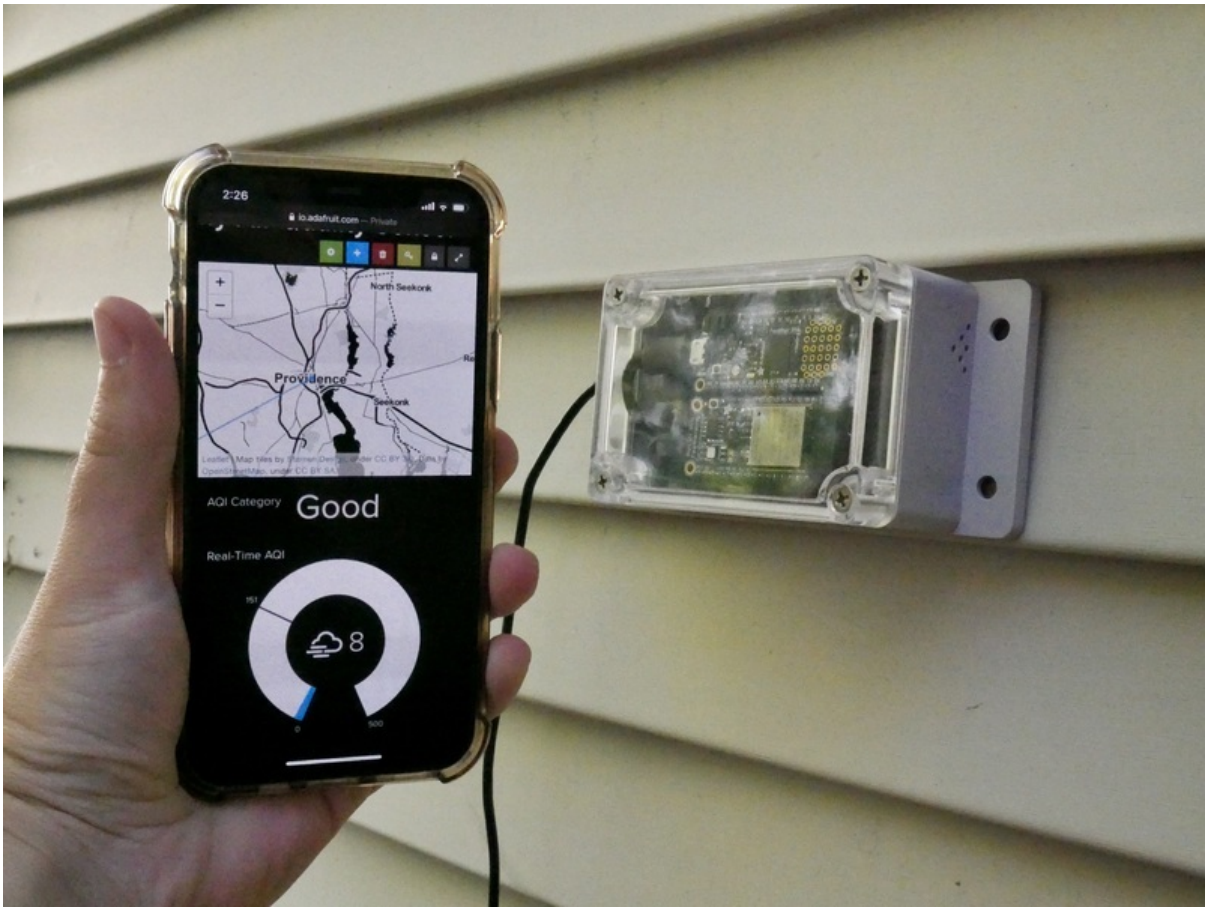
IoT Air Quality Sensor with Adafruit IO

Created by Brent Rubell

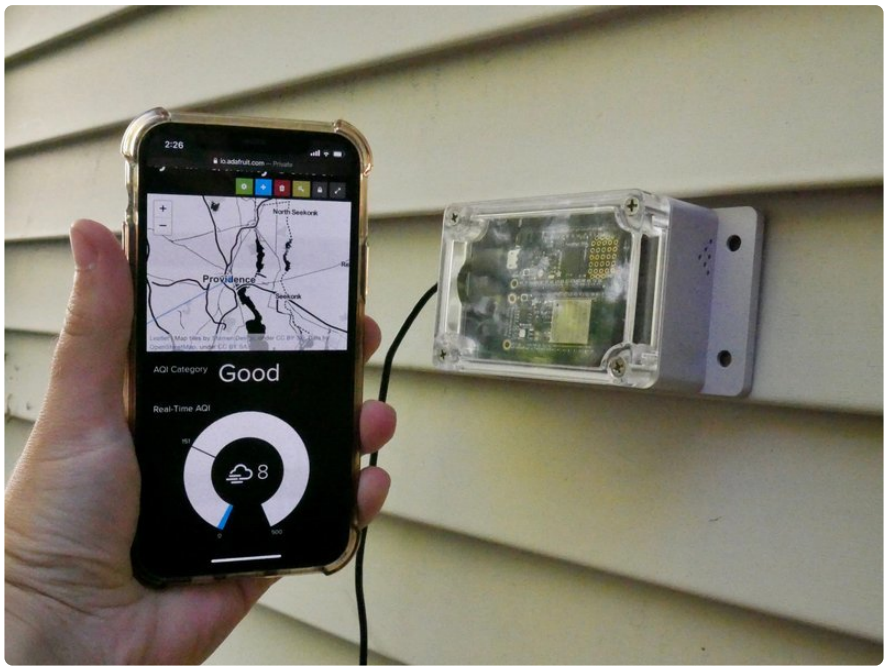


https://learn.adafruit.com/diy-air-quality-monitor

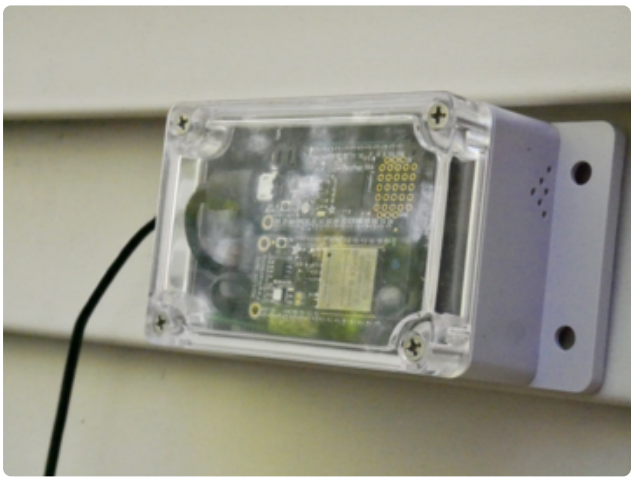
Last updated on 2024-06-03 03:13:57 PM EDT

Table of Contents

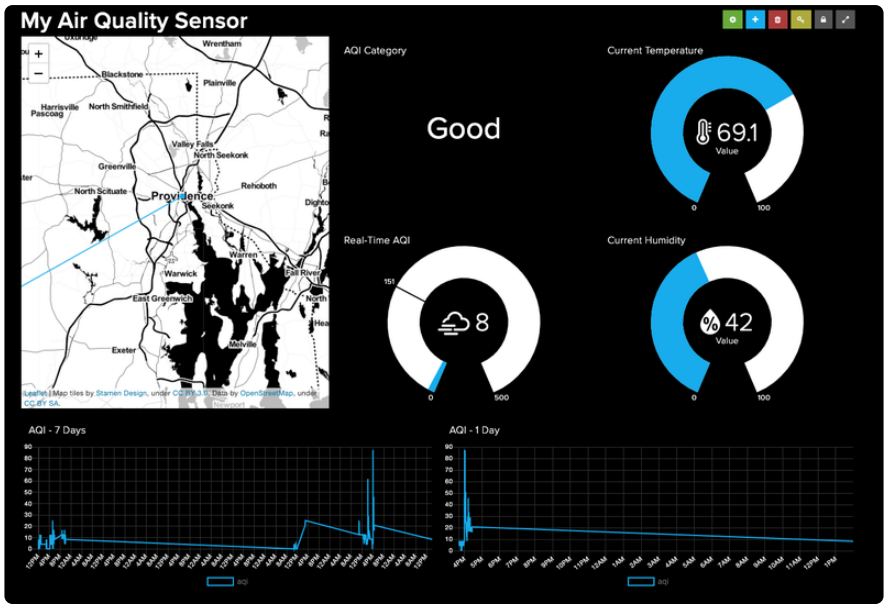
Overview



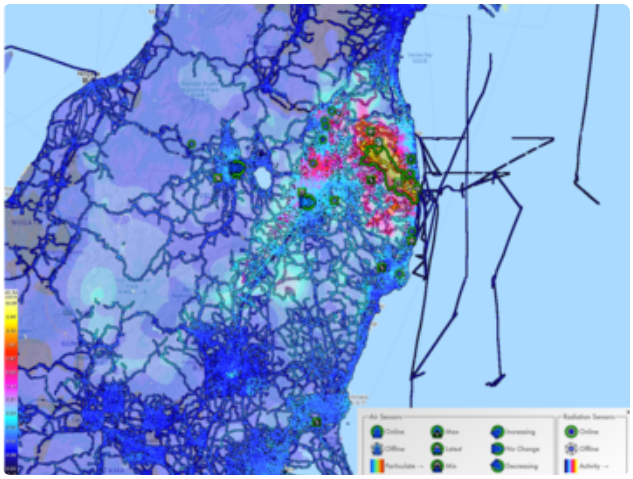
Breathe easy, knowing that you can track and sense the quality of the air (and environment!) around you with an **IoT Air Quality Sensor.** This sensor **measures PM2.5 (particles that are 2.5 microns or smaller in diameter) dust concentrations, temperature and humidity**. This sensor is small, wall-mountable (indoors or outdoors), weatherproof, and only requires a WiFi network connection and an AC outlet.



You'll **assemble an open source air quality sensor.** Then, you'll **program the sensor using CircuitPython** to measure air quality data and periodically **send measurements to** Adafruit IO, our incredible IoT Service **(https://adafru.it/Fm6).** Finally, you'll create a beautiful Adafruit IO dashboard to **visualize your sensor data from anywhere in the world**.



Why would I want to build an Air Quality Monitor?



Citizen Science

Soon after the Fukushima nuclear disaster in Japan, trustworthy information about radiation levels was publicly unavailable. An international volunteer organization, Safecast (https://adafru.it/O5B), designed devices for radiation mapping and openly shared their measurements to the public.

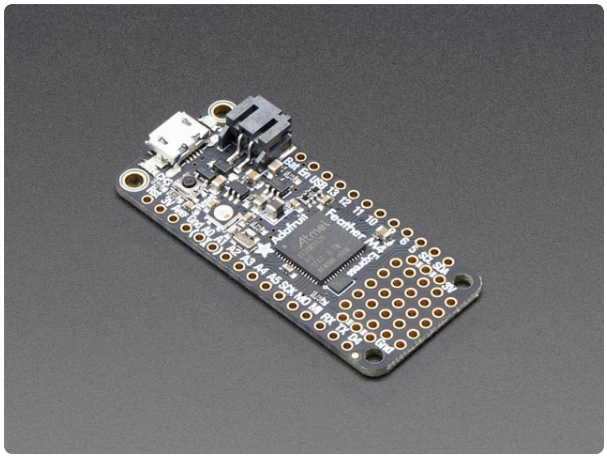
With the increasing amount of natural disasters, **building an open source air quality monitor is a step towards citizens being able to monitor and share data about essential environmental measurements without the need to trust an environmental regulatory body or wait for an official government response**.

Open Source Science

While building this sensor, we noticed a large amount of the current air quality sensor offerings on the market are closed source software and have a private API. This means it's impossible to send data from a DIY air quality sensor to their web service. If we purchased one of the company sponsored sensors and the company went out of business, we'd be left with a sensor which couldn't send data to the internet.

Building an own open source air quality monitoring sensor lets you control the hardware going into your IoT air quality sensor, the software running on the sensor (right down to the firmware!) and the web platform. We are using Adafruit.io (https://adafru.it/Fm6) for this guide, but you are free to modify the code to send data to other services such as Google Cloud Platform, Amazon AWS IoT, or Microsoft Azure IoT.

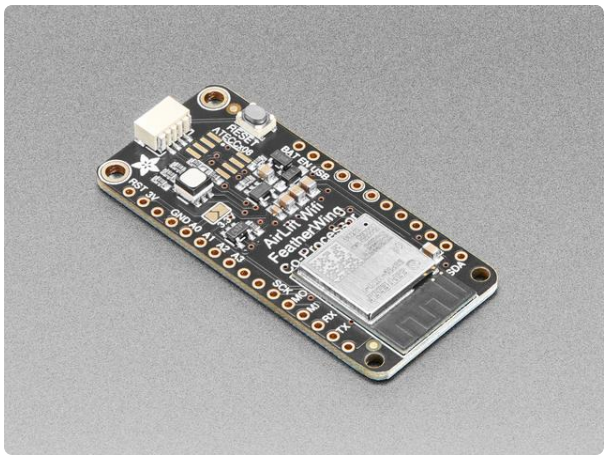
Parts



Adafruit Feather M4 Express - Featuring ATSAMD51

It's what you've been waiting for, the Feather M4 Express featuring ATSAMD51. This Feather is fast like a swift, smart like an owl, strong like a ox-bird (it's half ox,...

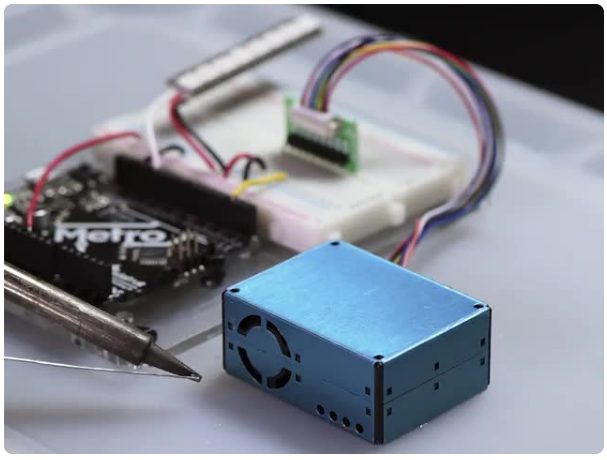
https://www.adafruit.com/product/3857



Adafruit AirLift FeatherWing – ESP32 WiFi Co-Processor

Give your Feather project a lift with the Adafruit AirLift FeatherWing - a FeatherWing that lets you use the powerful ESP32 as a WiFi co-processor. You probably have your...

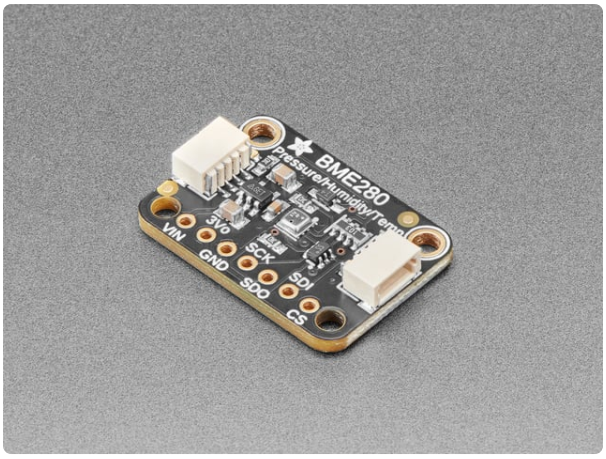
https://www.adafruit.com/product/4264



PM2.5 Air Quality Sensor and Breadboard Adapter Kit

Breathe easy, knowing that you can track and sense the quality of the air around you with the PM2.5 Air Quality Sensor with Breadboard Adapter particulate sensor....

