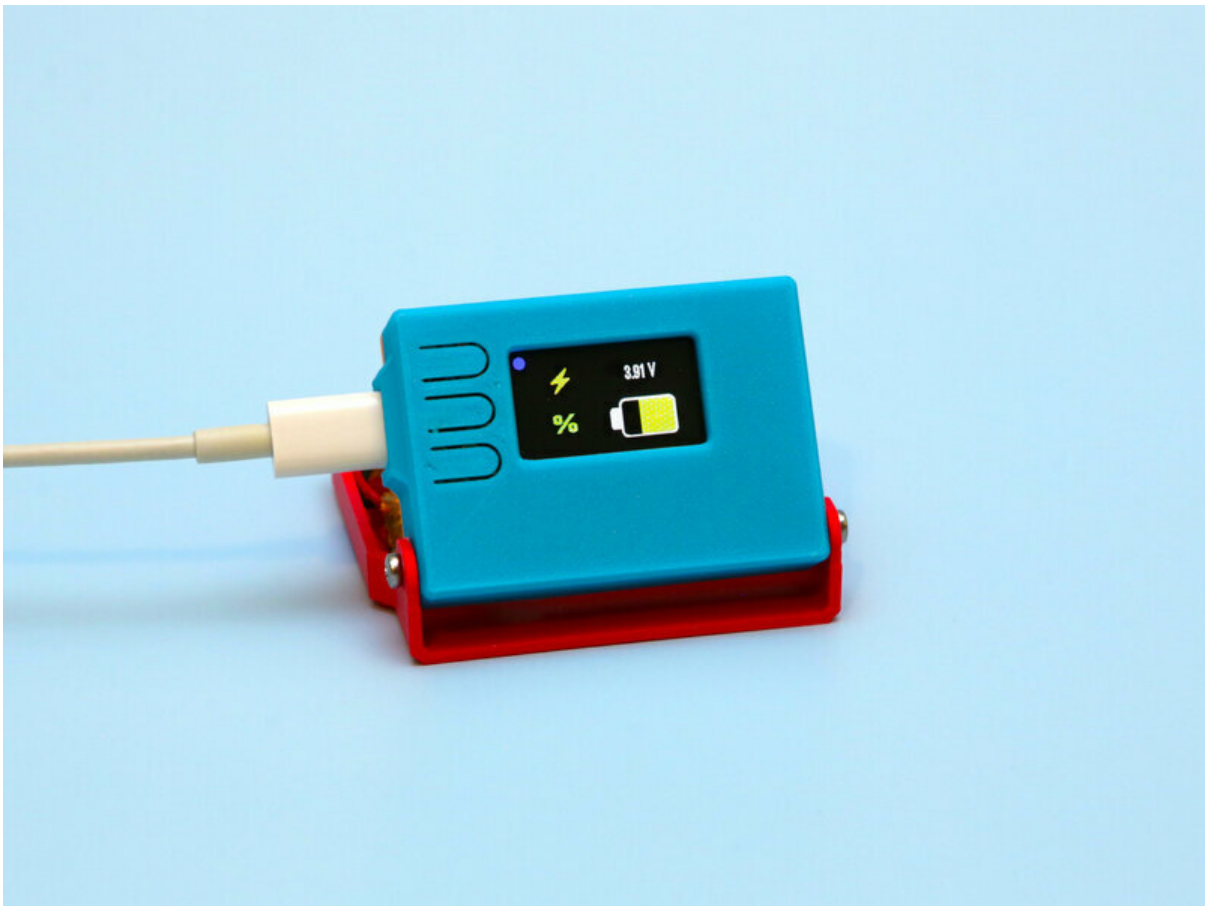
IoT Battery Monitor

Created by Ruiz Brothers



https://learn.adafruit.com/iot-battery-monitor

Last updated on 2025-01-14 01:59:02 PM EST

Table of Contents

Overview

Build an internet connected battery charger and monitor with Feather ESP32-S2 Reverse TFT, CircuitPython, and Adafruit IO.

The on-board MAX17048 LiPoly battery monitor chip reads the batteries voltage and is displayed on the Feather ESP32's built-in color TFT display. Use the on-board button to cycle between icons and text.

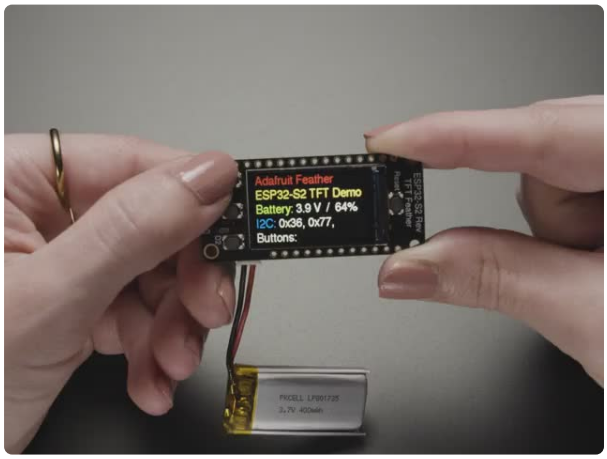
The project uses Adafruit IO actions to receive an SMS text or email when the battery is fully charged.

