PyPortal Pet Planter with Adafruit IO

Created by Ruiz Brothers

https://learn.adafruit.com/pyportal-pet-planter-with-adafruit-io

Last updated on 2024-04-09 02:04:35 PM EDT

Table of Contents

Overview 5

- PyPortal IoT Planter
- Audio & Animations with CircuitPython
- 3D Printed Case
- Parts
- Feed Me!

3D Printing 9

- 3D Printed Parts
- CAD Assembly
- Design Source Files
- Slicing Parts

PyPortal Assembly 11

- Screen and Face Plate
- Install Screen to Face Plate
- Install PyPortal to Screen Cover
- Secure PyPortal to Screen Cover

Sensor Assembly 12

- Hardware for Soil Sensor
- Install Soil Sensor
- Secure Soil Sensor
- Install STEMMA Cable

Planter Assembly 13

- Install Planter to Frame
- Install Drip Tray
- Snap Fit Drip Tray
- Snap Fit Planter
- Installed Planter
- Install Soil Sensor
- Install Sensor Plate
- Secure Sensor Plate
- Installed Soil Sensor Planter

Case Assembly 16

- Connect Soil Sensor to PyPortal
- Connect Speaker
- Install Face Plate to Frame
- Install Speaker to Frame
- Install Back Cover
- Final Build

PyPortal Wiring 18

Adafruit IO Setup 20

- Create Feeds
- Build a Dashboard
- Add a Line Chart
- Add Gauges

• Add Header Image
• Obtain Adafruit IO Key

PyPortal CircuitPython Setup 28

• Adafruit CircuitPython Bundle

Create Your settings.toml File 30

• CircuitPython settings.toml File
• settings.toml File Tips
• Accessing Your settings.toml Information in code.py

Internet Connect! 33

• Connect to WiFi
• Requests
• HTTP GET with Requests
• HTTP POST with Requests
• Advanced Requests Usage
• WiFi Manager

Code Setup 44

• CircuitPython Library Installation
• Add CircuitPython Code and Project Assets
• Install the Mu Editor
• settings.toml File Setup

Code Usage 52

• Viewing Sensor Data on PyPortal
• Viewing Sensor Data on Adafruit IO
• Adjusting Soil Moisture Levels

Overview

PyPortal IoT Planter

Build a smart planter with Adafruit PyPortal and CircuitPython. Monitor your plants vitals with Adafruit STEMMA Soil Sensor and plot moisture and temperature data. Use Adafruit IO to create a visual dashboard with gauges of your plants water levels.

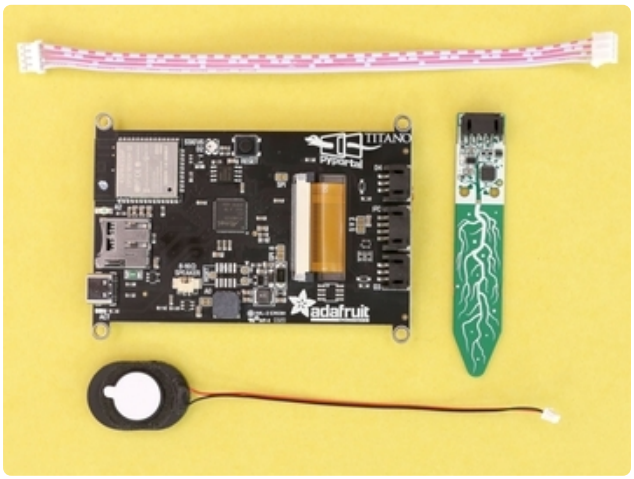
Audio & Animations with CircuitPython

The display shows an animation of the water levels and moisture and temperature values. The planter will play custom audio alerts when the water is too high or low.

3D Printed Case

The enclosure design mimics our PyPortal retro cases (https://adafru.it/IJc). Thick bezels and round edges makes the build look simple and cute. With the soil sensor hidden inside the planter, it's gives the plant a clean look and lets it show off its flora.

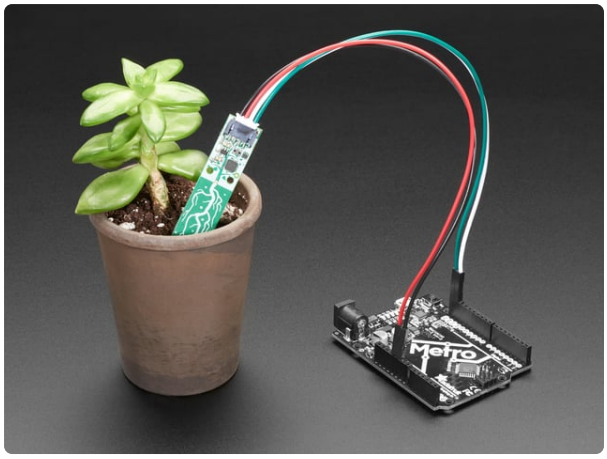
Parts



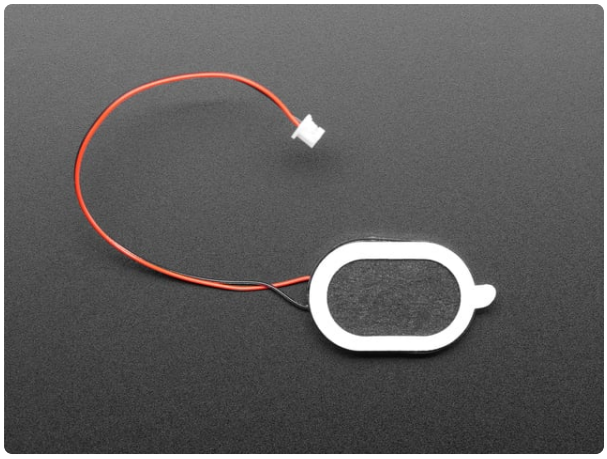
Parts used to build this project. The soil sensor and speaker connect directly into the PyPortal so there's no soldering needed!

PyPortal Titano (http://adafru.it/4444)
STEMMA Soil Sensor (http://adafru.it/4026)
STEMMA Cable – F/F (http://adafru.it/3568)
Mini Oval Speaker (http://adafru.it/3923)
Black Nylon Hardware Kit (http://adafru.it/3299)

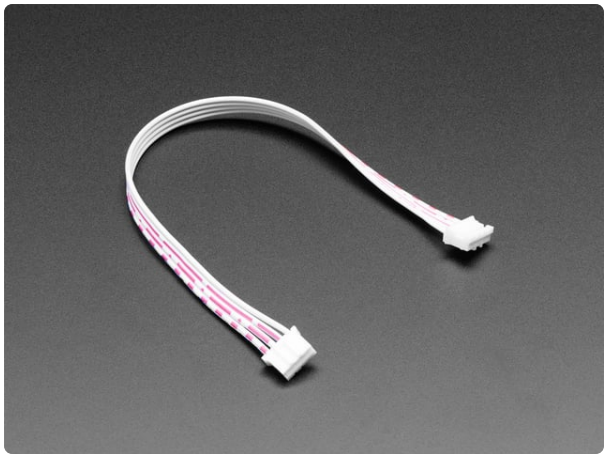
Adafruit PyPortal Titano
The PyPortal Titano is the big sister to our popular PyPortal now with twice as many pixels! The PyPortal...
https://www.adafruit.com/product/4444



[Adafruit STEMMA Soil Sensor - I2C Capacitive Moisture Sensor]
Most low cost soil sensors are resistive style, where there's two prongs and the sensor measures the conductivity between the two. These work OK at first, but eventually...
https://www.adafruit.com/product/4026

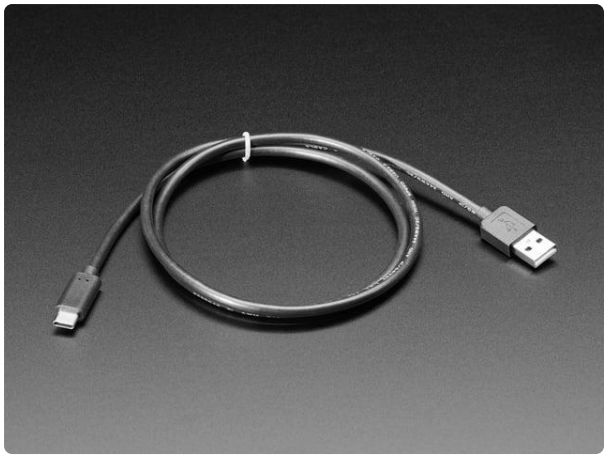


[Mini Oval Speaker - 8 Ohm 1 Watt]
Hear the good news! This wee speaker is a great addition to any audio project where you need 8 ohm impedance and 1W or less of power. We particularly like...
https://www.adafruit.com/product/3923



[STEMMA Cable - 4 Pin JST-PH 2mm Cable–Female/Female]
This 4-wire cable is a little over 150mm / 6" long and fitted with JST-PH female 4-pin connectors on each end. These types of JST cables are commonly found on small rechargeable...
https://www.adafruit.com/product/3568

[Black Nylon Machine Screw and Stand-off Set – M2.5 Thread]
Totaling 380 pieces, this M2.5 Screw Set is a must-have for your workstation. You'll have enough screws, nuts, and hex standoffs to fuel your maker...
https://www.adafruit.com/product/3299



USB Type A to Type C Cable - approx 1 meter / 3 ft long

As technology changes and adapts, so does Adafruit. This USB Type A to Type C cable will help you with the transition to USB C, even if you're still...

https://www.adafruit.com/product/4474

5V 1A (1000mA) USB port power supply - UL Listed

Need a USB jack for charging or powering a project, but don't want to lug around a computer? This switching supply gives a clean regulated output at up to 1000mA! 110 or 240 input,...

https://www.adafruit.com/product/501

8 x M3 Machine Screws

M3 x 4mm

https://www.albanycountyfasteners.com/Phillips-Pan-Head-Machine-Screw-M3-x-5-p/1066-1008.htm

---

Feed Me!

The plant used in this project are Venus Fly Traps. Their natural habitat are subtropical wetlands.

3D Printing

3D Printed Parts

STL files for 3D printing are oriented to print "as-is" on FDM style machines. Original design source may be downloaded using the links below.

