Espresso Water Tank Meter

Created by John Park

https://learn.adafruit.com/espresso-water-tank-meter

Last updated on 2025-02-16 05:57:02 PM EST

Table of Contents

Overview

Does your espresso machine have a water reservoir that's inconvenient to check on? I do! Let's build a contactless remote water level meter that harnesses the power of "The INTERNET" to keep us informed without having to move all the cups, lift the lid, and peer into it like some kind of luddite! That's right, we'll set up an Adafruit IO feed and dashboard to spy on the water tank.

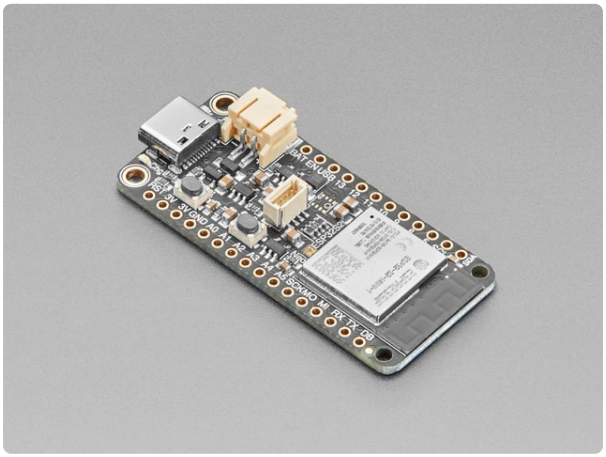
This can be done with an ESP32-S2 Feather and deep sleep mode to keep it low-power -- a 2200mAh battery should last over four months on a single charge.

Plus, it's ultrasonic, so you know your water is being secretly screamed at by a robot.

You'll have a beautiful dashboard to keep track of it all:

Note: the 3D printed tank lid in this guide fits the ECM Synchronika and Profitec Go but you should be able to modify the file to fit other brands/ makes.

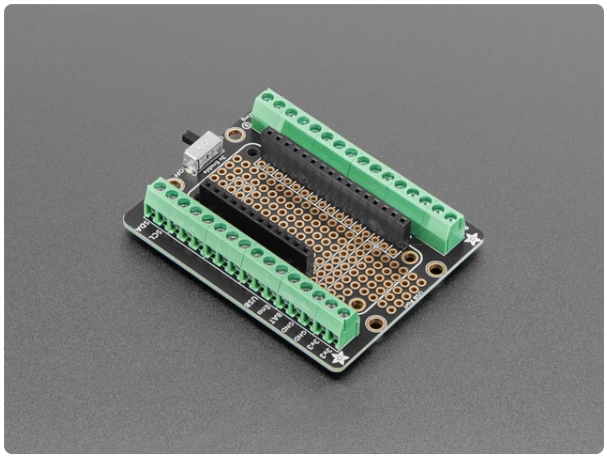
Parts



Adafruit ESP32-S2 Feather - 4 MB Flash + 2 MB PSRAM

What's Feather-shaped and has an ESP32-S2 WiFi module? What has a STEMMA QT connector for I2C devices? What has your favorite Espressif WiFi microcontroller and lots of Flash and...

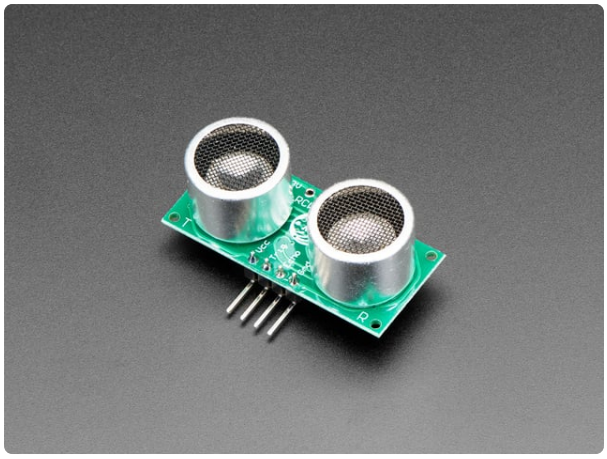
https://www.adafruit.com/product/5000



Assembled Terminal Block Breakout FeatherWing for all Feathers

The Terminal Block Breakout FeatherWing kit is like the Golden Eagle of prototyping FeatherWings (eg. majestic, powerful, good-looking). To start, you get a nice prototyping area...

https://www.adafruit.com/product/2926



Ultrasonic Distance Sensor - 3V or 5V - HC-SR04 compatible

If you're like me, you've dreamed of being a dolphin - smoothly gliding through the water. Using your echo-location abilities to detect tasty fish treats. Until genetic...

https://www.adafruit.com/product/4007

1 x 20-pin 0.1" Female Header — https://www.adafruit.com/product/4155

White - 5 pack

---

1 x Silicone Cover Stranded-Core Ribbon Cable — https://www.adafruit.com/product/3892

4 Wires 1 Meter Long - 26AWG Black

---

1 x Pre-Cut Multi-Colored Heat Shrink Pack Kit — https://www.adafruit.com/product/4559

280 pcs

---

1 x Black Nylon Machine Screw and Stand-off Set — https://www.adafruit.com/product/4685

M3 Thread

---

1 x Mini Magnet Feet for RGB LED Matrices — https://www.adafruit.com/product/4631

Pack of 4

CircuitPython

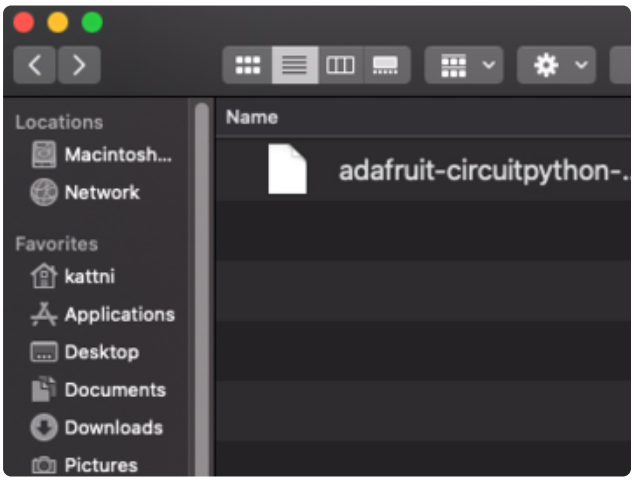
CircuitPython (https://adafru.it/tB7) is a derivative of MicroPython (https://adafru.it/BeZ) designed to simplify experimentation and education on low-cost microcontrollers. It makes it easier than ever to get prototyping by requiring no upfront desktop software downloads. Simply copy and edit files on the **CIRCUITPY** drive to iterate.

CircuitPython Quickstart

Follow this step-by-step to quickly get CircuitPython running on your board.

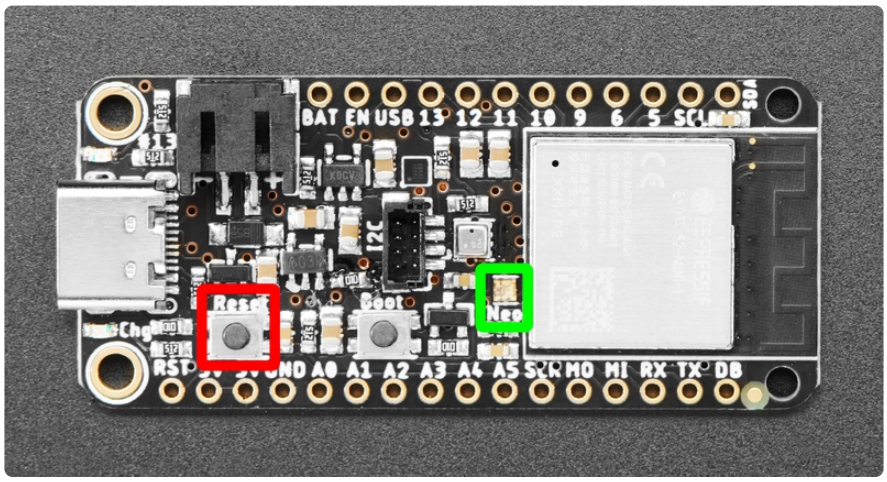
Download the latest version of CircuitPython for this board via circuitpython.org

https://adafru.it/WZd



Click the link above to download the latest CircuitPython UF2 file.

Save it wherever is convenient for you.



Plug your board into your computer, using a known-good data-sync cable, directly, or via an adapter if needed.

Click the **reset** button once (highlighted in red above), and then click it again when you see the **RGB status LED(s)** (highlighted in green above) turn purple (approximately half a second later). Sometimes it helps to think of it as a "slow double-click" of the reset button.

If you do not see the LED turning purple, you will need to reinstall the UF2 bootloader. See the **Factory Reset** page in this guide for details.

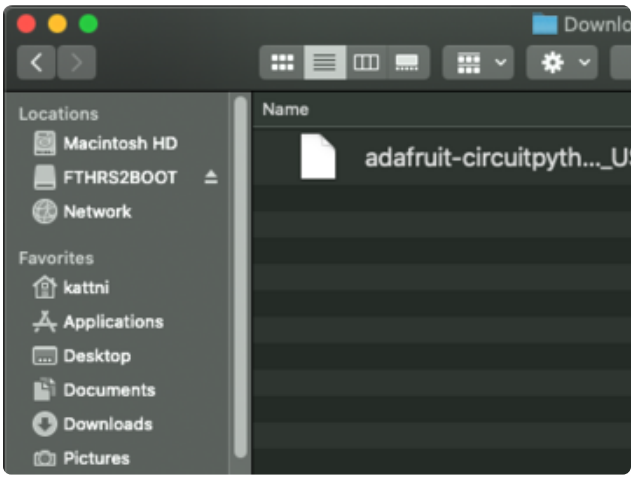
On some very old versions of the UF2 bootloader, the status LED turns red instead of purple.

Once successful, you will see the **RGB status LED(s)** turn green (highlighted in green above). If you see red, try another port, or if you're using an adapter or hub, try without the hub, or different adapter or hub.

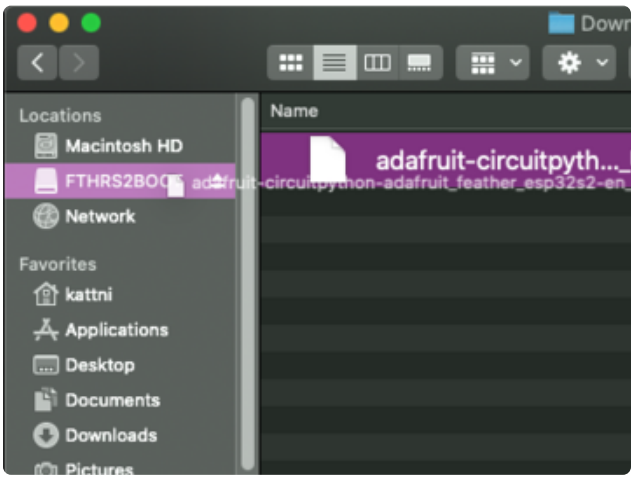
If double-clicking doesn't work the first time, try again. Sometimes it can take a few tries to get the rhythm right!

