

- **Rename**: You can rename your Particle App by simply double-clicking on the title of your app under the "Current App" header. You can modify the "Optional description" field in the same way.
- **My Apps**: Tired of working on your current project? Select the name of another app under the "My apps" header to open it in a tab of the Particle Build editor.
- **Files**: This header lists all known files associated with the open application. Click on a supporting file in your application to open it as an active tab in the editor.
- **Examples**: The "Example apps" header lists a continuously growing number of community-supported example apps. Use these apps as references for developing your own, or fork them outright to extend their functionality.

Flashing Your First App

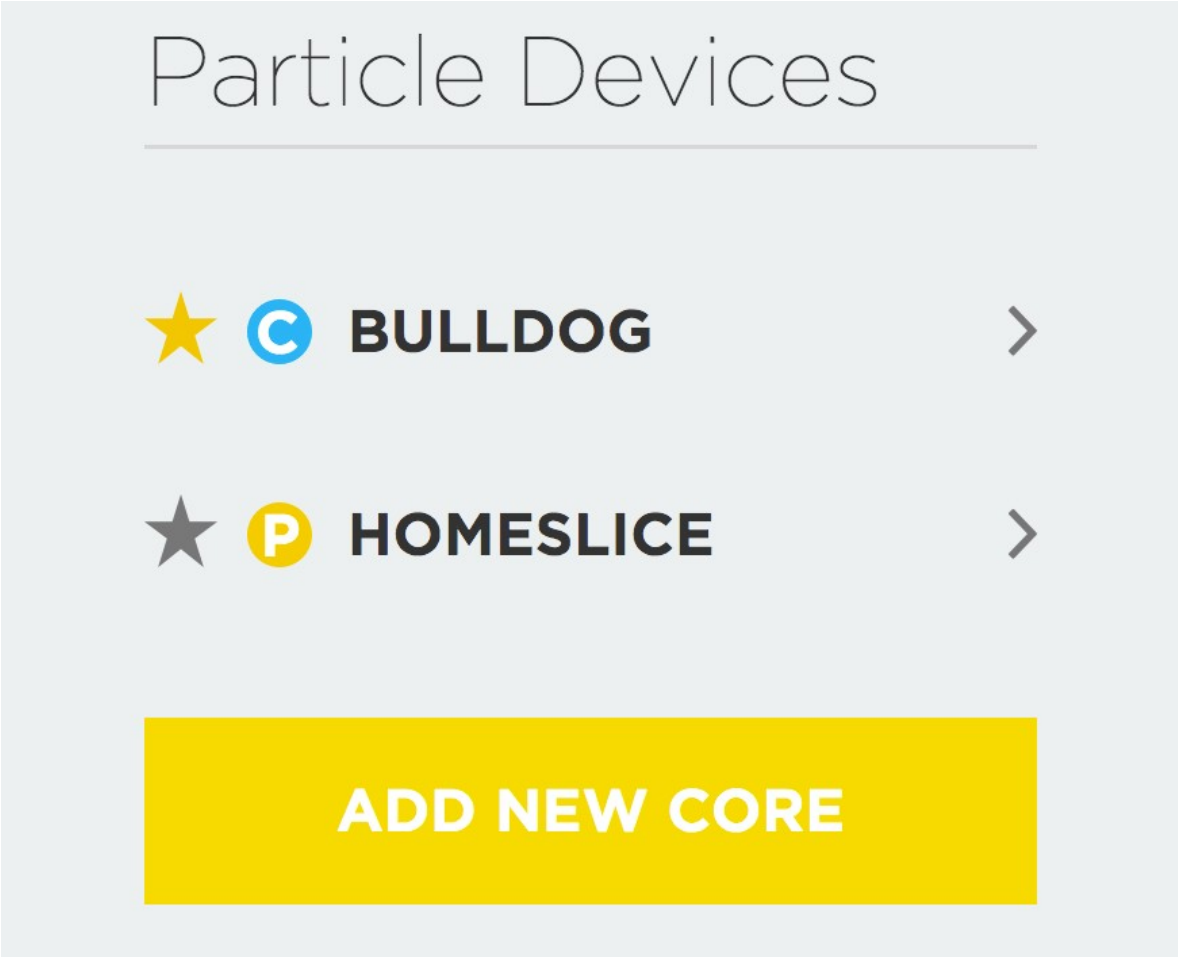
The best way to get started with the IDE is to start writing code:

- **Connect**: Make sure your device is powered and "breathing" Cyan, which indicates that it's connected to the Particle Cloud and ready to be updated.
- **Get Code**: Try clicking on the "Blink an LED" example under the "Example apps" header. The Particle Build editor should display the code for the example application in an active tab. Alternatively, you can copy and paste this snippet of code into a new application in the Build IDE.

```
//D7 LED Flash Example
int LED = D7;

void setup() {
    pinMode(LED, OUTPUT);
}

void loop() {
    digitalWrite(LED, HIGH);
    delay(1000);
    digitalWrite(LED, LOW);
    delay(1000);
}
```

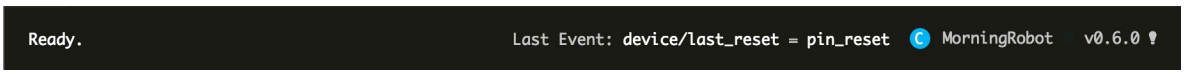


- **Select Your Device**: If you have more than one device you have to make sure that you've selected which of your devices to flash code to. Click on the "Devices" icon at the bottom left side of the navigation pane, then when you hover over device name the star will appear on the left. Click on it to set the device you'd like to update (it won't be visible if you have only one device). Once you've selected a device, the star associated with it will turn yellow. (If you only have one device, there is no need to select it, you can continue on to the next step).

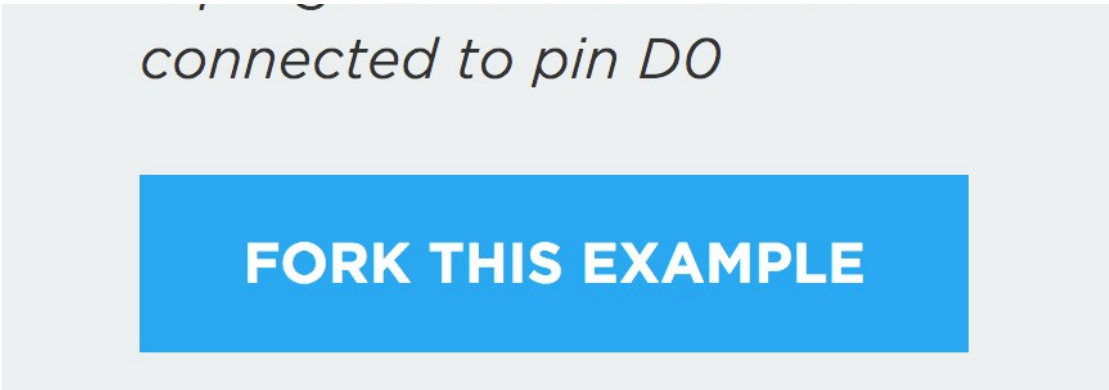
- **Device details**: To identify a device in the "Devices" list, click the chevron then click Signal to make the device LED shout rainbows. Click Signal again to stop. You can also use the firmware selector to opt-in to the latest cutting edge features!

NOTE: Devices are grouped by their platform. You can see the platform icon (circle with an letter) on the left of its name.

- **Status bar**: You can see more information about your currently selected device in the bottom right corner of the Web IDE. It contains the following: Last Event Name, Last Event Data, Device Type, Device Name, Device Status, Device Version. Clicking on the lightbulb will signal the device.



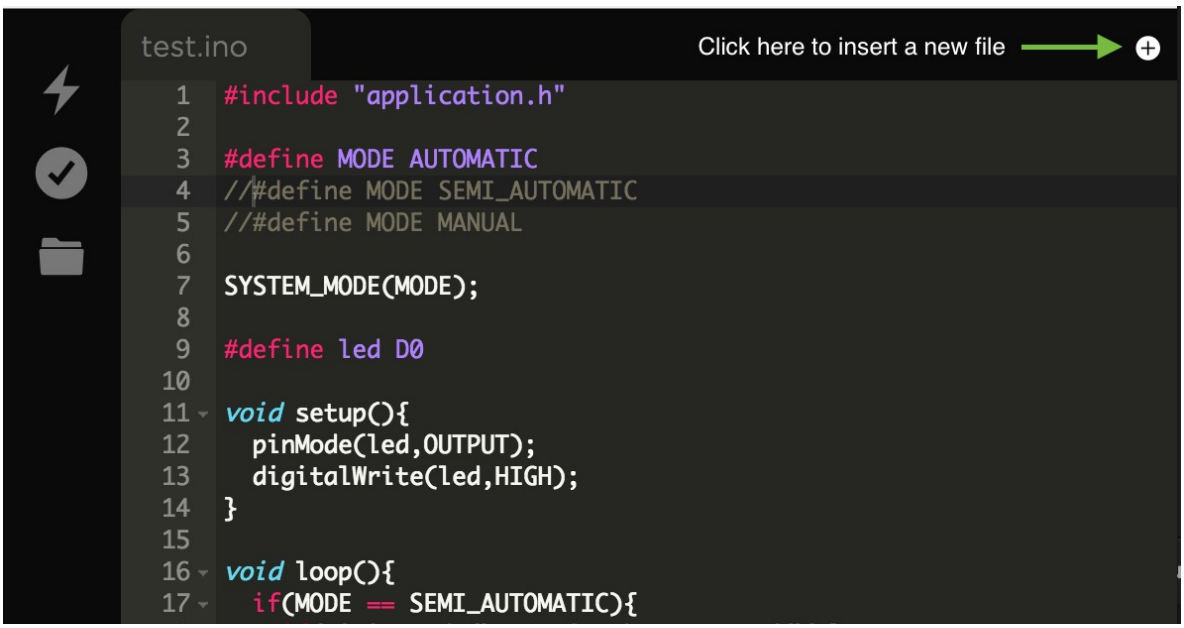
- **Flash**: Click the "Flash" button, and your code will be sent wirelessly to your device. If the flash was successful, the LED on your device will begin flashing magenta.