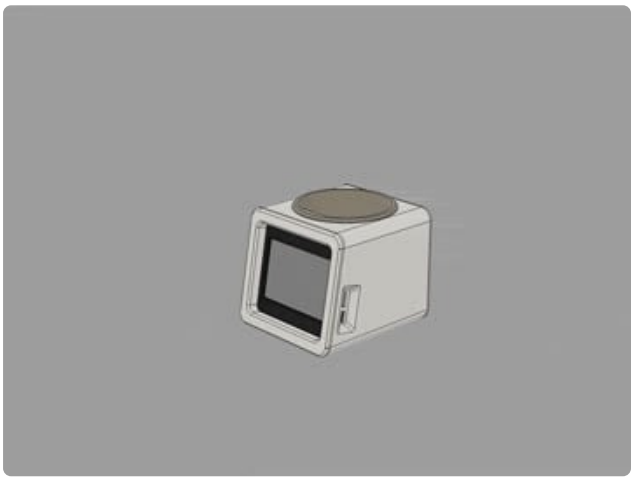
pyplant-frame.stl
pyplant-drip-tray.stl
pyplant-soil-plate.stl
pyplant-cover.stl
pyplant-screen.stl
pyplant-face.stl
pyplant-cup.stl

Download CAD files from Thingiverse

https://adafru.it/IFu

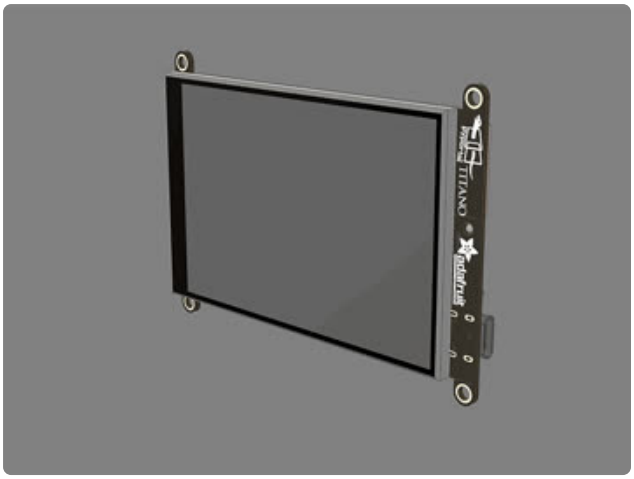
Download CAD files from Prusa Printers

https://adafru.it/IFE



CAD Assembly

The soil sensor is secured to the sensor plate with screws. The soil plate is mounted to the planter cup with machine screws. The PyPortal Titano is secured to the screen cover with screws. The screen cover is attached to the face plate with screws. The planter cup is installed to the enclosure with snap fits. A drip tray snap fits over the bottom of the planter cup. An open cover snap fits over the back of case.



Design Source Files

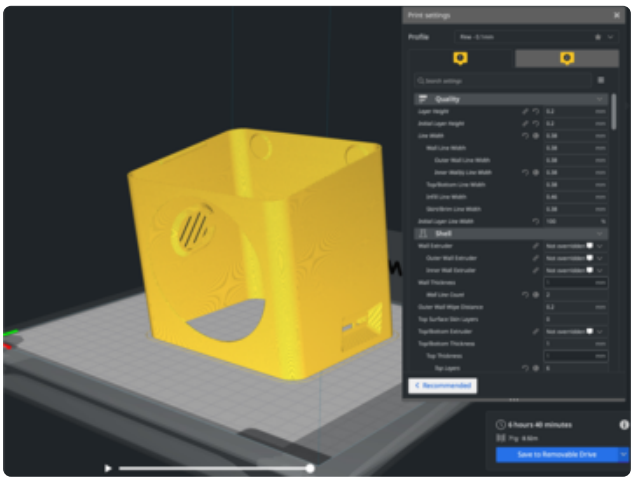
The project assembly was designed in Fusion 360. This can be downloaded in different formats like STEP, SAT and more. Electronic components like Adafruit's board, displays, connectors and more can be downloaded from the Adafruit CAD parts GitHub Repo (https://adafru.it/AW8).

Download 3D Models of PyPortal Titano

https://adafru.it/19kC

Download 3D Models of STEMMA Soil Sensor

https://adafru.it/19kD



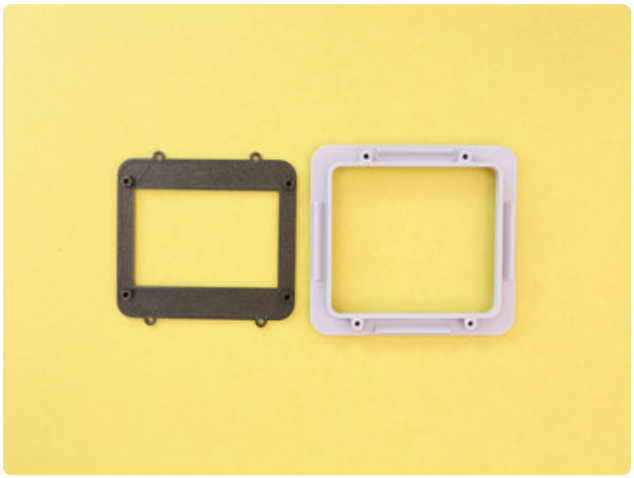
Slicing Parts

No supports are required. Slice with setting for PLA material.

The parts were sliced using CURA using the slice settings below.

PLA filament 220c extruder
0.2 layer height
10% gyroid infill
60mm/s print speed
60c heated bed

PyPortal Assembly

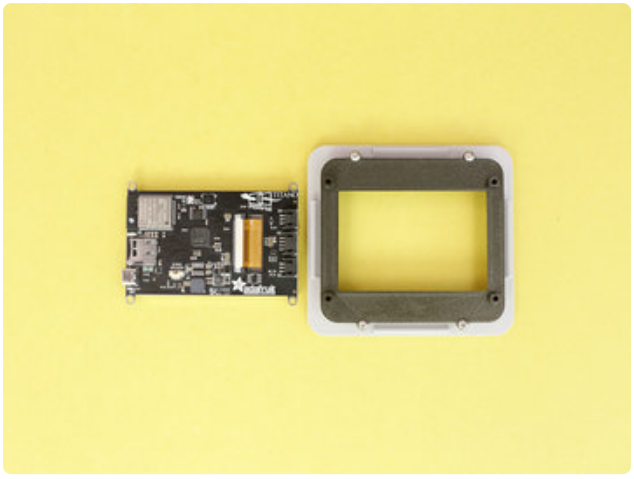


Screen and Face Plate

The screen cover is secured to the face plate using M3 machine screws. Place the screen cover over the face plate and line up the mounting tabs.

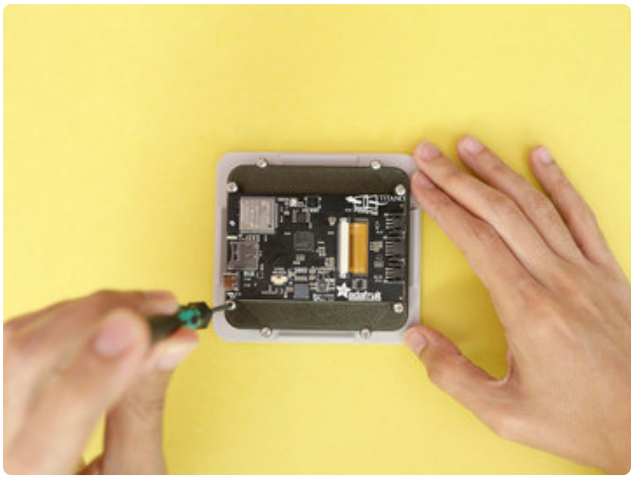
Install Screen to Face Plate

Insert and fasten M3x4mm long machine screws into the four mounting tabs.



Install PyPortal to Screen Cover

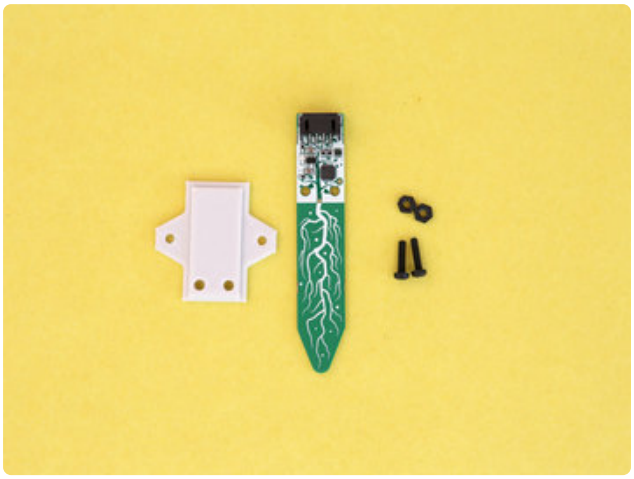
The PyPortal Titano is secured to the screen cover with four additional machine screws. Reference the photo for the correct orientation.



Secure PyPortal to Screen Cover

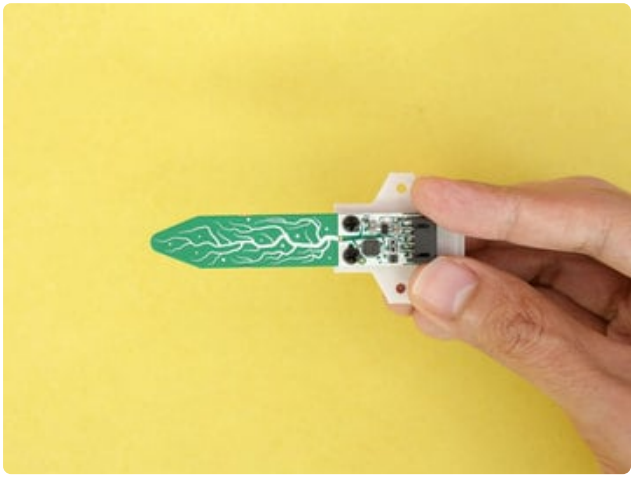
Place the PyPortal over the screen cover and line up the mounting tabs with the standoffs. Carefully fit the display in between the standoffs that feature the correct shape. Insert and fasten four M3 x 4mm long machine screws to the mounting tabs.

Sensor Assembly



Hardware for Soil Sensor

The STEMMA soil sensor is secured to the sensor plate using 2 x M2.5 x 10mm long screws and hex nuts.

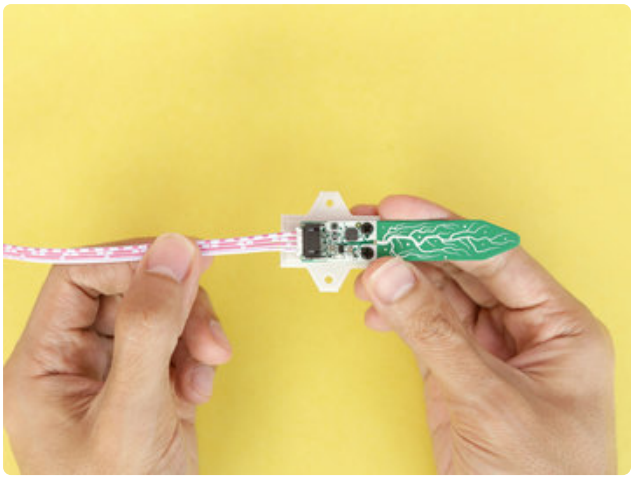


Install Soil Sensor

Place the STEMMA soil sensor over the sensor plate. Line up the two mounting holes on the PCB with the holes on the plate. Insert two M2.5 x 10mm long screws through the mounting holes.

Secure Soil Sensor

Insert and fasten an M2.5 hexnut onto the threads of the screws. Finger tighten the two hex nuts.



Install STEMMA Cable

Grab the F/F STEMMA cable and plug it into the port on the soil sensor.