A lot of people end up using charge-only USB cables and it is very frustrating! **Make sure you have a USB cable you know is good for data sync.**

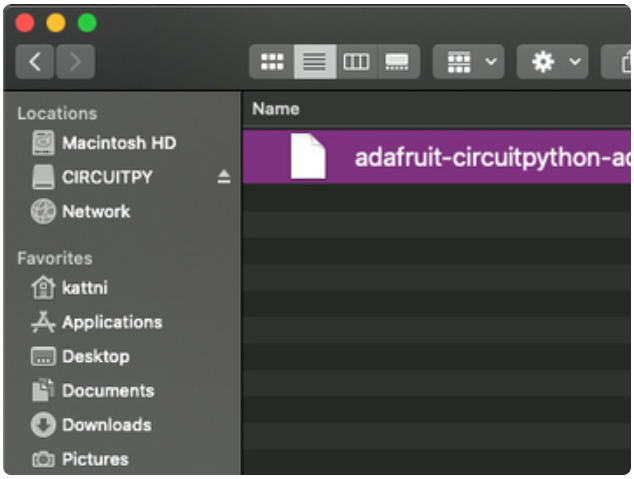
> If after several tries, and verifying your USB cable is data-ready, you still cannot get to the bootloader, it is possible that the bootloader is missing or damaged. Check out the Factory Reset page for details on resolving this issue.



You will see a new disk drive appear called **FTHRS2BOOT**.



Drag the **adafruit_circuitpython_etc.uf2** file to **FTHRS2BOOT**.



The **BOOT** drive will disappear and a new disk drive called **CIRCUITPY** will appear.

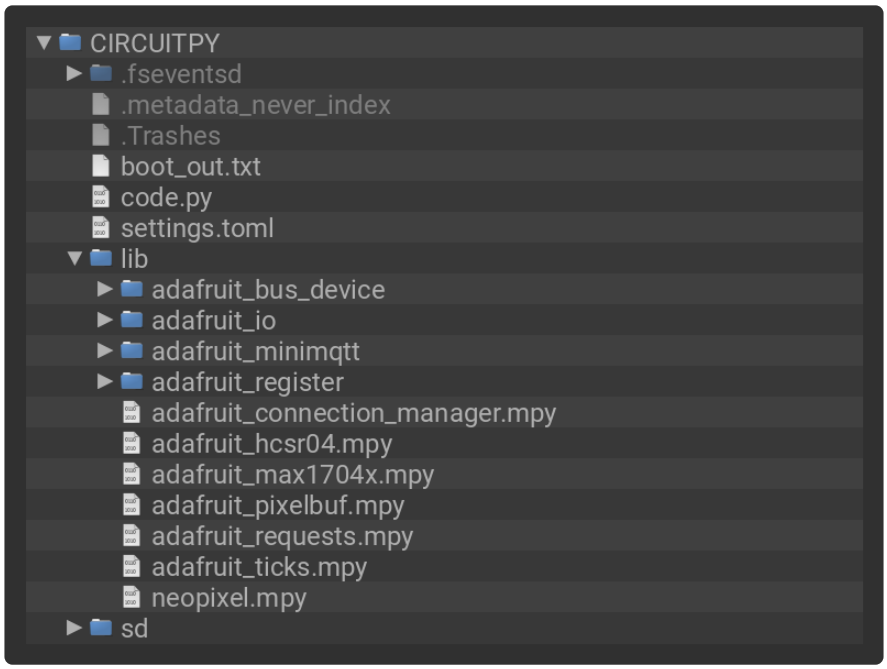
That's it!

Code the Espresso Water Tank Meter

Download the Project Bundle

Your project will use a specific set of CircuitPython libraries and the **code.py** file. To get everything you need, click on the **Download Project Bundle** link below, and uncompress the .zip file.

Drag the contents of the uncompressed bundle directory onto your board's **CIRCUITPY** drive, replacing any existing files or directories with the same names, and adding any new ones that are necessary.




```
# SPDX-FileCopyrightText: 2025 John Park for Adafruit Industries
#
# SPDX-License-Identifier: MIT
'''
Espresso Tank Meter
Feather ESP32-S2 with RCWL-1601 Ultrasonic distance sensor
```

```
'''

import time
import os
import ssl
import microcontroller
import supervisor
import socketpool
import wifi
import board
import digitalio
import alarm
import neopixel
import adafruit_hcsr04
import adafruit_minimqtt.adafruit_minimqtt as MQTT
from adafruit_io.adafruit_io import IO_MQTT
import adafruit_requests
import adafruit_max1704x

# Initialize the power pin for the sensor
sensor_power = digitalio.DigitalInOut(board.A2)
sensor_power.direction = digitalio.Direction.OUTPUT
sensor_power.value = False  # Start with sensor powered off

def power_sensor_on():
    """Turn on power to the ultrasonic sensor and wait for it to stabilize."""
    sensor_power.value = True
    time.sleep(0.55)  # Give sensor time to power up and stabilize

def power_sensor_off():
    """Turn off power to the ultrasonic sensor."""
    sensor_power.value = False

# Initialize the sonar sensor
sonar = adafruit_hcsr04.HCSR04(trigger_pin=board.A0, echo_pin=board.A1)

# Initialize the battery monitor
i2c = board.I2C()  # uses board.SCL and board.SDA
battery_monitor = adafruit_max1704x.MAX17048(i2c)

# Define colors (hex values)
WHITE = 0xFFFFFF
BLUE = 0x0000FF
GREEN = 0x00FF00
YELLOW = 0xFFFF00
RED = 0xFF0000
PINK = 0xbb00bb
CYAN = 0x00bbbb
OFF = 0x000000

# Initialize the NeoPixel
pixel = neopixel.NeoPixel(board.NEOPIXEL, 1, brightness=0.25)
# Show yellow on startup
pixel.fill(YELLOW)

# Operating hours (24-hour format with minutes, e.g., "6:35" and "16:00")
OPENING_TIME = "6:00"
CLOSING_TIME = "15:30"
# Normal operation check interval
NORMAL_CHECK_MINUTES = 10
# Sleep duration in seconds during operating hours
SLEEP_DURATION = 60 * NORMAL_CHECK_MINUTES
# Display duration in seconds
DISPLAY_DURATION = 1
# Number of samples to average
NUM_SAMPLES = 5

def parse_time(time_str):
    """Convert time string (HH:MM format) to hours and minutes."""
```

```
    # pylint: disable=redefined-outer-name
    parts = time_str.split(':')
    return int(parts[0]), int(parts[1])

def get_average_distance():
    """Take multiple distance readings and return the average."""
    power_sensor_on()  # Power on the sensor before taking measurements
    distances = []
    for _ in range(NUM_SAMPLES):
        try:
            distance = sonar.distance
            distances.append(distance)
            time.sleep(0.1)  # Short delay between readings
        except RuntimeError:
            print("Error reading distance")
            continue
    power_sensor_off()  # Power off the sensor after measurements

    # Only average valid readings
    if distances:
        return sum(distances) / len(distances)
    return None

def set_pixel_color(distance):
    """Set NeoPixel color based on distance."""
    if distance is None:
        pixel.fill(OFF)
        return

    if distance < 2:
        pixel.fill(WHITE)
    elif 2 <= distance < 10:
        pixel.fill(BLUE)
    elif 10 <= distance < 16:
        pixel.fill(GREEN)
    elif 18 <= distance < 20:
        pixel.fill(YELLOW)
    else:  # distance >= 22
        pixel.fill(RED)

# Wait for things to settle before reading sonar
time.sleep(0.1)

# Get average distance
avg_distance = get_average_distance()

if avg_distance is not None:

    if avg_distance >= 22:
        # pylint: disable=invalid-name
        avg_distance = 22
    print(f"Average distance: {avg_distance:.1f} cm")
    # Set color based on average distance
    set_pixel_color(avg_distance)

    # Check battery status
    battery_voltage = battery_monitor.cell_voltage
    battery_percent = battery_monitor.cell_percent
    print(f"Battery: {battery_percent:.1f}% ({battery_voltage:.2f}V)")

    # Try connecting to WiFi
    try:

        print("Connecting to %s" % os.getenv("CIRCUITPY_WIFI_SSID"))
        # Show pink while attempting to connect
        pixel.fill(PINK)
        wifi.radio.connect(os.getenv("CIRCUITPY_WIFI_SSID"),
os.getenv("CIRCUITPY_WIFI_PASSWORD"))
        print("Connected to %s" % os.getenv("CIRCUITPY_WIFI_SSID"))
```

```
        # Show cyan on successful connection
        pixel.fill(CYAN)
        time.sleep(1)  # Brief pause to show the connection success
    # pylint: disable=broad-except
    except Exception as e:
        print("Failed to connect to WiFi. Error:", e, "\nBoard will hard reset in
30 seconds.")
        pixel.fill(OFF)
        time.sleep(10)
        microcontroller.reset()

    # Create a socket pool
    pool = socketpool.SocketPool(wifi.radio)
    requests = adafruit_requests.Session(pool, ssl.create_default_context())

    # Initialize a new MQTT Client object
    mqtt_client = MQTT.MQTT(
        broker="io.adafruit.com",
        username=os.getenv("ADAFRUIT_AIO_USERNAME"),
        password=os.getenv("ADAFRUIT_AIO_KEY"),
        socket_pool=pool,
        ssl_context=ssl.create_default_context(),
    )

    # Initialize Adafruit IO MQTT "helper"
    io = IO_MQTT(mqtt_client)

    try:
        # If Adafruit IO is not connected...
        if not io.is_connected:
            print("Connecting to Adafruit IO...")
            io.connect()

        # Get current time from AIO time service
        aio_username = os.getenv("ADAFRUIT_AIO_USERNAME")
        aio_key = os.getenv("ADAFRUIT_AIO_KEY")
        timezone = os.getenv("TIMEZONE")
        # pylint: disable=line-too-long
        TIME_URL = f"https://io.adafruit.com/api/v2/{aio_username}/integrations/
time/strftime?x-aio-key={aio_key}&tz={timezone}"
        TIME_URL += "&fmt=%25Y-%25m-%25d+%25H%3A%25M%3A%25S.%25L+%25j+%25u+%25z+
%25Z"

        print("Getting time from Adafruit IO...")
        response = requests.get(TIME_URL)
        time_str = response.text.strip()  # Remove any leading/trailing whitespace
        print("Current time:", time_str)

        # Parse the current time from the time string
        time_parts = time_str.split()
        current_time = time_parts[1].split(':')
        current_hour = int(current_time[0])
        current_minute = int(current_time[1])

        # Get opening and closing hours and minutes
        opening_hour, opening_minute = parse_time(OPENING_TIME)
        closing_hour, closing_minute = parse_time(CLOSING_TIME)

        # Convert all times to minutes for easier comparison
        current_minutes = current_hour * 60 + current_minute
        opening_minutes = opening_hour * 60 + opening_minute
        closing_minutes = closing_hour * 60 + closing_minute

        # Check if we're within operating hours
        if opening_minutes <= current_minutes < closing_minutes:
            print(f"Within operating hours ({OPENING_TIME} to {CLOSING_TIME}),
proceeding with measurement")

            # Explicitly pump the message loop
```

```
            io.loop()

            # Send the distance data
            print(f"Publishing {avg_distance:.1f} to espresso water level feed")
            io.publish("espresso-water-tank-level", f"{avg_distance:.1f}")

            # Send the battery data
            print(f"Publishing {battery_percent:.1f} to battery level feed")
            io.publish("espresso-water-sensor-battery", f"{battery_percent:.1f}")


            # Make sure the message gets sent
            io.loop()

            print("Water level sent successfully")

            # Keep NeoPixel lit for DISPLAY_DURATION seconds
            time.sleep(DISPLAY_DURATION)

            # Use normal check interval during operating hours
            # # pylint: disable=invalid-name
            sleep_seconds = SLEEP_DURATION
            print(f"Next check in {NORMAL_CHECK_MINUTES} minutes")
        else:
            print(f"Outside operating hours ({OPENING_TIME} to {CLOSING_TIME}),
going back to sleep")
            # Calculate time until next opening
            if current_minutes >= closing_minutes:
                # After closing, calculate time until opening tomorrow
                minutes_until_open = (24 * 60 - current_minutes) + opening_minutes
            else:
                # Before opening, calculate time until opening today
                minutes_until_open = opening_minutes - current_minutes

            # Convert minutes to seconds for sleep duration
            sleep_seconds = minutes_until_open * 60
            hours_until_open = minutes_until_open // 60
            minutes_remaining = minutes_until_open % 60
            if minutes_remaining:
                print(f"Sleeping until {OPENING_TIME} ({hours_until_open} hours,
{minutes_remaining} minutes)")
            else:
                print(f"Sleeping until {OPENING_TIME} ({hours_until_open} hours)")

        response.close()

    # pylint: disable=broad-except
    except Exception as e:
        print("Failed to get or send data, or connect. Error:", e,
              "\nBoard will hard reset in 30 seconds.")
        pixel.fill(OFF)
        time.sleep(30)
        microcontroller.reset()

else:
    print("Failed to get valid distance readings")
    pixel.fill(OFF)
    # pylint: disable=invalid-name
    sleep_seconds = SLEEP_DURATION  # Use normal interval if we couldn't get
readings

# Prepare for deep sleep
power_sensor_off()  # Make sure sensor is powered off before sleep
pixel.brightness = 0  # Turn off NeoPixel

# Flush the serial output before sleep
# pylint: disable=pointless-statement
supervisor.runtime.serial_bytes_available
time.sleep(0.05)
```

```
# Create time alarm
time_alarm = alarm.time.TimeAlarm(monotonic_time=time.monotonic() + sleep_seconds)

# Enter deep sleep
alarm.exit_and_deep_sleep_until_alarms(time_alarm)
```