Enclosure

The 3D printed enclosure snap fits together, and features a tray for holding a 1200mAh battery. The enclosure can swivel forwards and back allowing for an adjustable viewing angle.

Use a JST extension cable to easily swap out batteries without having to unplug from the Feather.

Parts



Adafruit ESP32-S2 Reverse TFT Feather
Like Missy Elliot, we like to "put our [Feather] down, flip it and reverse it" and that's exactly what...
https://www.adafruit.com/product/5345

JST-PH Battery Extension Cable - 500mm

By popular demand, we now have a handy extension cord for all of our JST PH-terminated battery packs (such as our LiIon/LiPoly and 3xAAA holders). One end has a JST-PH compatible...

https://www.adafruit.com/product/1131

Pink and Purple Woven USB A to USB C Cable - 2 meters long

This cable is not only super-fashionable, with a woven pink and purple Blinka-like pattern, it's also made for USB C for our modernized breakout boards, Feathers and more.

https://www.adafruit.com/product/5044

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

Battery Options

1 x 100mAh — https://www.adafruit.com/product/1570

Lithium Ion Polymer Battery - 3.7v 100mAh

---

1 x 150mAh — https://www.adafruit.com/product/1317

Lithium Ion Polymer Battery - 3.7v 150mAh

---

1 x 350mAh — https://www.adafruit.com/product/4237

Lithium Ion Polymer Battery - 3.7v 350mAh

Lithium Ion Polymer Battery - 3.7v 400mAh

1 x 400mAh — https://www.adafruit.com/product/3898

Lithium Ion Polymer Battery - 3.7v 400mAh

---

1 x 420mAh — https://www.adafruit.com/product/4236

Lithium Ion Polymer Battery - 3.7v 420mAh

---

1 x 500mAh — https://www.adafruit.com/product/1578

Lithium Ion Polymer Battery - 3.7v 500mAh

---

1 x 1200mAh — https://www.adafruit.com/product/258

Lithium Ion Polymer Battery - 3.7v 1200mAh

---

1 x 2000mAh — https://www.adafruit.com/product/2011

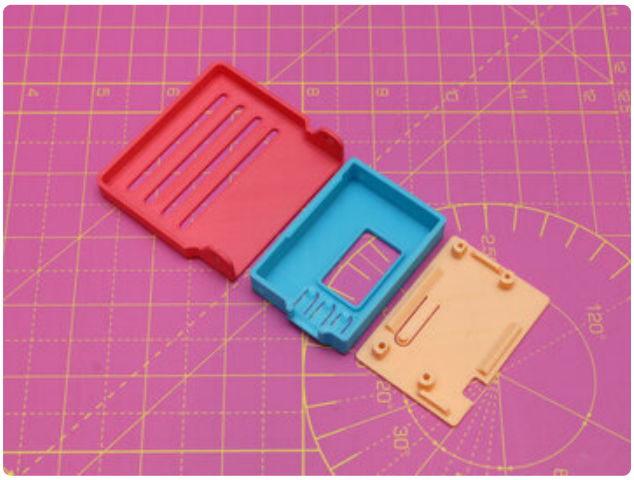
Lithium Ion Polymer Battery - 3.7v 2000mAh

Hardware

The following hardware is necessary for the case assembly.

- 4x M2.5 x 6mm long machine screws
- 2x M2 x 6mm long FF standoffs

CAD Files



3D Printed Parts

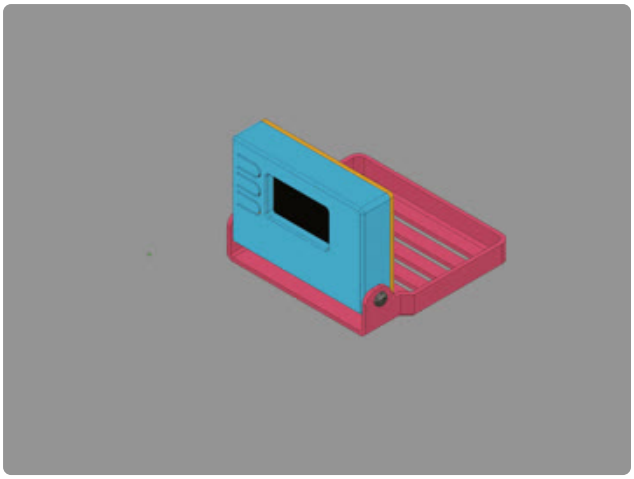
STL files for 3D printing are oriented to print "as-is" on FDM style machines. Parts are designed to 3D print without any support material using PLA filament. Original design source may be downloaded using the links below.