- **Fork**: Wish the timing of that LED flash was a little bit faster? Try clicking on the "Fork This Example" button after selecting the "Blink An LED" example application. You've now got a personal copy of that application that you can modify, save, and flash to all of your devices.
- **Edit**: Try changing the values in the delay() function from 1000 to 250, which changes the timing interval from 1000 milliseconds to only 250 milliseconds. Click the Verify button, then the Flash button. Is your device's LED blinking faster? Well done :)

Adding files to your app

As your code base grows, you will naturally create libraries to better manage your firmware development. To add a file to your app, simply hit the "+" button located at the top right hand corner.



```
18    if(WiFi.ready() && !Spark.connected()){
19      Spark.connect();
20    //  print_info();
```

This will create two new tabs, one with `.h` and one with `.cpp` extension. You can read more about why we need both in this C++ tutorial.

Sharing your app

When you want to share the app you created with someone, you can create a "snapshot" of it by clicking **Share this revision**: button.

Once you make the revision public anyone with the link can see a read-only version of your app.

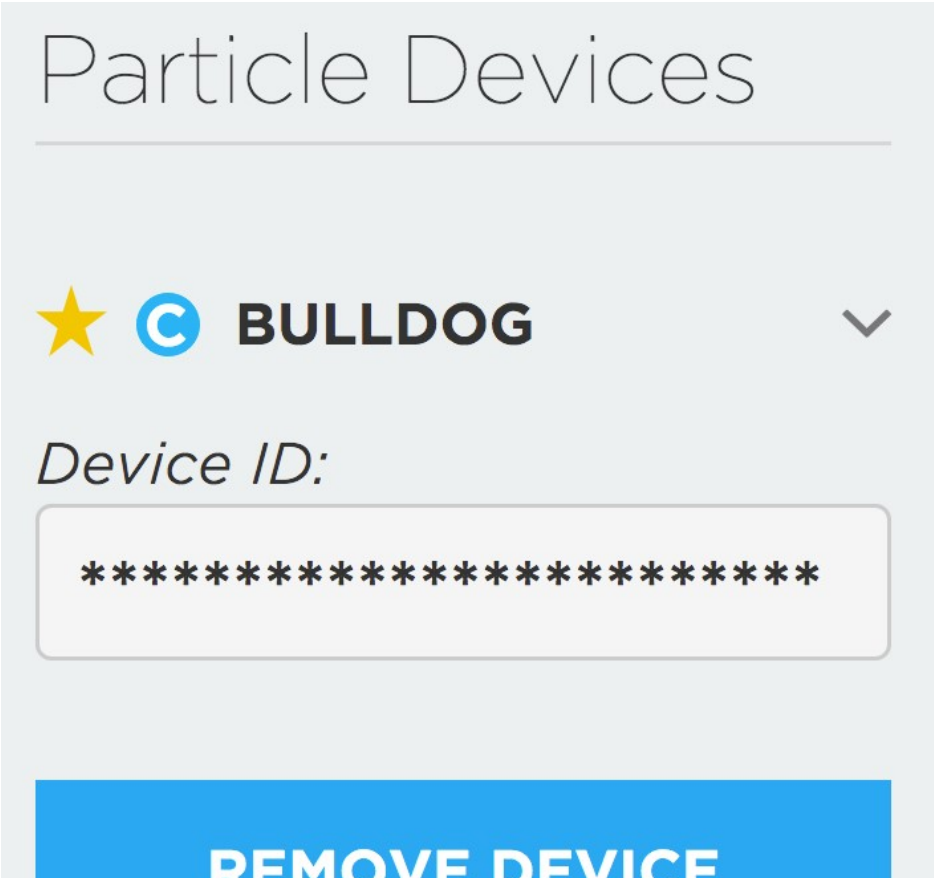
Note: you can't revert a revision to being private. Be careful about what you're making public.

Any changes you make to the app after this won't be reflected under this link. To share newer version of your app, you need to generate a new link for the latest revision.

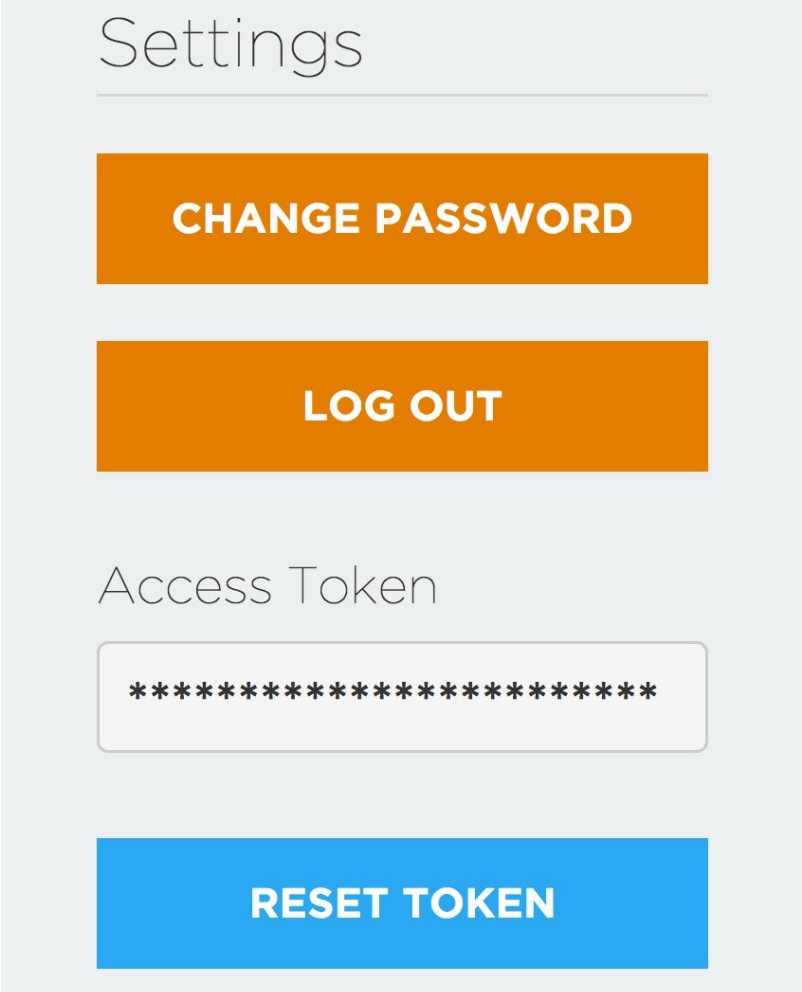
When viewing a shared app you can either flash it to any of your devices or copy it to your apps so you can modify it:

Account Information

There are a couple of other neat bells and whistles in Particle Build. The Particle Build IDE the best tool for viewing important information about your device, managing devices associated with your Particle account, and "unclaiming" them so they can be transferred to your friend.



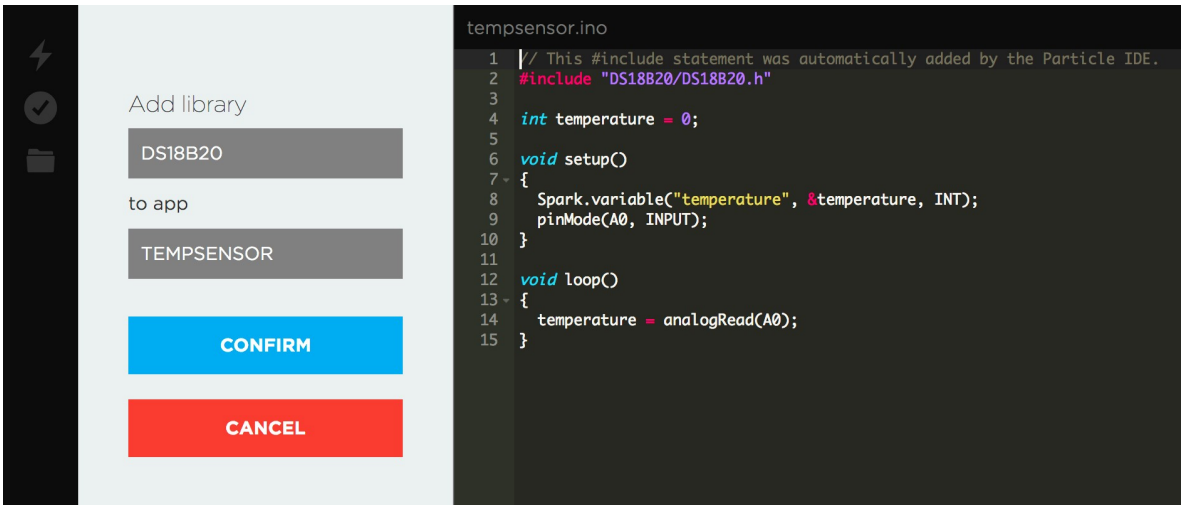
- **Device ID**: You can view your device's ID by clicking on the "Device" icon at the bottom of the navigation pane, then clicking the dropdown arrow next to the device of interest.
- **Unclaim**: You can "Unclaim" a device by pressing the "Remove Device" button that is revealed by clicking the dropdown arrow. Once a device has been unclaimed, it is available to be associated with any Particle users' account.



- **API Key**: You can find your most recent API Key listed under the "Settings" tab in your account. You can press the "Reset Token" button to assign a new

API Key to your account. *Note* that pressing this button will require you to update any hard-coded API Credentials in your Particle-powered projects!

Using Libraries



Firmware libraries are an important part of how you connect your Photon or Electron to sensors and actuators. They make it easy to reuse code across multiple Particle projects, or to leverage code written by other people in the Particle community. As an example, firmware libraries make it easy to get data out of your DS18B20 temperature sensor without writing any of the code yourself.

Particle libraries are hosted on GitHub, and can be easily accessed through through all of Particle's development tools including the Web IDE.