How it Works

Setup and Imports

```
import time, os, ssl, wifi, board, alarm, neopixel
import adafruit_hcsr04, adafruit_minimqtt, adafruit_max1704x
```

The code uses several libraries to handle hardware interfaces, WiFi connectivity, and power management. Key components are initialized:

- Ultrasonic sensor (HCSR04) on pins A0 and A1
- Battery monitor (MAX17048) via I2C
- NeoPixel LED for status indication

Configuration Constants

```
OPENING_TIME = "6:00"
CLOSING_TIME = "15:30"
NORMAL_CHECK_MINUTES = 5
SLEEP_DURATION = 60 * NORMAL_CHECK_MINUTES
NUM_SAMPLES = 5
```

These define the operating schedule and behavior:

- Operating hours from 6 AM to 3:30 PM
- Takes measurements every 5 minutes during operation
- Averages 5 samples per measurement

Distance Measurement

```
def get_average_distance():
    distances = []
    for _ in range(NUM_SAMPLES):
        try:
            distance = sonar.distance
            distances.append(distance)
            time.sleep(0.1)
        except RuntimeError:
            print("Error reading distance")
            continue
```

This function:

- Takes multiple readings from the sensor
- Handles potential read errors
- Returns the average of successful readings

Visual Feedback

```
def set_pixel_color(distance):
    if distance < 2:
        pixel.fill(WHITE)
    elif 2 <= distance < 10:
        pixel.fill(BLUE)
    # ... etc
```

The NeoPixel changes color based on the water level:

- Uses different colors to indicate various water levels
- Provides immediate visual feedback about tank status

WiFi and Data Transmission

```
wifi.radio.connect(os.getenv("CIRCUITPY_WIFI_SSID"), os.getenv("CIRCU
ITPY_WIFI_PASSWORD"))
mqtt_client = MQTT.MQTT(
    broker="io.adafruit.com",
    username=os.getenv("ADAFRUIT_AIO_USERNAME"),
    password=os.getenv("ADAFRUIT_AIO_KEY"),
    socket_pool=pool,
    ssl_context=ssl.create_default_context(),
)
```

The code:

- Connects to WiFi using credentials from environment variables
- Sets up MQTT connection to Adafruit IO
- Uses SSL for secure data transmission

Time Management

```
TIME_URL = f"https://io.adafruit.com/api/v2/{aio_username}/
integrations/time/strftime?..."
response = requests.get(TIME_URL)
time_str = response.text.strip()
```

The device:

- Gets current time from Adafruit IO
- Parses it to determine if it's within operating hours
- Calculates appropriate sleep duration

Data Reporting

```
io.publish("espresso-water-tank-level", f"{avg_distance:.1f}")
io.publish("espresso-water-sensor-battery", f"{battery_percent:.1f}")
```

Sends two pieces of data to Adafruit IO:

- Water level (distance measurement)
- Battery percentage

Power Management

```
time_alarm = alarm.time.TimeAlarm(monotonic_time=time.monotonic() +
sleep_seconds)
alarm.exit_and_deep_sleep_until_alarms(time_alarm)
```

The code implements these power-saving features:

- Uses deep sleep between measurements
- Calculates sleep duration based on operating hours
- Turns off NeoPixel during sleep

Connecting to the Adafruit IO MQTT Broker

If you do not want to host your own MQTT broker, using Adafruit IO (https://adafru.it/eZ8)'s MQTT broker is a great way to get started connecting your CircuitPython project to the internet. Best of all - it's free to use!

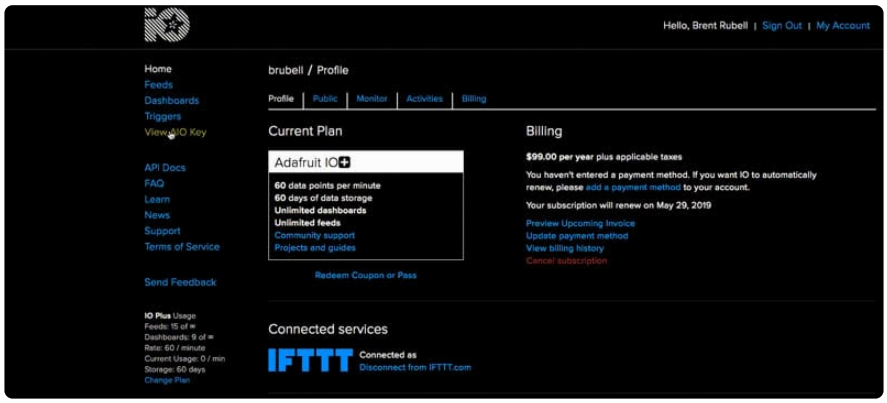
You're going to build an Adafruit IO Dashboard which can visualize incoming data from your CircuitPython board, and send data to it.

Obtain Adafruit IO Username and Key

If you have not already, sign up for an Adafruit IO account by clicking this link (https://adafru.it/Fm6).

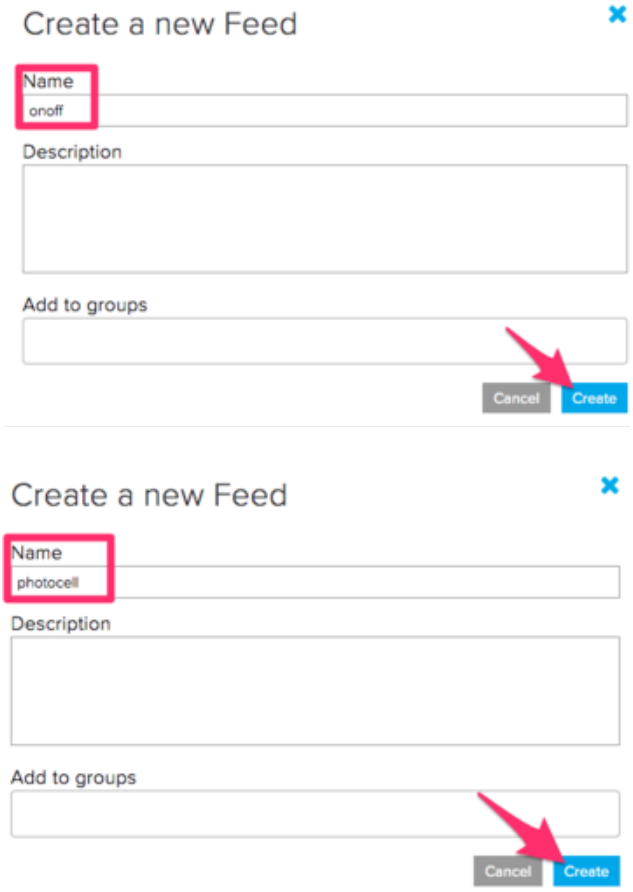
Next, you're going to need your Adafruit IO username and secret API key.

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



Create Adafruit IO Feeds

Adafruit IO uses a special type of MQTT Topic named a Feed to store data along with metadata (information about the data). You'll be publishing data to one feed, and subscribing to another.



Create two new Adafruit IO Feeds named onoff and photocell.

photocell - This feed will store light data published from your device to Adafruit IO
onoff - This feed will act as an on/off switch, publishing data to your device from Adafruit IO

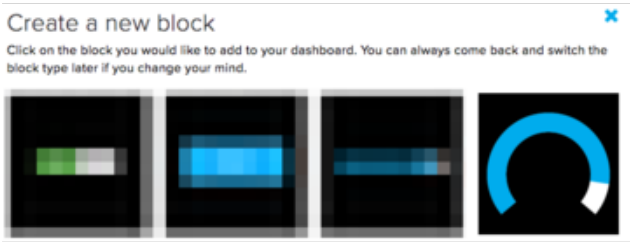
If you have not created an Adafruit IO Feed before, follow this page and come back once you've the created two feeds above (https://adafru.it/f5k).