Download STLs.zip

https://adafru.it/19OE

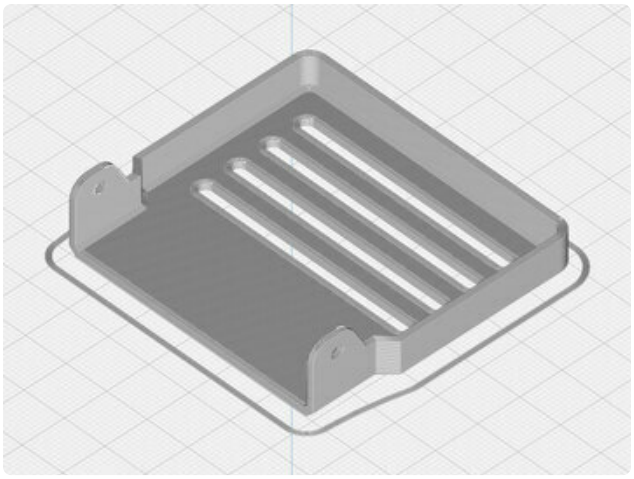
Download CAD source

https://adafru.it/19OF



CAD Assembly

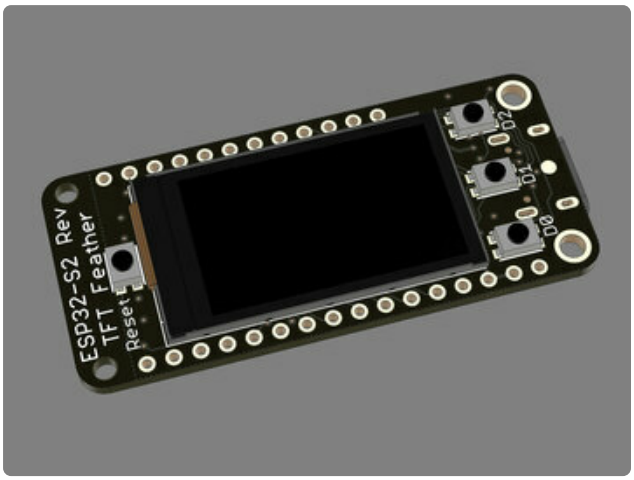
The Feather ESP32-S2 Reverse TFT Feather board is secured to the enclosure's back cover using both M2 and M2.5 fasteners. The back cover snap fits onto the front cover. The front cover is secured to the battery tray using M2.5 fastener.



Build Volume

The parts require a 3D printer with a minimum build volume.

68mm (X) x 62mm (Y) x 20mm (Z)



Design Source Files

The project assembly was designed in Fusion 360. This can be downloaded in different formats like STEP, STL and more.

Electronic components like Adafruit's boards, displays, connectors and more can be downloaded from the Adafruit CAD parts GitHub Repo (https://adafru.it/RvF).

CircuitPython

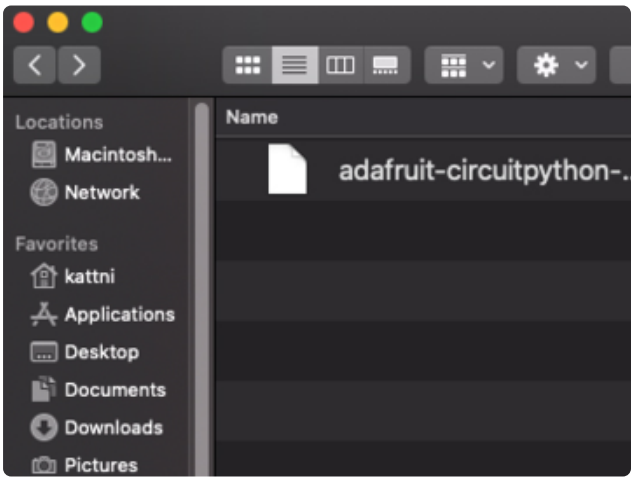
CircuitPython (https://adafru.it/tB7) is a derivative of MicroPython (https://adafru.it/BeZ) designed to simplify experimentation and education on low-cost microcontrollers. It makes it easier than ever to get prototyping by requiring no upfront desktop software downloads. Simply copy and edit files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