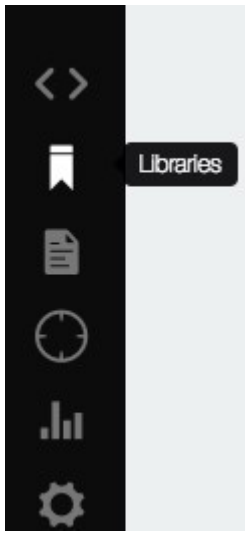
To include a firmware library in your Particle project, open the library drawer, search for the corresponding library for your sensor or actuator, and click the `Include in Project` button. Including a library in your project will add the library dependency to the `project.properties` file that will be compiled with your project when it is verified or flashed to your target device.

Read on for detailed instructions to include a firmware library in your Particle application with Build.

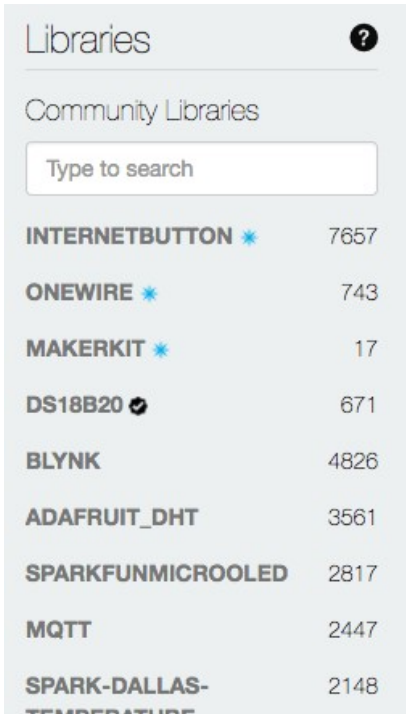
We have a detailed reference guide about libraries but for now here's a step by step guide on how to include a library in the Web IDE.

Step 1 - Go to the Libraries tab

Click on the libraries bookmark icon on the left hand side of the Build interface.



Step 2 - Find the library you need



Once you open the libraries tab, you'll be presented with a list of libraries. Libraries with the Particle logo next to them are Official libraries created by the Particle team for Particle hardware. Libraries that have a check mark next to them are Verified libraries. Verified libraries are popular community libraries that have been validated by the Particle team to ensure that they work and are well documented. Click here To learn more about the different kinds of Particle libraries.

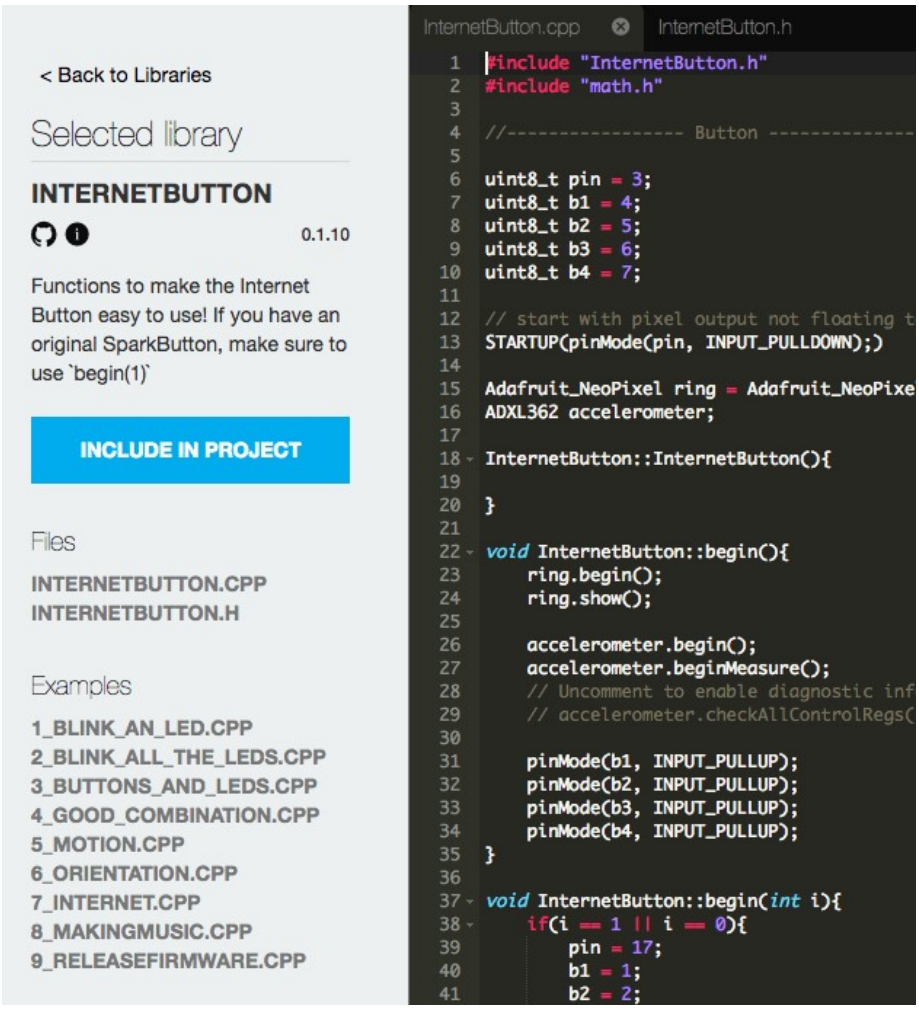
To find the right library for your project, you can either search for it directly or browse through popular firmware libraries using the browsing arrows at the bottom of the library list.

Search. To search for a library, begin typing in the search bar. Search results are ranked by match with the search term with a preference for official and verified libraries.

Browsing arrows. Not sure what library you're looking for? Use the browsing arrows beneath the library list to view additional Particle libraries in our firmware library manager. Pagination also works with search results.

Step 3 - Inspect a library

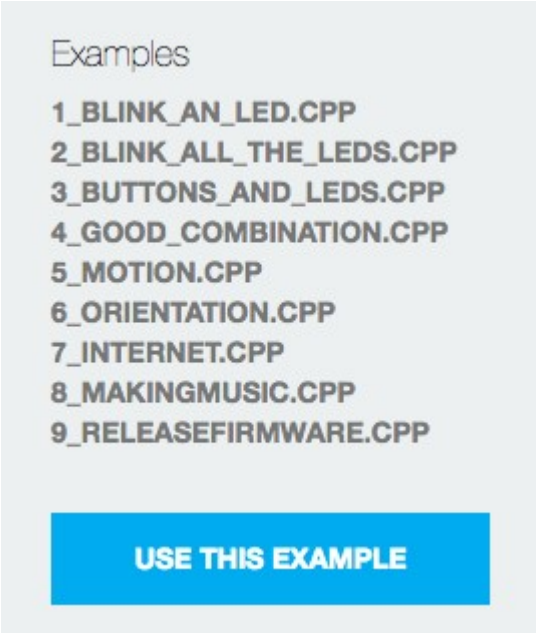
Clicking on a library from the library list shows you more detailed information about the library.



The detailed view for a library includes the following:

- `Library name` : The name of the library. The name must be unique, so there aren't two libraries with the same name.
- `Library version` : The version of the library. This follows the semver convention.

- `GitHub link` : Where the library is hosted. The code for public libraries must be open-sourced. See how to Contribute a library.
- `Library description` : Detailed information about the library
- `Library files` : What files come with the library. This follows the new library file structure.
- `Library examples` : Those are examples of usage. If you click on one of them, you will be shown the source code. To use it as one of your projects, click on 'Use this example'.

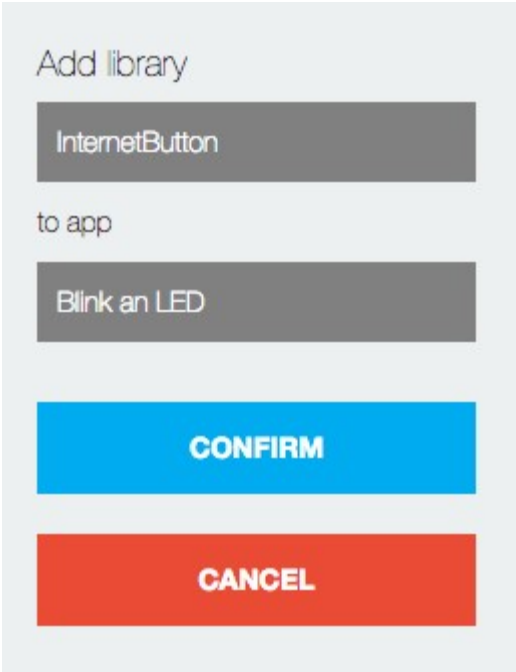


- `Library source` : In the editor you will see all the code of the library.

Step 4 - Click on 'Include in Project'

To add a firmware library to a project, click the `Include in Project` button. You will be presented with a list of your Particle projects that the library can be added

to. After you select your target project from the list, you'll be presented with a confirmation page.



Clicking the `Confirm` button will bring you back to your Particle project. The library include should appear at the top of your project source file. It should also be listed in the `Included libraries` section of the project.