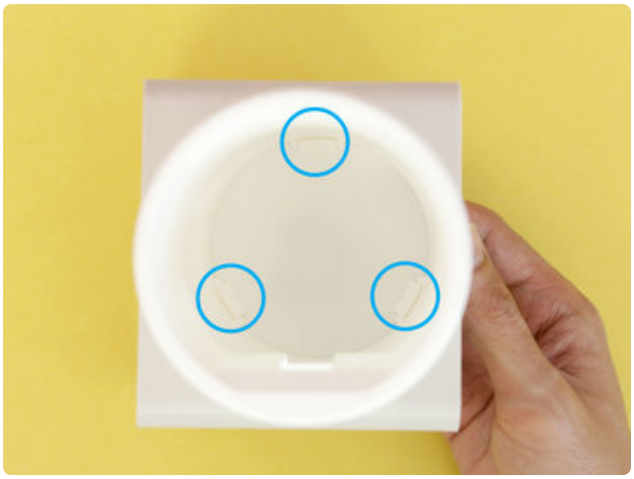
Planter Assembly

Install Planter to Frame

The planter cup is inserted into the frame. Insert the cup from the top of the frame so it's about halfway through.

Install Drip Tray

The drip tray will need to be installed next. Grab the drip tray and begin to place it under the planter cup.



Snap Fit Drip Tray

Line up the snap fits on the drip tray with the nubs on the bottom of the planter cup. Firmly press the two parts to snap fit them together.

Snap Fit Planter

With the drip tray now installed, begin to push the planter cup all the way into the frame. Nubs on the outside of the cup are used for snap fitting into the frame. Firmly press the planter cup into the frame.

Installed Planter

The planter cup is held in place with the snap fits. Rotate the planter cup so the flat surface is parallel with the front facing open end.

Install Soil Sensor

Insert the STEMMA cable through the hole in the planter cup. The cable is inserted through inside the planter.

Install Sensor Plate

Fit the sensor plate into the opening on the side of the planter cup. Position the plate so the mounting tabs are lined up with the mounting holes on the planter cup.

Secure Sensor Plate

Insert two M2.5 x 6mm long screws through the mounting holes on the planter cup. Install two M2.5 hex nuts onto the threads of the screws. Use a screw driver to fasten the screws onto the hex nuts.

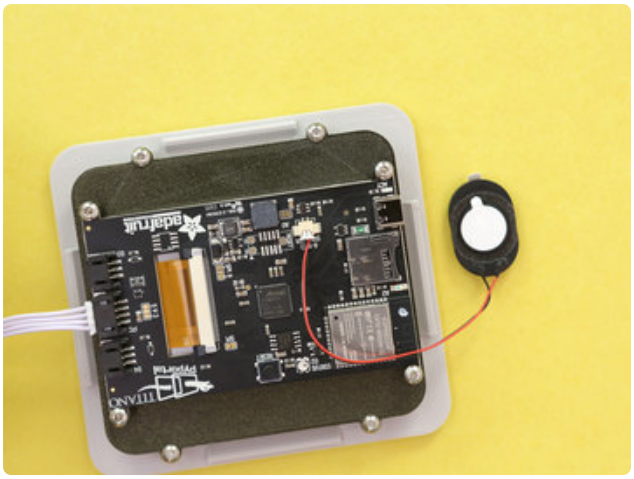
Installed Soil Sensor Planter

The soil sensor is mounted to the planter cup with the capacitive tip being inside and the electrical components on the outside. This keeps any dirt and water away from the components on-board the soil sensor.

Case Assembly

Connect Soil Sensor to PyPortal

Grab the STEMMA cable from the soil sensor and plug it into the STEMMA I2C port on the back of the PyPortal Titano.



Connect Speaker

Grab the cable from the speaker and connect it to the speaker port on the back of the PyPortal Titano.

Install Face Plate to Frame

Insert the PyPortal into the frame. Orient the PyPortal so the USB-C port is lined up with the cut out on the side of the frame. Adjust the wiring from the speaker and soil sensor so they're inside the frame and not being kinked. Press the face plate into the frame firmly to snap fit into place.

Install Speaker to Frame

Fit the speaker into the holder on the side of the case. Press the speaker into the holder from inside the frame.

Install Back Cover

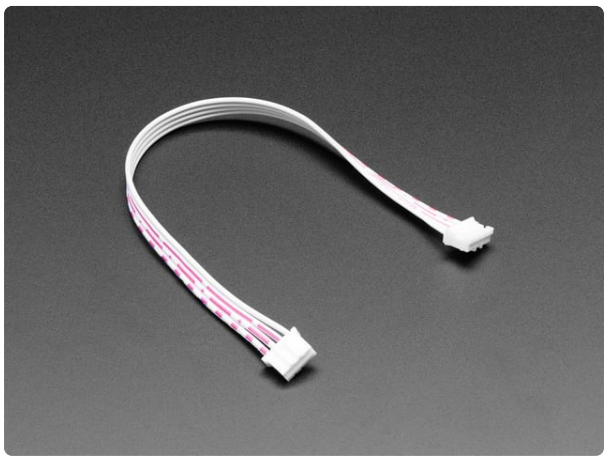
Grab the back cover and orient the snaps with the nubs on the side of the frame. Press the cover into the frame firmly to snap fit into place.

Final Build

And there you have it! The build is ready for potting with soil and watering. The planter has a diameter of 79mm (3in) and a depth of 93mm (3.6in). The drip tray has a depth of 16mm (0.6in).

PyPortal Wiring

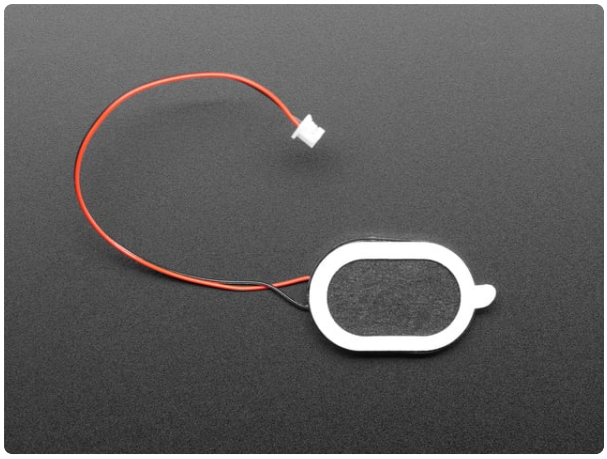
We recommend using a Female-to-Female Stemma Connector and plugging it in between the PyPortal and the STEMMA Soil Sensor. No soldering is involved - just connect the cable between the Stemma Soil Sensor and the PyPortal's I2C port.



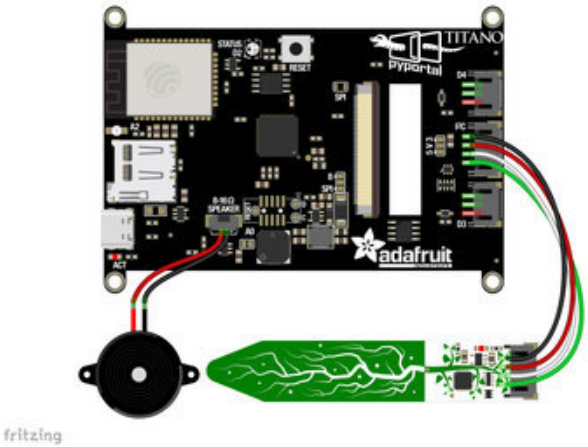
STEMMA Cable - 4 Pin JST-PH 2mm Cable–Female/Female

This 4-wire cable is a little over 150mm / 6" long and fitted with JST-PH female 4-pin connectors on each end. These types of JST cables are commonly found on small rechargeable...

https://www.adafruit.com/product/3568



[Mini Oval Speaker - 8 Ohm 1 Watt]
Hear the good news! This wee speaker is a great addition to any audio project where you need 8 ohm impedance and 1W or less of power. We particularly like...
https://www.adafruit.com/product/3923



The cable makes the following connections between the PyPortal's I2C port and the STEMMA Soil Sensor:

PyPortal 5V to **Sensor VIN**
PyPortal GND to **Sensor GND**
PyPortal SCL to **Sensor SCL**
PyPortal SDA to **Sensor SDA**
The speaker connects to the speaker port on the back of the PyPortal.

That's it - your PyPortal is wired up!

If you built a 3D Printed Planter, the sensor can be embedded into the planter cup.

If you do not want to 3D print an enclosure, you may use any planter and insert the STEMMA soil sensor into the pot. You may also want to position the sensor at the edge of your plant's pot.

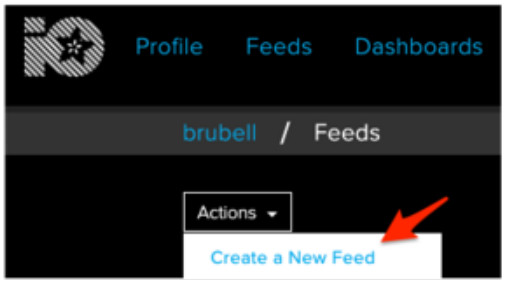
Adafruit IO Setup

Adafruit IO (https://adafru.it/fsU) is a platform designed by Adafruit to display, respond, and interact with your project's data. Adafruit keeps your data **private** (data feeds are private by default) and secure (we will never sell or give this data away to another company (https://adafru.it/BlI)) for you. It's the internet of things - for everyone!

If you do not already have an Adafruit IO account set up, head over to io.adafruit.com (https://adafru.it/fH9) to link your Adafruit.com account to Adafruit IO.

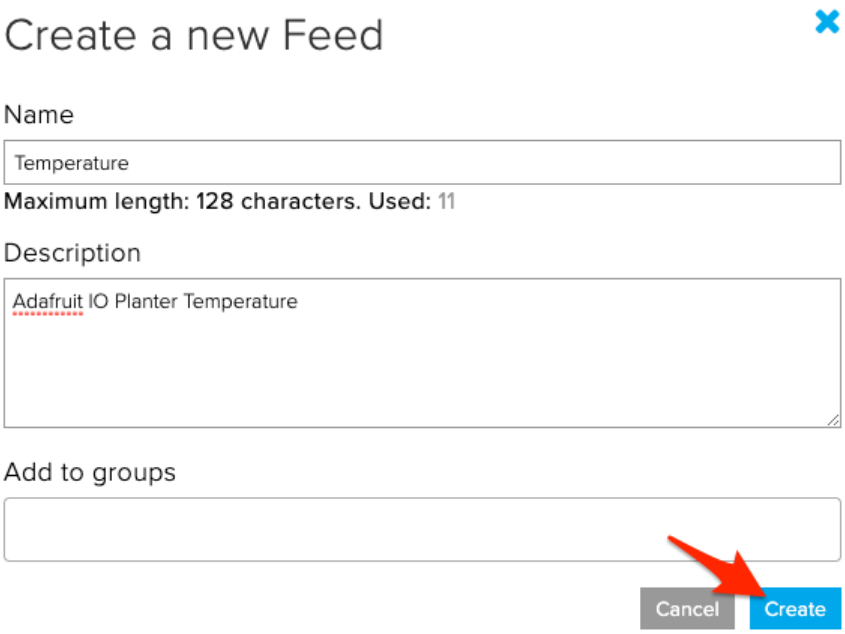
Create Feeds

Feeds are the core of the Adafruit IO system. The feed holds metadata about the data you push to Adafruit IO. We'll be creating two feeds - **temperature** and **moisture.**



Navigate to the Adafruit IO feeds page. (https://adafru.it/mxC)
Click Actions. From the dropdown, **click Create a New Feed**.