CircuitPython Quickstart

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

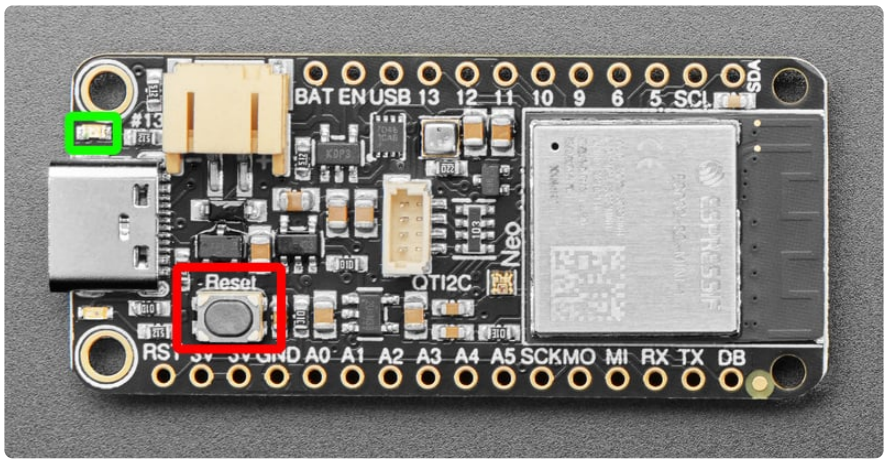
Download the latest version of CircuitPython for this board via circuitpython.org

https://adafru.it/18sB



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

Save it wherever is convenient for you.



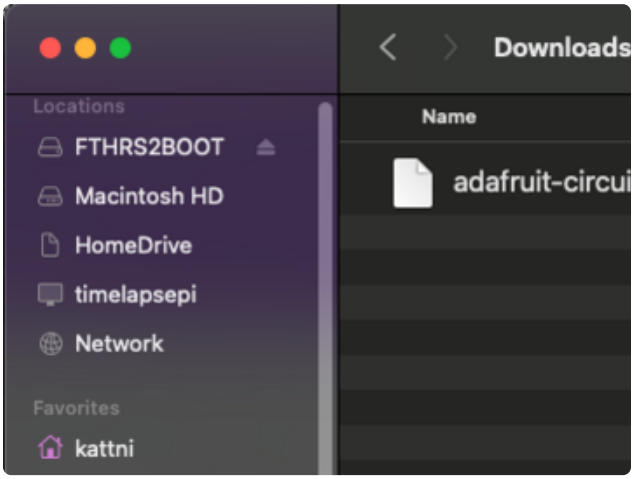
Plug your board into your computer, using a known-good data-sync cable, directly, or via an adapter if needed.

Double-click the **reset** button (highlighted in red above), and 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.

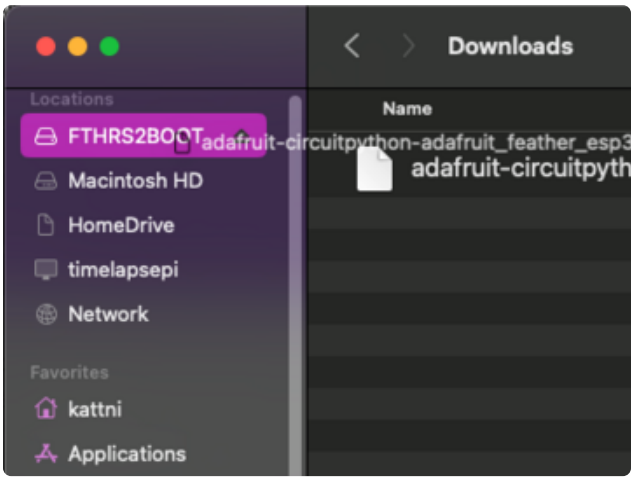
For this board, tap reset and wait for the LED to turn purple, and as soon as it turns purple, tap reset again. The second tap needs to happen while the LED is still purple.

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

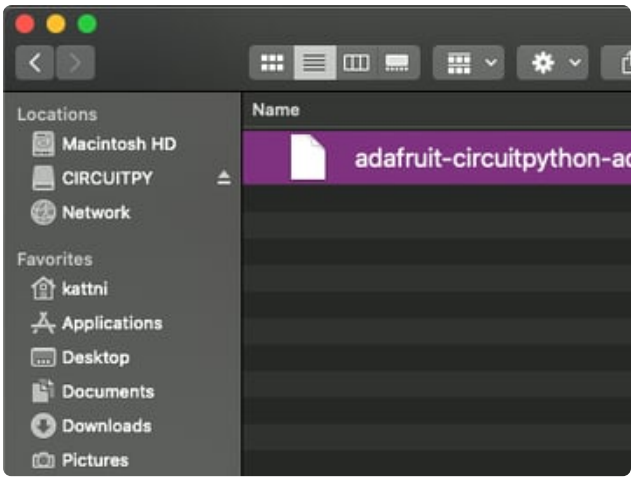
A lot of people end up using charge-only USB cables and it is very frustrating! **Make sure you have a USB cable you know is good for data sync.**



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!