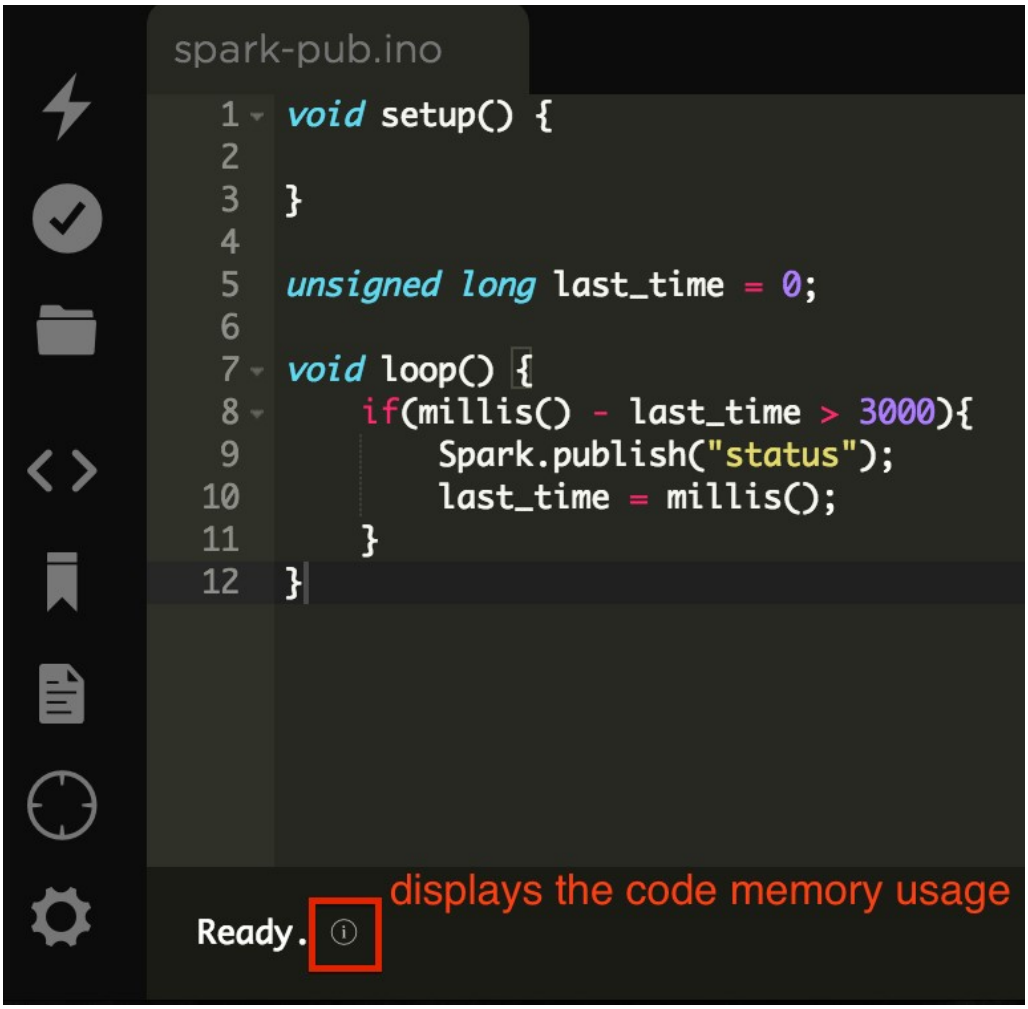
```
// This #include statement was automatically added by the Particle IDE.
#include <InternetButton.h>
```

Congrats! You have now added a firmware library to your Particle project!

Contribute a library

This functionality was moved to the Desktop IDE and the Command Line Interface (CLI). You can follow this link to find more about contributing a library.

Checking code memory usage



Whenever you **verify** your code and it compiles successfully, an "i" button will be displayed at the console output window.

This allows you to view the amount of FLASH and RAM used by this particular code/app that is currently being worked on.

*If there are no code changes and you **verify** code for the second time, the "i" button will not be available. You can simply add an extra blank space or new line before you **verify** and the "i" button will now appear.*

```
Output of arm-none-eabi-size:
text       data       bss        dec        hex
75840      1224       11860      88924      15

In a nutshell:
Flash used       77064 / 110592   69.7 %
RAM used         13084 / 20480    63.9 %
```

Wait, what is firmware?

An *embedded system* like the Core, Photon or Electron doesn't have an Operating System like a traditional computer. Instead, it runs a single application, often called *firmware*, which runs whenever the system is powered.

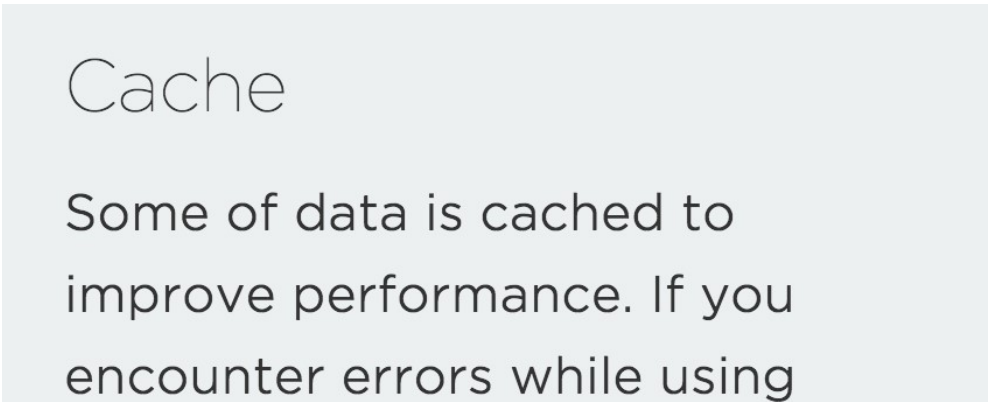
Firmware is so-called because it's harder than software and softer than hardware. Hardware is fixed during manufacturing, and doesn't change. Software can be updated anytime, so it's very flexible. Firmware is somewhere in between; hardware companies do issue firmware updates, but they tend to be very infrequent, because upgrading firmware can be difficult.

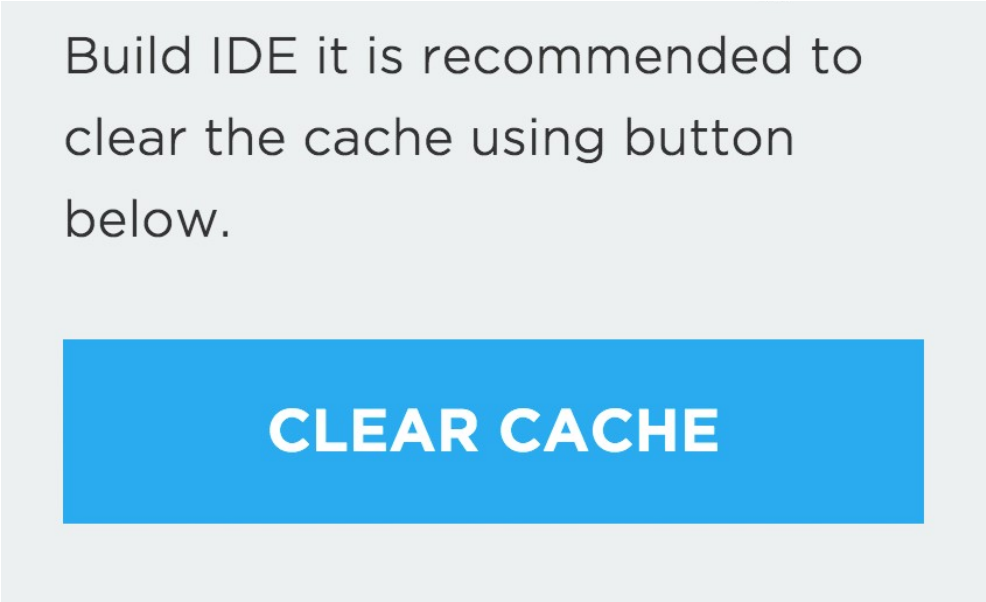
In our case, because the Cores, Photons and Electrons are connected to the internet, updating firmware is quite trivial; we send it over the network, and we have put in place safeguards to keep you from "bricking" your device.

When you flash code onto your device, you are doing an *over-the-air firmware update*. This firmware update overwrites almost all of the software on the device; the only piece that is untouched is the bootloader, which manages the process of loading new firmware and ensures you can always update the firmware over USB or through a factory reset.

For every device which version of our firmware you want to build against. In most cases you want to build with the latest firmware (which is used by default). If you need to target an older version (i.e. when newer version has some breaking changes) you can select it in dropdown located in device details.

Troubleshooting





Particle Build uses a local cache to improve its performance. In some cases this may cause errors or outdated information about libraries. If you encounter similar symptoms try clearing the cache by going to **Settings** and clicking **Clear cache** button.

Feeling oriented? Let's move on to some more interesting examples.

Also, check out and join our community forums for advanced help, tutorials, and troubleshooting.

Go to Community Forums >

ANNOTATED 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 (Included with Photon Kit and Maker Kit)
- **Software**
    - The online IDE
    - or the local Particle Dev
- **Experience**
    - Connecting your Device with your smartphone or over USB

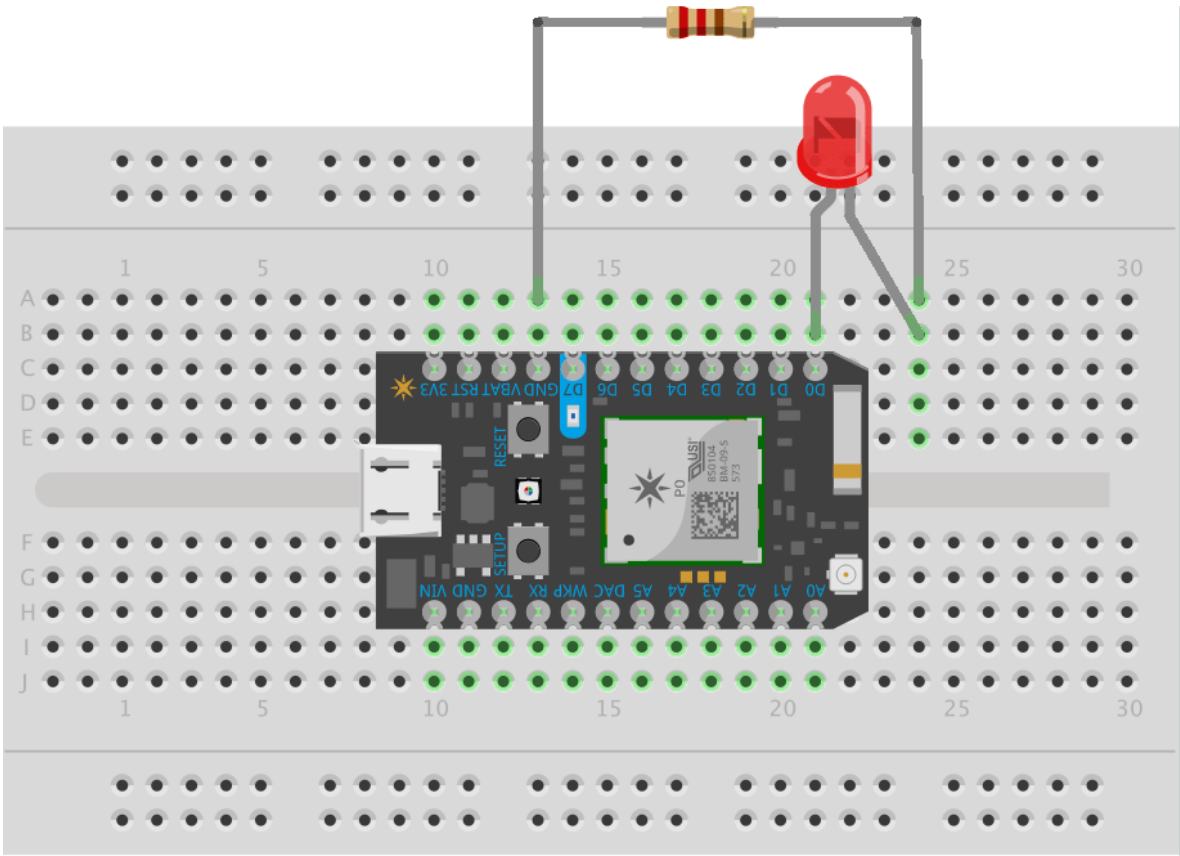
Blink an LED

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.