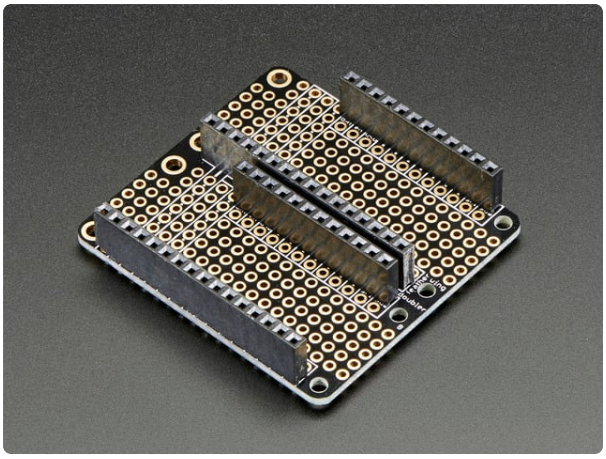
https://www.adafruit.com/product/3686



Adafruit BME280 I2C or SPI Temperature Humidity Pressure Sensor

Bosch has stepped up their game with their new BME280 sensor, an environmental sensor with temperature, barometric pressure and humidity! This sensor is great for all sorts...

https://www.adafruit.com/product/2652



FeatherWing Doubler - Prototyping Add-on For All Feather Boards

This is the FeatherWing Doubler - a prototyping add-on and more for all Feather boards. This is similar to our

https://www.adafruit.com/product/2890

Flanged Weatherproof Enclosure With PG-7 Cable Glands

Whether you're raiding tombs or traversing nuclear fallout wastelands, this is the most heavy-duty enclosure for your project! Weatherproof? Check. Tough polycarbonate cover?...

https://www.adafruit.com/product/3931

1 x Silicone Stranded Cable — https://www.adafruit.com/product/3889

Silicone Cover Stranded-Core Ribbon Cable - 4 Wires 1 Meter Long - 30 AWG Black

---

1 x Lipo Battery — https://www.adafruit.com/product/3898

Lithium Ion Polymer Battery, ideal For Feathers - 3.7V 400mAh

---

1 x USB Power Supply — https://www.adafruit.com/product/1994

5V 2A Switching Power Supply w/ USB-A Connector

USB A/Micro Cable - 2m
1 x 6ft Micro-USB Cable https://www.adafruit.com/product/2185
USB A/Micro Cable - 2m

Adafruit IO Setup

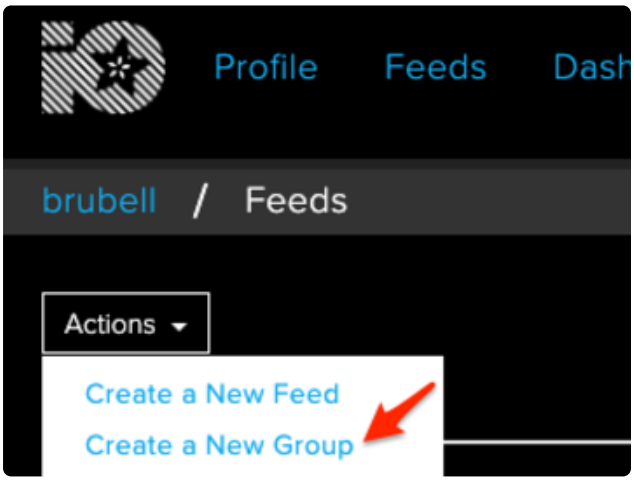
Obtain Adafruit IO Key

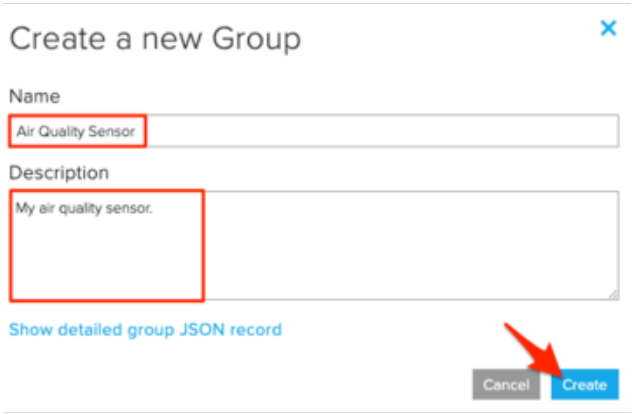
You will need your Adafruit IO username and secret API key.

Navigate to Adafruit IO (https://adafru.it/BmD) and **click the Adafruit IO Key button** to retrieve these values. Write them down in a safe place, you'll need them later.

Create Group

This guide will use multiple Adafruit IO feeds to store sensor values. To organize these feeds, you will need to create a new group.





Navigate to your Adafruit IO Feeds page (https://adafru.it/mxC).

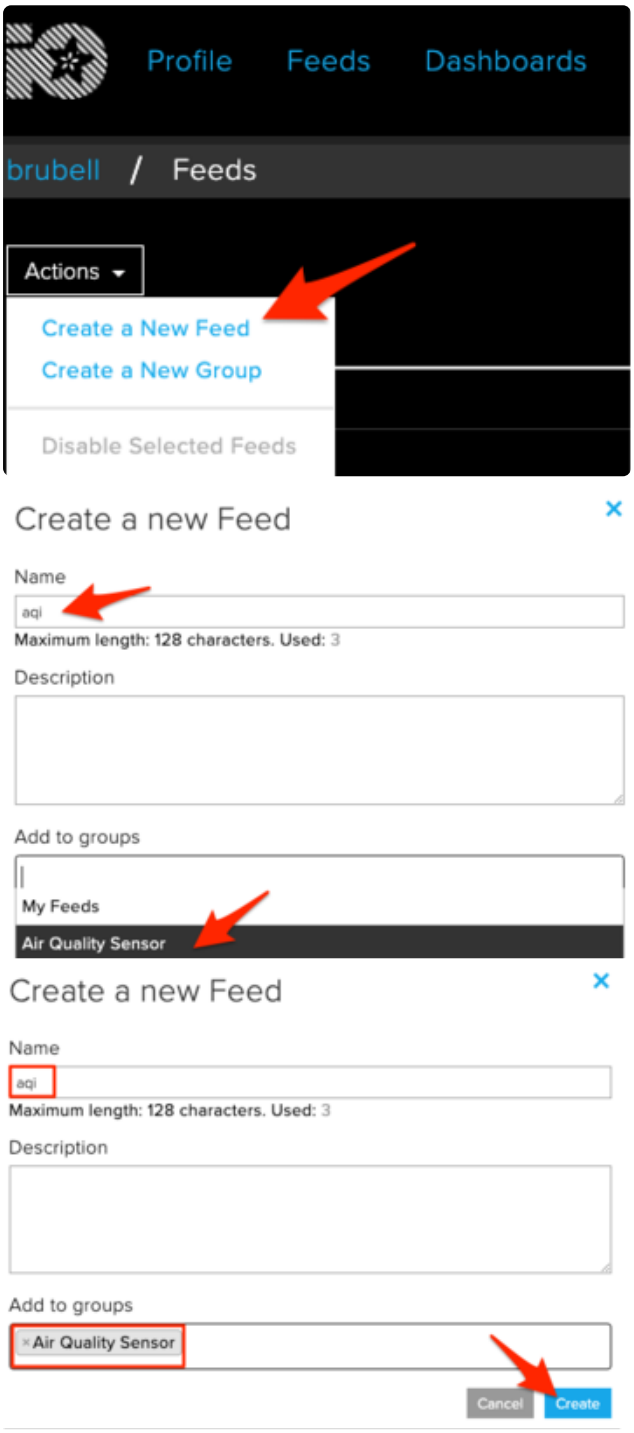
Click Actions -> Create a New Group

Name the group Air Quality Sensor. You can optionally set a description.

Click Create

Add Feeds to Group

Lets add a few feeds to the Air Quality Sensor group to hold sensor measurements and metadata.

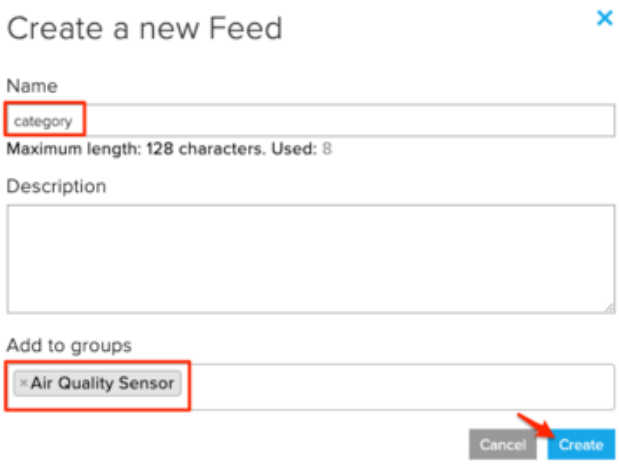


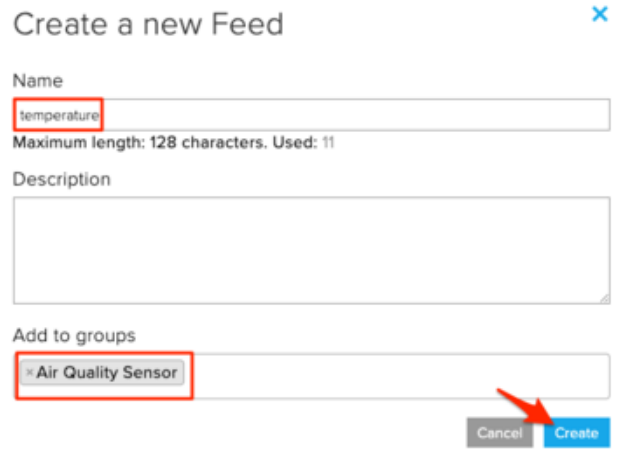
Click Actions -> Create a New Feed

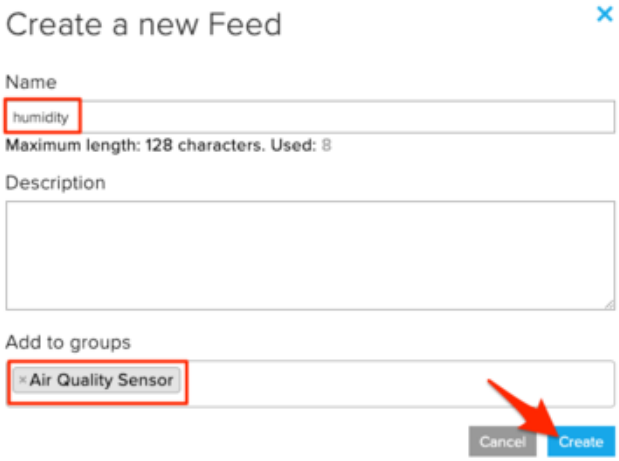
Name the feed AQI

Click Add to Groups and **select the Air Quality Sensor group**

Click Create

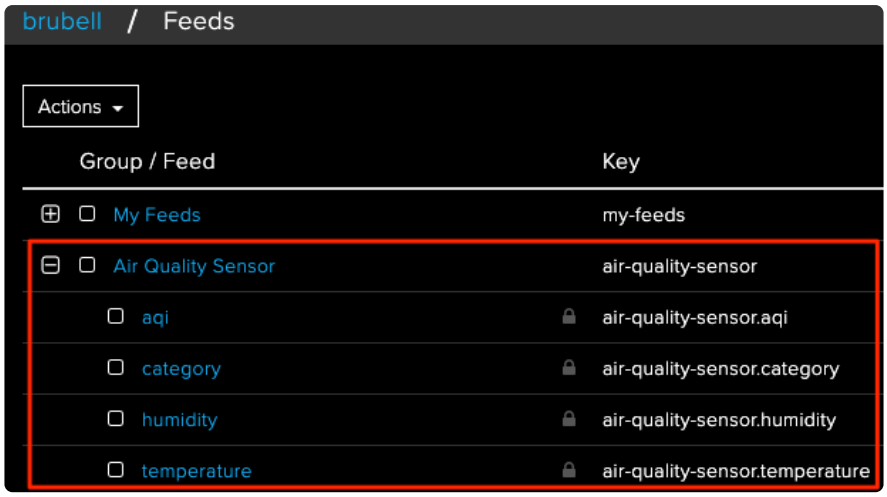






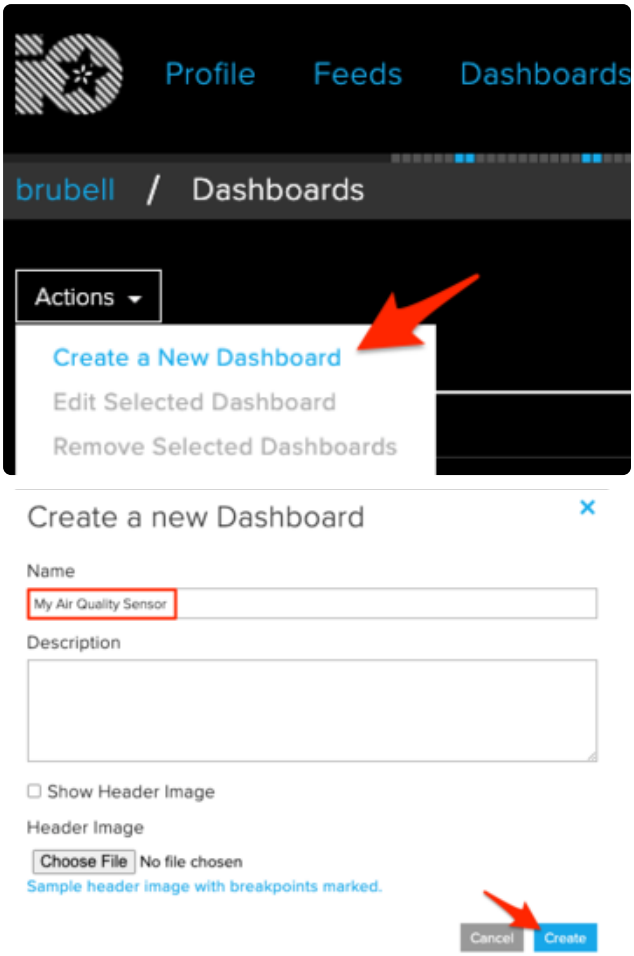
Repeat the process in the step above to **create feeds for category (AQI category), temperature, and humidity**.

Before proceeding, make sure your Air Quality Sensor group looks exactly like the screenshot below.



Adafruit IO Dashboard

Dashboards allow you to visualize data and control Adafruit IO connected projects from any modern web browser. We'll be adding gauge widgets to visualize data from the air quality sensor in real-time and charts to display data historically.



Navigate to the dashboards page on Adafruit IO (https://adafru.it/eIS).

Click Actions -> Create New Dashboard

Name the dashboard My Air Quality Sensor

Click Create

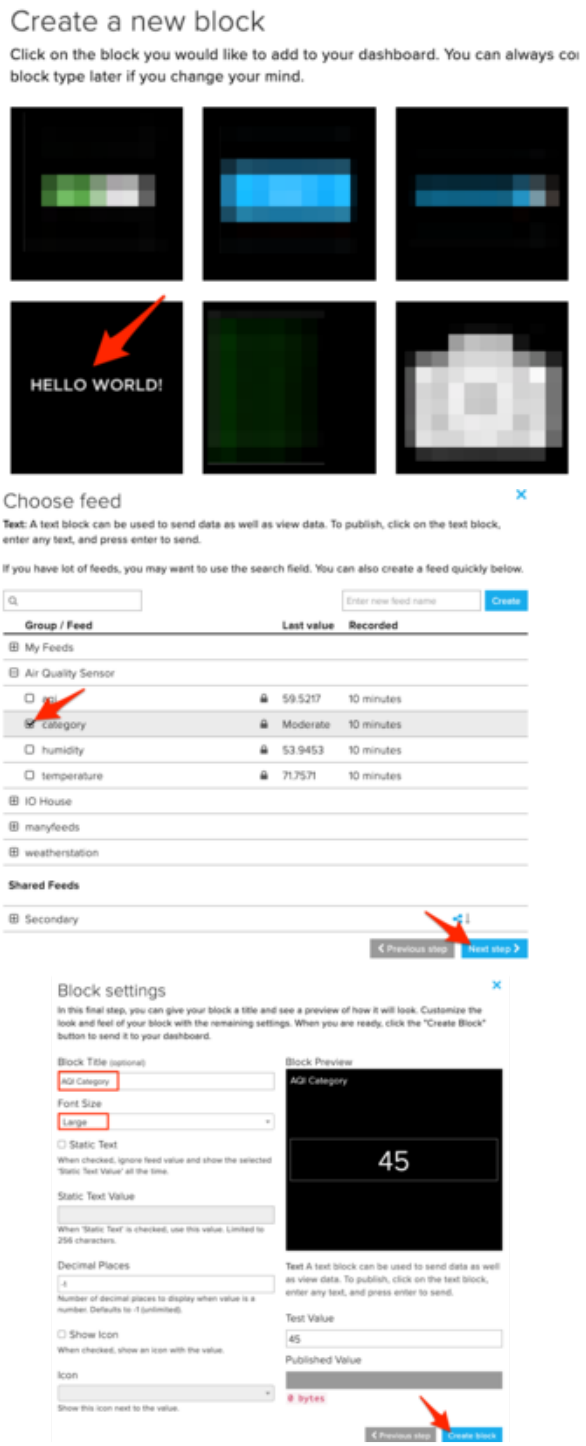
You should see your new dashboard pop-up in the list of Dashboards. **Click the My Air Quality Sensor dashboard link** to navigate to the dashboard page.

You should see an empty dashboard. Let's fill it with blocks!

Click the '+' button on your dashboard to add a new block.

Let's add a text block to display the air quality condition (Good, Acceptable, Moderate, etc) sent by the sensor.

From the Create a New Block picker, **click the Text Block**



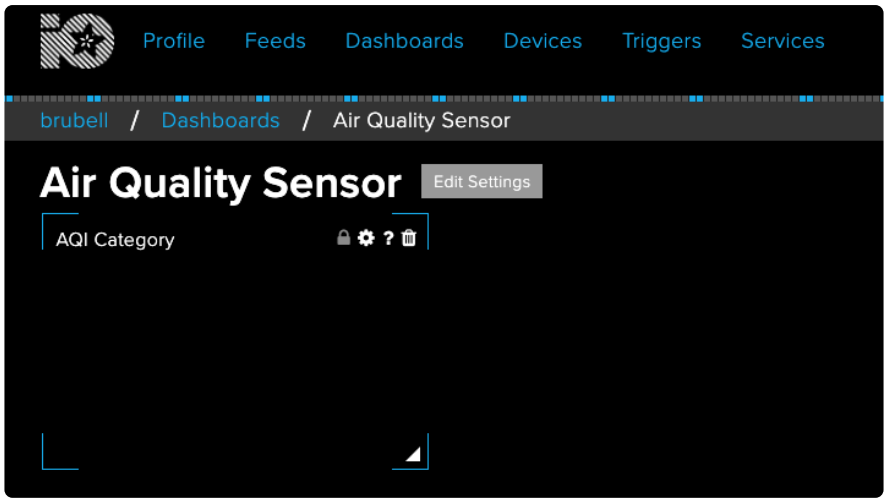
From the Create a New Block picker, **click the Text Block**

On the Choose Feed picker, **select the category feed**

Under Block Settings:

Set Block Title to AQI Category
Set Font Size to Large
Click Create Block

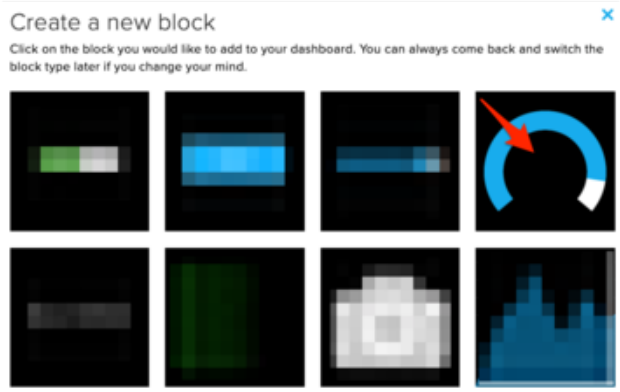
You should see the AQI Category text box appear on the dashboard. We'll organize the dashboard last - let's add the next block.