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

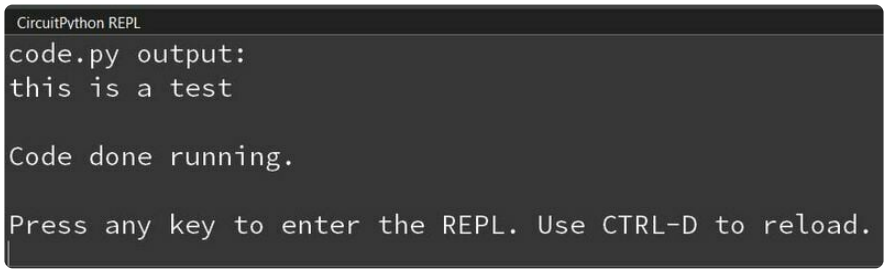
When your **settings.toml** file is ready, you can save it in your text editor with the **.toml** extension.

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.

Code the Battery Monitor

Once you've finished setting up your Feather ESP32-S2 Reverse TFT with CircuitPython, you can access the code and necessary libraries by downloading the Project Bundle.

To do this, click on the **Download Project Bundle** button in the window below. It will download to your computer as a zipped folder.

```
# SPDX-FileCopyrightText: 2024 Liz Clark for Adafruit Industries
# SPDX-License-Identifier: MIT

import ssl
import os
```

```
import socketpool
import wifi
import board
import digitalio
import displayio
import vectorio
from adafruit_bitmap_font import bitmap_font
from adafruit_display_text import bitmap_label
import adafruit_imageload
from adafruit_io.adafruit_io import IO_HTTP, AdafruitIO_RequestError
import adafruit_max1704x
import adafruit_requests
from simpleio import map_range
from adafruit_ticks import ticks_ms, ticks_add, ticks_diff

# states
send_io = True
bat_clock = ticks_ms()
bat_timer = 60 * 1000
first_run = True
# settings.toml imports
aio_username = os.getenv('AIO_USERNAME')
aio_key = os.getenv('AIO_KEY')
# connect to wifi
wifi.radio.connect(os.getenv('CIRCUITPY_WIFI_SSID'),
os.getenv('CIRCUITPY_WIFI_PASSWORD'))
pool = socketpool.SocketPool(wifi.radio)
requests = adafruit_requests.Session(pool, ssl.create_default_context())
io = IO_HTTP(aio_username, aio_key, requests)
try:
    # get feed
    battery_feed = io.get_feed("battery-monitor")
except AdafruitIO_RequestError:
    # if no feed exists, create one
    battery_feed = io.create_new_feed("battery-monitor")
# default group
group = displayio.Group()
# text only group
textOnly_group = displayio.Group()
board.DISPLAY.root_group = group
# palette for vector graphics
palette = displayio.Palette(5)
palette[0] = 0xFF0000
palette[1] = 0xFFFF00
palette[2] = 0x00FF00
palette[3] = 0x0000FF
palette[4] = 0x000000
# battery rectangle
rect = vectorio.Rectangle(pixel_shader=palette, width=72, height=45, x=140, y=70,
color_index = 0)
group.append(rect)
text_bg = vectorio.Rectangle(pixel_shader=palette, width=115, height=70,
                             x=120, y=60, color_index = 4)
# io indicator circle
circle = vectorio.Circle(pixel_shader=palette, radius=8, x=10, y=10, color_index=3)
textOnly_group.append(circle)
# graphics bitmap
bitmap, palette_bit = adafruit_imageload.load(
    "/bat_bg.bmp",
    bitmap=displayio.Bitmap,
    palette=displayio.Palette,
)
# purple is made transparent
palette_bit.make_transparent(0)
tile_grid = displayio.TileGrid(bitmap, pixel_shader=palette_bit)
group.append(tile_grid)
group.append(circle)
# font for graphics
sm_file = "/roundedHeavy-26.bdf"
```

```
sm_font = bitmap_font.load_font(sm_file)
# font for text only
lg_file = "/roundedHeavy-46.bdf"
lg_font = bitmap_font.load_font(lg_file)
volt_text = bitmap_label.Label(sm_font, text=" V", x=150, y=33)
group.append(volt_text)
big_volt_text = bitmap_label.Label(lg_font, text=" V")
big_volt_text.anchor_point = (0.5, 0.0)
big_volt_text.anchored_position = (board.DISPLAY.width / 2, 0)
textOnly_group.append(big_volt_text)
percent_text = bitmap_label.Label(sm_font, text=" %", x=150, y=90)
big_percent_text = bitmap_label.Label(lg_font, text=" %", x=board.DISPLAY.width//2,
y=90)
big_percent_text.anchor_point = (0.5, 1.0)
big_percent_text.anchored_position = (board.DISPLAY.width / 2, board.DISPLAY.height
- 15)
textOnly_group.append(big_percent_text)

# buttons
button0 = digitalio.DigitalInOut(board.D0)
button0.direction = digitalio.Direction.INPUT
button0.pull = digitalio.Pull.UP
button0_state = False
button1 = digitalio.DigitalInOut(board.D1)
button1.direction = digitalio.Direction.INPUT
button1.pull = digitalio.Pull.DOWN
button1_state = False
button2 = digitalio.DigitalInOut(board.D2)
button2.direction = digitalio.Direction.INPUT
button2.pull = digitalio.Pull.DOWN
button2_state = False

# MAX17048 instantiation
monitor = adafruit_max1704x.MAX17048(board.I2C())
monitor.activity_threshold = 0.01

# colors for battery graphic
def get_color(value):
    if value < 30:
        return 0
    elif 30 <= value <= 75:
        return 1
    else:
        return 2

while True:
    # reset button state on release
    if button0.value and button0_state:
        button0_state = False
    if not button1.value and button1_state:
        button1_state = False
    if not button2.value and button2_state:
        button2_state = False
    # toggle sending to adafruit io
    if not button0.value and not button0_state:
        button0_state = True
        send_io = not send_io
        if send_io:
            circle.color_index = 3
        else:
            circle.color_index = 4
    # toggle graphics or text only
    if button1.value and not button1_state:
        button1_state = True
        if board.DISPLAY.root_group == group:
            board.DISPLAY.root_group = textOnly_group
        else:
            board.DISPLAY.root_group = group
    # toggle battery graphic or % text
```