Setup

Connect everything together as shown in the image below. The negative (shorter) pin of the LED is connected to ground via a resistor and the positive (longer) pin is connected to D0.



Next, we're going to load code onto your device. Copy and paste this code into a new application on http://build.particle.io or on Particle Dev. We've heavily commented this code so that you can see what is going on in each line.

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

(In case you wonder how the pretty wiring diagram above was made, check out Fritzing and the Particle Fritzing parts library)

Code

```
// ------------
// 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 = D0; // 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

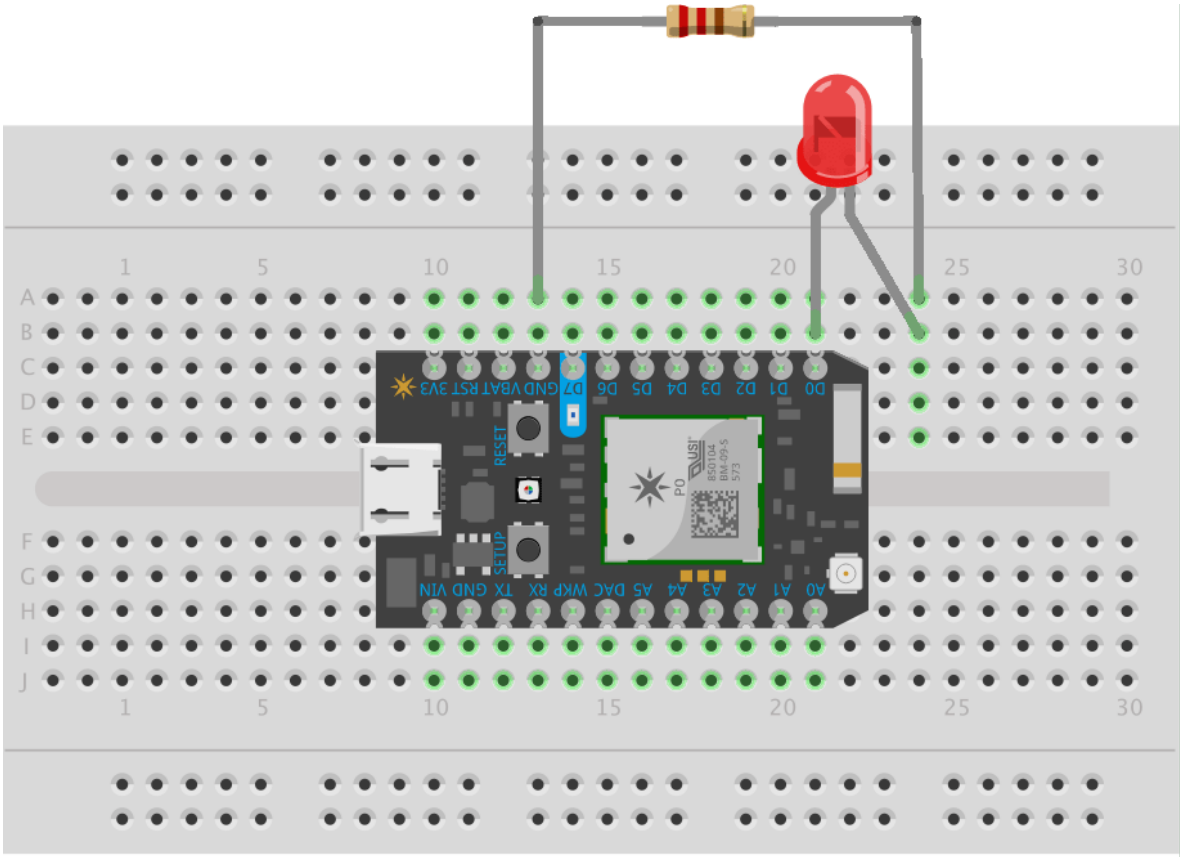
Now that we know how to blink an LED, how about we control it over the Internet? This is where the fun begins.

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

As in the previous example, connect everything together as shown in the image below. The negative (shorter) pin of the LED is connected to ground via a resistor and the positive (longer) pin is connected to D0.



Code

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

/* First, let's create our "shorthand" for the pins
Same as in the Blink an LED example:
led1 is D0, led2 is D7 */

int led1 = D0;
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;
    }
}
```

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 Photoresistor: Function and Variable

Intro

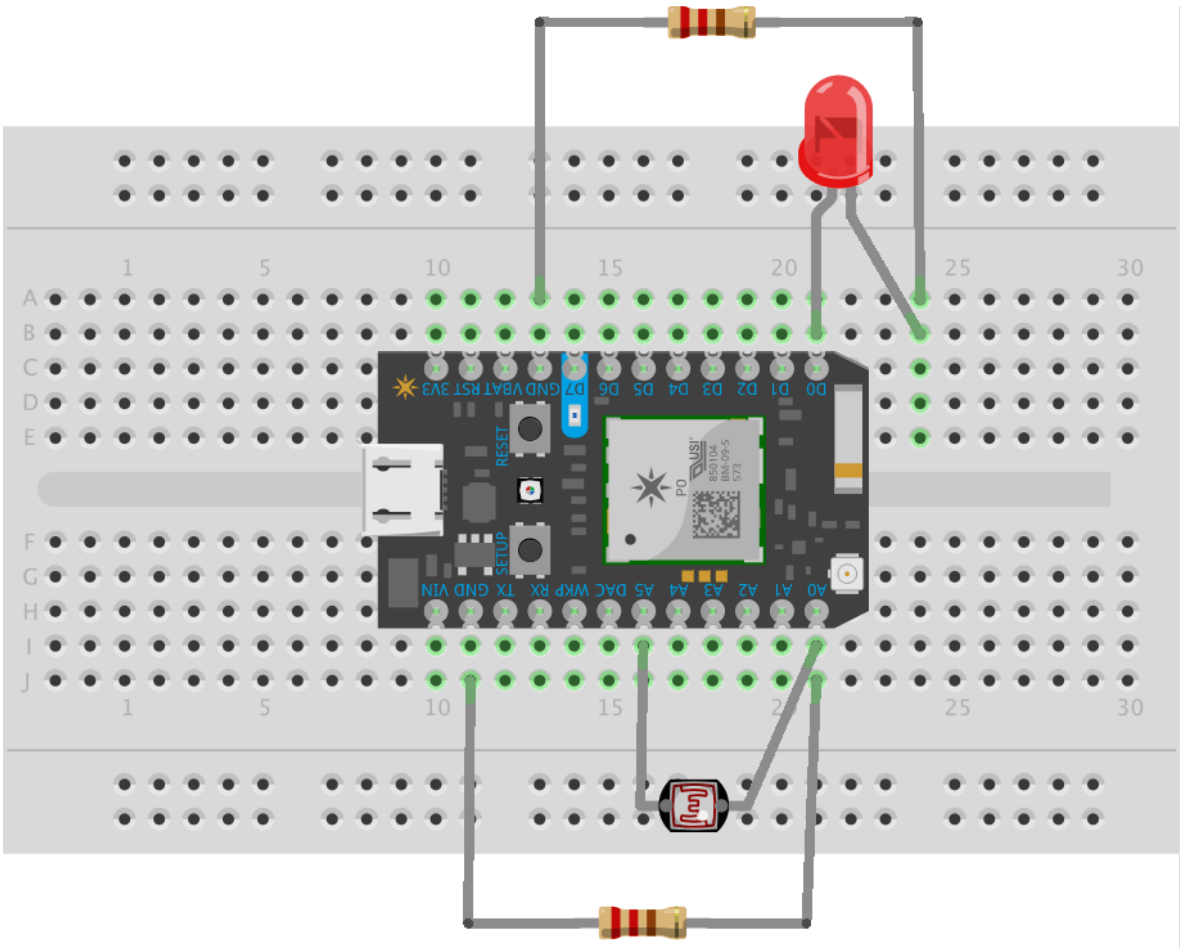
This example uses the same setup as the LED control example to make a `Particle.function`. This time, though, we're going to add a sensor.

We will get a value from a photoresistor and store it in the cloud.

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

Setup

Set up your breadboard as shown in the image below:



Make sure that the short leg of the LED is plugged into `GND`. The other orientations do not matter.

