Air Quality Sensor 3D Printed Enclosure
Created by Ruiz Brothers

Last updated on 2021-10-22 11:44:26 AM EDT
## Guide Contents

**Overview**
- IOT Air Sensor Monitor: 4
- Adafruit IO: 4
- Enclosure Design: 4
- Choose Your Sensors: 5
- Prerequisite Guides: 5
- Parts: 5
- Hardware List: 7

**Circuit Diagram**
- Adafruit Library for Fritzing: 9
- Choose Your Sensors: 9
- PMSA003i & BME680: 9
- PMS5003 & BME280: 10

**Wiring**
- Wire for BME280: 11
- Wired BME280: 11
- Wiring for PMS5003: 11
- Doubler FeatherWing + I2C (BME280 or BME680) Wiring: 12
- Doubler FeatherWing + PMS5003 Wiring: 12
- Feather Headers: 13
- Install Feathers to Doubler: 14
- USB micro B Connector: 14

**Adafruit IO Setup**
- Obtain Adafruit IO Key: 15
- Create Group: 15
- Add Feeds to Group: 15
- Adafruit IO Dashboard: 18
- Add Gauge Block for Real-Time AQI: 21
- Add Gauge Block for Humidity: 23
- Add Gauge Block for Temperature: 23
- Add Line Charts: 24
- Organize Dashboard: 26

**CircuitPython Setup**
- Install CircuitPython: 28
- CircuitPython Library Installation: 28

**Internet Connect!**
- What's a secrets file?: 29
- Connect to WiFi: 29

**Requests**
- HTTP GET with Requests: 37
- HTTP POST with Requests: 38
- Advanced Requests Usage: 38

**WiFi Manager**

**Code Usage**
- Text Editor: 43
- Secrets File Setup: 43
- Code: 44
- Modify Code: 48
- Code Usage: 49

**3D Printing**

© Adafruit Industries

https://learn.adafruit.com/air-quality-sensor-silo-house

Page 2 of 65
Parts List
Heat Set Inserts 50
Heat Set Rig 50
Install Heat Set Inserts 51

Assembly 52
Install PG-7 Cable Gland 52
Installing USB Extension Cable 52
Micro USB connector Wired 53
Hardware for Case Mounting Plate 53
Install Standoffs to Silo Case 54
Install Mounting Plate 55
Hardware for FeatherWing Doubler 55
FeatherWing Hardware Installed 56
Install PCB Mount to FeatherWing Doubler 56
Install and Secure PMSA003i 57
Secure and Install BME680 58
BME680 + PMSA003i Connect STEMMA 58
Install & Secure BME280 59
Install PMS5003 60
BME280 + PMS5003 Connect 60
Install & Secure PCB Mount 61
Install & Secure Bottom Cover 62
Install Top Roof 62
Final Build 63
Wall Mount 63
Overview

IOT Air Sensor Monitor

Build a 3D printed enclosure for your IOT Air Quality Sensor. This project is similar to our other air quality sensor guide (https://adafruit.it/OzE), except it uses Adafruit STEMMA sensors and has minimal soldering required.

This project uses sensors to measure PM2.5 (particles that are 2.5 microns or smaller in diameter) dust concentrations, temperature and humidity.

This weatherproof enclosure is modeled after a silo-home. We've included different types of mounting holes and brackets for different mounting configurations.

Adafruit IO

Build an Adafruit IO (https://adafruit.it/Eg2) Dashboard to visualize your air quality sensor's data at a glance. Adafruit IO (https://adafruit.it/Eg2) is free and includes up to 30 days of storage for your sensor's data.

You can view and interact with this data from anywhere in the world with a wireless connection.

Enclosure Design

The case is themed to resemble a modern silo grain bin house. The roof is removable by unscrewing the top from the case. The Feather boards are securely mounted inside via heat set inserts. A mounting plate is used to secure the enclosure to a surface. Weatherproof cable gland is used to keep moisture out. The bottom cover is modular allowing for a different set of sensors.
Choose Your Sensors
The enclosure and code accommodate for two different sets of sensors. Take a moment to choose your set up.

- BME280 & PMS5003
- BME680 & PMSA003i

Prerequisite Guides
Take a moment to walk through the following guides to get familiar with the board, sensors and Adafruit IO.