Add Gauge Block for Real-Time AQI

The United States Environmental Protection Agency uses an Air Quality Index (AQI) to communicate air quality. (https://adafru.it/O5C) While computing the AQI according to the EPA requires a 24 hour average, this gauge displays the real-time AQI.

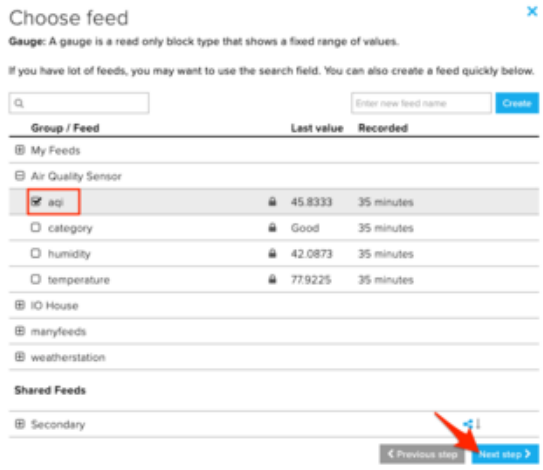
- **Note:** This guide uses air quality breakpoints and conditions developed by the USA EPA. Other countries have environmental protection agencies with similar countries with air quality indexes using PM2.5 particles. Check out this Wikipedia page for more info (https://adafru.it/O5D).



Click Create a New Block

Select the Gauge Block

Select the aqi feed



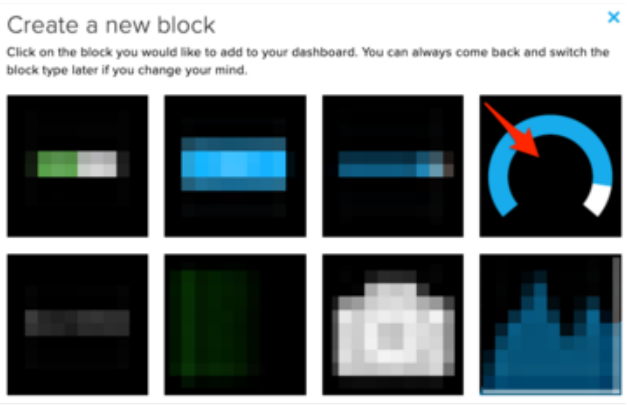
Under Block Settings:

Set Block Title to Real-Time AQI
Set the Gauge Max Value to 500
Set Gauge Width to 50px
Remove the "Value" text placeholder from Gauge Label, AQI is a unit-less value.
Set High Warning Value to 151
This AQI value corresponds with the EPA's "unhealthy" AQI category (https://adafru.it/O5D).
Set Decimal Places to **0**
Tick the Show Icon checkbox
Set Icon to **w:cloudy-windy**
Click Create Block

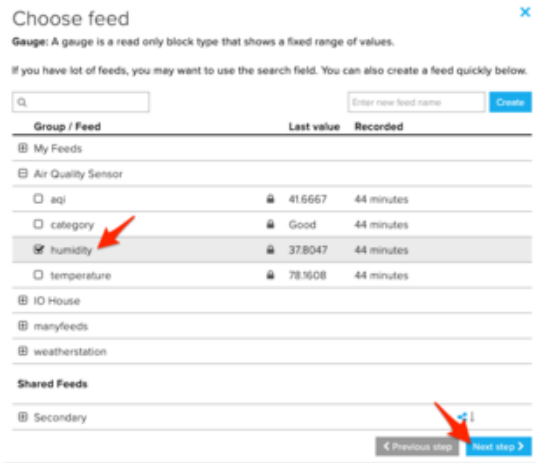
Your dashboard should now show the AQI category text block and the Real-Time AQI gauge.

Add Gauge Block for Humidity

We'll add another gauge block to display the BME280's humidity reading.



Click Create a New Block

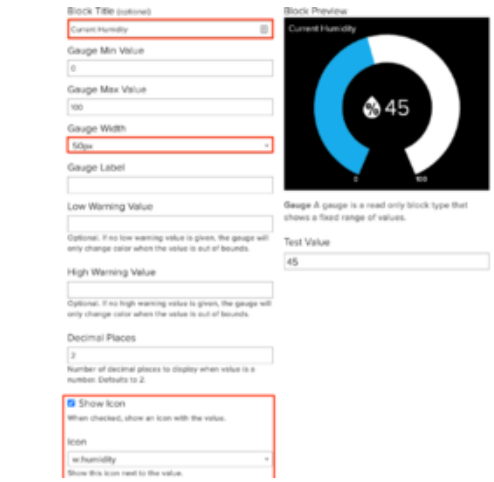


Select the Gauge Block

Select the humidity feed

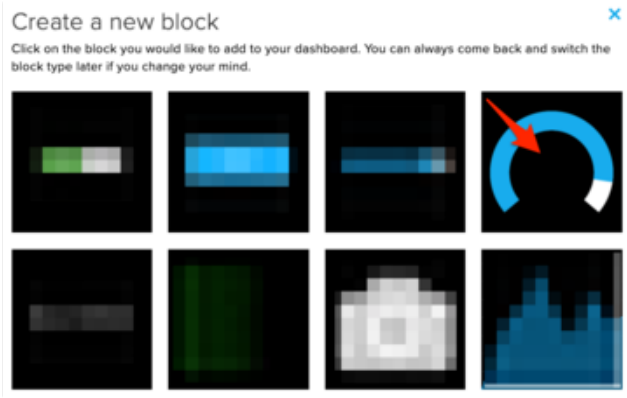
Under Block Settings:

Set Block Title to Current Humidity
Set Gauge Width to 50px
Remove the "Value" text placeholder from Gauge Label
Tick the Show Icon checkbox
Set Icon to **w:humidity**
Click Create Block



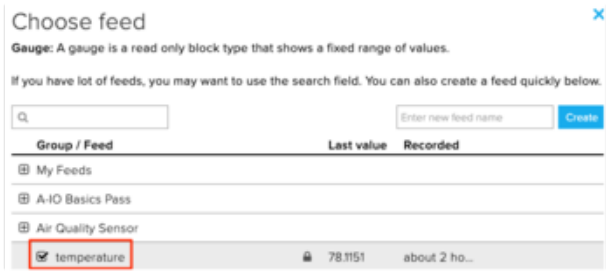
Add Gauge Block for Temperature

We'll add another gauge block to display the BME280's temperature reading.



Click Create a New Block

Select the **Gauge Block**



Select the humidity feed

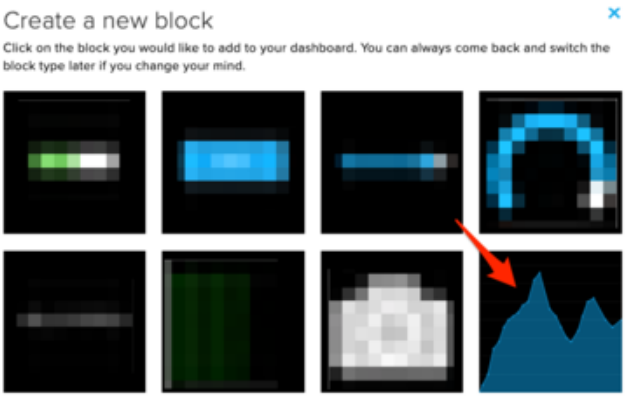
Under Block Settings:

Set Block Title to Current Temperature
Set Gauge Width to 50px
Set the Gauge label to **Degrees F or Degrees C**
Click the Show Icon checkbox
Set Icon to **thermometer**
Click Create Block

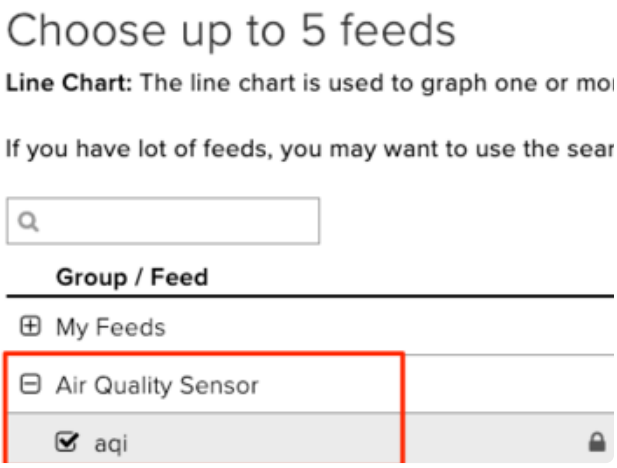
Add Line Charts

While real-time visualization of PM2.5 measurements over time is immediately useful - looking at the air quality index over time will help you understand the AQI as a more accurate average. Adafruit IO's Line Charts update dynamically whenever new values are pushed to the feed.

Since most environmental groups use a 24-hour average of AQI measurements, we'll create a new line chart block to display the AQI measurements for the past day.



Click Create a New Block



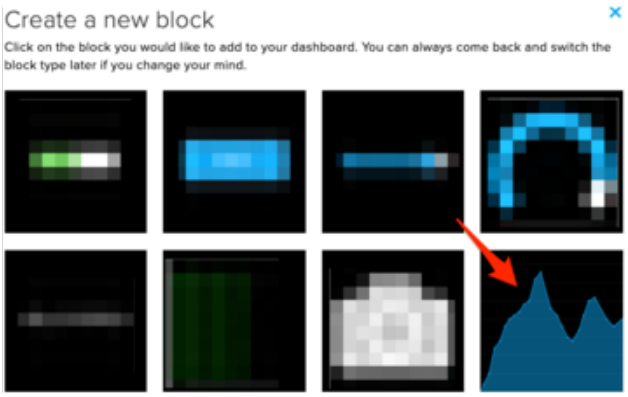
Select the **Line Chart Block**

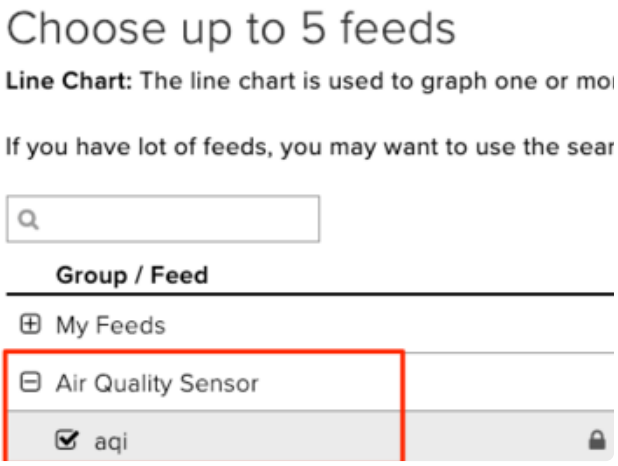
Select the aqi **feed**

Under Block Settings:

Set Block Title to **AQI - 24 Hours**
Set Show History to **24 Hours**
Click the **Draw Grid Lines** checkbox
Click Create Block

Let's make another line chart block to display the AQI from the past week.





Click Create a New Block

Select the **Line Chart Block**

Select the aqi **feed**

Under Block Settings:

Set Block Title to **AQI - 24 hours**
Set Show History to **7 Days**
Click the **Draw Grid Lines** checkbox
Click Create Block

Organize Dashboard

You can drag the dashboard blocks around to re-organize your dashboard.

Before moving on, make sure your dashboard contains the same blocks as the screenshot below

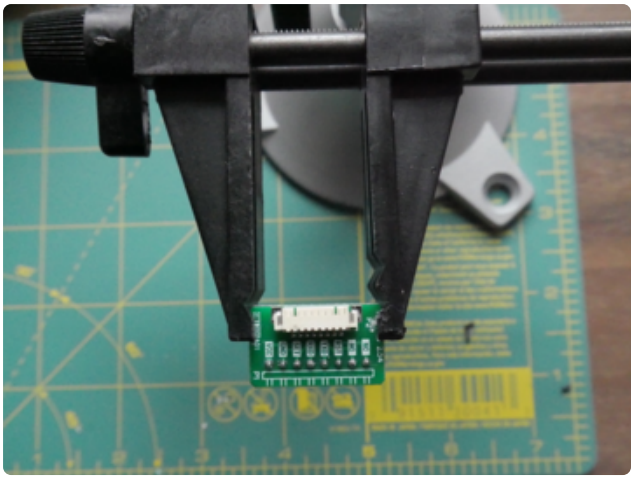
Wiring

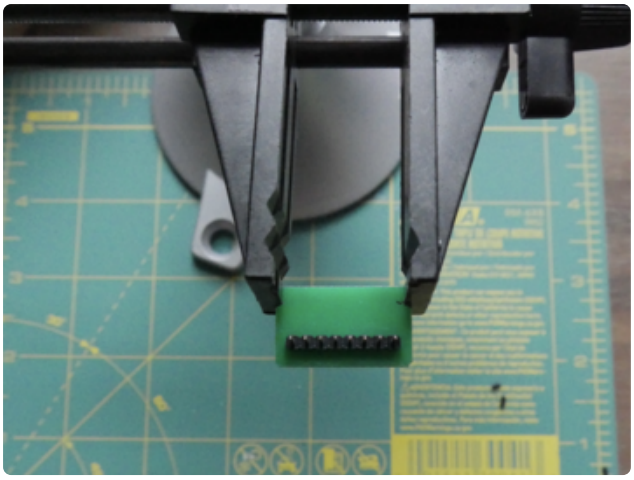
Attach Cables to BME280 and PMS5003 Adaptor

The PMS5003 comes with a breadboard adapter for you to easily hook the sensor up to a microcontroller with UART.

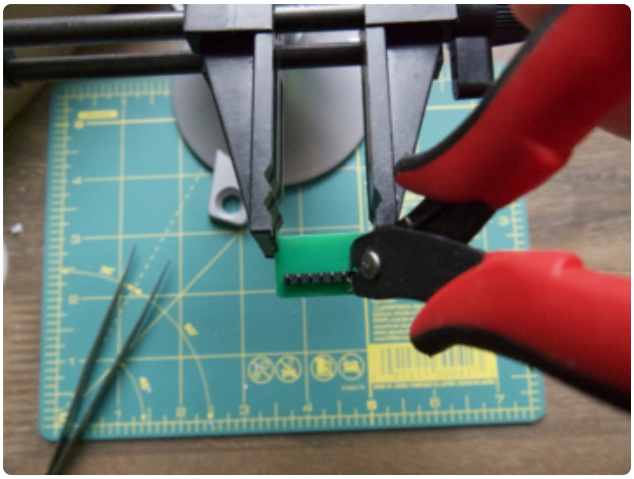
While you could use a female/female header adaptor, wires can easily disconnect. You'll want to solder cables from the FeatherWing Doubler to the breadboard adaptor for a stronger hold.

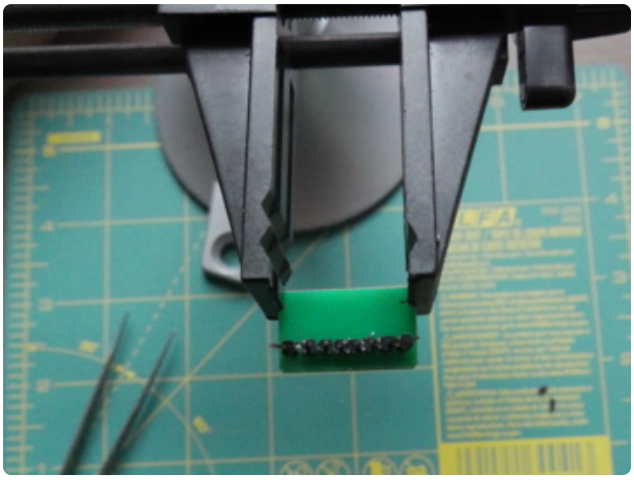
Let's begin by removing the pins from the breadboard adaptor.



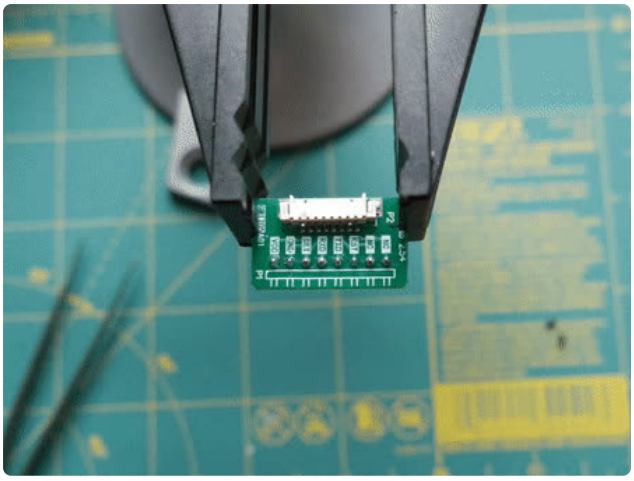


Place the breadboard adaptor into a vise (http://adafru.it/151) and flip the board over.