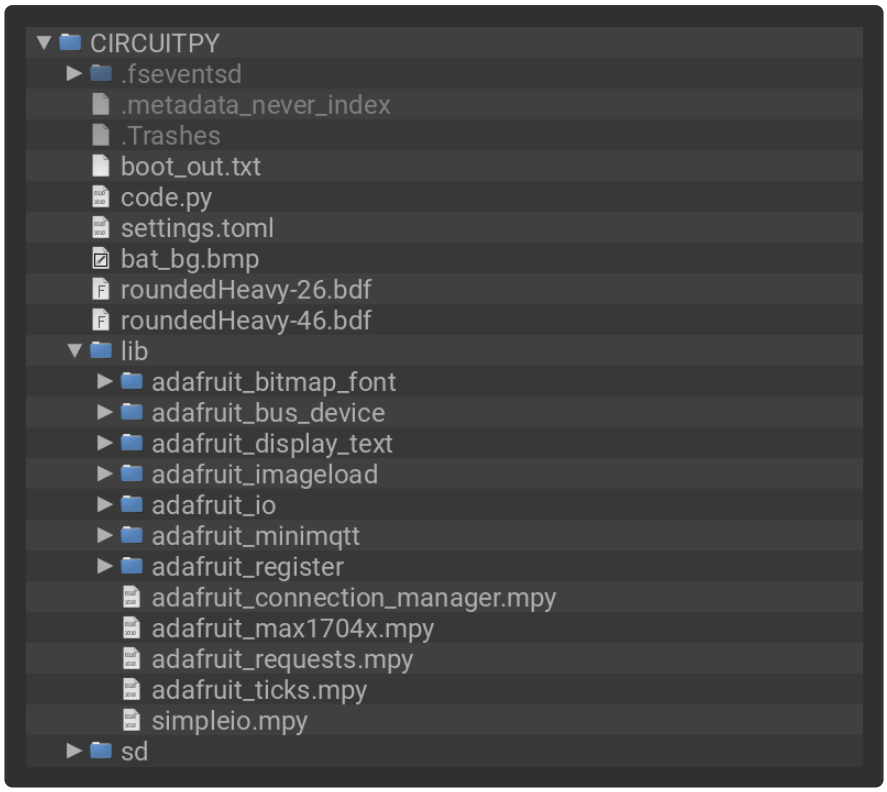
```
    if button2.value and not button2_state:
        button2_state = True
        if len(group) > 4:
            group.pop()
            group.pop()
        else:
            group.append(text_bg)
            group.append(percent_text)
    # read MAX17048 every 60 seconds
    if first_run or ticks_diff(ticks_ms(), bat_clock) >= bat_timer:
        first_run = False
        battery_volts = monitor.cell_voltage
        battery_percent = monitor.cell_percent
        print(f"Battery voltage: {battery_volts:.2f} Volts")
        print(f"Battery percentage: {battery_percent:.1f} %")
        print()
        battery_display = map_range(battery_percent, 0, 100, 0, 72)
        battery_x = map_range(battery_percent, 0, 100, 210, 140)
        #  update rectangle to reflect battery charge
        rect.width = int(battery_display)
        rect.x = int(battery_x)
        rect.color_index = get_color(battery_percent)
        volt_text.text = f"{battery_volts:.2f} V"
        percent_text.text = f"{battery_percent:.1f} %"
        big_volt_text.text = f"{battery_volts:.2f} V"
        big_percent_text.text = f"{battery_percent:.1f} %"
        if battery_percent >= 100 and send_io:
            io.send_data(battery_feed["key"], battery_percent)
        bat_clock = ticks_add(bat_clock, bat_timer)
```

Upload the Code and Libraries to the Feather ESP32-S2 Reverse TFT

After downloading the Project Bundle, plug your Feather ESP32-S2 Reverse TFT into the computer's USB port with a known good USB data+power cable. You should see a new flash drive appear in the computer's File Explorer or Finder (depending on your operating system) called **CIRCUITPY**. Unzip the folder and copy the following items to the Feather's **CIRCUITPY** drive:

- **lib** folder
- **code.py**
- **bat_bg.bmp**
- **roundedHeavy-26.bdf**
- **roundedHeavy-46.bdf**

Your Feather ESP32-S2 Reverse TFT **CIRCUITPY** drive should look like this after copying the **lib** folder, **.bdf** font files, **.bmp** image file and the **code.py** file.