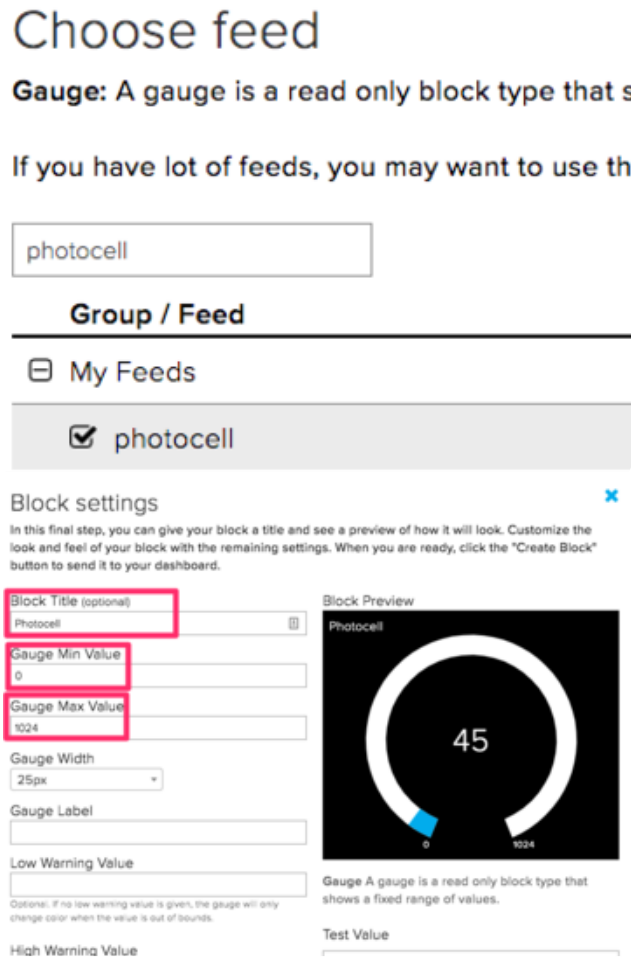
Create an Adafruit IO Dashboard

Adafruit IO Dashboards (https://adafru.it/f5m) are a way to interact with feeds. You can link blocks on dashboards to your feeds. The blocks can either display information about the feed (such as the current temperature) or allow you to interact with a feed by setting it to different values.

Start by creating a new dashboard. Name it whatever you'd like!

- **If you do not know how to create a dashboard**, head over to this page and come back here when you've successfully created a dashboard. (https://adafru.it/Fm7)



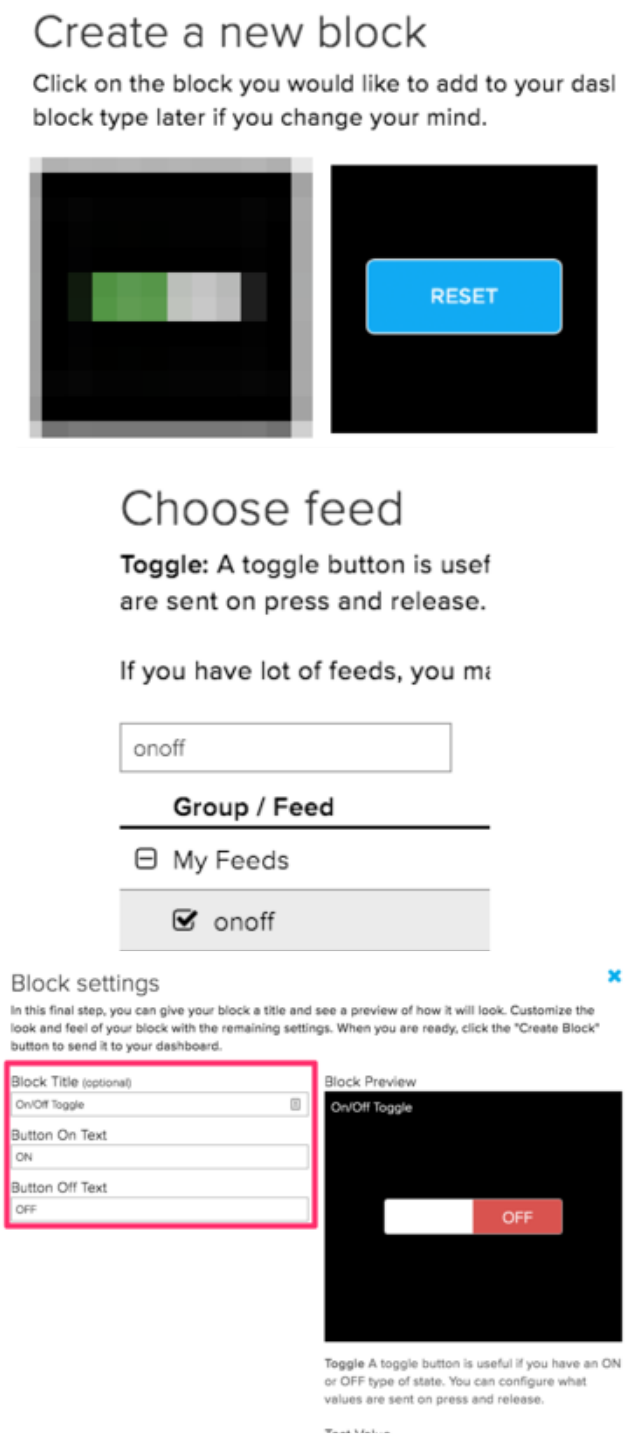


Create a Gauge Block

After creating a dashboard, **create a Gauge Block** to display the value of the Photocell feed.

Choose the photocell feed
Change the block title to Photocell
Set the Gauge Minimum Value to 0
Set the Gauge Maximum Value to 1024

If you do not know how to add blocks to a dashboard, head to over this page and come back when you've added a gauge block to your dashboard. (https://adafru.it/DZe)

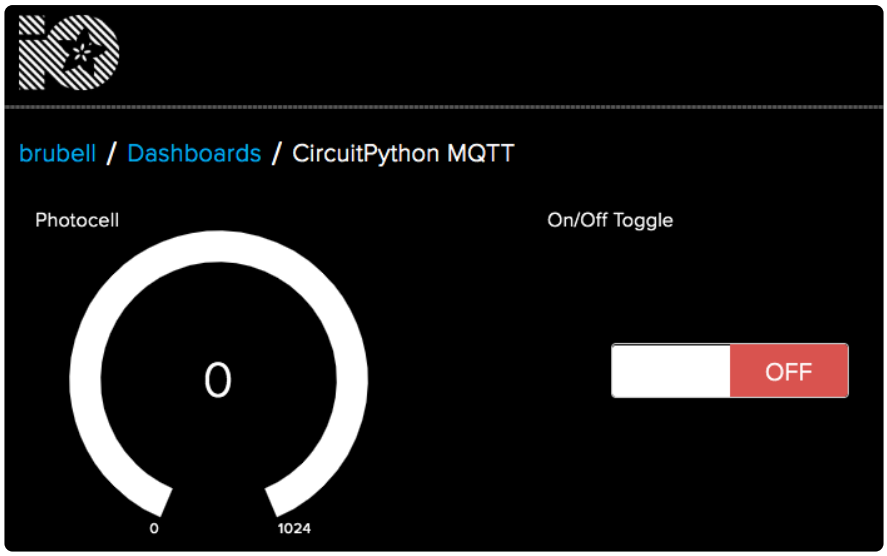


Create a Toggle Switch Block

To send values to the onoff feed you created - create a toggle switch block.

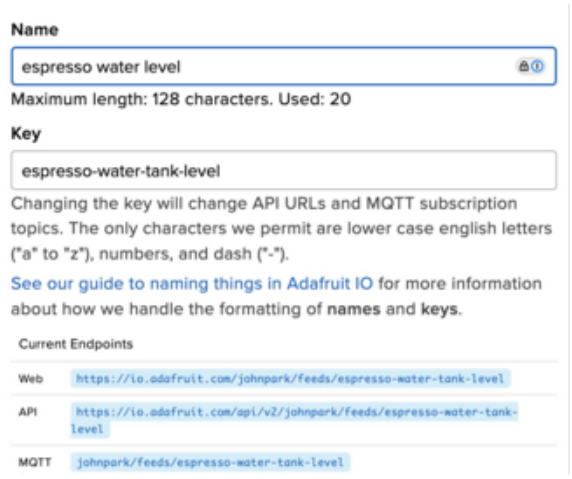
Choose the onoff **feed**
Set the block title to On/Off
Set the Button On Text to ON
Set the Button Off Text to OFF

Your dashboard should look like the following:

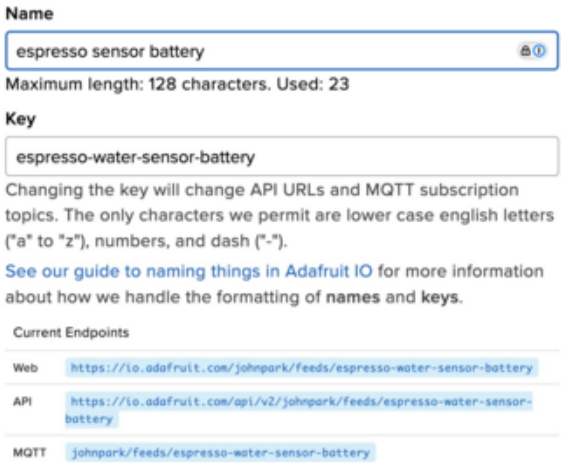


Make the Feeds & Dashboard

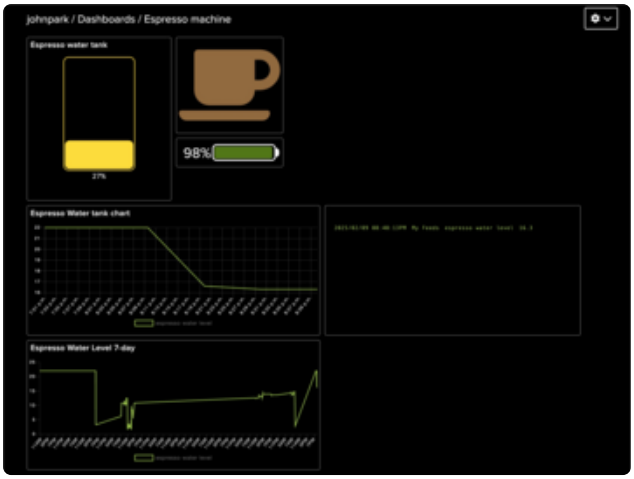
Following the detailed instructions here (https://adafru.it/ioA), create a feed for the water level and battery level with the following settings:

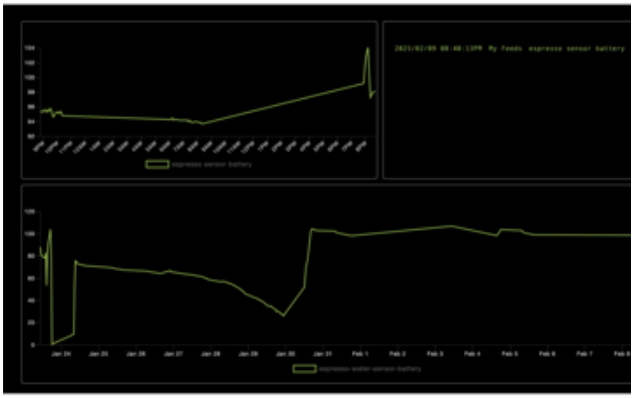


Create Water Level Feed



Create Battery Feed



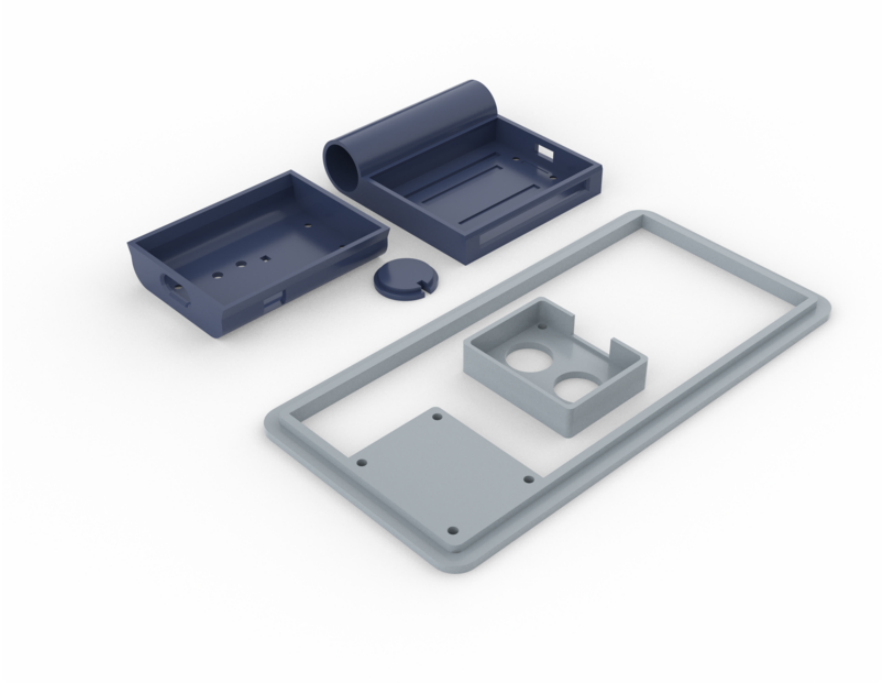


Create Dashboard

Follow this guide (https://adafru.it/f5m) for info on creating a dashboard for your feeds.

Here's a link to my publicly shared dashboard (https://adafru.it/1aes) to look at for an example. Let me know when the cauliflower steaks are done please!

Printed Parts

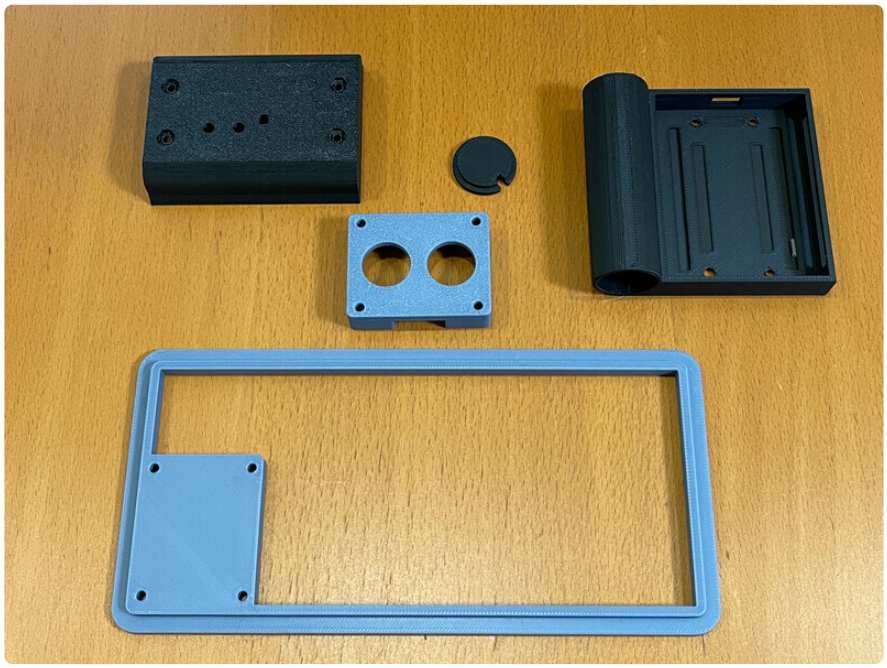


Espresso Water Tank Meter model files

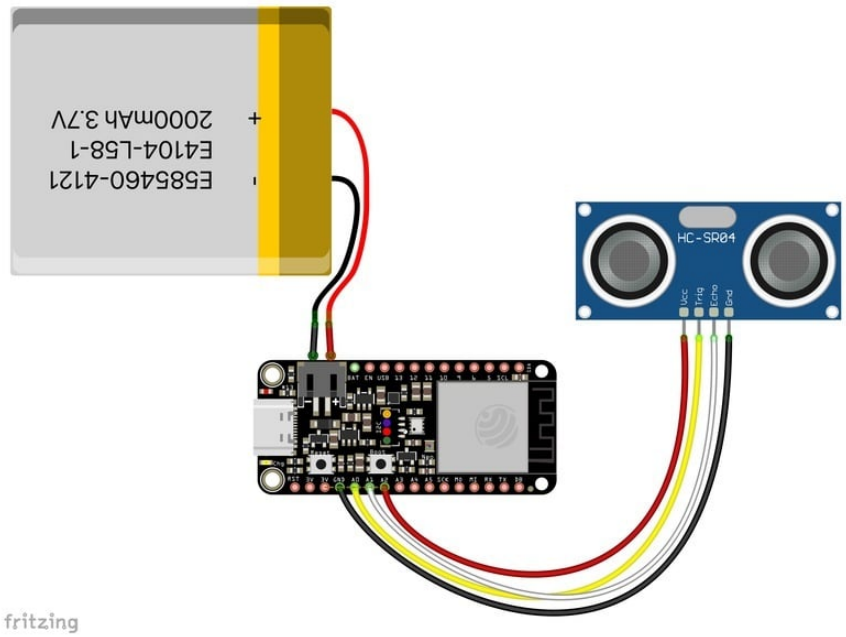
https://adafru.it/1aet

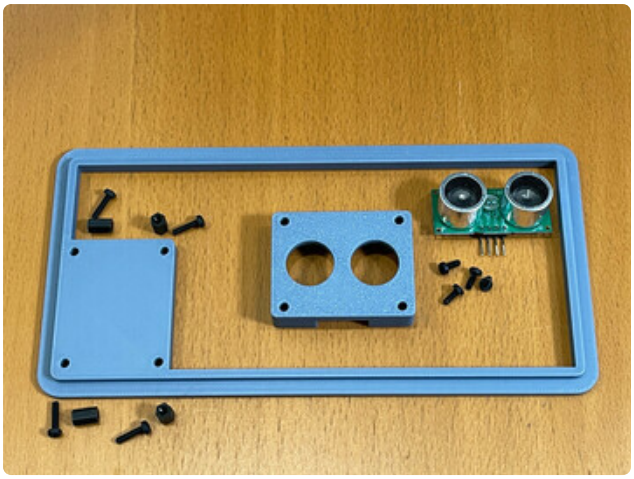
Download the attached files and print on standard settings (0.2mm layer height, 15% infill) in PLA (or PETG for added heat resistance).

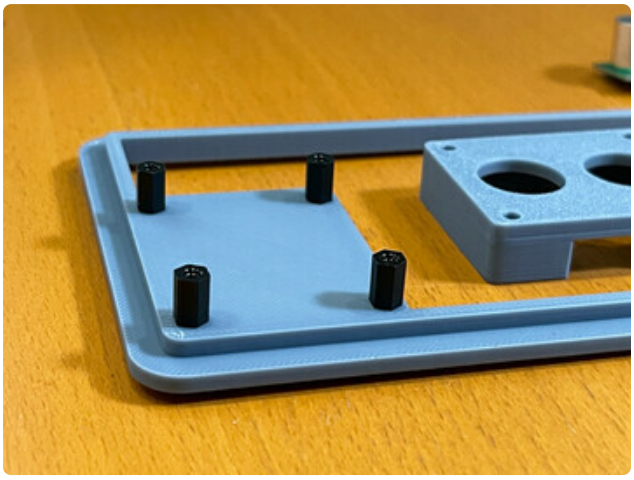
This design fits the 174mm x 80mm tank reservoir of an ECM Synchronika, you may need to modify the STEP file to fit a different tank.



Assemble the Meter

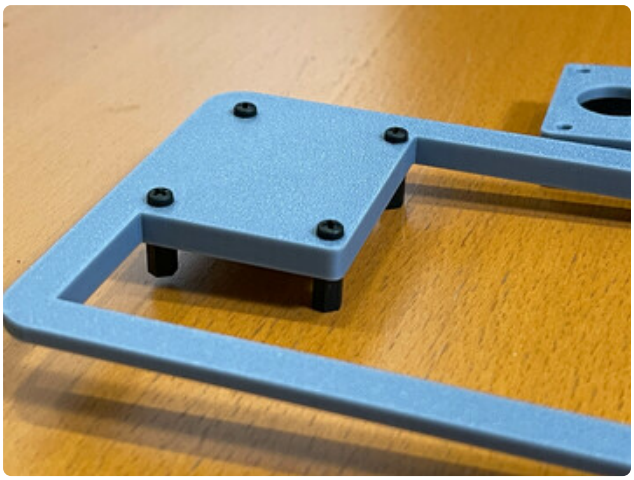


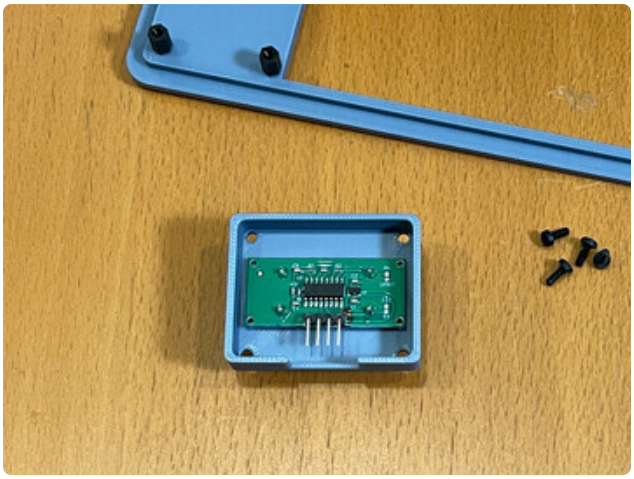


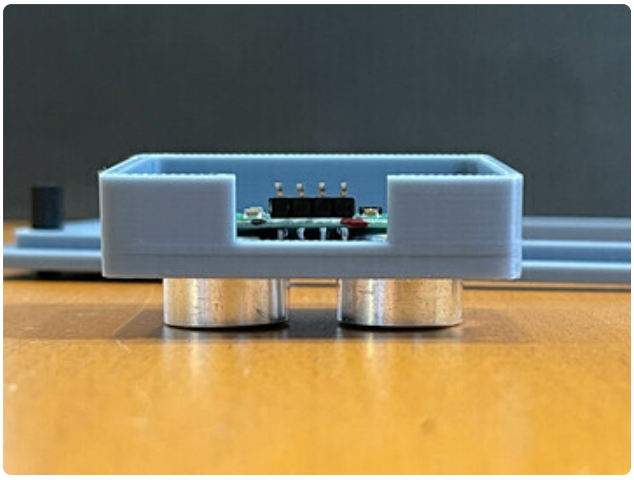


Sensor Tank Frame

Screw the standoffs to the underside of the frame as shown here using M2.5 x 12mm screws.