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

Code

Copy and paste the following code into your online IDE or Particle Dev environment.

```
// ---------------------------------------
// Function and Variable with Photoresistors
// ---------------------------------------
// In this example, we're going to register a Particle.variable() with the cloud so that we can read brightness levels from the photoresistor.
// 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 = D0; // This is where your LED is plugged in. The other side goes to a resistor connected to GND.

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

int power = A5; // This is the other end of your photoresistor. The other side is plugged into the "photoresistor" pin (above).
// The reason we have plugged one side into an analog pin instead of to "power" is because we want a very steady voltage to be sent to the photoresistor.
// That way, when we read the value from the other side of the photoresistor, we can accurately calculate a voltage drop.

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


// Next we go into the setup function.
```

```
void setup() {

    // 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)
    pinMode(photoresistor,INPUT);  // Our photoresistor pin is
input (reading the photoresistor)
    pinMode(power,OUTPUT); // The pin powering the photoresistor
is output (sending out consistent power)

    // Next, write one pin of the photoresistor to be the maximum
possible, so that we can use this for power.
    digitalWrite(power,HIGH);

    // We are going to declare a Particle.variable() here so that
we can access the value of the photoresistor 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 is and
store it in the int variable analogvalue
    analogvalue = analogRead(photoresistor);
    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 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 photoresistor 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

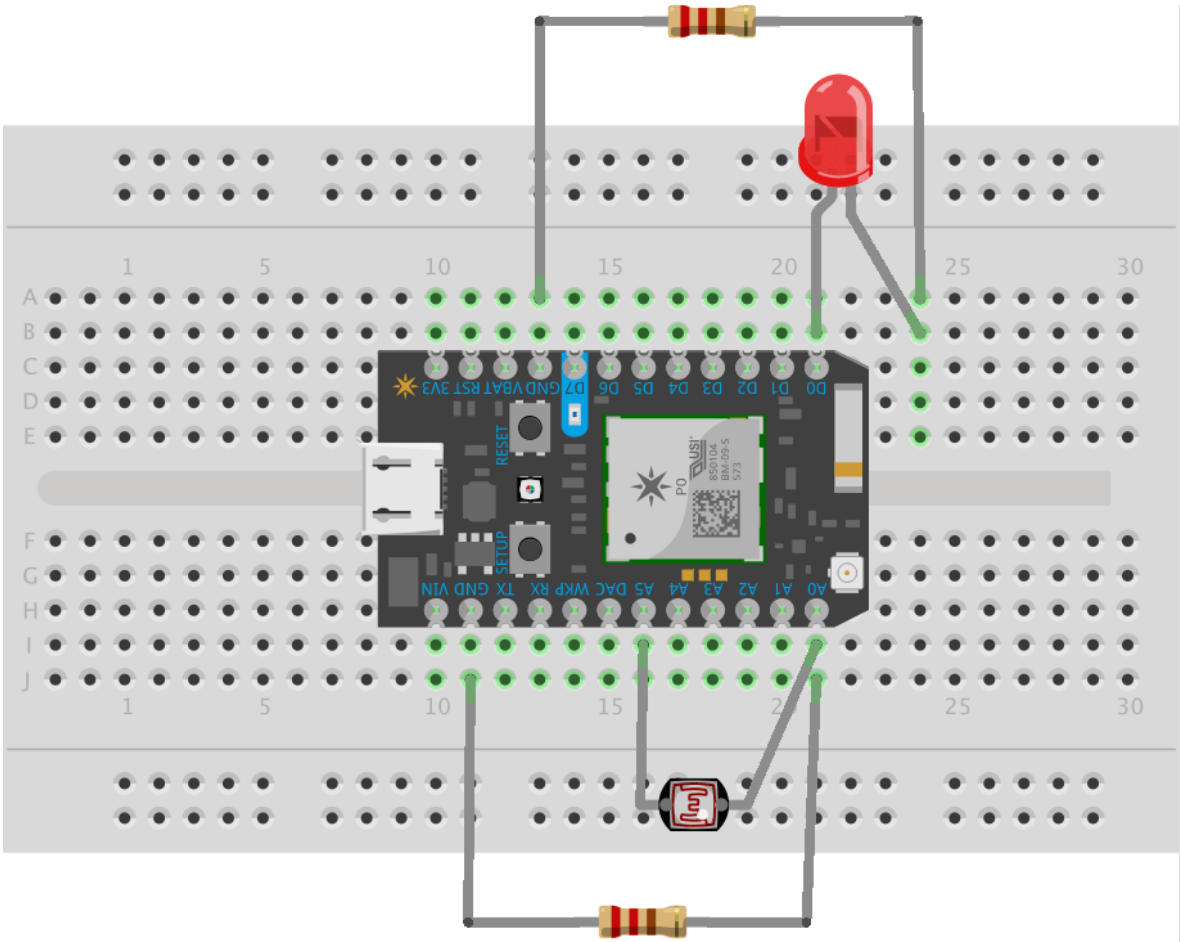
beam. If you mess up, don't worry-- you can just hit "reset" on your device and do it again!

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.

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

Setup

The setup is the same as in the last example. Set up your breadboard as follows:



Ensure that the short end of the LED is plugged into `GND` and that the LED and Photoresistor are bent to face each other. (You want the LED, when turned on, to

shine its beam of light directly at the photoresistor.) Try to leave enough space between the LED and the Photoresistor for your finger or a piece of paper.

Code

```
// ---------------------------------------
// Publish and Console with Photoresistors
// ---------------------------------------
// This app will publish an event when the beam of light between the LED and the photoresistor is broken.
// It will publish a different event when the light is intact again.

// Just like before, we're going to start by declaring which pins everything is plugged into.

int led = D0; // This is where your LED is plugged in. The other side goes to a resistor connected to GND.
int boardLed = D7; // This is the LED that is already on your device.
// On the Core, it's the LED in the upper right hand corner.
// On the Photon, it's next to the D7 pin.

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

int power = A5; // This is the other end of your photoresistor. The other side is plugged into the "photoresistor" pin (above).

// The following values get set up when your device boots up and calibrates:
int intactValue; // This is the average value that the photoresistor reads when the beam is intact.
int brokenValue; // This is the average value that the photoresistor reads when the beam is broken.
int beamThreshold; // This is a value halfway between ledOnValue and ledOffValue, above which we will assume the led is on and below which we will assume it is off.

bool beamBroken = false; // This flag will be used to mark if we have a new status or not. We will use it in the loop.
```

```
// 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
  pinMode(photoresistor,INPUT);  // Our photoresistor pin is input
(reading the photoresistor)
  pinMode(power,OUTPUT); // The pin powering the photoresistor is
output (sending out consistent power)

  // Next, write the power of the photoresistor to be the maximum
possible, which is 4095 in analog.
  digitalWrite(power,HIGH);

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

  // 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 on_1 = analogRead(photoresistor); // read photoresistor
  delay(200); // wait 200 milliseconds
  int on_2 = analogRead(photoresistor); // read photoresistor
  delay(300); // wait 300 milliseconds

  // 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 off_1 = analogRead(photoresistor); // read photoresistor
  delay(200); // wait 200 milliseconds
  int off_2 = analogRead(photoresistor); // read photoresistor
  delay(1000); // wait 1 second

  // 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);


  // Now we average the "on" and "off" values to get an idea of
what the resistance will be when the LED is on and off
  intactValue = (on_1+on_2)/2;
  brokenValue = (off_1+off_2)/2;

  // Let's also calculate the value between ledOn and ledOff,
above which we will assume the led is on and below which we assume
the led is off.
  beamThreshold = (intactValue+brokenValue)/2;

}


// Now for the loop.

void loop() {
```

```
  /* In this loop function, we're going to check to see if the
beam has been broken.
  When the status of the beam changes, we'll send a
Particle.publish() to the cloud
  so that if we want to, we can check from other devices when the
LED is on or off.

  We'll also turn the D7 LED on when the Photoresistor detects a
beam breakage.
  */

  if (analogRead(photoresistor)>beamThreshold) {

    /* If you are above the threshold, we'll assume the beam is
intact.
    If the beam was intact before, though, we don't need to change
anything.
    We'll use the beamBroken flag to help us find this out.
    This flag monitors the current status of the beam.
    After the beam is broken, it is set TRUE
    and when the beam reconnects it is set to FALSE.
    */

    if (beamBroken==true) {
        // If the beam was broken before, then this is a new
status.
        // We will send a publish to the cloud and turn the LED on.

        // Send a publish to your devices...
        Particle.publish("beamStatus","intact",60,PRIVATE);
        // And flash the on-board LED on and off.
        digitalWrite(boardLed,HIGH);
        delay(500);
        digitalWrite(boardLed,LOW);

        // Finally, set the flag to reflect the current status of
the beam.
        beamBroken=false;
    }
    else {
        // Otherwise, this isn't a new status, and we don't have
to do anything.
    }
  }
```

```
  else {
      // If you are below the threshold, the beam is probably
broken.
      if (beamBroken==false) {

        // Send a publish...
        Particle.publish("beamStatus","broken",60,PRIVATE);
        // And flash the on-board LED on and off.
        digitalWrite(boardLed,HIGH);
        delay(500);
        digitalWrite(boardLed,LOW);

        // Finally, set the flag to reflect the current status of
the beam.
        beamBroken=true;
      }
      else {
          // Otherwise, this isn't a new status, and we don't have
to do anything.
      }
  }