Add Your **settings.toml** File

As of CircuitPython 8.0.0, there is support for Environment Variables (https://adafru.it/11wE). Environment variables are stored in a **settings.toml** file. Similar to **secrets.py**, the **settings.toml** file separates your sensitive information from your main **code.py** file. Add your **settings.toml** file as described in the Create Your settings.toml File page (https://adafru.it/19ap) earlier in this guide. You'll need to include your WiFi network SSID and password as `CIRCUITPY_WIFI_SSID` and `CIRCUITPY_WIFI_PASSWORD` and your Adafruit IO username and key as `AIO_USERNAME` and `AIO_KEY`.

```
CIRCUITPY_WIFI_SSID = "YOUR-SSID-HERE"
CIRCUITPY_WIFI_PASSWORD = "YOUR-SSID-PASSWORD-HERE"
AIO_USERNAME = "YOUR-AIO-USERNAME-HERE"
AIO_KEY = "YOUR-AIO-KEY-HERE"
```

How the CircuitPython Code Works

At the top of the code are some states that are used in the loop. `send_io` enables sending the battery charging data to Adafruit IO. `bat_timer` is the amount of time that the battery monitor is checked and `first_run` and used to note that the code is running for the first time in the loop.

```
# states
send_io = True
bat_clock = ticks_ms()
```

```
bat_timer = 60 * 1000
first_run = True
```

After the states are the **settings.toml** imports and WiFi and Adafruit IO connections.

```
# settings.toml imports
aio_username = os.getenv('AIO_USERNAME')
aio_key = os.getenv('AIO_KEY')
# connect to wifi
wifi.radio.connect(os.getenv('CIRCUITPY_WIFI_SSID'),
os.getenv('CIRCUITPY_WIFI_PASSWORD'))
pool = socketpool.SocketPool(wifi.radio)
requests = adafruit_requests.Session(pool, ssl.create_default_context())
io = IO_HTTP(aio_username, aio_key, requests)
try:
    # get feed
    battery_feed = io.get_feed("battery-monitor")
except AdafruitIO_RequestError:
    # if no feed exists, create one
    battery_feed = io.create_new_feed("battery-monitor")
```

If your WiFi access point name or password are incorrect, an error will be generated.

Graphics

Next are the graphics. There are two display groups: one that includes graphical elements and one that is text only. You can switch between then in the loop depending on your preferences. the **bat_bg** bitmap file is loaded using the **adafruit_image** library and its background color is made transparent. **vectorio** shapes are used to show the battery charge graphic and if you are sending data to Adafruit IO.

```
# default group
group = displayio.Group()
# text only group
textOnly_group = displayio.Group()
board.DISPLAY.root_group = group
# palette for vector graphics
palette = displayio.Palette(5)
palette[0] = 0xFF0000
palette[1] = 0xFFFF00
palette[2] = 0x00FF00
palette[3] = 0x0000FF
palette[4] = 0x000000
# battery rectangle
rect = vectorio.Rectangle(pixel_shader=palette, width=72, height=45, x=140, y=70,
color_index = 0)
group.append(rect)
text_bg = vectorio.Rectangle(pixel_shader=palette, width=115, height=70,
                             x=120, y=60, color_index = 4)
# io indicator circle
circle = vectorio.Circle(pixel_shader=palette, radius=8, x=10, y=10, color_index=3)
textOnly_group.append(circle)
# graphics bitmap
bitmap, palette_bit = adafruit_imageload.load(
    "/bat_bg.bmp",
    bitmap=displayio.Bitmap,
    palette=displayio.Palette,
)
# purple is made transparent
```

```
palette_bit.make_transparent(0)
tile_grid = displayio.TileGrid(bitmap, pixel_shader=palette_bit)
group.append(tile_grid)
group.append(circle)
```