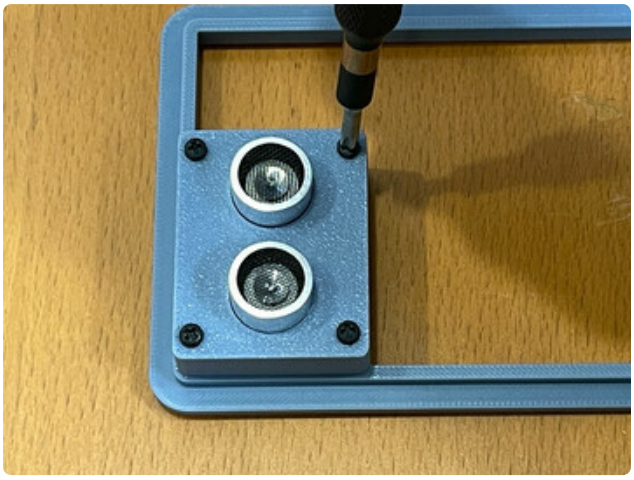






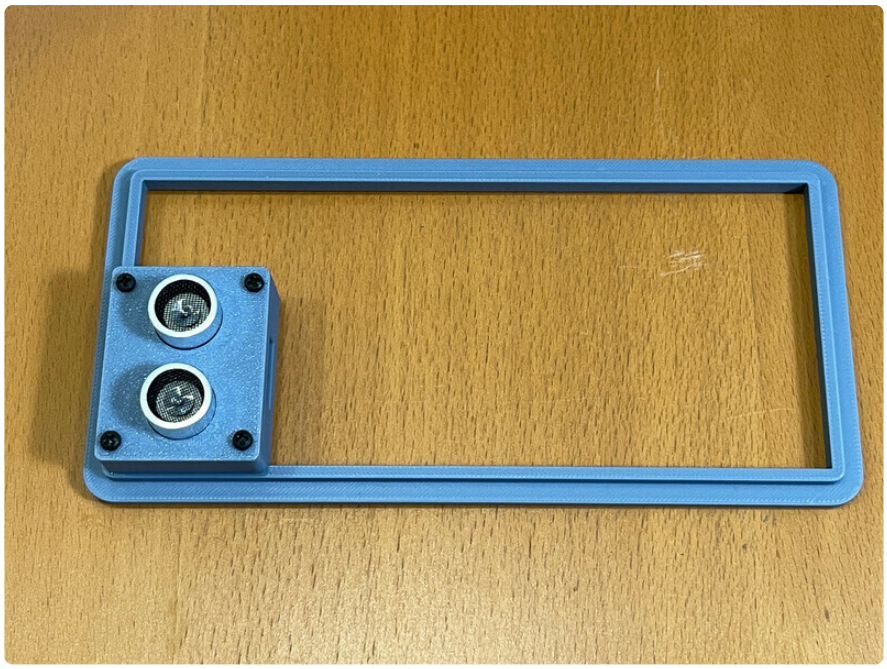
Sensor Cover

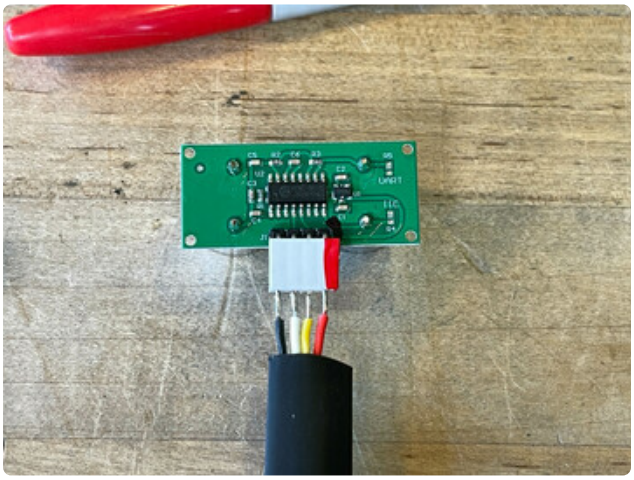
Press the sensor into the cover as shown.

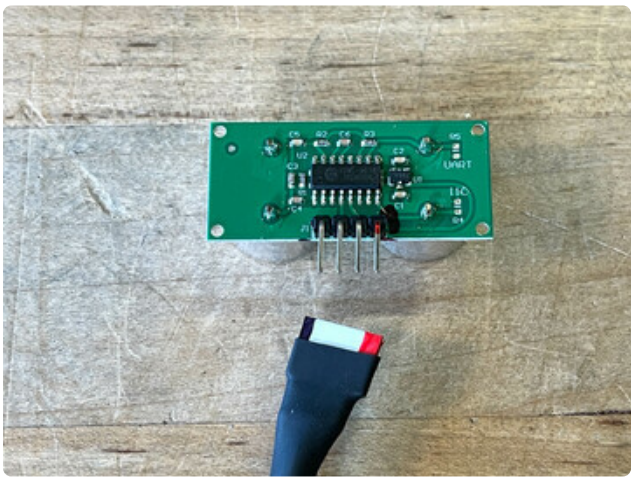


Secure Cover

Secure the cover to the frame using the M2.5 x 6mm screws.



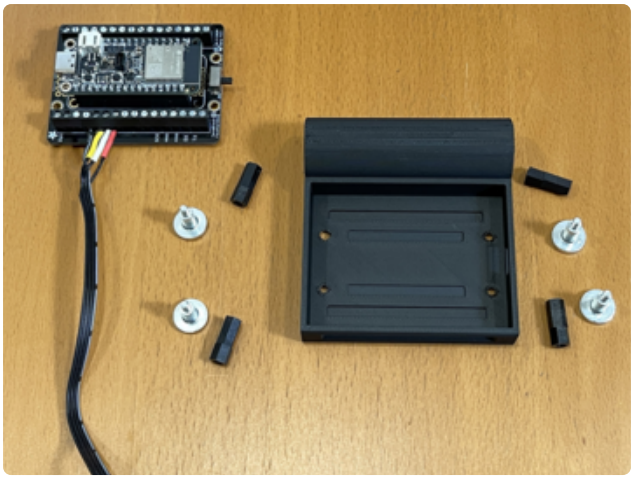


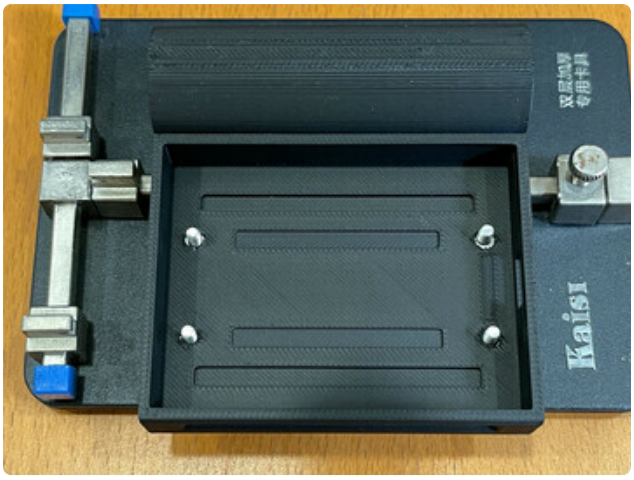


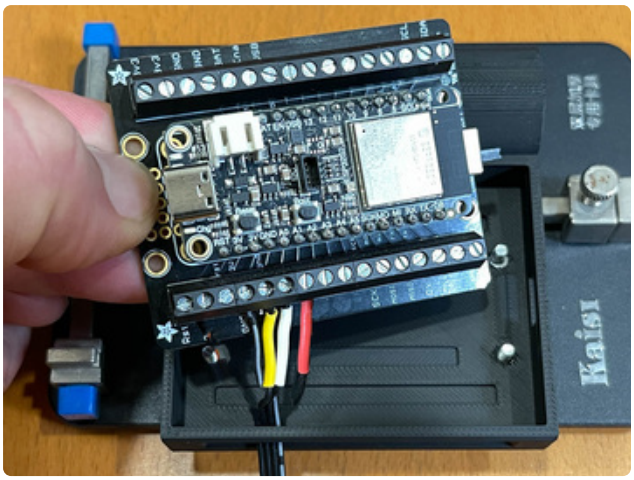
Wiring Harness

Use a 12" length of four-position ribbon cable and a female header to create a cable connector for the ultrasonic sensor. The other end will be screwed into the terminal block.

You can use some markers to color code them, and heat shrink tubing to neaten it up.