Change the feed name to Temperature. We may also optionally add a description here. **Click Create** to create a new feed.

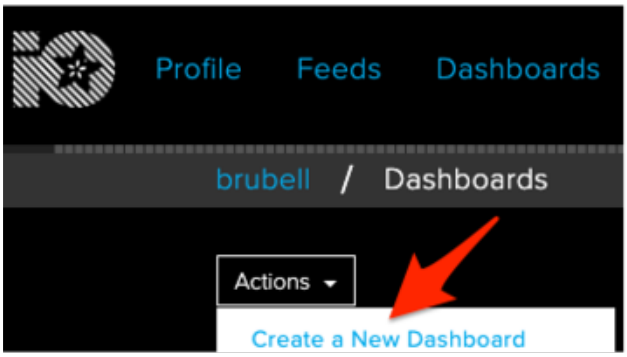


Then, **create another feed** called moisture.

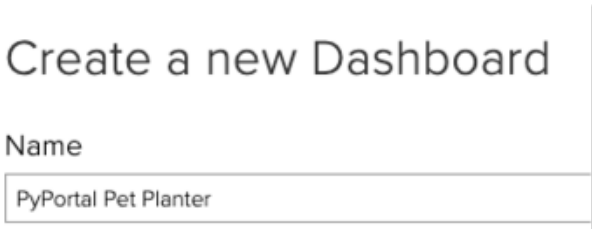
Build a Dashboard

Next, we'll create a dashboard to display the values from the feeds we created.

- For a primer about Adafruit IO Dashboards, head over to the Adafruit IO Basics: Dashboards (https://adafru.it/f5m) guide.



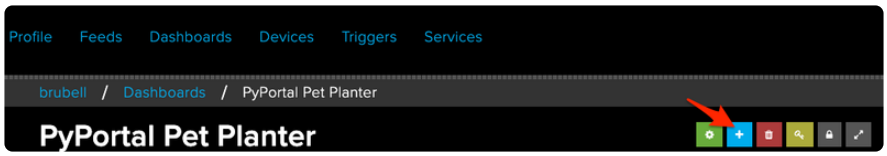
Navigate to the Adafruit IO Dashboards page (https://adafru.it/eIS).
From the Actions dropdown, **click Create a New Dashboard**.
Name the dashboard **PyPortal Pet Planter**



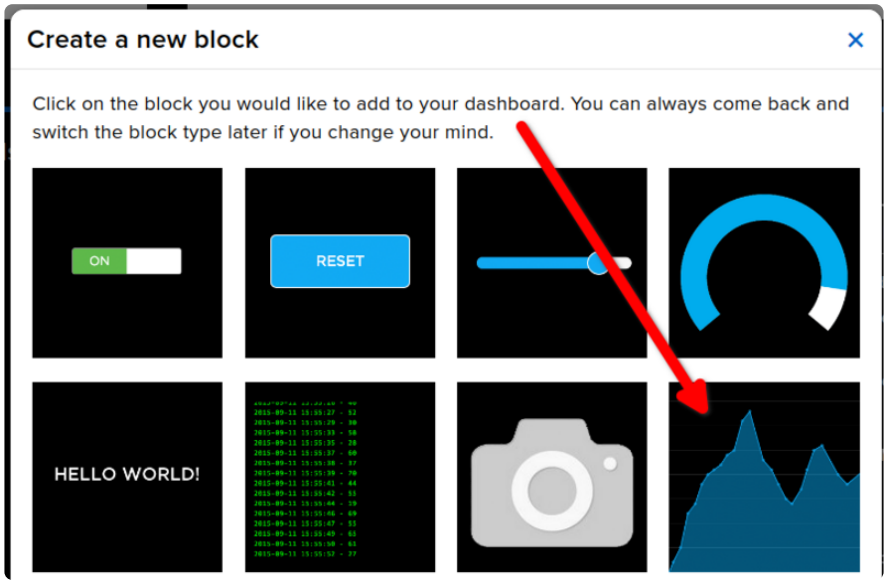
Add a Line Chart

The STEMMA Soil Sensor provides a soil moisture reading from very dry to very wet along with an ambient temperature reading from the internal temperature sensor on the microcontroller.

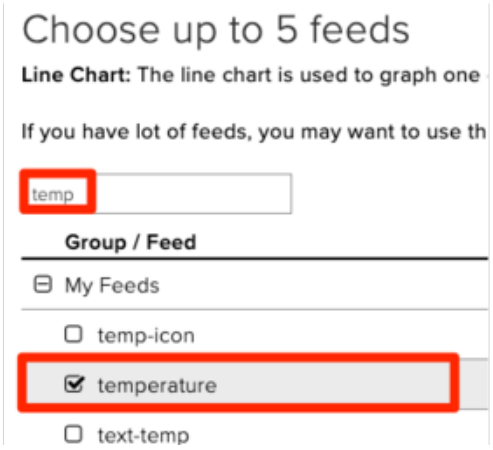
From the dashboard, **click the + button to create a new block.**



From the modal, **Click the Line Chart** block.



The line chart can graph one or more feeds.



The line chart can graph one or more feeds. We'll be graphing the moisture and temperature feed values to see how they change over time.

Search for the temperature feed. **Click the checkbox**. Then, **search** for the moisture feed. **Click the checkbox**.

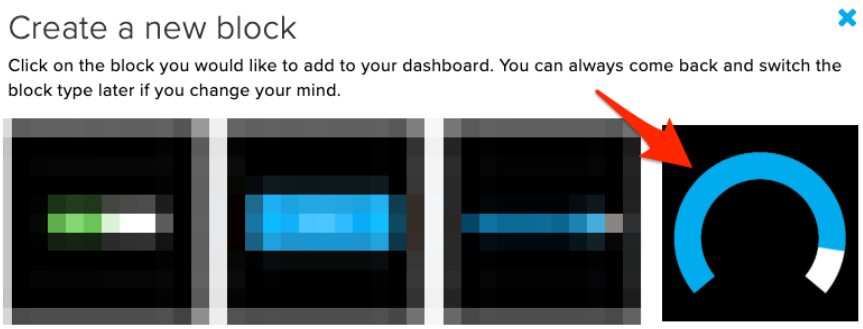
Click Next Step

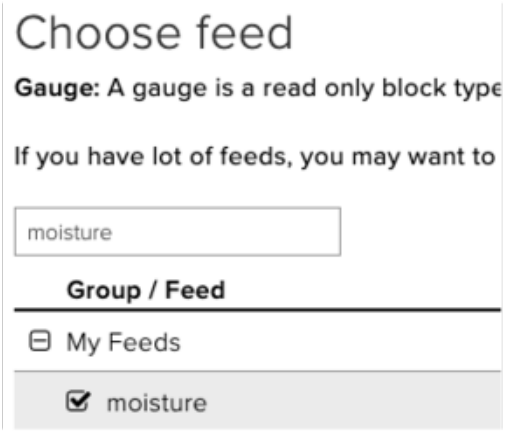
Name the Block Title as Temperature and Moisture

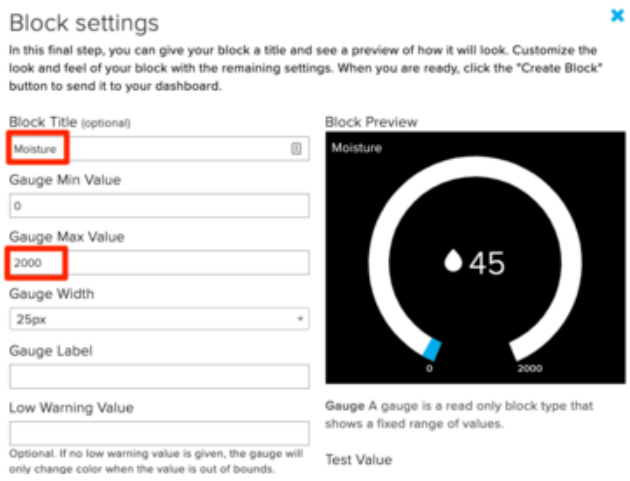
Set Show History to 4 hours

Add Gauges

A gauge is a read only block type that shows a fixed range of values. We'll add two gauge blocks to the dashboard to display values from the planter in real-time.







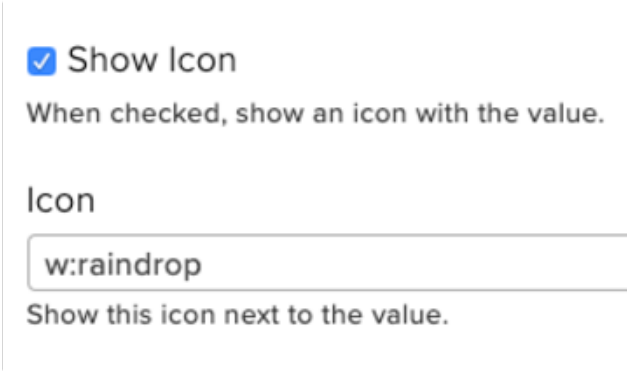
From the feed list, **select the moisture feed**.

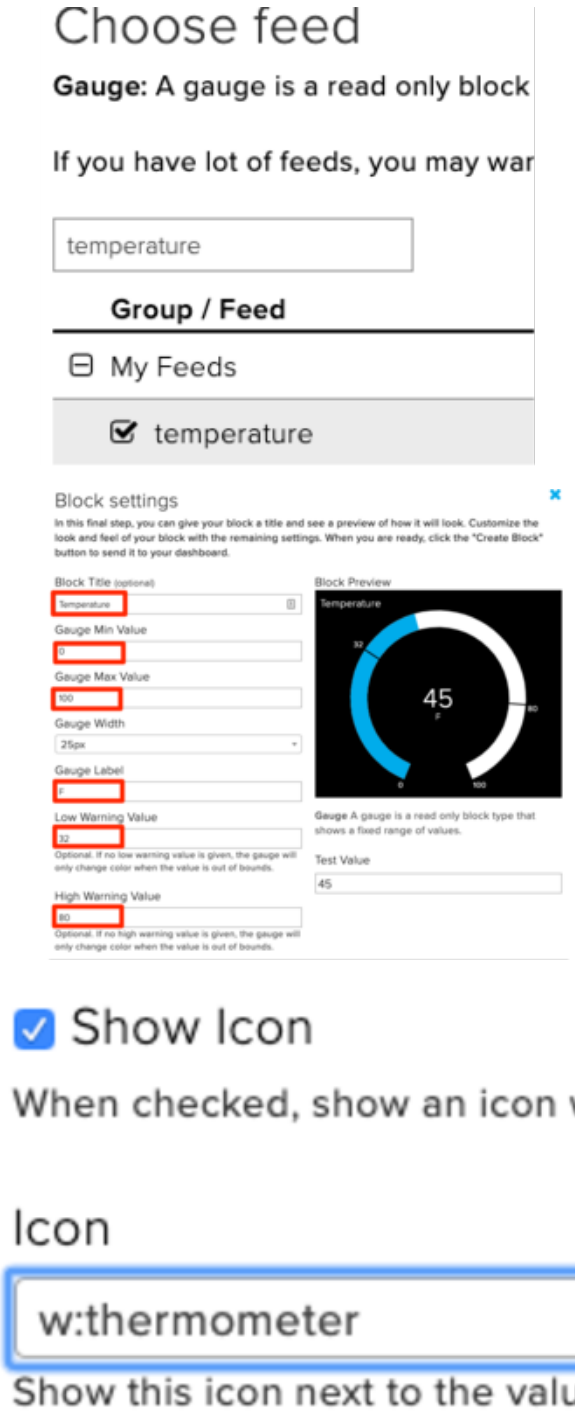
Set Block Title to **Moisture**.