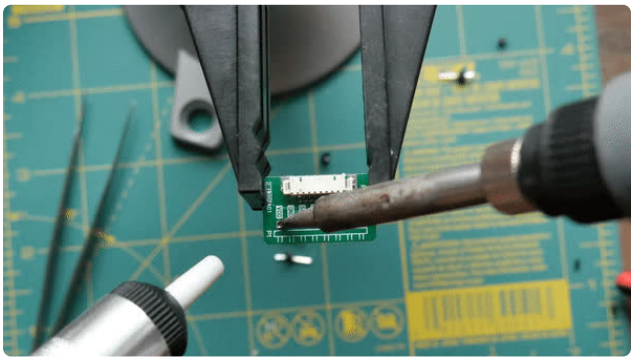


Using a pair of flush diagonal cutters (http://adafru.it/152), snip between the pins of the plastic header. Make sure to **snip the plastic between each pin**. This will make it easier to pull the pins from the adaptor when you de-solder them.

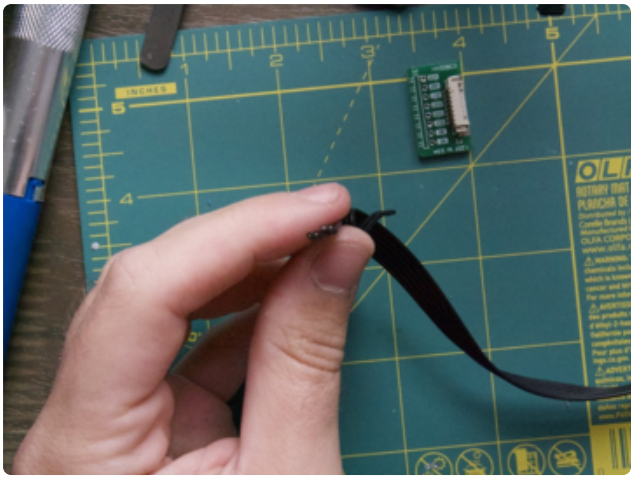


Tin the tip of your soldering iron. Then, **press the edge of your soldering iron against a solder-blob** on the adaptor to heat and liquify the solder.

While **heating the pin, use a pair of tweezers (or needle-nose pliers), wiggle the pin out of the hole in the PCB until it's removed**.



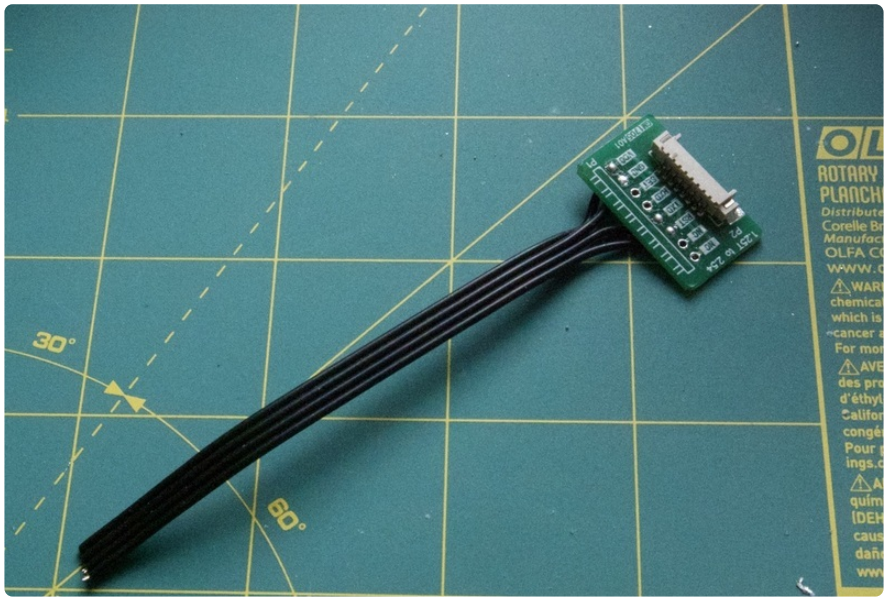
Using desoldering wick or a desoldering pump, r**emove old/excess solder from the adaptor**.



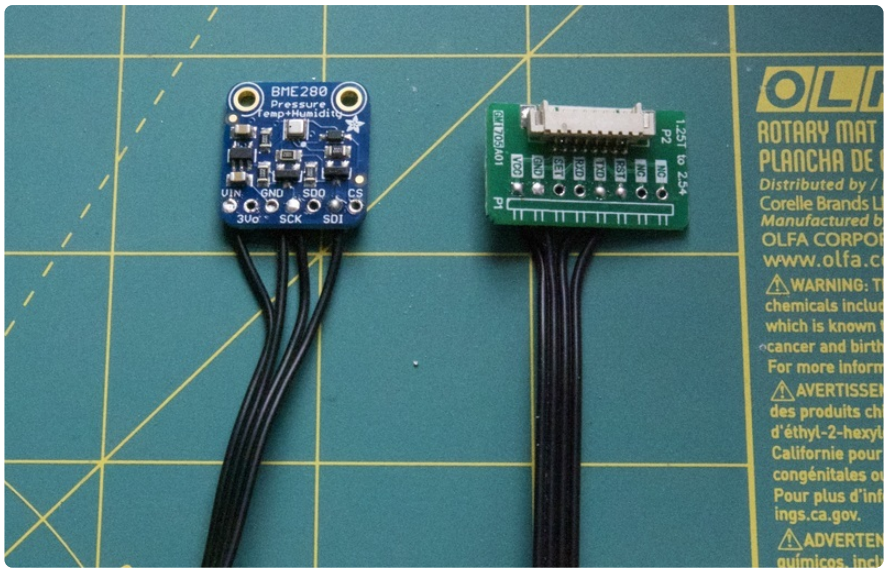
Cut a long piece of silicone stranded-core ribbon cable from the 1M of cable. It's always better to have more cable then less!

Then, **peel four cables off the silicone stranded-core ribbon cable. Strip a small amount of the silicone sheath from each wire and tin the end of it**.

Solder a length of wire to each of the VCC, GND, TXD, and RST pins on the adaptor. Your final adaptor should look like the following:

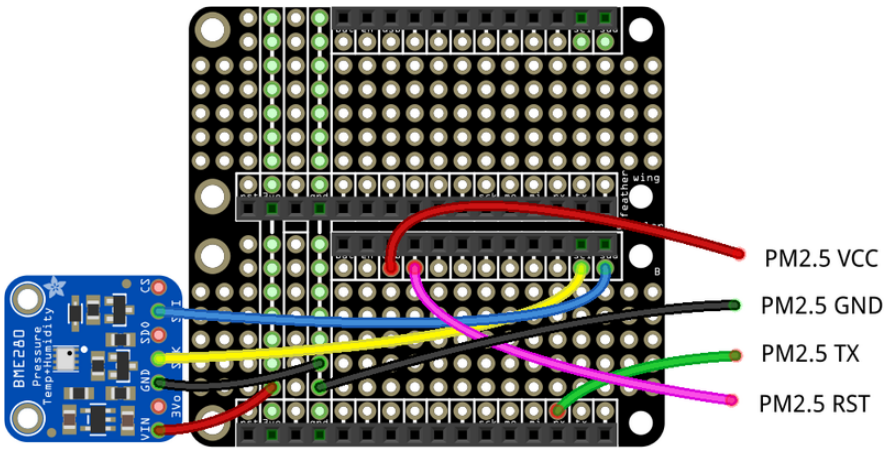


Cut a new length of silicone stranded-core ribbon cable and **repeat this process for the BME280** breakout. Both breakouts should now have cables attached.



Wiring

Next, you'll need to connect the BME280 and PM2.5 adaptor to the FeatherWing Doubler. We suggest using one of the two prototyping spots on the front of the FeatherWing Doubler.



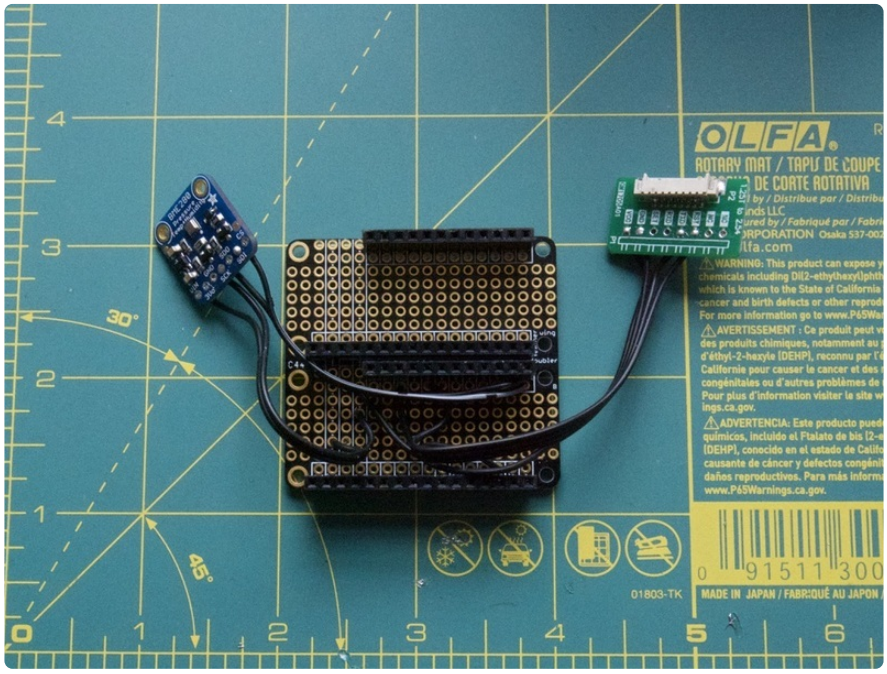
Make the following connections between the BME280 and the FeatherWing:

- **Board 3V** to **sensor VIN**
- **Board 3V** to **sensor CS**
- **Board GND** to **sensor GND**
- **Board SCL** to **sensor SCK**
- **Board SDA** to **sensor SDI**

Then, make the following connections between the PM2.5 adaptor and the FeatherWing:

- **Sensor VCC** to **board 5V**
- **Sensor GND** to **board GND**
- **Sensor TX** to **board RX**
    - Remember: **RX does not connect to RX!**

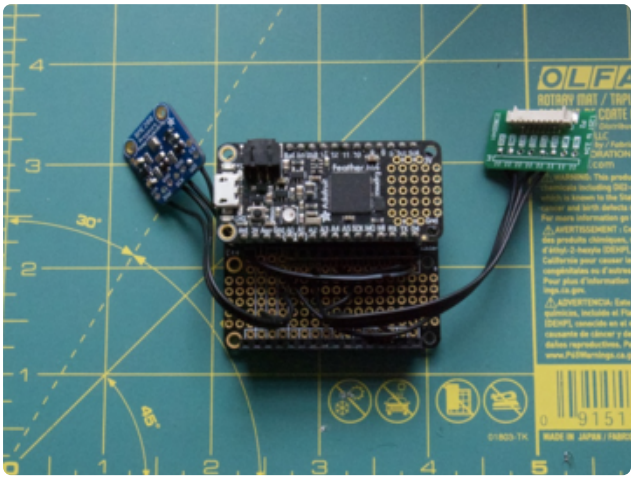
After making the connections above, your FeatherWing should look like the following.



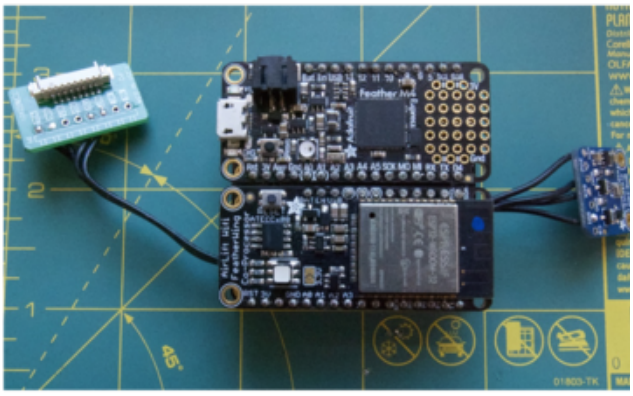
Let's move on to assembling the enclosure.

Assembly

Assemble the FeatherWing Doubler

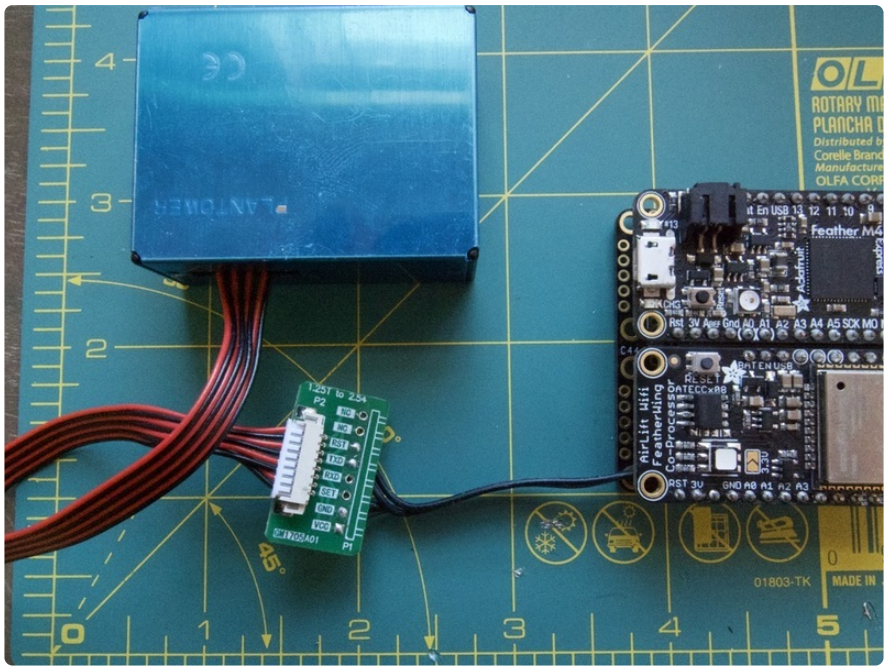


Plug the Feather M4 into the FeatherWing Doubler.



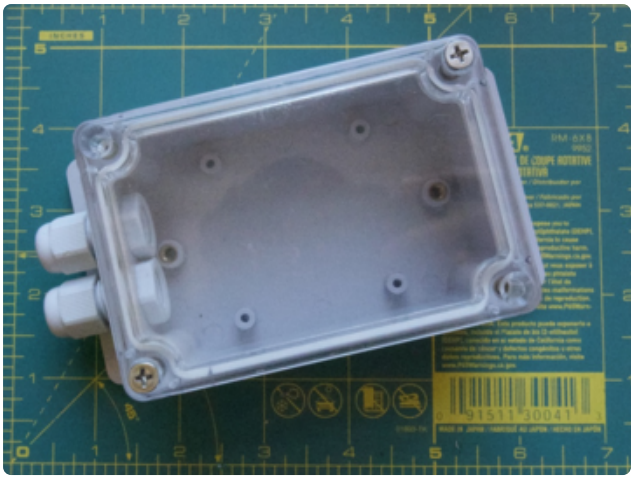
Then, plug the AirLift FeatherWing into the FeatherWing Doubler

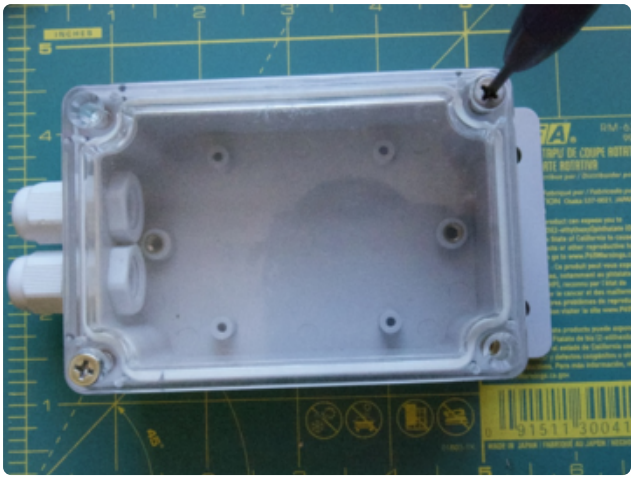
Plug the PM25 sensor into the adaptor breakout.



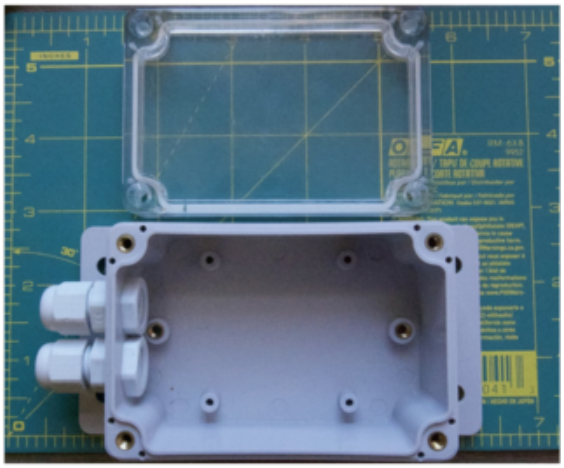
Test-Fit Enclosure

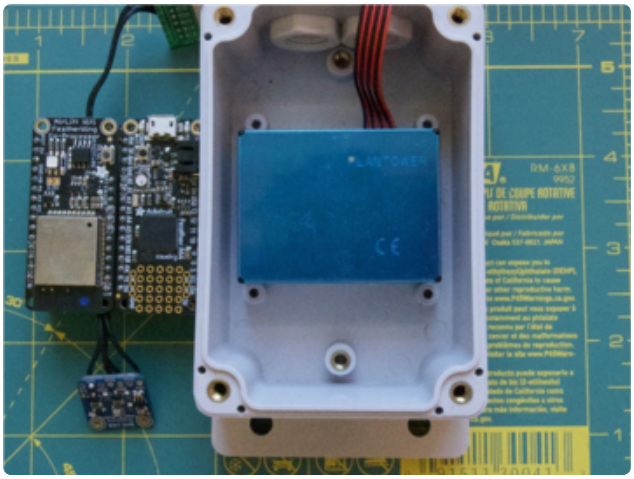
Next, you'll want to make sure the enclosure fits the hardware you assembled and check the lengths of the wires you soldered.