}
```

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.

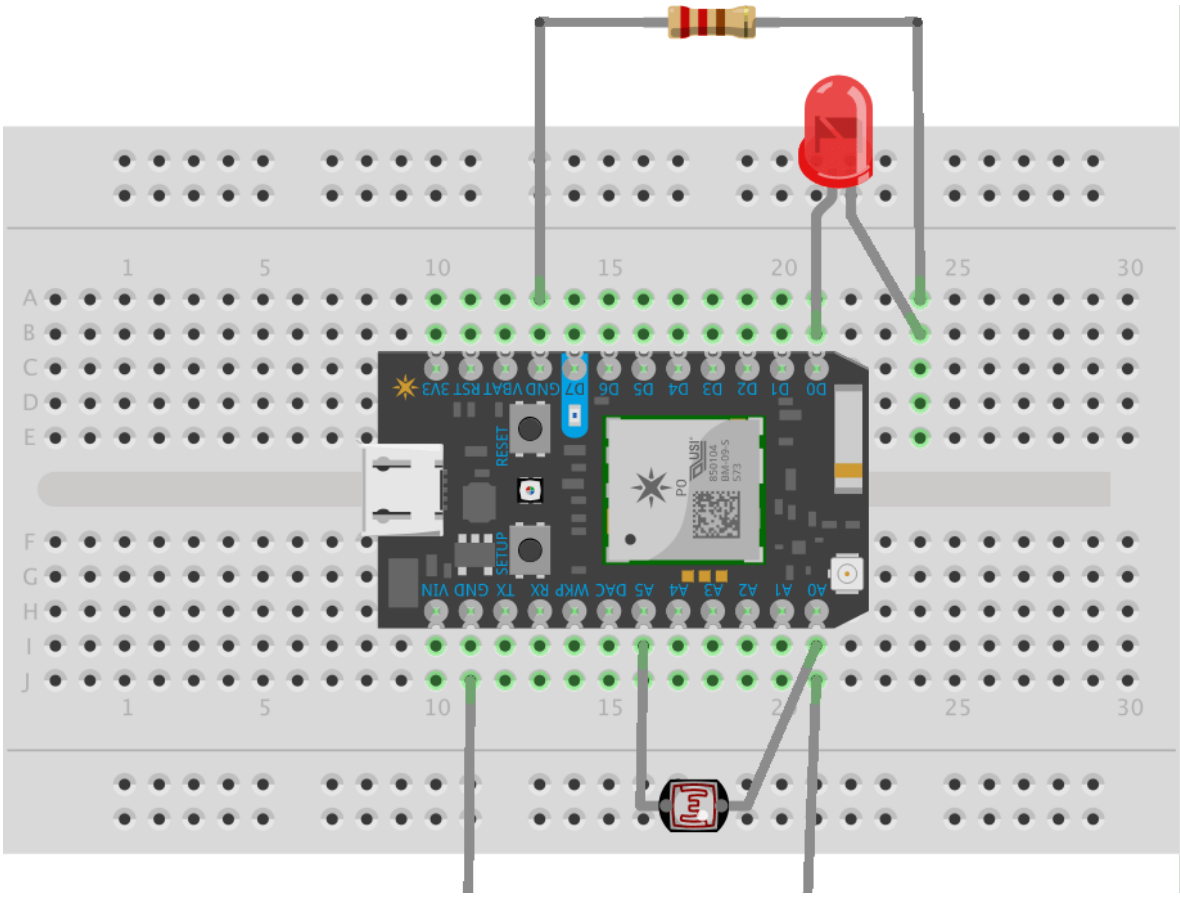
Have your buddy do the same thing, only with their event name and yours (swap 'em).

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 is the same as in the last example. Set up your breadboard as follows:



Ensure that the short end of the LED is plugged into `GND` and that the LED and Photoresistor are bent to face each other. (You want the LED, when turned on, to shine its beam of light directly at the photoresistor.) Try to leave enough space between the LED and the Photoresistor for your finger or a piece of paper.

Code

```
// ----------------------------------------
// Publish and Subscribe with Photoresistors
/* ----------------------------------------

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 = D0;
int boardLed = D7;
int photoresistor = A0;
int power = A5;

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
  pinMode(photoresistor,INPUT);  // Our photoresistor pin is input
(reading the photoresistor)
  pinMode(power,OUTPUT); // The pin powering the photoresistor is
output (sending out consistent power)

  // 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.

  // Next, deliver power to the photoresistor
  digitalWrite(power,HIGH);

  // Since everyone sets up their LEDs differently, we are also
going to start by calibrating our photoresistor.
  // 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(photoresistor); // read photoresistor
  delay(200); // wait 200 milliseconds
  int off_2 = analogRead(photoresistor); // read photoresistor
  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(photoresistor); // read photoresistor
  delay(200); // wait 200 milliseconds
  int on_2 = analogRead(photoresistor); // read photoresistor
  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(photoresistor)>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;
}
```

Also, check out and join our community forums for advanced help, tutorials, and troubleshooting.

Go to Community Forums >

HOW TO CONTRIBUTE TO PARTICLE'S OPEN SOURCE REPOSITORIES

Particle is an open-source platform. You might find something you'd like changed, a feature you'd like implemented, or a bug you see a fix for. If so, you can create a pull request or an issue on one of our open-source GitHub repos. Here's a quick run-down of some repos you might want to know about, as well as some information on how to contribute.

Open-Source Repos

Go to http://github.com/particle-iot to see all the available repositories. There are quite a few! Here's a guide to the most popular repos.

Style guides

Before you contribute to the code base, check out the `style-guides` repo. This will give you an idea of how we format our code. If you have additional suggestions on good code practices, please make a pull request.

Firmware

The `firmware` repo contains the system firmware that runs on the Core and Photon.

See the local build tools FAQ for steps to build the firmware on your machine.

CLI

The `particle-cli` repo contains the code used to run the Command Line Interface.

Particle API JS

The `particle-api-js` repo contains the code used to run Particle API JS, the official Particle JavaScript wrapper.

Mobile

There are several mobile repos, for both iOS and Android.

- iOS Repos
    - `spark-setup-ios` : the Particle Device Setup library
    - `spark-sdk-ios` : the Particle iOS Cloud SDK
    - `photon-tinker-ios` : the official Particle App
- Android Repos
    - `spark-setup-android` : the Particle Device Setup library
    - `spark-sdk-android` : the Particle Android Cloud SDK
    - `photon-tinker-android` : the official Particle App

Particle Dev

The `particle-dev` repo contains the code used to run Particle Dev, our professional, open source, hackable IDE, designed for use with Particle devices.

Documentation

The `docs` repo contains Particle's open source documentation. If you want something to be added or changed, just make a pull request.

Official Libraries

Particle maintains several open source libraries to be used with official Particle shields. They include:

- `InternetButton` : the library intended for use with the Internet Button.
- `RelayShield` : the library intended for use with the Photon Relay Shield

- `PowerShield` : the library intended for use with the Photon Power Shield

Local Cloud

The `spark-server` repo is an API compatible open source server for interacting with devices speaking the `spark-protocol` . If you are interested in the local cloud, this repo is for you.

Hardware Design

We share some hardware design files for each of our dev boards. These open source repos are designed mostly to give you a sense of what we are working on, but you are welcome to make contributions here as well if you have interest and expertise.

Current hardware design repos include:

- `electron`
- `photon`
- `core`
- Official Libraries such as the `InternetButton` , `RelayShield` , and `PowerShield`
- `hardware-libraries`

Using GitHub

If you're new to GitHub expert, go here to get a sense of GitHub flow. We also suggest the GitHub for Desktop application, which has a great built-in tutorial on forking, editing, merging, and creating a pull request.

If you prefer the command line, here's an extra fast tutorial on how to get started. Just make sure you fork a repo to your GitHub account before cloning your fork to edit on your local machine.

If you have your own favorite tutorials on how to `git` , make a pull request on this documentation to add them!

Also, check out and join our community forums for advanced help, tutorials, and troubleshooting.

Go to Community Forums >

HACKATHONING WITH PARTICLE

If you're at a hackathon, you'll want to get started with Particle, and fast! Here's some quick tips to get you up and running ASAP.

We'll go over how to set up your Photon, then go into some other tools you should install and features you should be aware of to get the most out of your hackathoning.

Setup

Making an Account

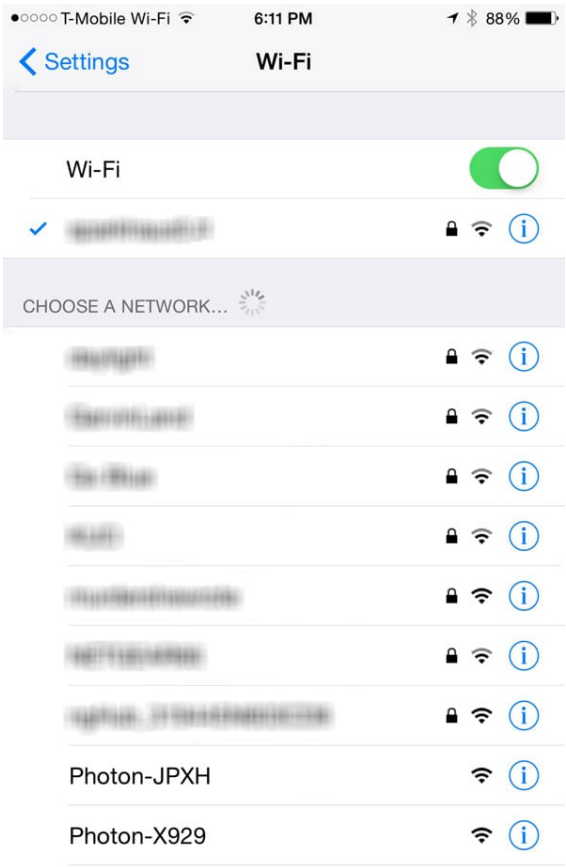
The first step, if you don't already have one, is to make an account. If you are the only one working on the firmware for this project, you can make an account with your personal email and a personal password. If you are working on a team and multiple people need to access the device, it may be a good idea to claim the device with a shared team email and password.

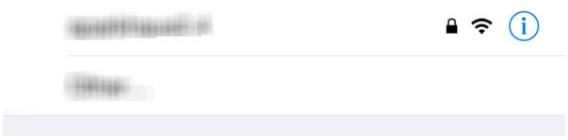
Claiming and Connecting

Once you have your account, it is time to claim your Photon to your account and get started coding. You'll be doing this with your iOS or Android device. Before you get started, examine the packaging that your Photon, Photon Kit, or Internet Button came in. You'll see a bar code like the one below. The bar code has a four-digit code (outlined in blue below) in the third part of the alphanumeric sequence.

In this case, you can see that four-digit code is `GG9J`.

During your smartphone setup, this code will appear after the word `Photon` in your Photon's Wi-Fi network:





In this iOS screenshot, you can see two Photons online. If you see this while you're setting up your PHoton, it just means you need to pick the Photon with your correct bar code (in this case, either `JPXH` or `X929`).

Continue to set up your Photon with your smartphone. Follow the instruction on the Particle app to get your Photon online and claim it to the account you set up earlier.

Particle Dev

Although the online IDE will allow you to flash code to your devices, we also suggest getting Particle Dev, the local IDE, for hackathons. It has a nice interface for monitoring Particle.variables and running Particle.functions without the CLI, which means you can develop faster.

Download Particle Dev here.

Particle Dev has a lot of wonderful features. Namely, you can monitor cloud variables, cloud functions, and serial output from your devices via the Devices part of the menu bar. More on that in the next section.

GitHub

If you plan to share code and develop with other team members, we recommend that you get a GitHub account. It's great for code sharing and will help you take advantage of your dev time.

Here's a basic guide on how to git for complete newcomers.

You can develop code, test it on your device, and push it to your repo. You and other team members can work on different features of your project on separate branches, develop and test separately, and merge features back into the master branch when they are finished.

Particle Dev does not play well with folder hierarchies. If you are including libraries in your repo, keep them in the same folder as the `.ino` file you plan to compile!

Mobile

Building a mobile app to hook up with a Particle device? Check out our guide on building mobile apps.

You can also read the reference for iOS and Android SDKs, and check out the mobile section of the community to ask for help and give feedback!

Connecting to Other Services

If you want to hook up different platforms quickly and without much code, try using our IFTTT integration. It's a simple way to connect your Particle device to email, text message, phone location, Twitter, Facebook, and more.

If you're looking to hit an API not listed in IFTTT and you have a little more experience with coding, try webhooks instead.

Now Do Stuff!

View Functions

Open up Particle Dev and go to the Particle menu. Select your device with `Select device`. Then select `Show cloud functions`.

A menu will appear on the lower half of the screen. It should have four functions: digitalWrite, digitalRead, analogWrite, and analogRead. These are the Tinker functions that are defaulted to your device before you flash your own code. Check out the Tinker docs for more info. You should be able to send commands to your device by identifying the pin and, for `write` functions, any necessary info. For example:

- `D7,HIGH` for digitalWrite

- `D7` for digitalRead
- `A0,100` for analogWrite
- `A0` for analogRead

Try it out!

View Variables

In addition to the `Show cloud function` ability, there are other elements of Particle Dev that you may find helpful:

If you go to the Particle menu you will also see `Show cloud variables`. This will allow you to see any variables you register with Particle.variable.

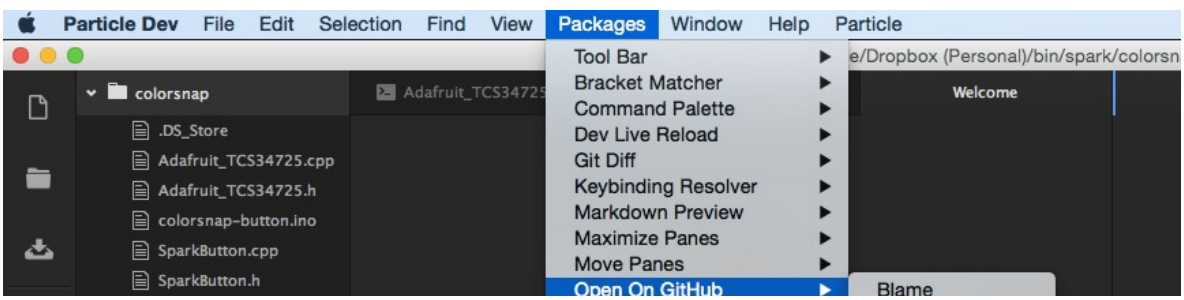
You can also monitor any serial output with the `Show serial monitor` command. Make sure the proper device is selected through `Particle` > `Select device` when you do this.

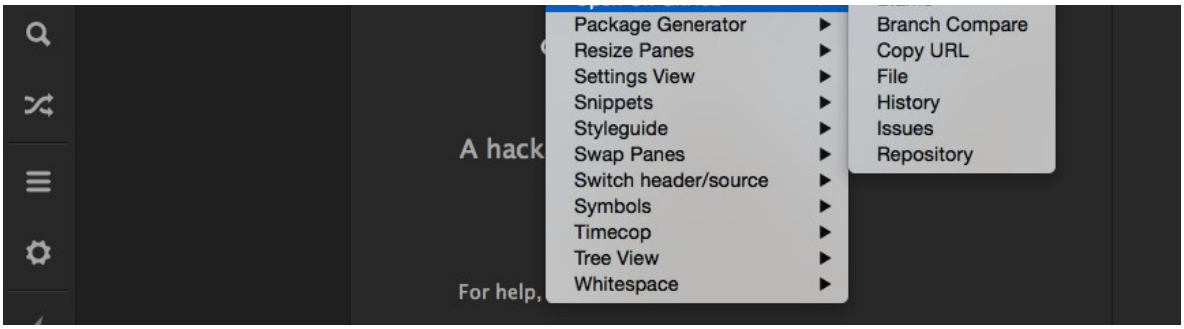
Flash Your Own Code

Although Tinker can be really useful in early stages, we're sure you'll want to put your real code on your device at some point.

For maximum sharability, make a repo on GitHub for your shared code. Hook up to this repo by using `git clone` on the clone URL. Open Particle Dev and navigate to the repo you just cloned. Make sure you have a single folder that will hold all of your .ino files and libraries-- the current version of Particle Dev gets confused when libraries are placed in subdirectories.

Note that now that you are in a GitHub repo, you can use some of the functions in Particle Dev that play nicely with git. Access these in the `Packages` menu:





Now you can compile and flash your first `.ino` file from Particle Dev. You are welcome to take any of these examples as a first try and flash it to your device by clicking the lightning bolt in the upper right corner. The examples explain how Variable, Function, Publish, and Subscribe are used, but documentation on coding these cloud-exposed elements is available here.

Pass Strings

In the quintessential example for Particle Function, we are calling a brewCoffee function with the argument "coffee". This is nice if we have an array of choices we want to identify with strings, but what if we want to pass `int` data to our `Particle.function`?

If you want to use `int` data from a `String`, you can create a `char` array to hold the `String` command, then use `atoi` to change this into an `int` value. Check out this example for setting the brightness of an LED.