The moisture sensor can read values up to 2000, **set Gauge Max Value** to **2000**.

Tick the Show Icon button and **set the icon** to **w:raindrop**.

Click Save.





Let's make another gauge block to display the temperature sensor's value.

Click Create a new block.

Set the **Block Title** to **Temperature**.

Set the **Gauge Label** to **F**

We can optionally set the Low Warning Value or High Warning Value. The gauge will change color when the value is out of bounds.

Tick the Show Icon button and **set the icon** to **w:thermometer**.

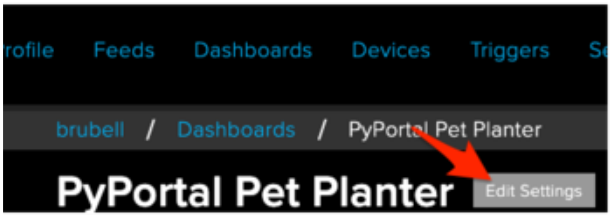
Click Save.

Add Header Image

This OPTIONAL step requires an an active Adafruit IO PLUS subscription (https://io.adafruit.com/plus)

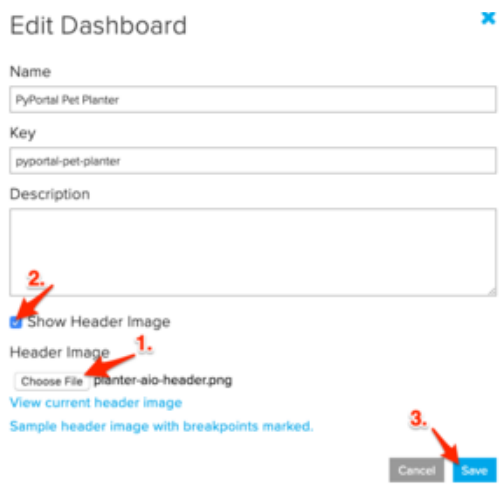
If your account has an active Adafruit IO **Plus** subscription (https://adafru.it/Eg3), you can optionally add a header image to customize your dashboard.

Download the image by right-clicking the button below and saving it to your computer.



Click the Edit Dashboard Layout (Green Gear) **button**.

Next to the dashboard's title, **click Edit Settings**.

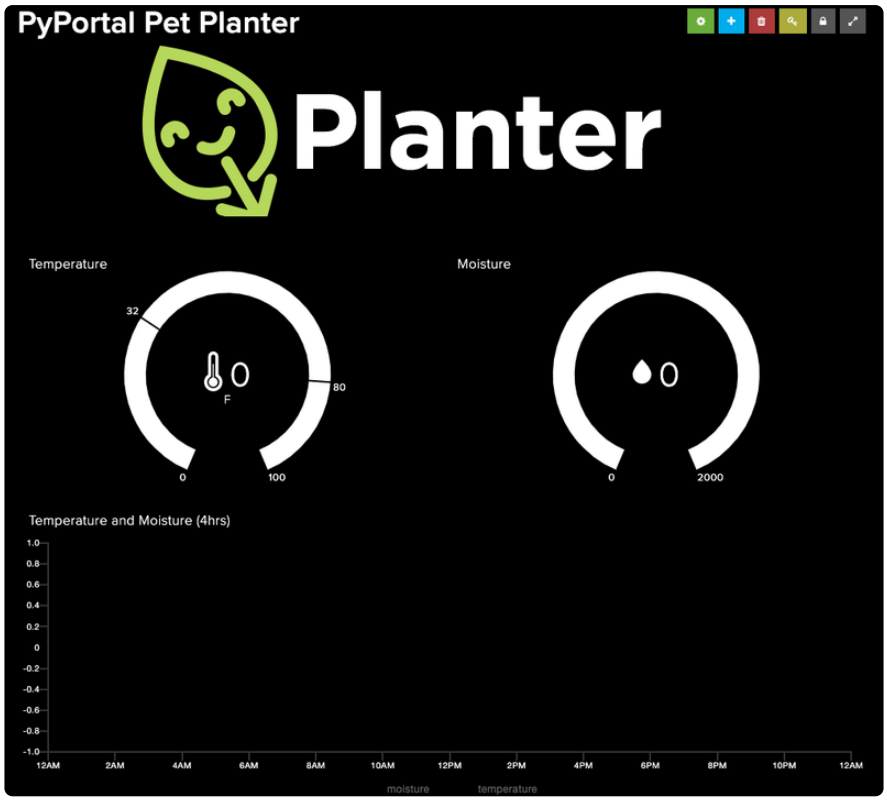


Under Header Image, **select the image** you downloaded in the previous step.

Tick the checkbox next to **Show Header Image**

Click Save.

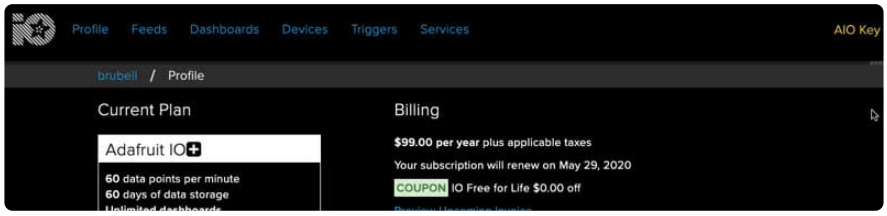
Your dashboard should look like the screenshot below.



Obtain Adafruit IO Key

You are also going to need your Adafruit IO username and secret API key.

Navigate to your profile **(https://adafru.it/EpC) and click the View AIO Key button** to retrieve them. Write them down in a safe place, you'll need them for the next step.



Now that we've set up our Adafruit IO account, let's move on to setting up the PyPortal Titano.

PyPortal CircuitPython Setup

To use all the amazing features of your PyPortal with CircuitPython, you must first install a number of libraries. This page covers that process.

Adafruit CircuitPython Bundle

Download the Adafruit CircuitPython Library Bundle. You can find the latest release here:

Latest Adafruit CircuitPython Library Bundle

https://adafru.it/ENC

Download the **adafruit-circuitpython-bundle-*.x-mpy-*.zip** bundle zip file where ***.x MATCHES THE VERSION OF CIRCUITPYTHON YOU INSTALLED**, and unzip a folder of the same name. Inside you'll find a **lib** folder. You have two options:

- You can add the **lib** folder to your **CIRCUITPY** drive. This will ensure you have all the drivers. But it will take a bunch of space on the 8 MB disk
- Add each library as you need it, this will reduce the space usage but you'll need to put in a little more effort.

At a minimum we recommend the following libraries, in fact we more than recommend. They're basically required. So grab them and install them into **CIRCUITPY/lib** now!

- **adafruit_esp32spi** - This is the library that gives you internet access via the ESP32 using (you guessed it!) SPI transport. You need this for anything Internet
- **adafruit_requests** - This library allows us to perform HTTP requests and get responses back from servers. GET/POST/PUT/PATCH - they're all in here!
- adafruit_connection_manager - used by adafruit_requests.
- **adafruit_pyportal** - This is our friendly wrapper library that does a lot of our projects, displays graphics and text, fetches data from the internet. Nearly all of our projects depend on it!
- **adafruit_portalbase** - This library is the base library that adafruit_pyportal library is built on top of.
- **adafruit_touchscreen** - a library for reading touches from the resistive touchscreen. Handles all the analog noodling, rotation and calibration for you.
- **adafruit_io** - this library helps connect the PyPortal to our free datalogging and viewing service
- **adafruit_imageload** - an image display helper, required for any graphics!
- **adafruit_display_text** - not surprisingly, it displays text on the screen
- **adafruit_bitmap_font** - we have fancy font support, and its easy to make new fonts. This library reads and parses font files.

- **adafruit_slideshow** - for making image slideshows - handy for quick display of graphics and sound
- **neopixel** - for controlling the onboard neopixel
- **adafruit_adt7410** - library to read the temperature from the on-board Analog Devices ADT7410 precision temperature sensor (not necessary for Titano or Pynt)
- **adafruit_bus_device** - low level support for I2C/SPI
- **adafruit_fakerequests** - This library allows you to create fake HTTP requests by using local files.

Create Your settings.toml File

CircuitPython works with WiFi-capable boards to enable you to make projects that have network connectivity. This means working with various passwords and API keys. As of CircuitPython 8 (https://adafru.it/Em8), there is support for a **settings.toml** file. This is a file that is stored on your **CIRCUITPY** drive, that contains all of your secret network information, such as your SSID, SSID password and any API keys for IoT services. It is designed to separate your sensitive information from your **code.py** file so you are able to share your code without sharing your credentials.

CircuitPython previously used a **secrets.py** file for this purpose. The **settings.toml** file is quite similar.

Your settings.toml file should be stored in the main directory of your CIRCUITPY drive. It should not be in a folder.

CircuitPython **settings.toml** File

This section will provide a couple of examples of what your **settings.toml** file should look like, specifically for CircuitPython WiFi projects in general.