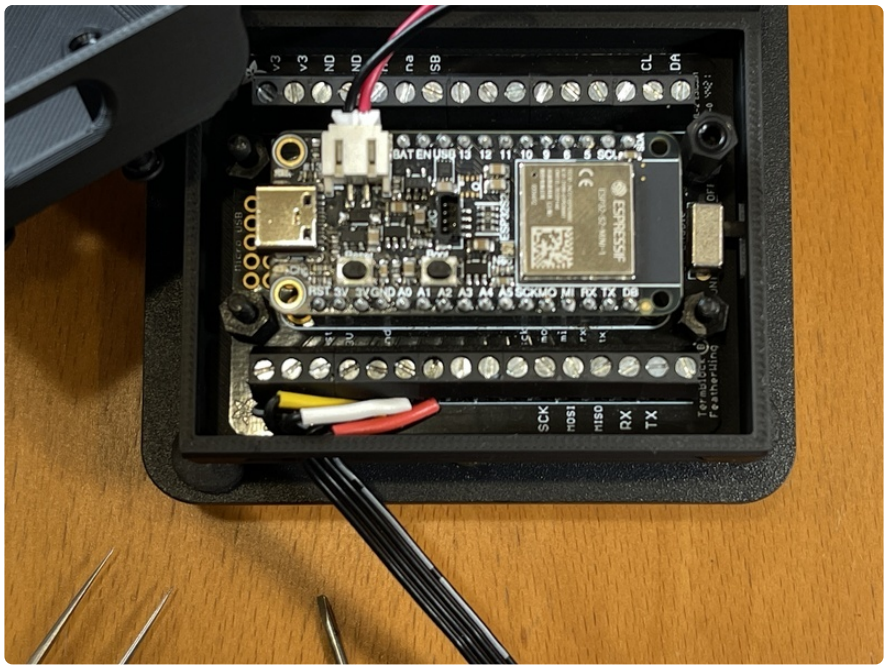
Screws & Feet

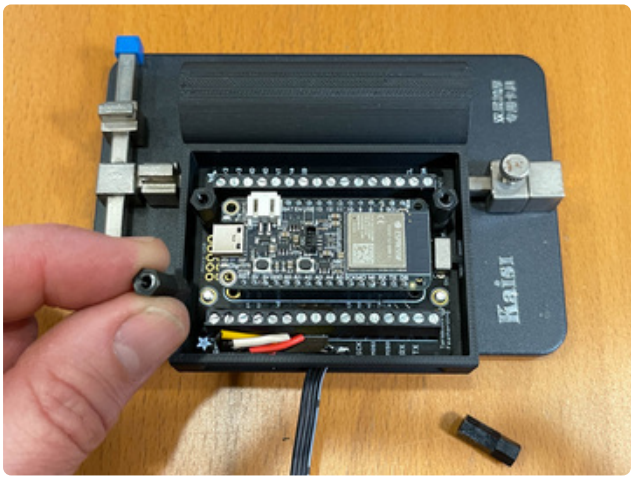
Screw the wires into the terminal blocks as shown to connect:

Sensor VCC pin to **Feather A2** (we'll use as switchable power source)
Sensor GND pin to **Feather GND**
Sensor TRIG pin to **Feather A0**
Sensor ECHO pin to **Feather A1**

Push the feet up through the case base holes. (Seen here on a ferrous base plate.)

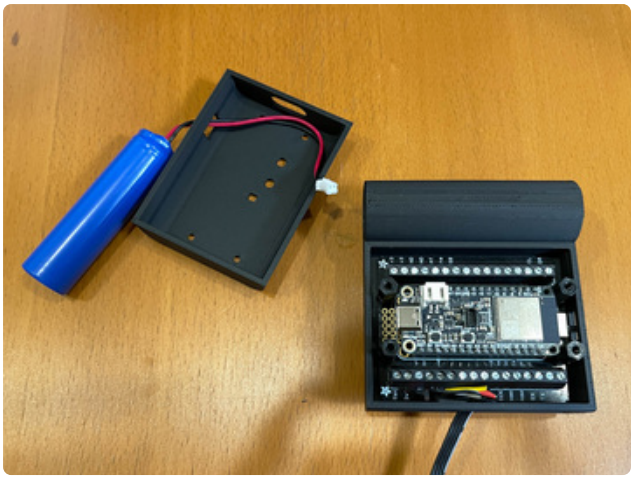
Feed the cable through the side slot, then place the Terminal Block FeatherWing onto the screws as shown.

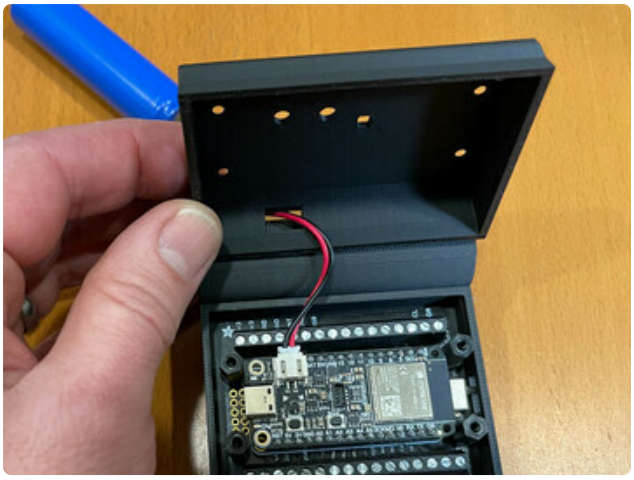


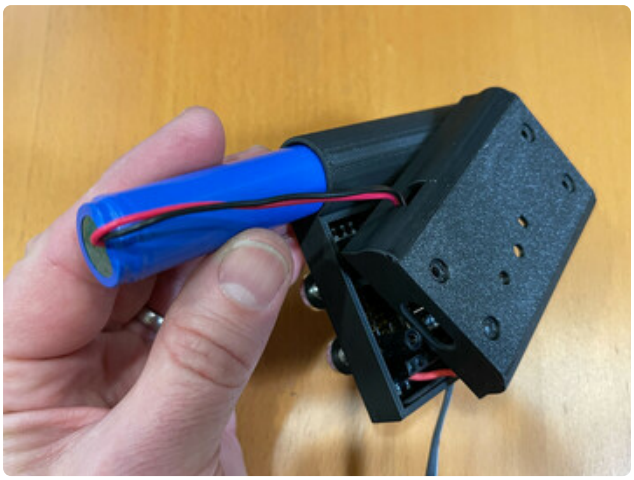


Standoffs

Thread the stacked standoffs onto the feet screws to secure the Feather in place and to provide fasteners for the case top.







Battery

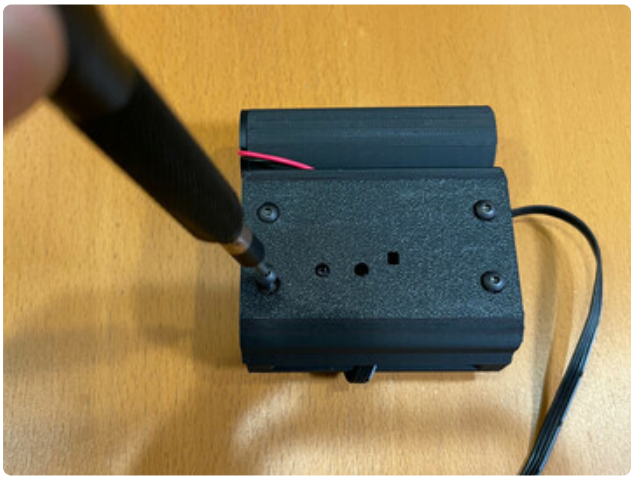
Feed the battery cable through the case top.

Then, plug the battery into the Feather's JST connector.

Slide the battery ever so satisfyingly into the battery holder.

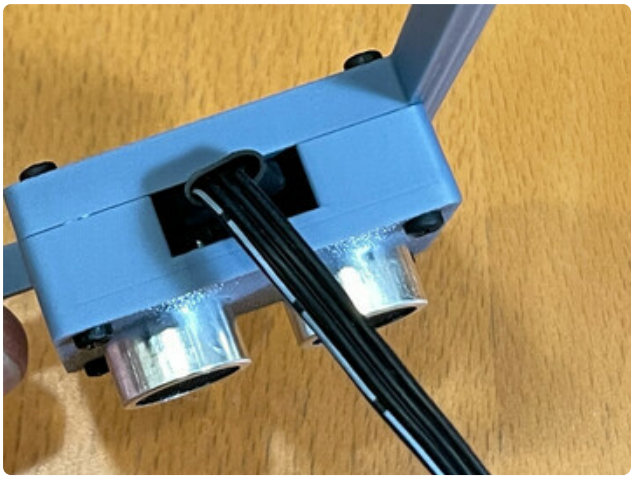
Cap It

Press the battery cap into place.



Topper

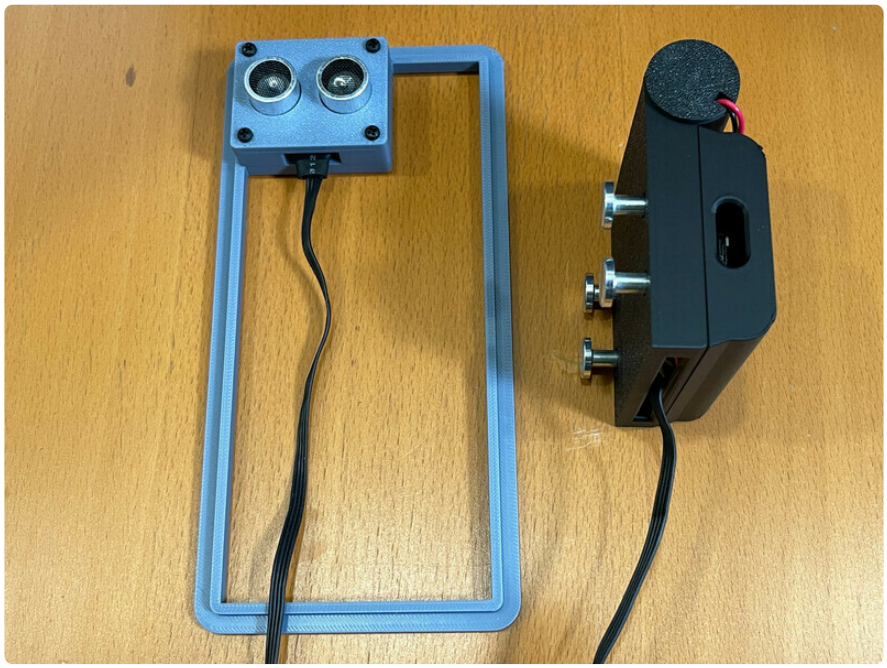
Screw the top into place with the four M2.5 x 12mm screws.

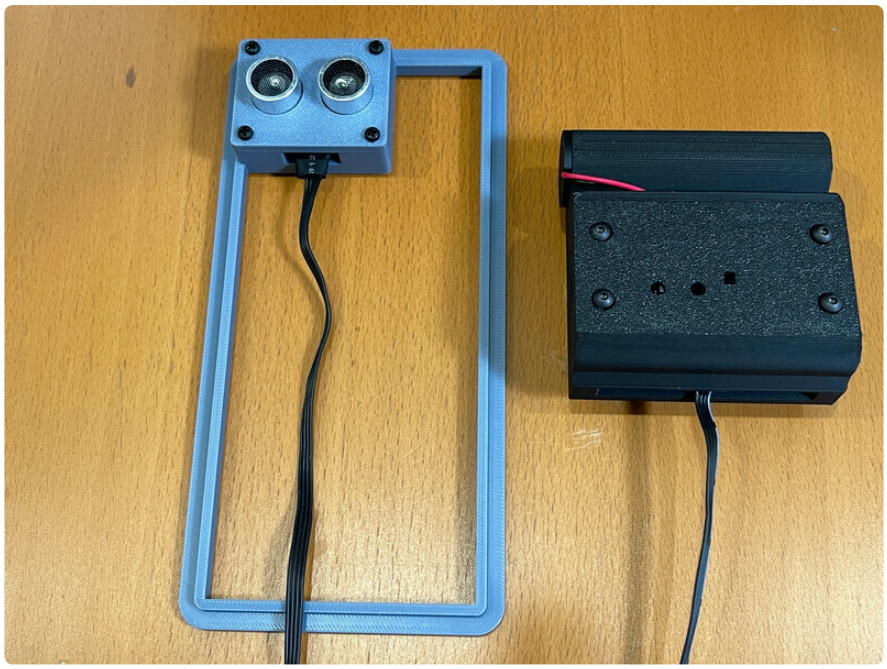


Wire It

Plug in the sensor cable as shown, with the GND on the left in this orientation.