Using a phillips head screwdriver, open the enclosure.





Place the PM25 sensor inside the enclosure. It should fit between the four mounting standoffs.

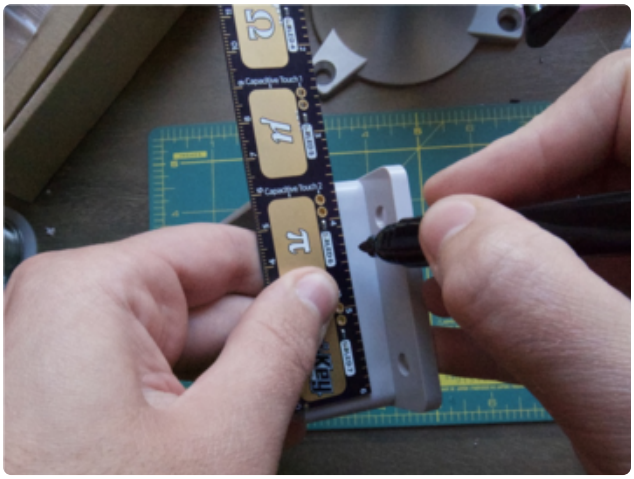
Make sure the sensor's fan faces the bottom of the enclosure and the cable faces the top of the enclosure (the side with the cable glands).

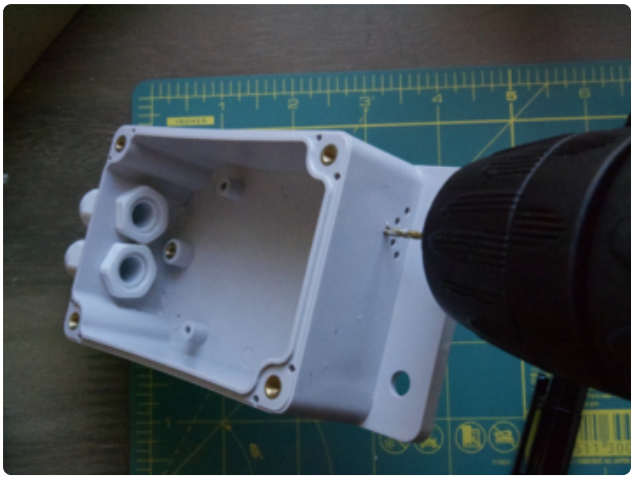
Place the FeatherWing on top of the PM25 sensor and cover it with the lid.

If you noticed the cables you cut were too long or if the solder joints broke, you'll want to fix them now.

Add Air-Holes to Enclosure

Next, you will use a handheld drill, a drill press, or a rotary tool to add some holes to the bottom of the enclosure. This will allow air particles to enter the enclosure.



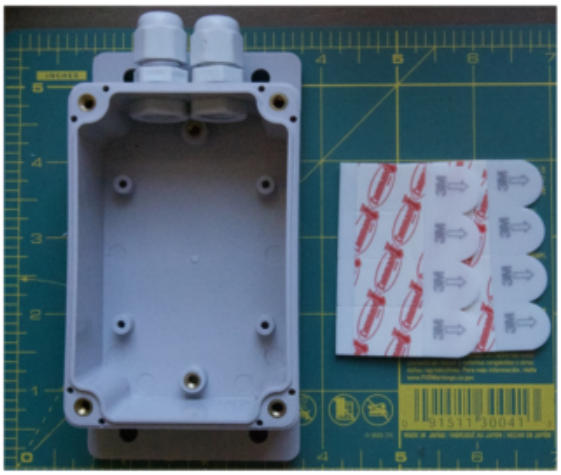


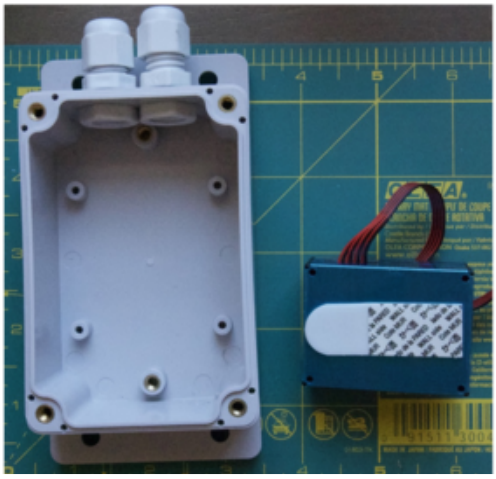
The PM2.5 sensor's fan intake is located on the bottom right hand side of the enclosure. The BME280 sensor will also be placed in the same location.

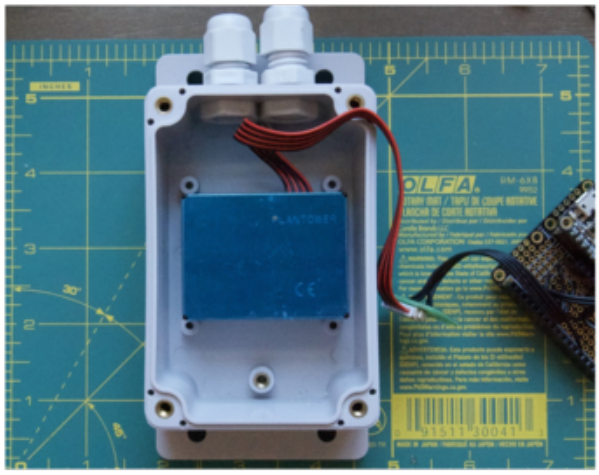
Using a marker and ruler, **mark a few spots at the bottom right of the enclosure**.

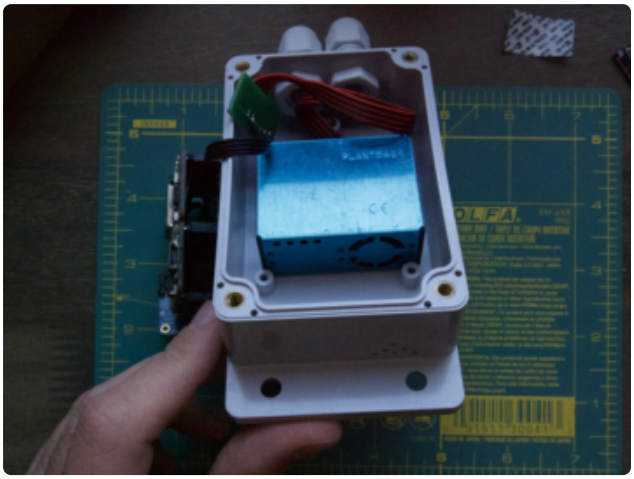
Use a drill and a small-diameter drill bit to drill holes into the enclosure. Be sure to **use a slow speed on your drill**, the enclosure's material is softer than it seems and you can accidentally drill through too quickly.

Do NOT add a filter to the bottom of the enclosure - it will prevent dust particles from being picked up by the fan.



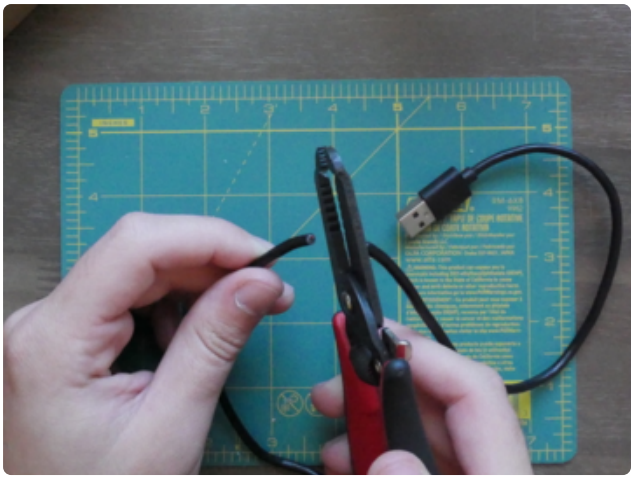


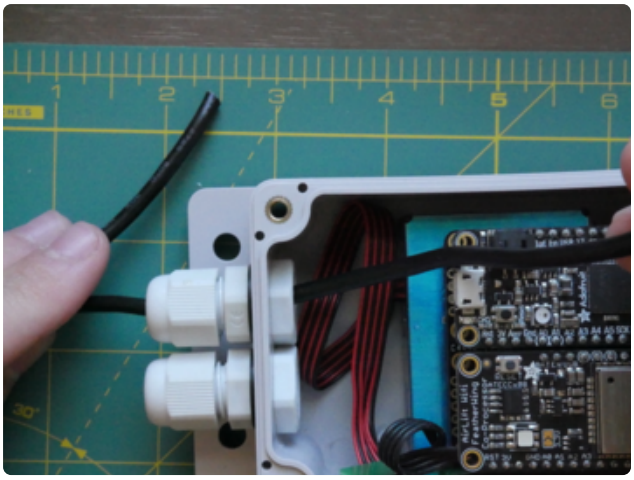


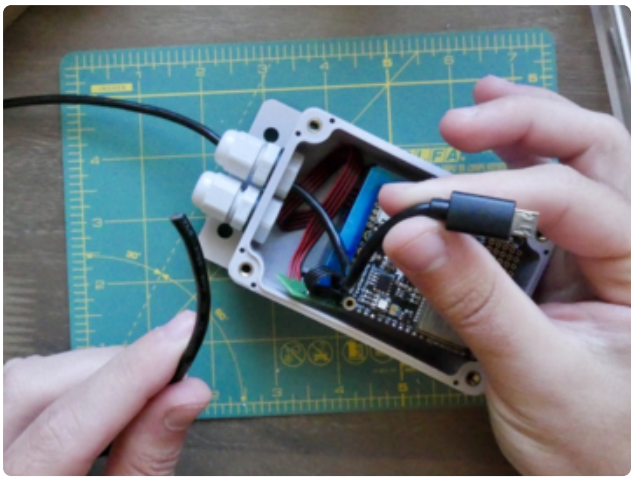


Using an adhesive of your choice - **affix the PM2.5 sensor to the bottom of the case.**

We used 3M Command Strips for a more temporary hold. If you're also using Command Strips - be sure to place the wall (black) side against the enclosure.





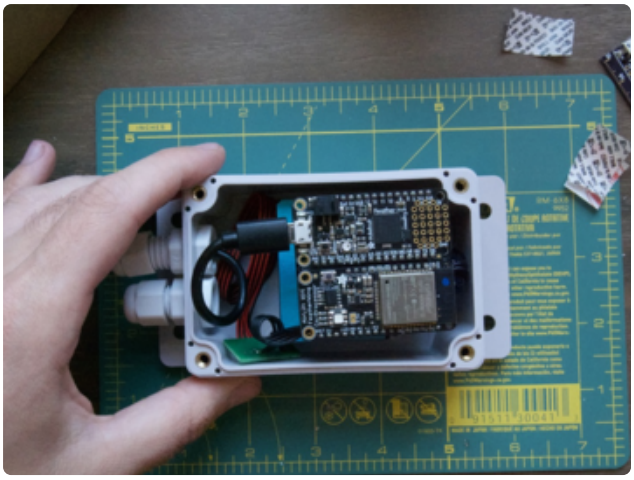


Cut a Micro-USB cable in half with a pair of wire cutters.

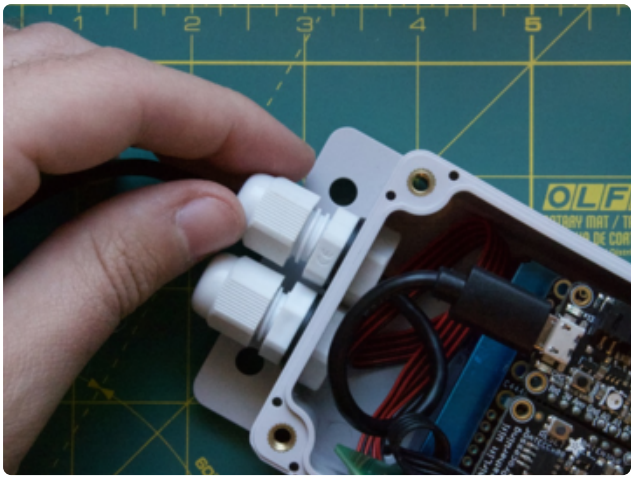
Thread the Micro-USB end through the right cable gland.

You'll need to strip and splice together the two ends of the USB cable. For more information about splicing wires, see this Learning System Guide (https://adafru.it/O5E).

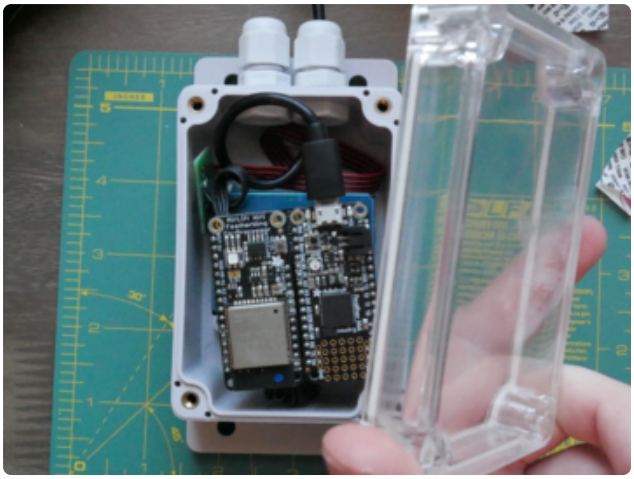
Add two adhesive strips to each side of the FeatherWing Doubler and insert the USB cable into the Feather M4.



You may need to re-organize cables at this point. We stored all of the cables in the top section in the case. **Then, press the FeatherWing Doubler onto the PM2.5 sensor**.



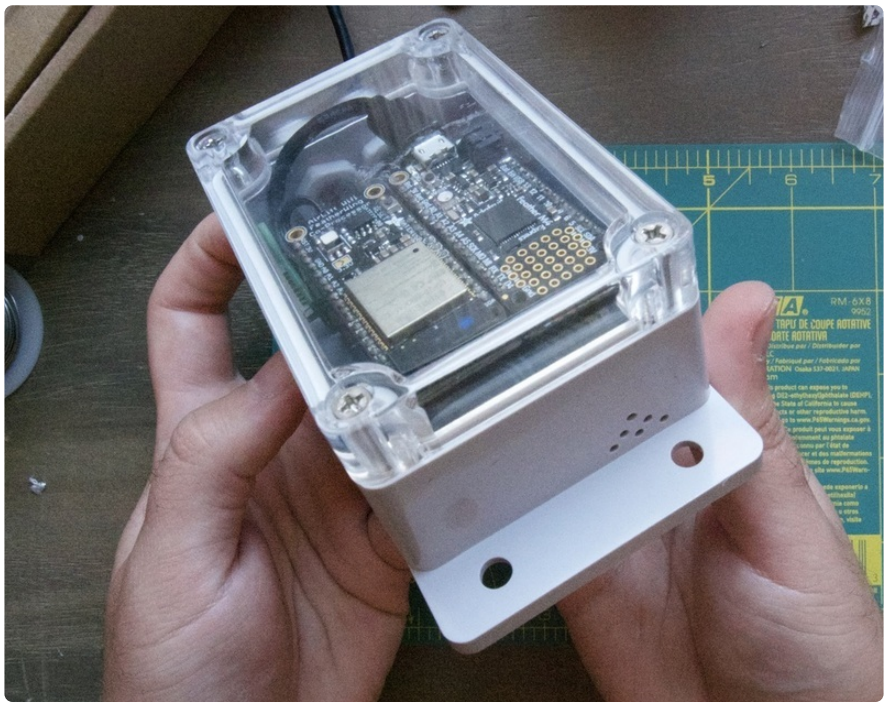
Tighten the cable gland to form a weatherproof seal around the USB cable. Make sure the left cable gland is also tightened.



Place the transparent cover over the enclosure, ensuring the gasket which runs around the enclosure is properly sealed.

Secure the cover to the enclosure using the four machine screws included with the enclosure.

That's it! Your air quality sensor is assembled and wired. The next steps will cover adding code to the Feather M4 and using the air quality sensor with Adafruit IO.



CircuitPython Setup

Install CircuitPython

Some CircuitPython compatible boards come with CircuitPython installed. Others are CircuitPython-ready, but need to have it installed. As well, you may want to update the version of CircuitPython already installed on your board. The steps are the same for installing and updating.

- To **install (or update) your CircuitPython board**, follow this page and come back here when you've successfully installed (or updated) CircuitPython. (https://adafru.it/Amd)

CircuitPython Library Installation

First make sure you are running the latest version of Adafruit CircuitPython (https://adafru.it/tBa) for your board.

Next you'll need to install the necessary libraries to use the hardware--carefully follow the steps to find and install these libraries from Adafruit's CircuitPython library bundle (https://adafru.it/zdx) matching your version of CircuitPython.

CircuitPython hardware shows up on your computer operating system as a flash drive when connected via usb. The flash drive is called **CIRCUITPY** and contains a number of files. You will need to add additional files to enable the features of this project.