The most minimal **settings.toml** file must contain your WiFi SSID and password, as that is the minimum required to connect to WiFi. Copy this example, paste it into your **settings.toml**, and update:

- `your_wifi_ssid`
- `your_wifi_password`

```
CIRCUITPY_WIFI_SSID = "your_wifi_ssid"
CIRCUITPY_WIFI_PASSWORD = "your_wifi_password"
```

Many CircuitPython network-connected projects on the Adafruit Learn System involve using Adafruit IO. For these projects, you must also include your Adafruit IO username and key. Copy the following example, paste it into your settings.toml file, and update:

- `your_wifi_ssid`
- `your_wifi_password`
- `your_aio_username`
- `your_aio_key`

```
CIRCUITPY_WIFI_SSID = "your_wifi_ssid"
CIRCUITPY_WIFI_PASSWORD = "your_wifi_password"
ADAFRUIT_AIO_USERNAME = "your_aio_username"
ADAFRUIT_AIO_KEY = "your_aio_key"
```

Some projects use different variable names for the entries in the **settings.toml** file. For example, a project might use `ADAFRUIT_AIO_ID` in the place of `ADAFRUIT_AIO_USERNAME`. **If you run into connectivity issues, one of the first things to check is that the names in the settings.toml file match the names in the code.**

Not every project uses the same variable name for each entry in the settings.toml file! Always verify it matches the code.

settings.toml File Tips

Here is an example **settings.toml** file.

```
# Comments are supported
CIRCUITPY_WIFI_SSID = "guest wifi"
CIRCUITPY_WIFI_PASSWORD = "guessable"
CIRCUITPY_WEB_API_PORT = 80
CIRCUITPY_WEB_API_PASSWORD = "passw0rd"
test_variable = "this is a test"
thumbs_up = "\U0001f44d"
```

In a **settings.toml** file, it's important to keep these factors in mind:

- Strings are wrapped in double quotes; ex: `"your-string-here"`
- Integers are **not** quoted and may be written in decimal with optional sign (`+1`, `-1`, `1000`) or hexadecimal (`0xabcd`).
    - Floats, octal (`0o567`) and binary (`0b11011`) are not supported.

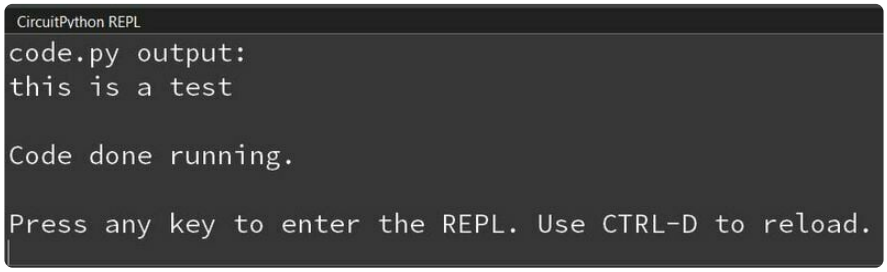
- Use `\u` escapes for weird characters, `\x` and `\ooo` escapes are not available in **.toml** files
    - Example: `\U0001f44d` for (thumbs up emoji) and `\u20ac` for € (EUR sign)
- Unicode emoji, and non-ASCII characters, stand for themselves as long as you're careful to save in "UTF-8 without BOM" format

When your **settings.toml** file is ready, you can save it in your text editor with the **.toml** extension.

Accessing Your **settings.toml** Information in **code.py**

In your **code.py** file, you'll need to `import` the `os` library to access the **settings.toml** file. Your settings are accessed with the `os.getenv()` function. You'll pass your settings entry to the function to import it into the **code.py** file.

```
import os

print(os.getenv("test_variable"))
```



In the upcoming CircuitPython WiFi examples, you'll see how the **settings.toml** file is used for connecting to your SSID and accessing your API keys.

Internet Connect!

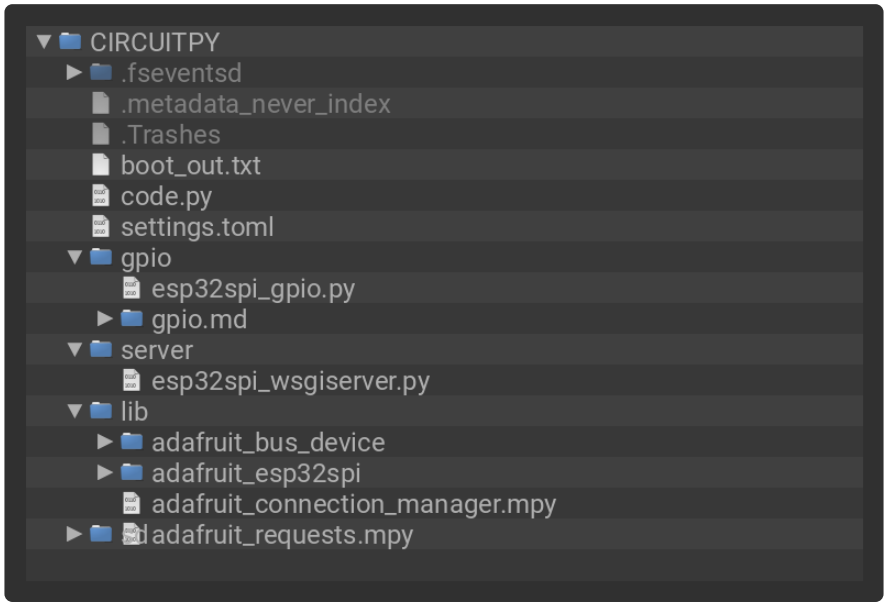
Connect to WiFi

OK, now that you have your **settings.toml** file set up - you can connect to the Internet.

To do this, you need to first install a few libraries, into the lib folder on your **CIRCUITPY** drive. Then you need to update **code.py** with the example script.

Thankfully, we can do this in one go. In the example below, click the **Download Project Bundle** button below to download the necessary libraries and the **code.py** file in a zip file. Extract the contents of the zip file, open the directory **examples/** and then click on the directory that matches the version of CircuitPython you're using and copy the contents of that directory to your **CIRCUITPY** drive.

Your **CIRCUITPY** drive should now look similar to the following image:




```
# 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"),
```

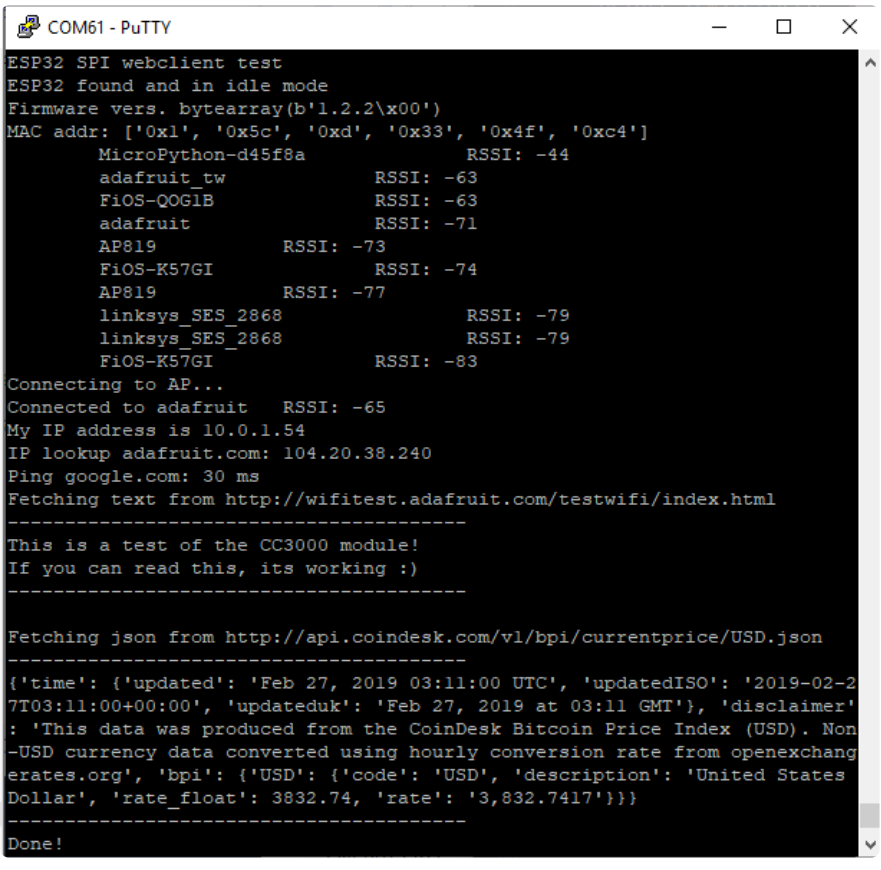
```
    "password": getenv("CIRCUITPY_WIFI_PASSWORD"),
}
if secrets == {"ssid": None, "password": None}:
    try:
        # Fallback on secrets.py until depreciation is over and option is removed
        from secrets import secrets
    except ImportError:
        print("WiFi secrets are kept in settings.toml, 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**.

Don't forget you'll also need to create the **settings.toml** file as seen above, with your WiFi ssid and password.

In a serial console, you should see something like the following. For more information about connecting with a serial console, view the guide Connecting to the Serial Console (https://adafru.it/Bec).



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)
```

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...")
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)
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"))
```

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/Kpa) 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.

To use with CircuitPython, you need to first install a few libraries, into the lib folder on your **CIRCUITPY** drive. Then you need to update **code.py** with the example script.

Thankfully, we can do this in one go. In the example below, click the **Download Project Bundle** button below to download the necessary libraries and the **code.py** file in a zip file. Extract the contents of the zip file, open the directory **examples/** and then click on the directory that matches the version of CircuitPython you're using and copy the contents of that directory to your **CIRCUITPY** drive.

Your **CIRCUITPY** drive should now look similar to the following image:

CIRCUITPY

```
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 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(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)
```

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`.

Having requested data from the server, we'd now like to see 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.

To use with CircuitPython, you need to first install a few libraries, into the lib folder on your **CIRCUITPY** drive. Then you need to update **code.py** with the example script.

Thankfully, we can do this in one go. In the example below, click the **Download Project Bundle** button below to download the necessary libraries and the **code.py** file in a zip file. Extract the contents of the zip file, open the directory **examples/** and then click on the directory that matches the version of CircuitPython you're using and copy the contents of that directory to your **CIRCUITPY** drive.

Your **CIRCUITPY** drive should now look similar to the following image:

CIRCUITPY

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

WiFi Manager

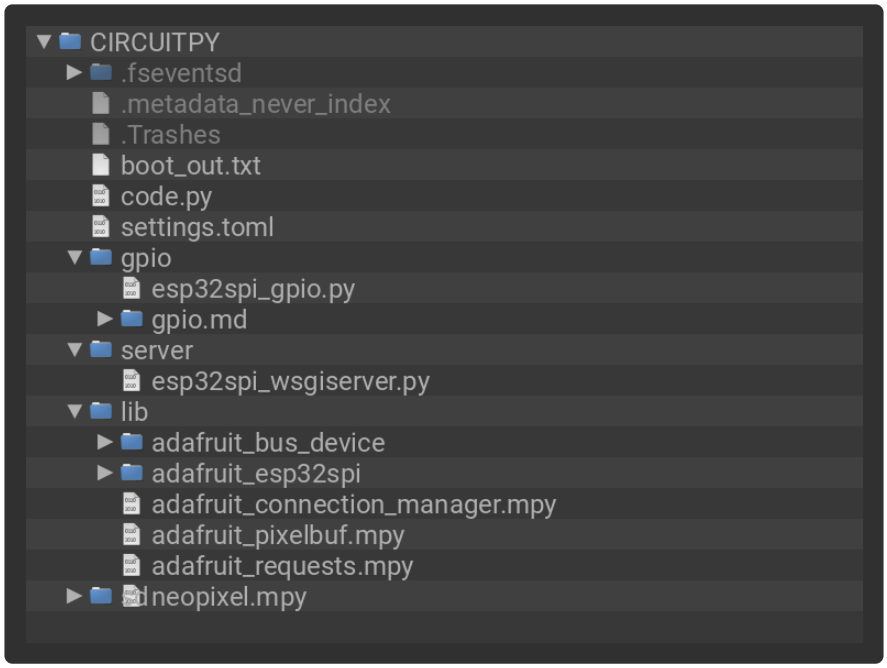
That simpletest example works but it's a little finicky - you need to constantly check WiFi status and have many loops to manage connections and disconnections. For more advanced uses, we recommend using the WiFiManager object. It will wrap the connection/status/requests loop for you - reconnecting if 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:

To use with CircuitPython, you need to first install a few libraries, into the lib folder on your **CIRCUITPY** drive. Then you need to update **code.py** with the example script.

Thankfully, we can do this in one go. In the example below, click the **Download Project Bundle** button below to download the necessary libraries and the **code.py** file in a zip file. Extract the contents of the zip file, open the directory **examples/** and then click on the directory that matches the version of CircuitPython you're using and copy the contents of that directory to your **CIRCUITPY** drive.

Your **CIRCUITPY** drive should now look similar to the following image:




```
# 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)
```

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_',
    }
```

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)

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 Setup

CircuitPython Library Installation

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

Add CircuitPython Code and Project Assets

In the embedded code element below, click on the **Download Project Bundle** button below, and save the .zip archive file to your computer.