```
void setup() {
    Particle.function("brightness", setBrightness);
}

int setBrightness(String command) {
    if (command) {
        char inputStr[64];
        command.toCharArray(inputStr,64);
        int value = atoi(inputStr);
        analogWrite(led,value);
        return value;
    }
    else {
        return 0;
```

```
    }
}
```

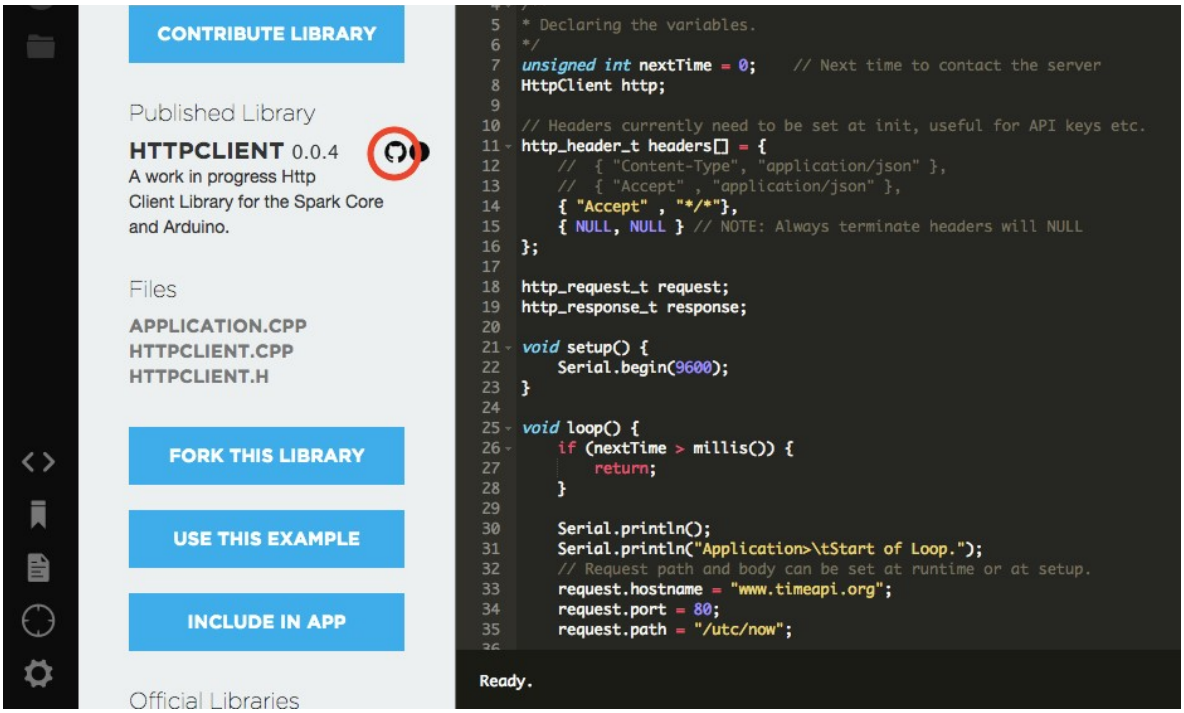
Sometimes you want to send multiple values through a `Particle.function`. If you want to do that, use `strtok`. Here's an example of a `Particle.function` you could use to set the color of an RGB LED.

```
void setup() {
    Particle.function("color", setColor);
}

int setColor(String command) {
    if (command) {
        char inputStr[64];
        command.toCharArray(inputStr,64);
        char *p = strtok(inputStr,",");
        int red = atoi(p);
        p = strtok(NULL,",");
        int grn = atoi(p);
        p = strtok(NULL,",");
        int blu = atoi(p);
        p = strtok(NULL,",");
        b.allLedsOn(red,grn,blu);
        return 1;
    }
    else {
        return 0;
    }
}
```

Add Libraries

If you want to add a library, you can search in the Libraries tab of Particle Build. To include this in a dev app, download a fork of the library to be included in your app. You can get to the repo by clicking the GitHub icon next to the library's name:



Note that some libraries may not yet be updated for the Photon.

Other Resources

Happy hacking! If you get stuck, try out:

- These docs, specifically the ones referring to the kit you have (standard examples or Internet Button docs)
- The Particle Community
- The Particle Support Page

Best of luck!