First, create a folder on the drive named lib if it is not already there.

Ensure your board's **lib** folder has the following files and folders copied over. The version of the files must be the same major version as your version of CircuitPython (i.e. 5.x for 5.x, 6.x for 6.x etc.)

- **adafruit_bus_device**
- **adafruit_esp32spi**
- **adafruit_io**
- **adafruit_logging.mpy**
- **adafruit_requests.mpy**
- **adafruit_pm25**
- **neopixel.mpy**

- simpleio.mpy
- adafruit_bme280.mpy

Internet Connect!

Once you have CircuitPython setup and libraries installed we can get your board connected to the Internet.

To get connected, you will need to start by creating a secrets file.

What's a secrets file?

We expect people to share tons of projects as they build CircuitPython WiFi widgets. What we want to avoid is people accidentally sharing their passwords or secret tokens and API keys. So, we designed all our examples to use a `secrets.py` file, that is in your **CIRCUITPY** drive, to hold secret/private/custom data. That way you can share your main project without worrying about accidentally sharing private stuff.

Your **secrets.py** file should look like this:

```
# This file is where you keep secret settings, passwords, and tokens!
# If you put them in the code you risk committing that info or sharing it

secrets = {
    'ssid' : 'home ssid',
    'password' : 'my password',
    'timezone' : "America/New_York", # http://worldtimeapi.org/timezones
    'github_token' : 'fawfj23rakjnfawiefa',
    'hackaday_token' : 'h4xx0rs3kret',
    }
```

Inside is a python dictionary named secrets with a line for each entry. Each entry has an entry name (say `'ssid'`) and then a colon to separate it from the entry key `'home ssid'` and finally a comma ,

At a minimum you'll need the `ssid` and `password` for your local WiFi setup. As you make projects you may need more tokens and keys, just add them one line at a time. See for example other tokens such as one for accessing github or the hackaday API. Other non-secret data like your timezone can also go here, just cause its called secrets doesn't mean you can't have general customization data in there!

For the correct time zone string, look at http://worldtimeapi.org/timezones (https://adafru.it/EcP) and remember that if your city is not listed, look for a city in the same

time zone, for example Boston, New York, Philadelphia, Washington DC, and Miami are all on the same time as New York.

Of course, don't share your **secrets.py** - keep that out of GitHub, Discord or other project-sharing sites.

Connect to WiFi

OK now you have your secrets setup - you can connect to the Internet using the ESP32SPI and the Requests modules.

First make sure you are running the latest version of Adafruit CircuitPython (https://adafru.it/Amd) for your board.

Next you'll need to install the necessary libraries to use the hardware--carefully follow the steps to find and install these libraries from Adafruit's CircuitPython library bundle (https://adafru.it/zdx). Our introduction guide has a great page on how to install the library bundle (https://adafru.it/ABU) for both express and non-express boards.

Remember for non-express boards like the Feather M0, you'll need to manually install the necessary libraries from the bundle:

- **adafruit_bus_device**
- **adafruit_esp32_spi**
- **adafruit_requests**
- **neopixel**

Before continuing make sure your board's lib folder or root filesystem has the above files copied over.

Next connect to the board's serial REPL (https://adafru.it/Awz) so you are at the CircuitPython >>> prompt.

Into your `lib` folder. Once that's done, load up the following example using Mu or your favorite editor:

```
# SPDX-FileCopyrightText: 2019 ladyada for Adafruit Industries
# SPDX-License-Identifier: MIT

from os import getenv
import board
import busio
```

```
from digitalio import DigitalInOut
import adafruit_connection_manager
import adafruit_requests
from adafruit_esp32spi import adafruit_esp32spi

# Get wifi details and more from a settings.toml file
# tokens used by this Demo: CIRCUITPY_WIFI_SSID, CIRCUITPY_WIFI_PASSWORD
secrets = {
    "ssid": getenv("CIRCUITPY_WIFI_SSID"),
    "password": getenv("CIRCUITPY_WIFI_PASSWORD"),
}
if secrets == {"ssid": None, "password": None}:
    try:
        # Fallback on secrets.py until depreciation is over and option is removed
        from secrets import secrets
    except ImportError:
        print("WiFi secrets are kept in settings.toml, please add them there!")
        raise

print("ESP32 SPI webclient test")

TEXT_URL = "http://wifitest.adafruit.com/testwifi/index.html"
JSON_URL = "http://api.coindesk.com/v1/bpi/currentprice/USD.json"


# If you are using a board with pre-defined ESP32 Pins:
esp32_cs = DigitalInOut(board.ESP_CS)
esp32_ready = DigitalInOut(board.ESP_BUSY)
esp32_reset = DigitalInOut(board.ESP_RESET)

# If you have an AirLift Shield:
# esp32_cs = DigitalInOut(board.D10)
# esp32_ready = DigitalInOut(board.D7)
# esp32_reset = DigitalInOut(board.D5)

# If you have an AirLift Featherwing or ItsyBitsy Airlift:
# esp32_cs = DigitalInOut(board.D13)
# esp32_ready = DigitalInOut(board.D11)
# esp32_reset = DigitalInOut(board.D12)

# If you have an externally connected ESP32:
# NOTE: You may need to change the pins to reflect your wiring
# esp32_cs = DigitalInOut(board.D9)
# esp32_ready = DigitalInOut(board.D10)
# esp32_reset = DigitalInOut(board.D5)

# Secondary (SCK1) SPI used to connect to WiFi board on Arduino Nano Connect RP2040
if "SCK1" in dir(board):
    spi = busio.SPI(board.SCK1, board.MOSI1, board.MISO1)
else:
    spi = busio.SPI(board.SCK, board.MOSI, board.MISO)
esp = adafruit_esp32spi.ESP_SPIcontrol(spi, esp32_cs, esp32_ready, esp32_reset)

pool = adafruit_connection_manager.get_radio_socketpool(esp)
ssl_context = adafruit_connection_manager.get_radio_ssl_context(esp)
requests = adafruit_requests.Session(pool, ssl_context)

if esp.status == adafruit_esp32spi.WL_IDLE_STATUS:
    print("ESP32 found and in idle mode")
print("Firmware vers.", esp.firmware_version.decode("utf-8"))
print("MAC addr:", ":".join("%02X" % byte for byte in esp.MAC_address))

for ap in esp.scan_networks():
    print("\t%-23s RSSI: %d" % (str(ap["ssid"], "utf-8"), ap["rssi"]))

print("Connecting to AP...")
while not esp.is_connected:
    try:
        esp.connect_AP(secrets["ssid"], secrets["password"])
```

```
    except OSError as e:
        print("could not connect to AP, retrying: ", e)
        continue
print("Connected to", str(esp.ssid, "utf-8"), "\tRSSI:", esp.rssi)
print("My IP address is", esp.pretty_ip(esp.ip_address))
print(
    "IP lookup adafruit.com: %s" %
esp.pretty_ip(esp.get_host_by_name("adafruit.com"))
)
print("Ping google.com: %d ms" % esp.ping("google.com"))

# esp._debug = True
print("Fetching text from", TEXT_URL)
r = requests.get(TEXT_URL)
print("-" * 40)
print(r.text)
print("-" * 40)
r.close()

print()
print("Fetching json from", JSON_URL)
r = requests.get(JSON_URL)
print("-" * 40)
print(r.json())
print("-" * 40)
r.close()

print("Done!")
```

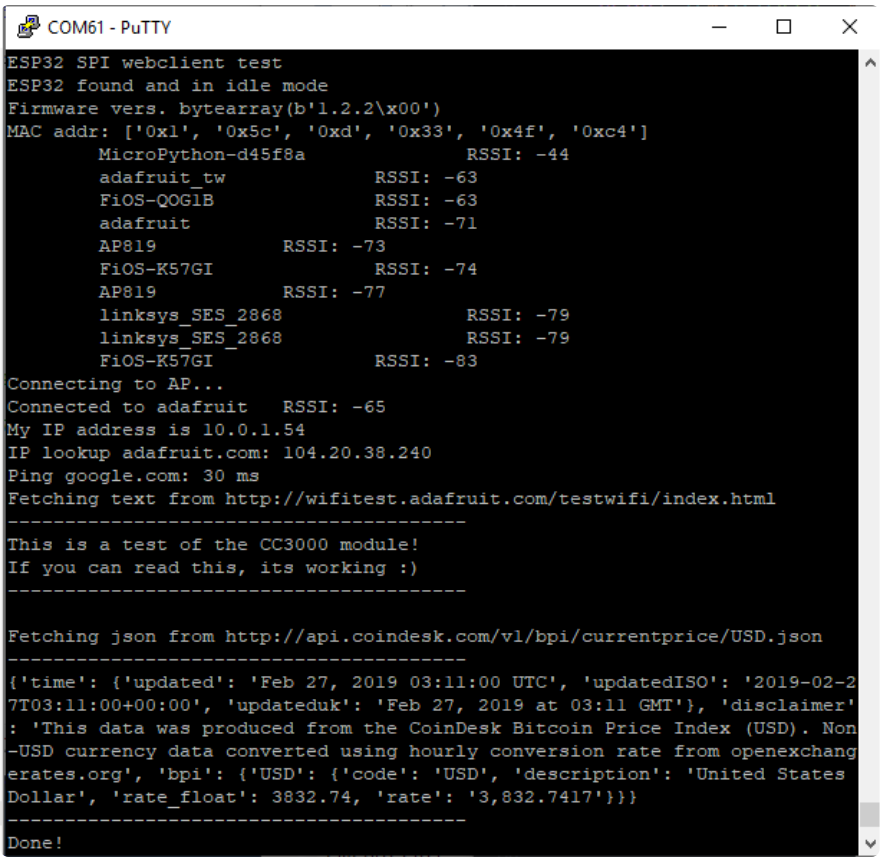
And save it to your board, with the name `code.py`.

This first connection example doesn't use a secrets file - you'll hand-enter your SSID/password to verify connectivity first!

Then go down to this line

```
esp.connect_AP(b'MY_SSID_NAME', b'MY_SSID_PASSWORD')
```

and change `MY_SSID_NAME` and `MY_SSID_PASSWORD` to your access point name and password, keeping them within the '' quotes. (This example doesn't use the secrets' file, but its also very stand-alone so if other things seem to not work you can always re-load this. You should get something like the following:



In order, the example code...

Initializes the ESP32 over SPI using the SPI port and 3 control pins:

```
esp32_cs = DigitalInOut(board.ESP_CS)
esp32_ready = DigitalInOut(board.ESP_BUSY)
esp32_reset = DigitalInOut(board.ESP_RESET)

spi = busio.SPI(board.SCK, board.MOSI, board.MISO)
esp = adafruit_esp32spi.ESP_SPIcontrol(spi, esp32_cs, esp32_ready, esp32_reset)
```

To use the AirLift FeatherWing's pins, replace the following lines into your code:

```
esp32_cs = DigitalInOut(board.D13)
esp32_ready = DigitalInOut(board.D11)
esp32_reset = DigitalInOut(board.D12)
```

Tells our `requests` library the type of socket we're using (socket type varies by connectivity type - we'll be using the `adafruit_esp32spi_socket` for this example). We'll also set the interface to an `esp` object. This is a little bit of a hack, but it lets us use `requests` like CPython does.

```
requests.set_socket(socket, esp)
```

Verifies an ESP32 is found, checks the firmware and MAC address

```
if esp.status == adafruit_esp32spi.WL_IDLE_STATUS:
    print("ESP32 found and in idle mode")
print("Firmware vers.", esp.firmware_version)
print("MAC addr:", [hex(i) for i in esp.MAC_address])
```

Performs a scan of all access points it can see and prints out the name and signal strength:

```
for ap in esp.scan_networks():
    print("\t%s\t\tRSSI: %d" % (str(ap['ssid'], 'utf-8'), ap['rssi']))
```

Connects to the AP we've defined here, then prints out the local IP address, attempts to do a domain name lookup and ping google.com to check network connectivity (note sometimes the ping fails or takes a while, this isn't a big deal)

```
    print("Connecting to AP...")
esp.connect_AP(b'MY_SSID_NAME', b'MY_SSID_PASSWORD')
print("Connected to", str(esp.ssid, 'utf-8'), "\tRSSI:", esp.rssi)
print("My IP address is", esp.pretty_ip(esp.ip_address))
print("IP lookup adafruit.com: %s" %
esp.pretty_ip(esp.get_host_by_name("adafruit.com")))
print("Ping google.com: %d ms" % esp.ping("google.com"))
```

OK now we're getting to the really interesting part. With a SAMD51 or other large-RAM (well, over 32 KB) device, we can do a lot of neat tricks. Like for example we can implement an interface a lot like requests (https://adafru.it/E9o) - which makes getting data really really easy

To read in all the text from a web URL call `requests.get` - you can pass in `https` URLs for SSL connectivity

```
TEXT_URL = "http://wifitest.adafruit.com/testwifi/index.html"
print("Fetching text from", TEXT_URL)
r = requests.get(TEXT_URL)
print('-'*40)
print(r.text)
print('-'*40)
r.close()
```

Or, if the data is in structured JSON, you can get the json pre-parsed into a Python dictionary that can be easily queried or traversed. (Again, only for nRF52840, M4 and other high-RAM boards)

```
JSON_URL = "http://api.coindesk.com/v1/bpi/currentprice/USD.json"
print("Fetching json from", JSON_URL)
r = requests.get(JSON_URL)
print('-'*40)
print(r.json())
print('-'*40)
r.close()
```

Requests

We've written a requests-like (https://adafru.it/FpT) library for web interfacing named Adafruit_CircuitPython_Requests (https://adafru.it/FpW). This library allows you to send HTTP/1.1 requests without "crafting" them and provides helpful methods for parsing the response from the server.

Here's an example of using Requests to perform GET and POST requests to a server.

```
Temporarily unable to load content:
```

The code first sets up the ESP32SPI interface. Then, it initializes a `request` object using an ESP32 `socket` and the `esp` object.

```
import board
import busio
from digitalio import DigitalInOut
import adafruit_esp32spi.adafruit_esp32spi_socket as socket
from adafruit_esp32spi import adafruit_esp32spi
import adafruit_requests as requests

# If you have an externally connected ESP32:
esp32_cs = DigitalInOut(board.D9)
esp32_ready = DigitalInOut(board.D10)
esp32_reset = DigitalInOut(board.D5)

spi = busio.SPI(board.SCK, board.MOSI, board.MISO)
esp = adafruit_esp32spi.ESP_SPIcontrol(spi, esp32_cs, esp32_ready, esp32_reset)

print("Connecting to AP...")
while not esp.is_connected:
    try:
        esp.connect_AP(b'MY_SSID_NAME', b'MY_SSID_PASSWORD')
    except RuntimeError as e:
        print("could not connect to AP, retrying: ",e)
        continue
print("Connected to", str(esp.ssid, 'utf-8'), "\tRSSI:", esp.rssi)

# Initialize a requests object with a socket and esp32spi interface
requests.set_socket(socket, esp)
```

Make sure to set the ESP32 pinout to match your AirLift breakout's connection:

```
esp32_cs = DigitalInOut(board.D9)
esp32_ready = DigitalInOut(board.D10)
esp32_reset = DigitalInOut(board.D5)
```

HTTP GET with Requests

The code makes a HTTP GET request to Adafruit's WiFi testing website - http://wifitest.adafruit.com/testwifi/index.html (https://adafru.it/Fp-).

To do this, we'll pass the URL into `requests.get()`. We're also going to save the response from the server into a variable named `response`.

While we requested data from the server, we'd what the server responded with. Since we already saved the server's `response`, we can read it back. Luckily for us, **requests automatically decodes the server's response into human-readable text,** you can read it back by calling `response.text`.

Lastly, we'll perform a bit of cleanup by calling `response.close()`. This closes, deletes, and collect's the response's data.

```
    print("Fetching text from %s"%TEXT_URL)
response = requests.get(TEXT_URL)
print('-'*40)

print("Text Response: ", response.text)
print('-'*40)
response.close()
```

While some servers respond with text, some respond with json-formatted data consisting of attribute–value pairs.

CircuitPython_Requests can convert a JSON-formatted response from a server into a CPython `dict.` object.

We can also fetch and parse **json** data. We'll send a HTTP get to a url we know returns a json-formatted response (instead of text data).

Then, the code calls `response.json()` to convert the response to a CPython `dict`.

```
    print("Fetching JSON data from %s"%JSON_GET_URL)
response = requests.get(JSON_GET_URL)
print('-'*40)

print("JSON Response: ", response.json())
print('-'*40)
response.close()
```

HTTP POST with Requests

Requests can also **POST data to a server by calling the `requests.post` method, passing it a `data` value.**

```
data = '31F'
print("POSTing data to {0}: {1}".format(JSON_POST_URL, data))
response = requests.post(JSON_POST_URL, data=data)
print('-'*40)

json_resp = response.json()
# Parse out the 'data' key from json_resp dict.
print("Data received from server:", json_resp['data'])
print('-'*40)
response.close()
```

You can also post json-formatted data to a server by passing `json` data into the `requests.post` method.

```
json_data = {"Date" : "July 25, 2019"}
print("POSTing data to {0}: {1}".format(JSON_POST_URL, json_data))
response = requests.post(JSON_POST_URL, json=json_data)
print('-'*40)

json_resp = response.json()
# Parse out the 'json' key from json_resp dict.
print("JSON Data received from server:", json_resp['json'])
print('-'*40)
response.close()
```

Advanced Requests Usage

Want to send custom HTTP headers, parse the response as raw bytes, or handle a response's http status code in your CircuitPython code?