The rig is ready for deployment!





Now you're ready to install and use the meter.

Use the Espresso Water Tank Meter

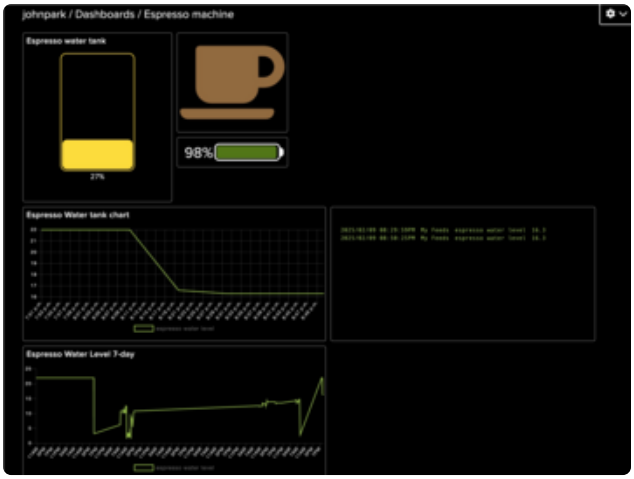
Add Sensor

Open the tank lid and fit the sensor bracket into place. Not a bad time to refill the tank while you're in there!

Mount Board

Run the sensor cable along the back of the machine and then mount the board case to the side or back using the magnetic feet.

Turn on the Feather and it'll connect to WiFi and send the sensor and battery data to your Feeds.

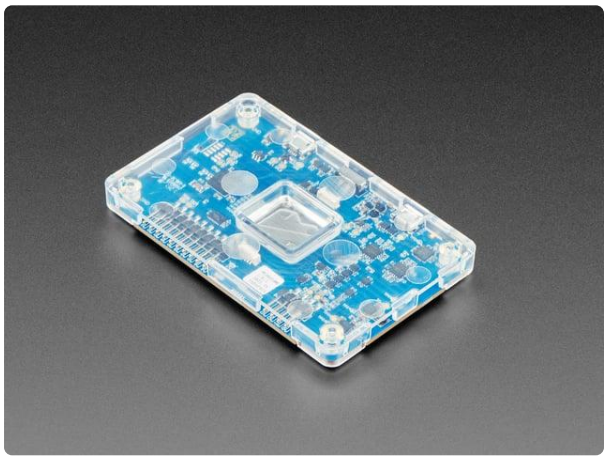


Check Dashboard

You can keep an eye on the dashboard, or even create alerts using things like SMS power-ups or It's a Snap (https://adafru.it/1a59) and Shortcuts on an iOS device.

Now, enjoy making a delicious espresso or cappuccino with confidence that you won't run out of water mid-shot!

Power Profiling



Nordic nRF-PPK2 - Power Profiler Kit II

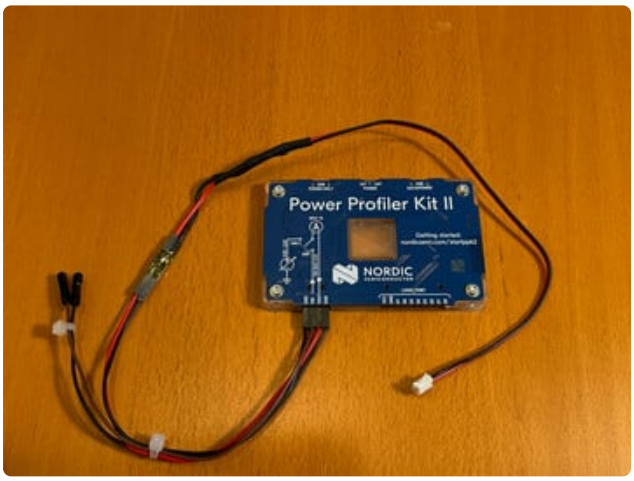
The Power Profiler Kit II is a standalone unit, which can measure and optionally supply currents all the way from sub-uA and as high as 1A on all Nordic DKs, in...

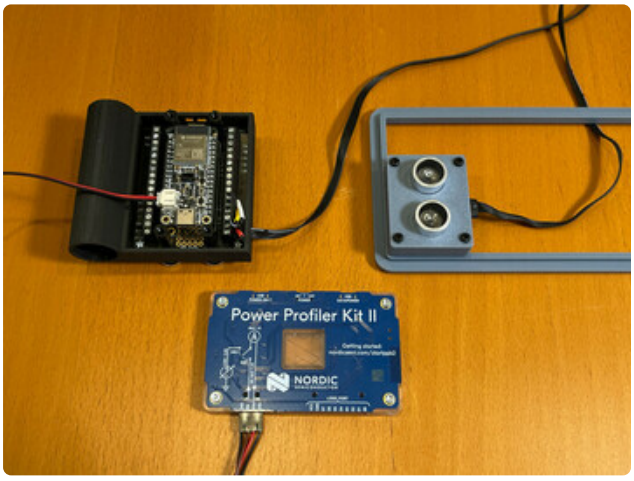
https://www.adafruit.com/product/5048

If you want to run your meter off of battery from perhaps a remote location, it's good to know what kind of battery life you can expect. Even when using deep sleep mode on the microcontroller, there is a tiny amount of current draw. And while awake, the radios can chew up battery a bite at a time. So, rather than guess, one may use science!

The Nordic Power Profiler Kit II works great for measuring current draw on a project like this. Check out this guide for setup instruction here (https://adafru.it/1aeu). Once you've installed the nRFConnect for Desktop app and have the Power Profiler app running, you can wire it up.

This is totally optional for this project.





Wiring up the PPK

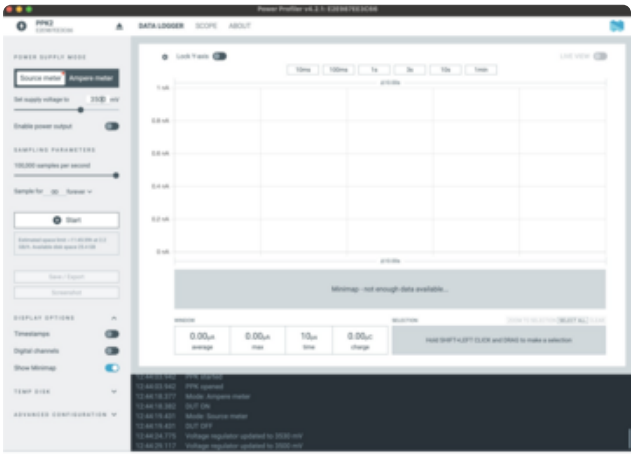
The first step is to connect your microcontroller to the PPK using the JST battery connector.

The PPK comes with two cables. For this, you'll need the 1x4 pin cable comprised of a black wire, red wire, brown wire and black wire, in that order.

Plug it into the 4-pin header on the PPK, so that GND is black, VIN is brown, VOUT is red, and GND is black, as shown

Then, connect it to the Feather JST-PH battery connector as follows:

PPKII VOUT (red wire) to **positive on the Feather JST-PH connector**
PPKII VOUT-GND (black wire on the right) to **negative on the Feather JST-PH connector**



Profiler App

Plug the PPKII into your computer using a micro USB cable, being sure to plug into the PPKII's **USB DATA/POWER** port, NOT the **USB POWER ONLY** port. Ask me how I know this.

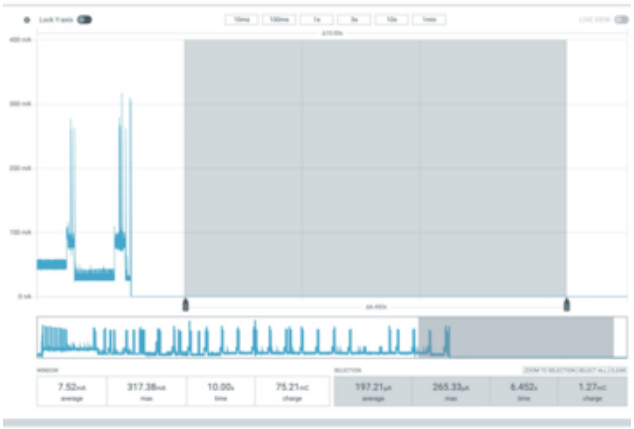
Turn on the PPKII (LEDs will glow and blink!), then select **PPK2** from the dropdown list at the top left of the app.

Set the **Power Supply Mode** to **Source Meter**. This allows us to supply power to the Feather while measuring the current draw

Set the supply voltage to `3500 mV`

Set **Enable power output** to the **on** position

Click the **Start** button and you will see the Feather power up and the graph will record the current draw.



Power Report

You should see something like the graph here. The Feather powers on, connects to WiFi and Adafruit IO, checks the ultrasonic sensor, reports the data, and then goes into deep sleep mode until the alarm wakes it up again to repeat. Press stop when you're done measuring.

You can shift + click-drag a section of the graph to see values for that selection. The data tells us these things about the Feather while it's awake:

it's active for about 8.636 seconds
it draws a maximum of 335.50mA
it draws an average of 58.88ma

By checking a selection while it's in deep sleep mode we learn:

max draw is 2.65µA
average draw is 197.21µAma

Battery Life Analysis

We can use the power report data to determine our ideal battery life expectancy between charges. I'm using a PKCELL ICR18650 2200mAh 3.7V Lithium Ion battery (http://adafru.it/1781).

These are my values going in:

- Capacity: `2200mAh`
- Voltage: `3.7V`
- Sleep current: `197µA`
- Active current: `58.88mA`
- Wake-up interval: `10 minutes`
- Active duration: `9 seconds`
- Operating hours per day: `9` (6am to 3pm)

This comes out to (thanks to Claude AI for doing the math):

Operating hours consumption (9 hours):

- Active duty cycle: `1.67% × 58.88mA = 0.98mA`
- Sleep duty cycle: `98.33% × 197µA = 0.194mA`
- Average current when operating: `1.174mA`
- Operating period usage: `9 hours × 1.174mA = 10.57mAh`

Non-operating hours consumption (15 hours):

- Sleep current: `197µA`
- Non-operating period usage: `15 hours × 197µA = 2.96mAh`

Total daily power consumption:

- Operating period: `10.57mAh`
- Non-operating period: `2.96mAh`
- Total daily usage: `13.53mAh`

Battery life calculation:

- Battery capacity: `2200mAh`
- Daily usage: `13.53mAh`
- Theoretical battery life = `2200mAh ÷ 36.96mAh = 162.6 days`

Practical estimated life (at 80% usable capacity): `162.60 × 0.8 = 130.08 days`

Four months, not bad! You can adjust some parameters pretty easily if needed, for example with a larger capacity battery, or shorter operating hours.