- Feather M4 Guide (https://adafruit.it/CJN)
- Airlift FeatherWing Guide (https://adafruit.it/OzF)
- PM2.5 Air Quality Sensor (https://adafruit.it/OA0)
- Adafruit PMSA003I Air Quality Breakout (https://adafruit.it/OA1)
- Adafruit IO Welcome Guide (https://adafruit.it/BRB)

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,...
$22.95
In Stock
Add to Cart

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...
$12.95
In Stock
Add to Cart

Your browser does not support the video tag.
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...
$39.95
In Stock
Add to Cart

Adafruit PMSA003I Air Quality Breakout
Breathe easy, knowing that you can track and sense the quality of the air around you with this Adafruit PMSA003I Air Quality Breakout. This sensor is great for...
$44.95
In Stock
Add to Cart

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...
$14.95
In Stock
Add to Cart

Adafruit BME680 - Temperature, Humidity, Pressure and Gas Sensor
The long awaited BME680 from Bosch gives you all the environmental sensing you want in one small package. This little sensor...
$18.95
In Stock
Add to Cart

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
$7.50
In Stock
Add to Cart

1 x STEMMA QT Cable
Qwiic JST SH 4-pin Cable - 100mm Long
Add to Cart

1 x USB Extension Cable
3 meters / 10 ft long
Add to Cart

1 x USB A/Micro Cable - 2m
USB A/Micro Cable - 2m
Add to Cart
1 x **PG7 Cable Gland**  
Cable Gland PG-7 size - 0.118” to 0.169” Cable Diameter - PG-7  
Add to Cart

1 x **USB Micro B Connector**  
USB DIY Slim Connector Shell - MicroB Plug  
Add to Cart

1 x **M3 Heat-Set Inserts**  
Brass Heat-Set Inserts for Plastic - M3 x 4mm - 50 pack  
Out of Stock

1 x **Silicone Cover Stranded-Core Ribbon Cable**  
10 Wire 1 Meter Long - 28AWG Black  
Add to Cart

1 x **M2.5 Hardware**  
Black Nylon Screw and Stand-off Set – M2.5 Thread  
Add to Cart

4 x **M3 Hardware**  
M3 screws, hex nuts and standoffs  
Add to Cart

4 x **M3 x 20mm Standoffs**  
M3 x 20mm Standoffs  
Buy Now

1 x **USB Power Supply**  
5V 1A (1000mA) USB port power supply - UL Listed  
Add to Cart

**Hardware List**

- 4x M3 x 4mm heat-set inserts (silo case)
- 4x M3 x 6mm flat head machine screws (silo standoffs)
- 4x M3 hex nuts (silo standoffs)
- 2x M3 x 8mm button head machine screws (PCB mount for silo)
- 2x M3 x 6mm thumb screws (bottom cover for silo)
- 8x M2.5 x 4mm button head machine screws (PCB mount for FeatherWing Doubler)
- 4x M2.5 x 6mm FF standoffs (PCB mount for FeatherWing Doubler)
- 2x M5 x 8mm button head screws (silo mounting plate to wall)
For PMSA003i

- 2x M3 x 6mm button head machine screws

For BME280

- 2x M2.5 x 4mm flat head machine screws

For BME680

- 4x M2.5 x 4mm flat head machine screws
Circuit Diagram

The diagram below provides a visual reference for wiring of the components. This diagram was created using the software package Fritzing (https://adafru.it/oEP).

Adafruit Library for Fritzing

Use Adafruit's Fritzing parts library to create circuit diagrams for your projects. Download the library or just grab individual parts. Get the library and parts from GitHub - Adafruit Fritzing Parts (https://adafru.it/AYZ).

Choose Your Sensors

This project can use two different sets of sensors. The first set uses the BME280 and the PMS5003. The second set uses the BME680 and the PMSA003i. The wiring is slightly different and noted below.

The code accommodates for both setups.

PMSA003i & BME680

Make the following connections between the BME680 and the FeatherWing:

- Board 3V to sensor VIN
- Board GND to sensor GND
- Board SCL to sensor SCL
- Board SDA to sensor SDA

Then, make the following connections between the PMSA003i and the BME680:

- Sensor VIN to board 5V
- Sensor GND to board GND
- Sensor SCL to board SCL
- Sensor SDA to board SDA
PMS5003 & BME280

Make the following connections between the BME280 and the FeatherWing:

- Board 3V to sensor VIN
- 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!
Wiring

Wire for BME280
Use a piece of 4-wire silicone ribbon cable, 15cm (6in) in length.