Then, **uncompress the .zip file**, it will unpack to a folder named **pyportal_pet_planter**.

Copy the contents of **pyportal_pet_planter** directory to your PyPortal **CIRCUITPY** drive.

Be sure to copy the **fonts**, **images**, and **sounds** directories from the Project Bundle zip file to the PyPortal **CIRCUITPY** drive in addition to the libraries in **lib** and **code.py**.

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

import time

import board
import busio
from digitalio import DigitalInOut
import adafruit_connection_manager
from adafruit_esp32spi import adafruit_esp32spi, adafruit_esp32spi_wifimanager
import adafruit_imageload
import displayio
import neopixel
from adafruit_bitmap_font import bitmap_font
from adafruit_display_text.label import Label
from adafruit_io.adafruit_io import IO_MQTT
import adafruit_minimqtt.adafruit_minimqtt as MQTT
from adafruit_pyportal import PyPortal
from adafruit_seesaw.seesaw import Seesaw
from simpleio import map_range

#---| User Config |---------------

# How often to poll the soil sensor, in seconds
# Polling every 30 seconds or more may cause connection timeouts
DELAY_SENSOR = 15

# How often to send data to adafruit.io, in minutes
DELAY_PUBLISH = 5

# Maximum soil moisture measurement
SOIL_LEVEL_MAX = 500.0

# Minimum soil moisture measurement
SOIL_LEVEL_MIN= 350.0

#---| End User Config |---------------

# Background image
BACKGROUND = "/images/roots.bmp"
# Icons for water level and temperature
ICON_LEVEL = "/images/icon-wetness.bmp"
ICON_TEMP = "/images/icon-temp.bmp"
WATER_COLOR = 0x16549E

# Audio files
```

```
wav_water_high = "/sounds/water-high.wav"
wav_water_low = "/sounds/water-low.wav"

# the current working directory (where this file is)
cwd = ("/"+__file__).rsplit('/', 1)[0]

# 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

# Set up i2c bus
i2c_bus = busio.I2C(board.SCL, board.SDA)

# Initialize soil sensor (s.s)
ss = Seesaw(i2c_bus, addr=0x36)

# PyPortal ESP32 AirLift 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)
status_light = neopixel.NeoPixel(board.NEOPIXEL, 1, brightness=0.2)
wifi = adafruit_esp32spi_wifimanager.ESPSPI_WiFiManager(esp, secrets, status_light)

# Initialize PyPortal Display
display = board.DISPLAY

WIDTH = board.DISPLAY.width
HEIGHT = board.DISPLAY.height

# Initialize new PyPortal object
pyportal = PyPortal(esp=esp,
                    external_spi=spi)

# Set backlight level
pyportal.set_backlight(0.5)

# Create a new DisplayIO group
splash = displayio.Group()

# show splash group
display.root_group = splash

# Palette for water bitmap
palette = displayio.Palette(2)
palette[0] = 0x000000
palette[1] = WATER_COLOR
palette.make_transparent(0)

# Create water bitmap
water_bmp = displayio.Bitmap(display.width, display.height, len(palette))
water = displayio.TileGrid(water_bmp, pixel_shader=palette)
splash.append(water)

print("drawing background..")
# Load background image
try:
    bg_bitmap, bg_palette = adafruit_imageload.load(BACKGROUND,
                                                    bitmap=displayio.Bitmap,
                                                    palette=displayio.Palette)
# Or just use solid color
except (OSError, TypeError):
    BACKGROUND = BACKGROUND if isinstance(BACKGROUND, int) else 0x000000
    bg_bitmap = displayio.Bitmap(display.width, display.height, 1)
```

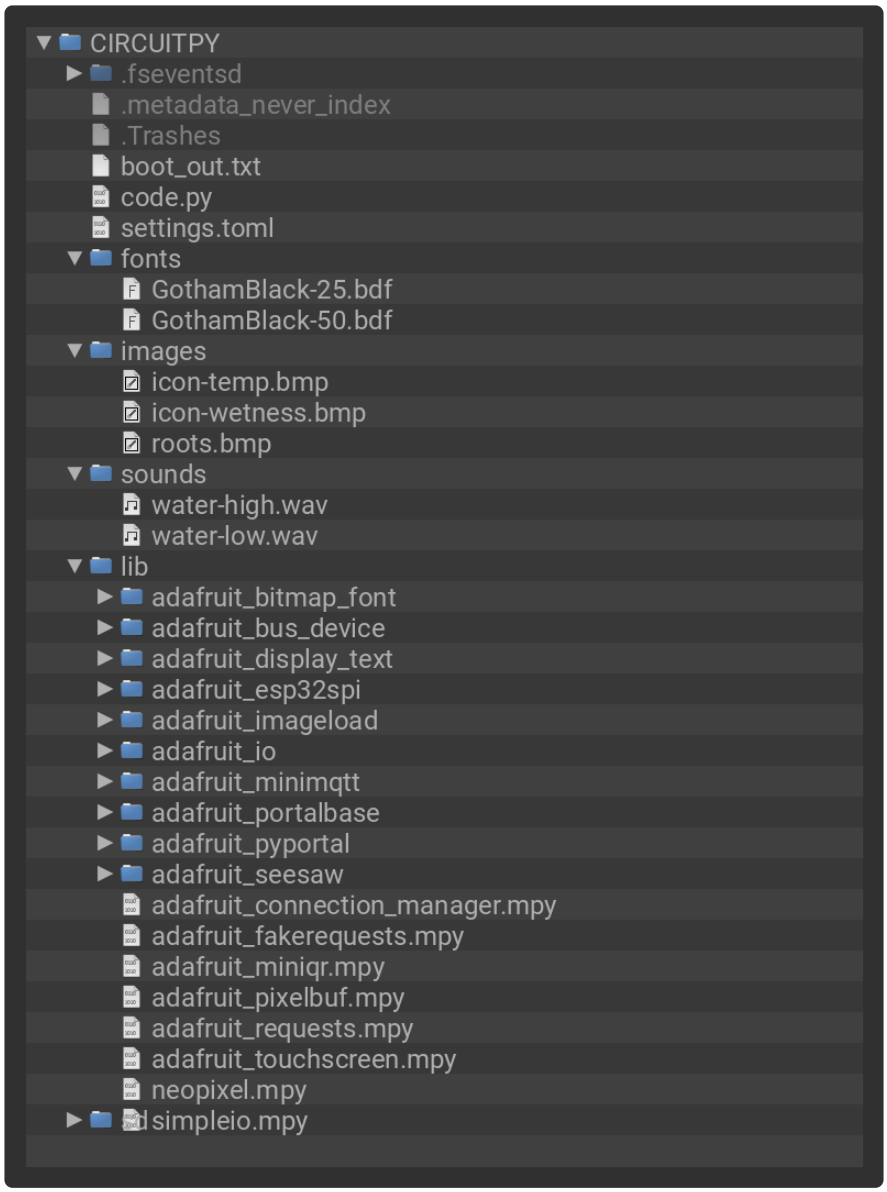
```
    bg_palette = displayio.Palette(1)
    bg_palette[0] = BACKGROUND
bg_palette.make_transparent(0)
background = displayio.TileGrid(bg_bitmap, pixel_shader=bg_palette)

# Add background to display
splash.append(background)

print('loading fonts...')
# Fonts within /fonts/ folder
font = cwd+"/fonts/GothamBlack-50.bdf"
font_small = cwd+"/fonts/GothamBlack-25.bdf"

# pylint: disable=syntax-error
data_glyphs = b'0123456789FC-* '
font = bitmap_font.load_font(font)
font.load_glyphs(data_glyphs)

font_small = bitmap_font.load_font(font_small)
full_glyphs = b'0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ-,.: '
font_small.load_glyphs(full_glyphs)

# Label to display Adafruit IO status
label_status = Label(font_small)
label_status.x = 305
label_status.y = 10
splash.append(label_status)

# Create a label to display the temperature
label_temp = Label(font)
label_temp.x = 35
label_temp.y = 300
splash.append(label_temp)

# Create a label to display the water level
label_level = Label(font)
label_level.x = display.width - 130
label_level.y = 300
splash.append(label_level)

print('loading icons...')
# Load temperature icon
icon_tmp_bitmap, icon_palette = adafruit_imageload.load(ICON_TEMP,
                                                        bitmap=displayio.Bitmap,
                                                        palette=displayio.Palette)
icon_palette.make_transparent(0)
icon_tmp_bitmap = displayio.TileGrid(icon_tmp_bitmap,
                                     pixel_shader=icon_palette,
                                     x=0, y=280)
splash.append(icon_tmp_bitmap)

# Load level icon
icon_lvl_bitmap, icon_palette = adafruit_imageload.load(ICON_LEVEL,
                                                        bitmap=displayio.Bitmap,
                                                        palette=displayio.Palette)
icon_palette.make_transparent(0)
icon_lvl_bitmap = displayio.TileGrid(icon_lvl_bitmap,
                                     pixel_shader=icon_palette,
                                     x=315, y=280)
splash.append(icon_lvl_bitmap)

# Connect to WiFi
label_status.text = "Connecting..."
while not esp.is_connected:
    try:
        wifi.connect()
    except (RuntimeError, ConnectionError) as e:
        print("could not connect to AP, retrying: ",e)
        wifi.reset()
```

```
        continue
print("Connected to WiFi!")

pool = adafruit_connection_manager.get_radio_socketpool(esp)
ssl_context = adafruit_connection_manager.get_radio_ssl_context(esp)

# Initialize a new MQTT Client object
mqtt_client = MQTT.MQTT(broker="io.adafruit.com",
                        username=secrets["aio_user"],
                        password=secrets["aio_key"],
                        socket_pool=pool,
                        ssl_context=ssl_context)

# Adafruit IO Callback Methods
# pylint: disable=unused-argument
def connected(client):
    # Connected function will be called when the client is connected to Adafruit IO.
    print('Connected to Adafruit IO!')

def subscribe(client, userdata, topic, granted_qos):
    # This method is called when the client subscribes to a new feed.
    print('Subscribed to {0} with QOS level {1}'.format(topic, granted_qos))

# pylint: disable=unused-argument
def disconnected(client):
    # Disconnected function will be called if the client disconnects
    # from the Adafruit IO MQTT broker.
    print("Disconnected from Adafruit IO!")