We've written an example to show advanced usage of the requests module below.

```
Temporarily unable to load content:
```

WiFi Manager

That simpletest example works but its a little finicky - you need to constantly check WiFi status and have many loops to manage connections and disconnections. For more advanced uses, we recommend using the WiFiManager object. It will wrap the connection/status/requests loop for you - reconnecting if WiFi drops, resetting the ESP32 if it gets into a bad state, etc.

Here's a more advanced example that shows the WiFi manager and also how to POST data with some extra headers:

```
# SPDX-FileCopyrightText: 2019 ladyada for Adafruit Industries
# SPDX-License-Identifier: MIT

import time
```

```
from os import getenv
import board
import busio
from digitalio import DigitalInOut
import neopixel
from adafruit_esp32spi import adafruit_esp32spi
from adafruit_esp32spi import adafruit_esp32spi_wifimanager

print("ESP32 SPI webclient test")

# Get wifi details and more from a settings.toml file
# tokens used by this Demo: CIRCUITPY_WIFI_SSID, CIRCUITPY_WIFI_PASSWORD
#                           CIRCUITPY_AIO_USERNAME, CIRCUITPY_AIO_KEY
secrets = {}
for token in ["ssid", "password"]:
    if getenv("CIRCUITPY_WIFI_" + token.upper()):
        secrets[token] = getenv("CIRCUITPY_WIFI_" + token.upper())
for token in ["aio_username", "aio_key"]:
    if getenv("CIRCUITPY_" + token.upper()):
        secrets[token] = getenv("CIRCUITPY_" + token.upper())

if not secrets:
    try:
        # Fallback on secrets.py until depreciation is over and option is removed
        from secrets import secrets
    except ImportError:
        print("WiFi secrets are kept in settings.toml, please add them there!")
        raise

# If you are using a board with pre-defined ESP32 Pins:
esp32_cs = DigitalInOut(board.ESP_CS)
esp32_ready = DigitalInOut(board.ESP_BUSY)
esp32_reset = DigitalInOut(board.ESP_RESET)

# If you have an externally connected ESP32:
# esp32_cs = DigitalInOut(board.D9)
# esp32_ready = DigitalInOut(board.D10)
# esp32_reset = DigitalInOut(board.D5)

# Secondary (SCK1) SPI used to connect to WiFi board on Arduino Nano Connect RP2040
if "SCK1" in dir(board):
    spi = busio.SPI(board.SCK1, board.MOSI1, board.MISO1)
else:
    spi = busio.SPI(board.SCK, board.MOSI, board.MISO)
esp = adafruit_esp32spi.ESP_SPIcontrol(spi, esp32_cs, esp32_ready, esp32_reset)
"""Use below for Most Boards"""
status_light = neopixel.NeoPixel(board.NEOPIXEL, 1, brightness=0.2)
"""Uncomment below for ItsyBitsy M4"""
# status_light = dotstar.DotStar(board.APA102_SCK, board.APA102_MOSI, 1,
brightness=0.2)
"""Uncomment below for an externally defined RGB LED (including Arduino Nano
Connect)"""
# import adafruit_rgbled
# from adafruit_esp32spi import PWMOut
# RED_LED = PWMOut.PWMOut(esp, 26)
# GREEN_LED = PWMOut.PWMOut(esp, 27)
# BLUE_LED = PWMOut.PWMOut(esp, 25)
# status_light = adafruit_rgbled.RGBLED(RED_LED, BLUE_LED, GREEN_LED)

wifi = adafruit_esp32spi_wifimanager.ESPSPI_WiFiManager(esp, secrets, status_light)

counter = 0

while True:
    try:
        print("Posting data...", end="")
        data = counter
        feed = "test"
        payload = {"value": data}
```

```
        response = wifi.post(
            "https://io.adafruit.com/api/v2/"
            + secrets["aio_username"]
            + "/feeds/"
            + feed
            + "/data",
            json=payload,
            headers={"X-AIO-KEY": secrets["aio_key"]},
        )
        print(response.json())
        response.close()
        counter = counter + 1
        print("OK")
    except OSError as e:
        print("Failed to get data, retrying\n", e)
        wifi.reset()
        continue
    response = None
    time.sleep(15)
```

Next, **set up an Adafruit IO feed named `test`**

- If you do not know how to set up a feed, follow this page and come back when you've set up a feed named `test`. (https://adafru.it/f5k)

You'll note here we use a secrets.py file to manage our SSID info. The wifimanager is given the ESP32 object, secrets and a neopixel for status indication.

Note, you'll need to add a some additional information to your secrets file so that the code can query the Adafruit IO API:

- `aio_username`
- `aio_key`

You can go to your adafruit.io View AIO Key link to get those two values and add them to the secrets file, which will now look something like this:

```
# This file is where you keep secret settings, passwords, and tokens!
# If you put them in the code you risk committing that info or sharing it

secrets = {
    'ssid' : '_your_ssid_',
    'password' : '_your_wifi_password_',
    'timezone' : "America/Los_Angeles", # http://worldtimeapi.org/timezones
    'aio_username' : '_your_aio_username_',
    'aio_key' : '_your_aio_key_',
    }
```

We can then have a simple loop for posting data to Adafruit IO without having to deal with connecting or initializing the hardware!

Take a look at your **test** feed on Adafruit.io and you'll see the value increase each time the CircuitPython board posts data to it!

Code Usage

Text Editor

Adafruit recommends using the Mu editor for editing your CircuitPython code. You can get more info in this guide (https://adafru.it/ANO).

Alternatively, you can use any text editor that saves simple text files.

Secrets File Setup

Open the **secrets.py** file on your CircuitPython device using Mu or your favorite text editor. You're going to edit this file to enter your WiFi credentials along with your keys.

- **Change** `ssid` to **the name of your WiFi network**
- **Change** `password` to **your WiFi network's password**
- **Change** `aio_user` to **your Adafruit IO Username**

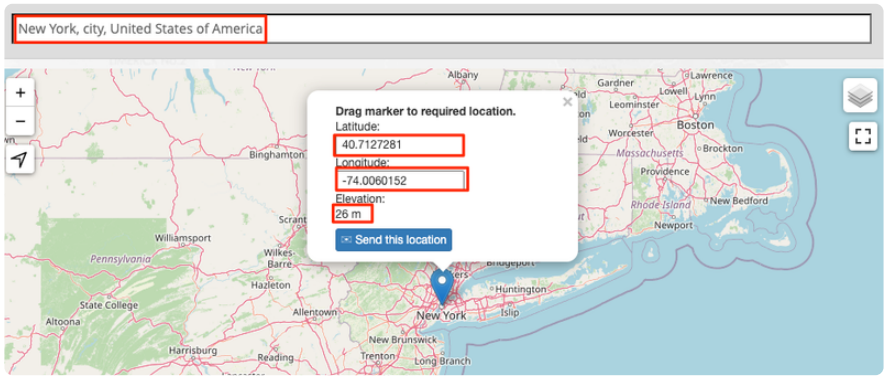
- **Change `aio_key` to your Adafruit IO Key.**

```
secrets = {
    'ssid' : 'home ssid',
    'password' : 'my password',
    'timezone' : "America/New_York", # http://worldtimeapi.org/timezones
    'aio_username' : 'MY_ADAFRUIT_IO_USERNAME',
    'aio_key' : 'MY_ADAFRUIT_IO_KEY',
    'latitude': MY_LAT, # https://www.latlong.net/
    'longitude': MY_LON,
    'elevation': MY_ELE
    }
```

Next, let's to add your location's latitude, longitude and altitude data to the secrets file. Entering your location will allow the Map Block to show an image of your sensor's location.

For privacy reasons, we suggest limiting your location data to your city, town, or municipality. Instead of setting our sensor's location to Adafruit's exact address, we'll set it to New York City.

Navigate to this website to find your location's GPS latitude, longitude and altitude coordinates (https://adafru.it/O9E) and enter your city/town.



In the secrets file, change `MY_LAT`, `MY_LON`, and `MY_ELE` to the values obtained from the website above.

Code

Click the Download: Project Zip File link below in the code window to get a zip file with all the files needed for the project. Copy **code.py** from the zip file and place on the **CIRCUITPY** drive.

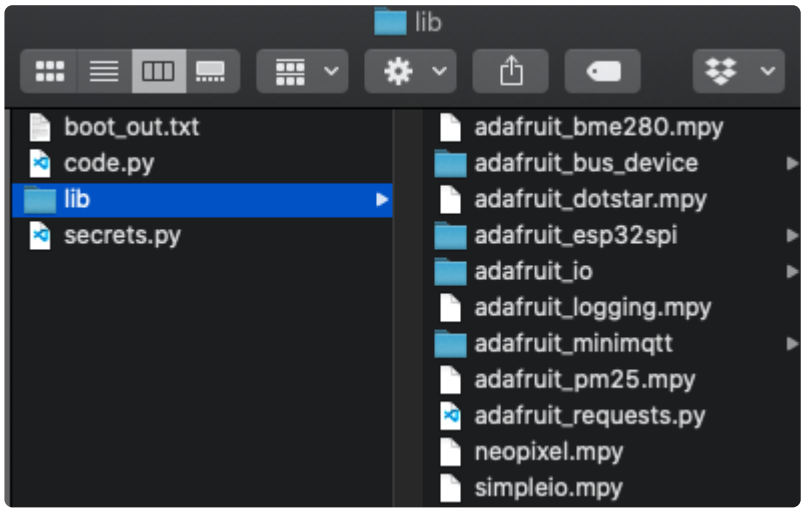
```
# SPDX-FileCopyrightText: 2020 Brent Rubell for Adafruit Industries
#
# SPDX-License-Identifier: MIT

import time
```

```
import board
import busio
from digitalio import DigitalInOut
import neopixel
from adafruit_esp32spi import adafruit_esp32spi, adafruit_esp32spi_wifimanager
from adafruit_io.adafruit_io import IO_HTTP
from simpleio import map_range
from adafruit_pm25.uart import PM25_UART

# Uncomment below for PMSA003I Air Quality Breakout
# from adafruit_pm25.i2c import PM25_I2C
import adafruit_bme280

### Configure Sensor ###
# Return environmental sensor readings in degrees Celsius
USE_CELSIUS = False
# Interval the sensor publishes to Adafruit IO, in minutes
PUBLISH_INTERVAL = 10

### WiFi ###

# Get wifi details and more from a secrets.py file
try:
    from secrets import secrets
except ImportError:
    print("WiFi secrets are kept in secrets.py, please add them there!")
    raise

# AirLift FeatherWing
esp32_cs = DigitalInOut(board.D13)
esp32_reset = DigitalInOut(board.D12)
esp32_ready = DigitalInOut(board.D11)

spi = busio.SPI(board.SCK, board.MOSI, board.MISO)
esp = adafruit_esp32spi.ESP_SPIcontrol(spi, esp32_cs, esp32_ready, esp32_reset)
status_light = neopixel.NeoPixel(board.NEOPIXEL, 1, brightness=0.2)
wifi = adafruit_esp32spi_wifimanager.ESPSPI_WiFiManager(esp, secrets, status_light)

# Connect to a PM2.5 sensor over UART
reset_pin = None
uart = busio.UART(board.TX, board.RX, baudrate=9600)
pm25 = PM25_UART(uart, reset_pin)

# Create i2c object
i2c = busio.I2C(board.SCL, board.SDA, frequency=100000)

# Connect to a BME280 over I2C
bme_sensor = adafruit_bme280.Adafruit_BME280_I2C(i2c)

# Uncomment below for PMSA003I Air Quality Breakout
# pm25 = PM25_I2C(i2c, reset_pin)

# Uncomment below for BME680
# import adafruit_bme680
# bme_sensor = adafruit_bme680.Adafruit_BME680_I2C(i2c)

### Sensor Functions ###
def calculate_aqi(pm_sensor_reading):
    """Returns a calculated air quality index (AQI)
    and category as a tuple.
    NOTE: The AQI returned by this function should ideally be measured
    using the 24-hour concentration average. Calculating a AQI without
    averaging will result in higher AQI values than expected.
    :param float pm_sensor_reading: Particulate matter sensor value.

    """
    # Check sensor reading using EPA breakpoint (Clow-Chigh)
    if 0.0 <= pm_sensor_reading <= 12.0:
        # AQI calculation using EPA breakpoints (Ilow-IHigh)
```

```
        aqi_val = map_range(int(pm_sensor_reading), 0, 12, 0, 50)
        aqi_cat = "Good"
    elif 12.1 <= pm_sensor_reading <= 35.4:
        aqi_val = map_range(int(pm_sensor_reading), 12, 35, 51, 100)
        aqi_cat = "Moderate"
    elif 35.5 <= pm_sensor_reading <= 55.4:
        aqi_val = map_range(int(pm_sensor_reading), 36, 55, 101, 150)
        aqi_cat = "Unhealthy for Sensitive Groups"
    elif 55.5 <= pm_sensor_reading <= 150.4:
        aqi_val = map_range(int(pm_sensor_reading), 56, 150, 151, 200)
        aqi_cat = "Unhealthy"
    elif 150.5 <= pm_sensor_reading <= 250.4:
        aqi_val = map_range(int(pm_sensor_reading), 151, 250, 201, 300)
        aqi_cat = "Very Unhealthy"
    elif 250.5 <= pm_sensor_reading <= 350.4:
        aqi_val = map_range(int(pm_sensor_reading), 251, 350, 301, 400)
        aqi_cat = "Hazardous"
    elif 350.5 <= pm_sensor_reading <= 500.4:
        aqi_val = map_range(int(pm_sensor_reading), 351, 500, 401, 500)
        aqi_cat = "Hazardous"
    else:
        print("Invalid PM2.5 concentration")
        aqi_val = -1
        aqi_cat = None
    return aqi_val, aqi_cat


def sample_aq_sensor():
    """Samples PM2.5 sensor
    over a 2.3 second sample rate.

    """
    aq_reading = 0
    aq_samples = []

    # initial timestamp
    time_start = time.monotonic()
    # sample pm2.5 sensor over 2.3 sec sample rate
    while time.monotonic() - time_start <= 2.3:
        try:
            aqdata = pm25.read()
            aq_samples.append(aqdata["pm25 env"])
        except RuntimeError:
            print("Unable to read from sensor, retrying...")
            continue
        # pm sensor output rate of 1s
        time.sleep(1)
    # average sample reading / # samples
    for sample in range(len(aq_samples)):
        aq_reading += aq_samples[sample]
    aq_reading = aq_reading / len(aq_samples)
    aq_samples.clear()
    return aq_reading


def read_bme(is_celsius=False):
    """Returns temperature and humidity
    from BME280/BME680 environmental sensor, as a tuple.

    :param bool is_celsius: Returns temperature in degrees celsius
                            if True, otherwise fahrenheit.
    """
    humid = bme_sensor.humidity
    temp = bme_sensor.temperature
    if not is_celsius:
        temp = temp * 1.8 + 32
    return temp, humid
```

```
# Create an instance of the Adafruit IO HTTP client
io = IO_HTTP(secrets["aio_user"], secrets["aio_key"], wifi)

# Describes feeds used to hold Adafruit IO data
feed_aqi = io.get_feed("air-quality-sensor.aqi")
feed_aqi_category = io.get_feed("air-quality-sensor.category")
feed_humidity = io.get_feed("air-quality-sensor.humidity")
feed_temperature = io.get_feed("air-quality-sensor.temperature")

# Set up location metadata from secrets.py file
location_metadata = {
    "lat": secrets["latitude"],
    "lon": secrets["longitude"],
    "ele": secrets["elevation"],
}

elapsed_minutes = 0
prv_mins = 0

while True:
    try:
        print("Fetching time...")
        cur_time = io.receive_time()
        print("Time fetched OK!")
        # Hourly reset
        if cur_time.tm_min == 0:
            prv_mins = 0
    except (ValueError, RuntimeError, ConnectionError, OSError) as e:
        print("Failed to fetch time, retrying\n", e)
        wifi.reset()
        wifi.connect()
        continue

    if cur_time.tm_min >= prv_mins:
        print("%d min elapsed.." % elapsed_minutes)
        prv_mins = cur_time.tm_min
        elapsed_minutes += 1

    if elapsed_minutes >= PUBLISH_INTERVAL:
        print("Sampling AQI...")
        aqi_reading = sample_aq_sensor()
        aqi, aqi_category = calculate_aqi(aqi_reading)
        print("AQI: %d" % aqi)
        print("Category: %s" % aqi_category)

        # temp and humidity
        print("Sampling environmental sensor...")
        temperature, humidity = read_bme(USE_CELSIUS)
        print("Temperature: %0.1f F" % temperature)
        print("Humidity: %0.1f %%" % humidity)

        # Publish all values to Adafruit IO
        print("Publishing to Adafruit IO...")
        try:
            io.send_data(feed_aqi["key"], str(aqi), location_metadata)
            io.send_data(feed_aqi_category["key"], aqi_category)
            io.send_data(feed_temperature["key"], str(temperature))
            io.send_data(feed_humidity["key"], str(humidity))
            print("Published!")
        except (ValueError, RuntimeError, ConnectionError, OSError) as e:
            print("Failed to send data to IO, retrying\n", e)
            wifi.reset()
            wifi.connect()
            continue
        # Reset timer
        elapsed_minutes = 0
    time.sleep(30)
```

Once all the files are copied from your computer to the Feather, you should have the following files on your **CIRCUITPY** drive:

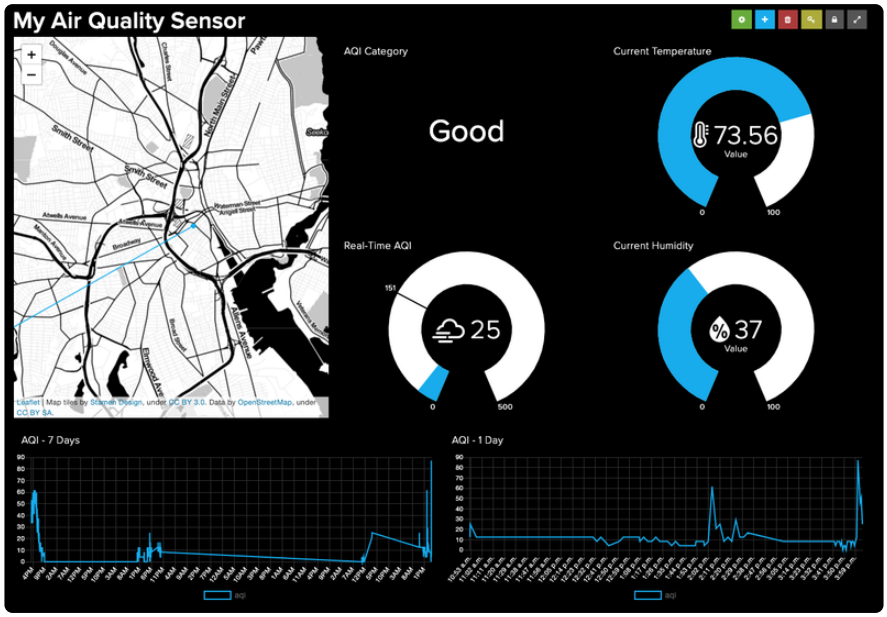


Code Usage

Before permanently installing the sensor, you should test the sensor to make sure the sensors are wired correctly and the board can publish data to Adafruit IO.