Wired BME280
Solder the ribbon cable to the pins on the BME280 breakout board.

- VIN
- GND
- SCK
- SDI

Wiring for PMS5003
Use a piece of 4-wire silicone ribbon cable, 7cm (3in) in length. Solder the wires to the pins on the breakout adapter for the PMS5003.

- VCC
- GND
- TXD
- RST
Doubler FeatherWing + I2C (BME280 or BME680) Wiring
Use the photo and labels to reference the wiring. The BME280 and BME680 use exact same pins on the FeatherWing Doubler.

Doubler FeatherWing + PMS5003 Wiring
Use the photo and labels to reference wiring. The PMS5003 requires dedicated wiring to the Doubler FeatherWing.

The PMSA003i plugs directly into the STEMMA port on the BME680, essentially daisy chained I2C.
Feather Headers
Install the male 12pin and 16pin headers to the Feather M4 and Airlift FeatherWing. Solder the headers to the pins.
Install Feathers to Doubler
Line up the male header pins with the female header pins on the FeatherWing Doubler. Press firmly to fully seat the headers.

USB micro B Connector
Wire up the micro USB b connector to a piece of 4-wire ribbon cable, about 12cm (5in) in length. This is used as the power cable for the Feather. Use the USB extension cable to increase the length of the cable.

Do not wire the micro USB connector to the USB extension cable just yet. The USB cable must be installed into the cable gland in the case first before wiring.
Adafruit IO Setup

Obtain Adafruit IO Key

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

Navigate to Adafruit IO (https://adafruit.io/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://adafruit.io/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://adafruit.io/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. 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

Choose up to 5 feeds

*Line Chart:* The line chart is used to graph one or more feeds.

If you have lot of feeds, you may want to use the search function.

**Group / Feed**

**My Feeds**

- **Air Quality Sensor**

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.
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:

```python
# 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': 'fawfj23rakjnfwiefa',
    '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](http://worldtimeapi.org/timezones) 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://adafruit.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://adafruit.it/zdx). Our introduction guide has a great page on how to install the library bundle (https://adafruit.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_espm32_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://adafruit.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:

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

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

# 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

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)

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

requests.set_socket(socket, esp)

if esp.status == 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])

for ap in esp.scan_networks():
    print("SSID:", ap["ssid"], ", RSSI: %d" % (ap["rssi"]))

print("Connecting to AP...")
while not esp.is_connected:
    try:
        esp.connect_AP(secrets["ssid"], secrets["password"])
    except RuntimeError as e:
        print("could not connect to AP, retrying: ", e)
        continue
print("Connected to", str(esp.ssid, "utf-8"), ", RSSI: ", 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())
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:

```python
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:

![Image of ESP32 serial output showing successful connection and fetching JSON data from Coindesk API](image)

In order, the example code...
Initializes the ESP32 over SPI using the SPI port and 3 control pins:

```python
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:

```python
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.

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

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

```python
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:

```python
for ap in esp.scan_networks():
    print("%s	RSSI: %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)

```python
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](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

```python
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)

```python
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](https://adafru.it/FpT)) library for web interfacing named Adafruit_CircuitPython_Requests ([https://adafru.it/FpW](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.

```python
# SPDX-FileCopyrightText: 2021 ladyada for Adafruit Industries
# SPDX-License-Identifier: MIT

# adafruit_requests usage with an esp32spi_socket
import board
import busio
from digitalio import DigitalInOut
import adafruit_esp32spi.adafruit_esp32spi_socket as socket
import adafruit_esp32spi
import adafruit_requests as requests

# Add a secrets.py to your filesystem that has a dictionary called secrets with "ssid" and "password" keys with your WiFi credentials. DO NOT share that file or commit it into Git or other
# source control.
```
try:
    from secrets import secrets
except ImportError:
    print("WiFi secrets are kept in secrets.py, 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)

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

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(secrets["ssid"], secrets["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
socket.set_interface(esp)
requests.set_socket(socket, esp)

TEXT_URL = "http://wifitest.adafruit.com/testwifi/index.html"
JSON_GET_URL = "https://httpbin.org/get"
JSON_POST_URL = "https://httpbin.org/post"

print("Fetching text from %s" % TEXT_URL)
response = requests.get(TEXT_URL)
print("-" * 40)
print("Text Response: ", response.text)
response.close()

print("Fetching JSON data from %s" % JSON_GET_URL)
response = requests.get(JSON_GET_URL)
print("-" * 40)
print("JSON Response: ", response.json())
response.close()
response.close()
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()

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()

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:

```python
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://adafruit.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.

```python
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.

# SPDX-FileCopyrightText: 2021 ladyada for Adafruit Industries
# SPDX-License-Identifier: MIT
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

# Add a secrets.py to your filesystem that has a dictionary called secrets with "ssid" and
# "password" keys with your WiFi credentials. DO NOT share that file or commit it into Git or
# other
# source control.
# pylint: disable=no-name-in-module,wrong-import-order
try:
    from secrets import secrets
except ImportError:
    print("WiFi secrets are kept in secrets.py, 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)

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(secrets["ssid"], secrets["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
socket.set_interface(esp)
requests.set_socket(socket, esp)

JSON_GET_URL = "http://httpbin.org/get"

# Define a custom header as a dict.
headers = {"user-agent": "blinka/1.0.0"}

print("Fetching JSON data from %s..." % JSON_GET_URL)
response = requests.get(JSON_GET_URL, headers=headers)
print("." * 60)

json_data = response.json()
headers = json_data["headers"]
print("Response's Custom User-Agent Header: {0}".format(headers["User-Agent"]))
print("." * 60)
# Read Response's HTTP status code
print("Response HTTP Status Code: ", response.status_code)
print("-" * 60)

# Close, delete and collect the response data
response.close()

## WiFi Manager

That simplest example works but it's 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:

```python
# SPDX-FileCopyrightText: 2019 ladyada 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
from adafruit_esp32spi import adafruit_esp32spi_wifimanager

print("ESP32 SPI webclient test")
# 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

# 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)

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 for Most Boards
"""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
# 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 (ValueError, RuntimeError) 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://adafruit.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:
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:

```python
# 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://adafruit.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.

```python
secrets = {
    'ssid' : 'home ssid',
    'password' : 'my password',
    'timezone' : "America/New_York", # http://worldtimeapi.org/timezones
    'aio_user' : '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 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://adafruit.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 which appears when your board is plugged into your computer via a known good USB cable.

```python
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)
```

© Adafruit Industries  https://learn.adafruit.com/air-quality-sensor-silo-house  Page 44 of 65
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, 600)
        aqi_cat = "Injurious to Life and Health"
    else:
        aqi_val = map_range(int(pm_sensor_reading), 501, 600, 601, 1000)
        aqi_cat = "Highly Unhealthy"

    return aqi_val, aqi_cat
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 degree 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_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) 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..\n" % 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) 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:

![Image of file structure]

Modify Code

The code.py file is configured for an Air Quality Sensor using a BME280 and PMS5003 sensors. You’ll need to open a text editor and make the following modifications in the code.py file for the BME680 and PMSA003i.

Change out the following lines from

```
uart = busio.UART(board.TX, board.RX, baudrate=9600)

pm25 = adafruit_pm25.PM25_UART(uart, reset_pin)
```

to

```
# uart = busio.UART(board.TX, board.RX, baudrate=9600)

# pm25 = adafruit_pm25.PM25_UART(uart, reset_pin)
```

Change the following line from

```
bme_sensor = adafruit_bme280.Adafruit_BME280_I2C(i2c)
```