# Initialize an Adafruit IO MQTT Client
io = IO_MQTT(mqtt_client)

# Connect the callback methods defined above to the Adafruit IO MQTT Client
io.on_connect = connected
io.on_subscribe = subscribe
io.on_disconnect = disconnected

# Connect to Adafruit IO
print("Connecting to Adafruit IO...")
io.connect()
label_status.text = " "
print("Connected!")

fill_val = 0.0
def fill_water(fill_percent):
    """Fills the background water.
    :param float fill_percent: Percentage of the display to fill.

    """
    assert fill_percent <= 1.0, "Water fill value may not be > 100%"
    # pylint: disable=global-statement
    global fill_val

    if fill_val > fill_percent:
        for _y in range(int((board.DISPLAY.height-1) -
((board.DISPLAY.height-1)*fill_val)),
                        int((board.DISPLAY.height-1) -
((board.DISPLAY.height-1)*fill_percent))):
            for _x in range(1, board.DISPLAY.width-1):
                water_bmp[_x, _y] = 0
    else:
        for _y in range(board.DISPLAY.height-1,
                        (board.DISPLAY.height-1) -
((board.DISPLAY.height-1)*fill_percent), -1):
            for _x in range(1, board.DISPLAY.width-1):
                water_bmp[_x, _y] = 1
    fill_val = fill_percent

def display_temperature(temp_val, is_celsius=False):
```

```
    """Displays the temperature from the STEMMA soil sensor
    on the PyPortal Titano.
    :param float temp: Temperature value.
    :param bool is_celsius:

    """
    if not is_celsius:
        temp_val = (temp_val * 9 / 5) + 32 - 15
        print('Temperature: %0.0fF'%temp_val)
        label_temp.text = '%0.0fF'%temp_val
        return int(temp_val)
    else:
        print('Temperature: %0.0fC'%temp_val)
        label_temp.text = '%0.0fC'%temp_val
        return int(temp_val)

# initial reference time
initial = time.monotonic()
while True:
    # Explicitly pump the message loop
    # to keep the connection active
    try:
        io.loop()
    except (ValueError, RuntimeError, ConnectionError, OSError) as e:
        print("Failed to get data, retrying...\n", e)
        wifi.reset()
        continue
    now = time.monotonic()

    print("reading soil sensor...")
    # Read capactive
    moisture = ss.moisture_read()
    label_level.text = str(moisture)

    # Convert into percentage for filling the screen
    moisture_percentage = map_range(float(moisture), SOIL_LEVEL_MIN, SOIL_LEVEL_MAX,
0.0, 1.0)

    # Read temperature
    temp = ss.get_temp()
    temp = display_temperature(temp)

    # fill display
    print("filling disp..")
    fill_water(moisture_percentage)
    print("disp filled..")

    print("temp: " + str(temp) + "  moisture: " + str(moisture))

    # Play water level alarms
    if moisture <= SOIL_LEVEL_MIN:
        print("Playing low water level warning...")
        pyportal.play_file(wav_water_low)
    elif moisture >= SOIL_LEVEL_MAX:
        print("Playing high water level warning...")
        pyportal.play_file(wav_water_high)


    if now - initial > (DELAY_PUBLISH * 60):
        try:
            print("Publishing data to Adafruit IO...")
            label_status.text = "Sending to IO..."
            io.publish("moisture", moisture)
            io.publish("temperature", temp)
            print("Published")
            label_status.text = "Data Sent!"

            # reset timer
            initial = now
```

```
        except (ValueError, RuntimeError, ConnectionError, OSError) as e:
            label_status.text = "ERROR!"
            print("Failed to get data, retrying...\n", e)
            wifi.reset()
    time.sleep(DELAY_SENSOR)
```

This is what the final contents of the **CIRCUITPY** drive will look like:



Before continuing make sure your board's **lib** folder has the following files and folders copied over:

- **adafruit_bitmap_font**
- **adafruit_io**
- **adafruit_bus_device**
- **adafruit_logging.mpy**
- **adafruit_seesaw**
- **adafruit_display_text**

- adafruit_minimqtt.mpy
- adafruit_touchscreen.mpy
- adafruit_esp32spi
- adafruit_pyportal.mpy
- neopixel.mpy
- adafruit_imageload
- adafruit_requests.mpy
- simpleio.mpy

Install the Mu Editor

This guide requires you to edit and interact with CircuitPython code. While you can use any text editor of your choosing, **Mu** is a simple code editor that works with the Adafruit CircuitPython boards. It's written in Python and works on Windows, MacOS, Linux and Raspberry Pi. The serial console is built right in, so you get immediate feedback from your board's serial output!

Before proceeding, **click the button below to install the Mu Editor**. There are versions for PC, mac, and Linux.

Install Mu Editor

https://adafru.it/ANO

settings.toml File Setup

Open the **settings.toml** file on your **CIRCUITPY** drive using Mu or your own text editor. If you don't have one, use a text editor and copy the text below and edit from there. Add the credentials shown below, substituting your own credentials.

```
CIRCUITPY_WIFI_SSID = "your_wifi_ssid"
CIRCUITPY_WIFI_PASSWORD = "your_wifi_password"
AIO_USERNAME = "your_aio_username"
AIO_KEY = "your_aio_key"
```

Code Usage

Viewing Sensor Data on PyPortal

You should see the PyPortal display update displaying the temperature value and moisture level.

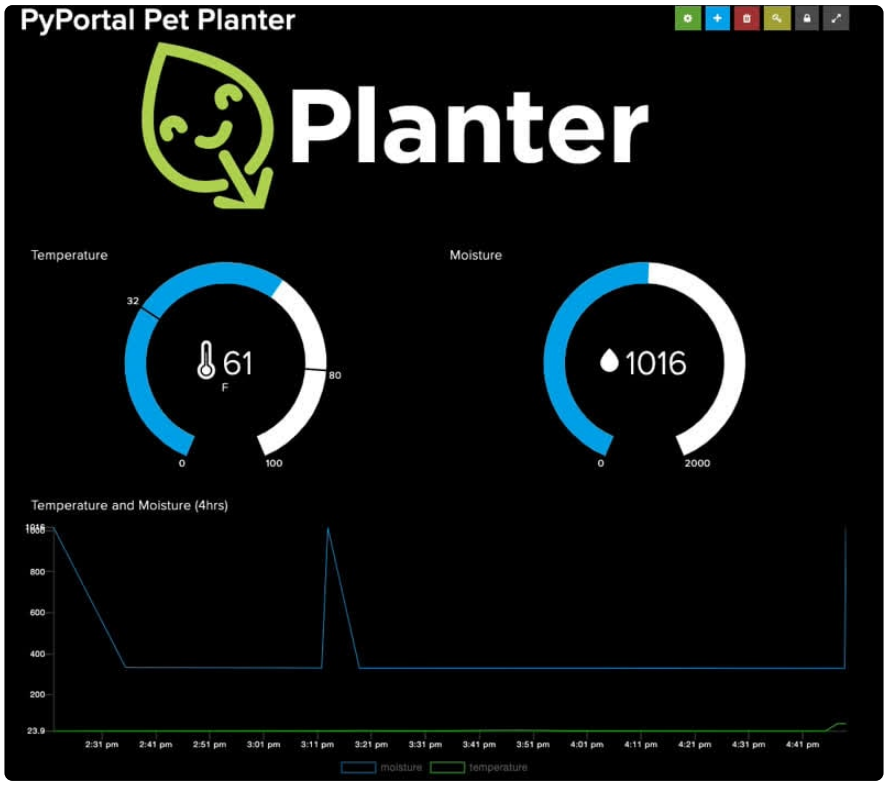
The code reads the sensor every `DELAY_SENSOR` seconds. Adjust this value in the code to increase or decrease this delay.

The status indicator at the bottom of the PyPortal will display when it's sending data to Adafruit IO.

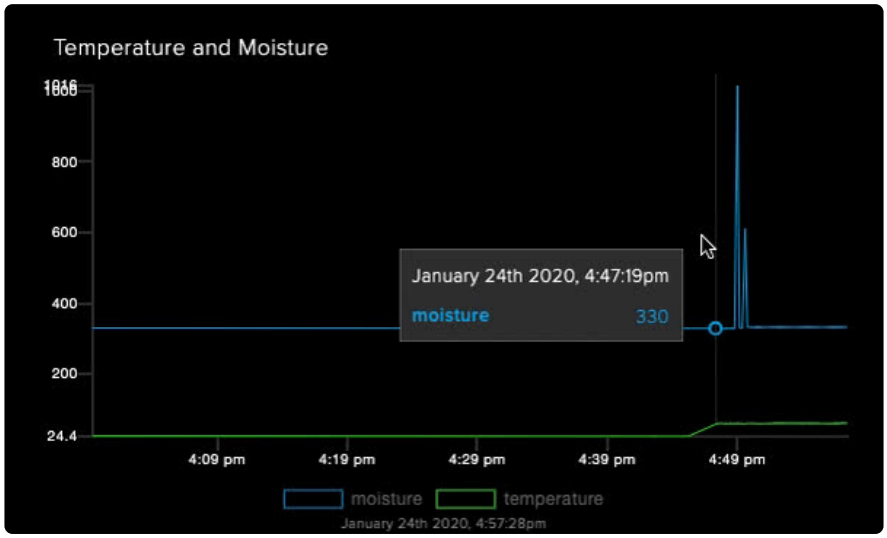
The PyPortal only sends data to Adafruit IO every `DELAY_PUBLISH` minutes. Adjust this value in the code to increase or decrease the delay.

Viewing Sensor Data on Adafruit IO

Navigate to the dashboard we created earlier. You'll notice the values on the gauges changing in real-time. The line graph block will also update depending on the interval you set.



Hovering your mouse over the line graph will display both the moisture and temperature values.



Adjusting Soil Moisture Levels

In soil, the moisture level will generally range from about 300 to 500. This value does depend on how packed/loose the soil is. If you'd like to adjust the threshold for if the soil is dry or wet, use this example sketch in the STEMMA Soil Sensor guide (https://adafru.it/IFv) to quickly determine the soil's moisture level.

Once you have a better idea of the moisture thresholds, adjust the `SOIL_LEVEL_MAX` and `SOIL_LEVEL_MIN` variables in your code.