Plug the sensor into a mini-USB power cable. and **navigate to the Adafruit IO Dashboard you created earlier**. Every ten minutes all the blocks populate with values.

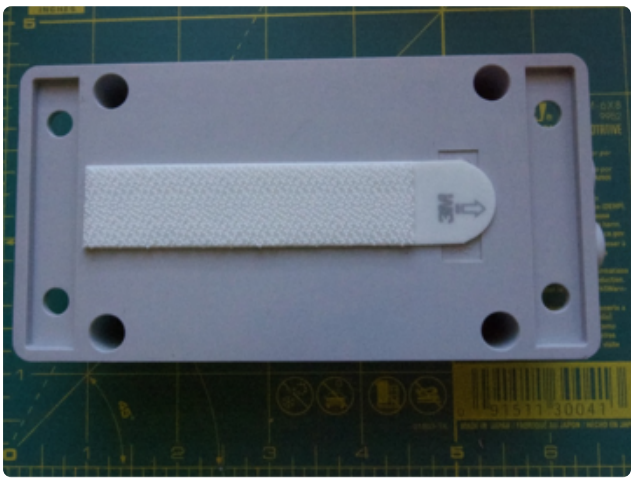
Since the air quality index values are measured in real-time, they may be higher than EPA NowCast real-time AQI values. After a day of the sensor capturing and logging data, the AQI - 1 Day line chart block will display air quality measurements every hour for the previous day.



Install Sensor

Before deploying your air quality monitor, make sure there's a WiFi network in the location you're planning on deploying to. If you're unsure about connectivity - stand exactly where you want to install the sensor, open your mobile phone/tablet, connect to your WiFi network and navigate to Adafruit IO (https://adafru.it/fH9) with your web browser.

If the test above was successful, let's move on to installing the sensor. The mounting technique you may use for this sensor varies by installation type.

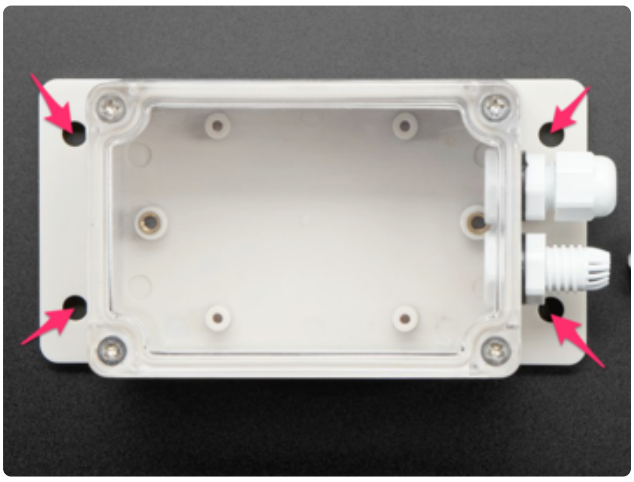


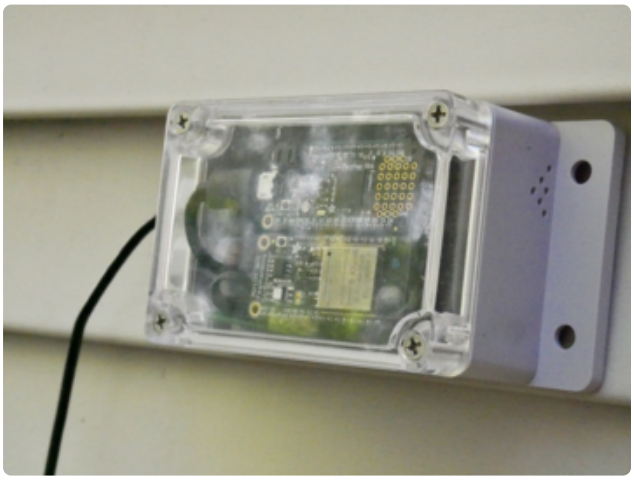
Indoor Mounting

If you want to mount your sensor indoors, we suggest using 3M Command Strips.

Peel the plastic backing off a command strip and affix it to the back of the enclosure. Attach a second command strip to the wall.

Press the enclosure into the wall and hold it for a few seconds.

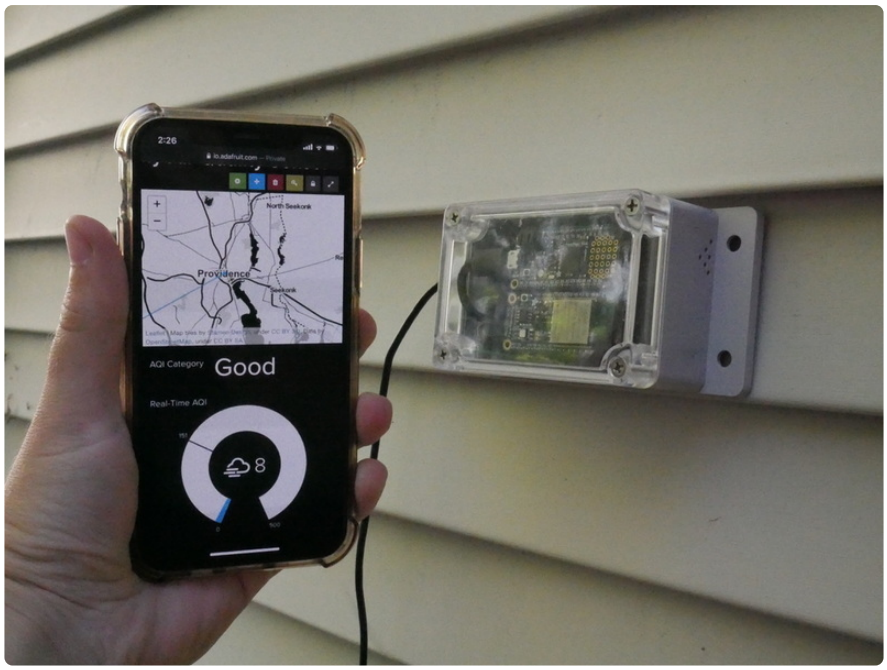




Outdoor Mounting

You can also install this sensor outdoors. Make sure you have a WiFi connection and A/C power available.

The flanged weatherproof enclosure has four mounting holes with a 6mm diameter. You can pick up the appropriate screws from your local hardware store's website.



Code Walkthrough

Hardware Setup and Configuration

The first chunk of code imports the **secrets.py** file containing WiFi details, Adafruit IO credentials, and location metadata.

```
# Get wifi details and more from a secrets.py file
try:
    from secrets import secrets
except ImportError:
    print("WiFi secrets are kept in secrets.py, please add them there!")
    raise
```

```
# AirLift FeatherWing
esp32_cs = DigitalInOut(board.D13)
esp32_reset = DigitalInOut(board.D12)
esp32_ready = DigitalInOut(board.D11)

spi = busio.SPI(board.SCK, board.MOSI, board.MISO)
esp = adafruit_esp32spi.ESP_SPIcontrol(spi, esp32_cs, esp32_ready, esp32_reset)
status_light = neopixel.NeoPixel(board.NEOPIXEL, 1, brightness=0.2)
wifi = adafruit_esp32spi_wifimanager.ESPSPI_WiFiManager(esp, secrets, status_light)
```

The PM2.5 sensor is initialized with UART. The BME280 sensor is also initialized with I2C.

```
# Connect to a PM2.5 sensor over UART
reset_pin = DigitalInOut(board.G0)
reset_pin.direction = Direction.OUTPUT
reset_pin.value = False
uart = busio.UART(board.TX, board.RX, baudrate=9600)
```

```
pm25 = PM25_UART(uart, reset_pin)

# Connect to a BME280 sensor over I2C
i2c = busio.I2C(board.SCL, board.SDA)
bme280 = adafruit_bme280.Adafruit_BME280_I2C(i2c)
```

Reading and Calculating AQI

The `calculate_aqi` function calculates a real-time qualitative Air Quality Index (AQI). This function does not use the quantitative EPA NOWCast Real-Time AQI function since this code does not store values over time. Values are stored and processed by Adafruit IO.

The function takes a sensor reading of PM2.5 size particles and returns both the AQI value and the AQI category.

The Air Quality Index developed by the EPA is divided into **six** categories. Each of the six category names has an index. These categories range from the least amount of health concern ("Good") to immediate health concern where you'll need to don a respirator ("Hazardous").

The breakpoints (CLow, CHigh, ILow, IHigh) in this function are provided by the United State's Environmental Protection Agency (EPA), for more information on the AQI calculation check out this Wikipedia article (https://adafru.it/O5D).

- If you are in a country other than the United States, this Wikipedia article contains air quality indices for locations all over the world (https://adafru.it/O5D).

```
### Sensor Functions ###
def calculate_aqi(pm_sensor_reading):
    """Returns a calculated air quality index (AQI)
    and category as a tuple.
    NOTE: The AQI returned by this function should ideally be measured
    using the 24-hour concentration average. Calculating a AQI without
    averaging will result in higher AQI values than expected.
    :param float pm_sensor_reading: Particulate matter sensor value.

    """
    # Check sensor reading using EPA breakpoint (Clow-Chigh)
    if 0.0 &lt;= pm_sensor_reading &lt;= 12.0:
        # AQI calculation using EPA breakpoints (Ilow-IHigh)
        aqi_val = map_range(int(pm_sensor_reading), 0, 12, 0, 50)
        aqi_cat = "Good"
    elif 12.1 &lt;= pm_sensor_reading &lt;= 35.4:
        aqi_val = map_range(int(pm_sensor_reading), 12, 35, 51, 100)
        aqi_cat = "Moderate"
    elif 35.5 &lt;= pm_sensor_reading &lt;= 55.4:
        aqi_val = map_range(int(pm_sensor_reading), 36, 55, 101, 150)
        aqi_cat = "Unhealthy for Sensitive Groups"
    elif 55.5 &lt;= pm_sensor_reading &lt;= 150.4:
        aqi_val = map_range(int(pm_sensor_reading), 56, 150, 151, 200)
        aqi_cat = "Unhealthy"
```

```
    elif 150.5 &lt;= pm_sensor_reading &lt;= 250.4:
        aqi_val = map_range(int(pm_sensor_reading), 151, 250, 201, 300)
        aqi_cat = "Very Unhealthy"
    elif 250.5 &lt;= pm_sensor_reading &lt;= 350.4:
        aqi_val = map_range(int(pm_sensor_reading), 251, 350, 301, 400)
        aqi_cat = "Hazardous"
    elif 350.5 &lt;= pm_sensor_reading &lt;= 500.4:
        aqi_val = map_range(int(pm_sensor_reading), 351, 500, 401, 500)
        aqi_cat = "Hazardous"
    else:
        print("Invalid PM2.5 concentration")
        aqi_val = -1
        aqi_cat = None
    return aqi_val, aqi_cat
```

The `sample_aq_sensor` function samples a PM2.5 sensor. Since the PlanTower updates the counts every 2.3 seconds, yet outputs data every 1 second. The sensor could possibly take two successive, identical, samples. This function samples the sensor for 2.3 seconds and averages the amount of samples over the number of samples obtained in that time interval.

- For more information about how the PM2.5 sensor calculates and outputs data, check out this page of the PM2.5 sensor's learn guide featuring StanJ's analysis report (https://adafru.it/O5F).

```
def sample_aq_sensor():
    """Samples PM2.5 sensor
    over a 2.3 second sample rate.

    """
    aq_reading = 0
    aq_samples = []

    # initial timestamp
    time_start = time.monotonic()
    # sample pm2.5 sensor over 2.3 sec sample rate
    while time.monotonic() - time_start &lt;= 2.3:
        try:
            aqdata = pm25.read()
            aq_samples.append(aqdata["pm25 env"])
        except RuntimeError:
            print("Unable to read from sensor, retrying...")
            continue
        # pm sensor output rate of 1s
        time.sleep(1)
    # average sample reading / # samples
    for sample in range(len(aq_samples)):
        aq_reading += aq_samples[sample]
    aq_reading = aq_reading / len(aq_samples)
    aq_samples.clear()
    return aq_reading
```

The `read_bme_280` function reads the BME280 sensor temperature and humidity and returns it as a tuple. If you set `USE_CELSIUS` at the top of the code to `True`, the temperature value will be returned in Celsius instead of Fahrenheit.

```
def read_bme280(is_celsius=False):
    """Returns temperature and humidity
    from BME280 environmental sensor, as a tuple.

    :param bool is_celsius: Returns temperature in degrees celsius
                            if True, otherwise fahrenheit.
    """
    humid = bme280.humidity
    temp = bme280.temperature
    if not is_celsius:
        temp = temp * 1.8 + 32
    return temperature, humid
```

Adafruit IO Setup and Configuration

Next is the Adafruit IO initialization and configuration. An instance of the Adafruit IO HTTP client is created.

```
# Create an instance of the Adafruit IO HTTP client
io = IO_HTTP(secrets['aio_user'], secrets['aio_key'], wifi)
```

The feeds you created earlier are initialized using calls to `get_feed`.

```
# Describes feeds used to hold Adafruit IO data
feed_aqi = io.get_feed("air-quality-sensor.aqi")
feed_aqi_category = io.get_feed("air-quality-sensor.category")
feed_humidity = io.get_feed("air-quality-sensor.humidity")
feed_temperature = io.get_feed("air-quality-sensor.temperature")
```

The location values (latitude, longitude, elevation) are pulled from your secrets file and initialized as a tuple, `location_metadata`.

```
# Set up location metadata from secrets.py file
location_metadata = (secrets['latitude'], secrets['longitude'],
secrets['elevation'])
```

Main Loop

The `while True` loop fetches the current time from Adafruit IO's time API (we don't need a real-time-clock or timezone calculations) and reads/publishes sensor data to Adafruit IO when a time interval elapses.

Obtaining time from Adafruit IO

The Adafruit IO time service does not replace a time-synchronization service like NTP, but it can help you figure out your local time on an Internet of Things device that doesn't have a built in clock.

Instead of using a software-based timer, this code will fetch the current time from Adafruit IO using a call to `receive_time()` every 30 seconds. Then, it will keep track of the minutes elapsed.

```
try:
        print("Fetching time...")
        cur_time = io.receive_time()
        print("Time fetched OK!")
        # Hourly reset
        if cur_time.tm_min == 0:
            prv_mins = 0
    except (ValueError, RuntimeError) as e:
        print("Failed to fetch time, retrying\n", e)
        wifi.reset()
        wifi.connect()
        continue

    if cur_time.tm_min &gt;= prv_mins:
        print("%d min elapsed.."%elapsed_minutes)
        prv_mins = cur_time.tm_min
        elapsed_minutes += 1
```

Sample and Publish Data to Adafruit IO

When `PUBLISH_INTERVAL` elapses, the loop will sample the air quality sensor and environmental sensor. Values are printed to the REPL.

Once values are obtained, each value is published to its respective Adafruit IO Feed using calls to `io.send_data`.

Finally, the sensor sleeps 30 seconds before running again.

```
if elapsed_minutes &gt;= PUBLISH_INTERVAL:
        print("Sampling AQI...")
        aqi_reading = sample_aq_sensor()
        aqi, aqi_category = calculate_aqi(aqi_reading)
        print("AQI: %d"%aqi)
        print("Category: %s"%aqi_category)

        # temp and humidity
        print("Sampling environmental sensor...")
        temperature, humidity = read_bme280(USE_CELSIUS)
        print("Temperature: %0.1f F" % temperature)
        print("Humidity: %0.1f %%" % humidity)

        # Publish all values to Adafruit IO
        print("Publishing to Adafruit IO...")
        try:
            io.send_data(feed_aqi["key"], str(aqi), location_metadata)
            io.send_data(feed_aqi_category["key"], aqi_category)
            io.send_data(feed_temperature["key"], str(temperature))
            io.send_data(feed_humidity["key"], str(humidity))
            print("Published!")
        except (ValueError, RuntimeError) as e:
            print("Failed to send data to IO, retrying\n", e)
            wifi.reset()
            wifi.connect()
```

```
            continue
          # Reset timer
          elapsed_minutes = 0
        time.sleep(30)
```