to

```
# bme_sensor = adafruit_bme280.Adafruit_BME280_I2C(i2c)
```
Then, uncomment the following lines to enable the PMSA003I and BME680 sensors:

```python
# Uncomment below for PMSA003I Air Quality Breakout
pm25 = adafruit_pm25.PM25_I2C(i2c, reset_pin)

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

**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.
3D Printing

Parts List
STL files for 3D printing are oriented to print "as-is" on FDM style machines. Parts are designed to 3D print without any support material. Original design source may be downloaded using the links below.

File names

- silo-case.stl
- silo-roof.stl
- bottom-pms5003.stl
- bottom-pmsa003i.stl
- mounting-plate.stl
- pcb-mount.stl
- tripod-plate.stl
- PG7-plug.stl
- pmsa003i-bracket.stl

Heat Set Inserts
The case uses 4x M3 x 4mm long heat set inserts to secure the PCB mount and bottom cover. The case is sized for the heat set inserts that are stocked in the Adafruit shop.
Heat Set Rig
Optionally use our 3D printed Heat Set rig to install the heat sets.

Install Heat Set Inserts
Use a heat-set insert for your soldering iron to install the heat set inserts.

Heat-Set Insert For Soldering Irons - #4-40 / M3 Inserts
Wanna improve the connection strength between your project's 3D-printed parts, and also have nice clean surfaces? Instead of gluing bits together, or screwing plastic screws...
$9.95
In Stock
Add to Cart

Brass Heat-Set Inserts for Plastic - M3 x 4mm - 50 pack
Wanna improve the connection strength between your project's 3D-printed parts, and also have nice clean surfaces? Instead of gluing bits together, or screwing plastic screws...
Out of Stock
Install PG-7 Cable Gland
Fit the PG-7 size cable gland into the 3d printed PG7-plug.stl part. Fit the cable gland into the hole on the side of the case. It should have a tight fitting. Optionally use hot glue to seal the hole.

Installing USB Extension Cable
Cut the USB Micro B end off the USB extension cable, leaving the Type A connector in-tacked. Insert and thread the USB wires through the PG7 size cable gland.
Micro USB connector Wired
Solder the wires from the USB cable to the micro USB B connector from earlier. Use pieces of heat shrink tubing to insulate exposed wires.

Hardware for Case Mounting Plate
Use the following hardware to secure the mounting plate to the case.

- 4x M3 hex nuts
- 4x M3 x 20mm M-F standoffs
- 4x M3 x 8mm flat head machine screws
Install Standoffs to Silo Case
Insert and place the M3 hex nuts into the recesses inside the case. While holding hex nuts in place, insert and fasten the M3 standoffs. Repeat process for all four standoffs.
Install Mounting Plate
Place the mounting plate over the standoffs and line up the mounting holes. Insert and fasten 4x M3 x 8mm flat head machine screws to secure the mounting plate.

Hardware for FeatherWing Doubler
Use the following hardware for the FeatherWing Doubler.

- 8x M2.5 x 6mm button head machine screws
- 4x M2.5 x 6mm FF standoffs
FeatherWing Hardware Installed
Secure the M2.5 screws and standoffs to the corner mounting holes on the FeatherWing Doubler. Use the screws to secure the standoffs.

Install PCB Mount to FeatherWing Doubler
Place the PCB mount over the standoffs and line up the mounting holes. Use the M2.5 screws to secure the PCB mount to the FeatherWing Doubler.
Install and Secure PMSA003i

Use 2x M3 x 6mm long button head machine screws to secure the PMSA003i to the bottom cover. Fit the PMSA003i into the bottom cover and line up the holes. Fit the 3D printed bracket over the PMSA003i and secure using the 2x M3 screws.
Secure and Install BME680
Use 2x M2.5 x 6mm flat head machine screws to secure the BME680 to the bottom cover. Use the photo to reference correct placement and orientation.

BME680 + PMSA003i Connect STEMMA
Use the STEMMA cable to connect the BME680 to the PMSA003i.
Install & Secure BME280
Use 2x M2.5 x 6mm flat head machine screws to secure the BME280 to the bottom cover. Use the photo to reference correct placement and orientation.
Install PMS5003
The PMS5003 press fits into the holder that's built into the bottom cover. Use the holes in the bottom to correctly line up the PMS5003.

BME280 + PMS5003 Connect
Connect the included cable to the PMS5003 and the breakout adapter. The breakout can optionally be secured to the bottom of the PCB mount.
Install & Secure PCB Mount

Fit the PCB mount through the bottom of the case and pull it out from the top. Plug in the AirLift FeatherWing and Feather M4 to the Doubler. Plug in the micro USB cable to the Feather M4.

Use 2x M3 x 6mm button head machine screws to secure the PCB mount to the tabs inside the case. Reference photo for best placement and orientation.
Install & Secure Bottom Cover
Fit any excess wiring into the case. Fit the bottom cover into the case and line up the mounting holes. Press fit the bottom cover into the case.

Use 2x M3 x 6mm thumb screws to secure the bottom cover to the case.

Install Top Roof
Fit the roof piece over the top of the case. Line up the threads and begin to screw the top cover. Unscrew the top cover to remove it.
Final Build
Hurray! The build is complete. Start to plan where this thing is going to be mounted. Depending on your application, you may need to create your own method of mounting.

Wall Mount
Find a good spot to secure the mounting plate. Outside our entrance was a key lockbox that we repurposed for the screw holes. Use M5 (sheet metal or wood) screws to secure the mounting plate. Optionally use double-sided foam tape, hotglue, 3M adhesive or the like.

To power the Feather, plug in the USB cable to a 5V wall adapter.