Then all of the text attributes are created. There are two font files, one for a smaller font and one for a larger font. The larger font is used for the text only graphics group.

```
# font for graphics
sm_file = "/roundedHeavy-26.bdf"
sm_font = bitmap_font.load_font(sm_file)
# font for text only
lg_file = "/roundedHeavy-46.bdf"
lg_font = bitmap_font.load_font(lg_file)
volt_text = bitmap_label.Label(sm_font, text=" V", x=150, y=33)
group.append(volt_text)
big_volt_text = bitmap_label.Label(lg_font, text=" V")
big_volt_text.anchor_point = (0.5, 0.0)
big_volt_text.anchored_position = (board.DISPLAY.width / 2, 0)
textOnly_group.append(big_volt_text)
percent_text = bitmap_label.Label(sm_font, text=" %", x=150, y=90)
big_percent_text = bitmap_label.Label(lg_font, text=" %", x=board.DISPLAY.width//2,
y=90)
big_percent_text.anchor_point = (0.5, 1.0)
big_percent_text.anchored_position = (board.DISPLAY.width / 2, board.DISPLAY.height
- 15)
textOnly_group.append(big_percent_text)
```

Buttons and I2C

The three buttons on the Feather are setup as **digitalio** inputs. The onboard MAX17048 is instantiated over I2C.

```
# buttons
button0 = digitalio.DigitalInOut(board.D0)
button0.direction = digitalio.Direction.INPUT
button0.pull = digitalio.Pull.UP
button0_state = False
button1 = digitalio.DigitalInOut(board.D1)
button1.direction = digitalio.Direction.INPUT
button1.pull = digitalio.Pull.DOWN
button1_state = False
button2 = digitalio.DigitalInOut(board.D2)
button2.direction = digitalio.Direction.INPUT
button2.pull = digitalio.Pull.DOWN
button2_state = False

# MAX17048 instantiation
monitor = adafruit_max1704x.MAX17048(board.I2C())
monitor.activity_threshold = 0.01
```

Colors

There is a simple function called `get_color()` that maps the charge percentage of the battery to a color in a palette. This changes the color of the battery from red to yellow to green as it charges up.

```
# colors for battery graphic
def get_color(value):
    if value &lt; 30:
        return 0
    elif 30 &lt;= value &lt;= 75:
        return 1
    else:
        return 2
```

The Loop

In the loop, the buttons control the UI appearance and code functionality. Button D0 toggles sending the battery info to Adafruit IO. Button D1 changes between the text only display group and the graphical group. Button D2 changes between representing the battery charge percentage in text or via the battery graphic.

```
# toggle sending to adafruit io
    if not button0.value and not button0_state:
        button0_state = True
        send_io = not send_io
        if send_io:
            circle.color_index = 3
        else:
            circle.color_index = 4
    # toggle graphics or text only
    if button1.value and not button1_state:
        button1_state = True
        if board.DISPLAY.root_group == group:
            board.DISPLAY.root_group = textOnly_group
        else:
            board.DISPLAY.root_group = group
    # toggle battery graphic or % text
    if button2.value and not button2_state:
        button2_state = True
        if len(group) &gt; 4:
            group.pop()
            group.pop()
        else:
            group.append(text_bg)
            group.append(percent_text)
```

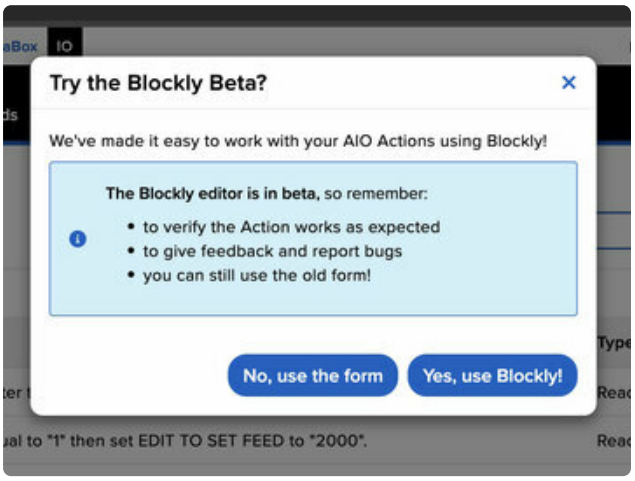
Monitor the Battery

Every minute the MAX17048 is read to check on the battery voltage and charge percentage. The graphics are updated with the data to show on the TFT. If you chose to send the data to IO, the charge percentage is sent to your feed when the battery is fully charged.

```
# read MAX17048 every 60 seconds
    if first_run or ticks_diff(ticks_ms(), bat_clock) &gt;= bat_timer:
        first_run = False
        battery_volts = monitor.cell_voltage
        battery_percent = monitor.cell_percent
        print(f"Battery voltage: {battery_volts:.2f} Volts")
        print(f"Battery percentage: {battery_percent:.1f} %")
        print()
        battery_display = map_range(battery_percent, 0, 100, 0, 72)
        battery_x = map_range(battery_percent, 0, 100, 210, 140)
```

```
#  update rectangle to reflect battery charge
rect.width = int(battery_display)
rect.x = int(battery_x)
rect.color_index = get_color(battery_percent)
volt_text.text = f"{battery_volts:.2f} V"
percent_text.text = f"{battery_percent:.1f} %"
big_volt_text.text = f"{battery_volts:.2f} V"
big_percent_text.text = f"{battery_percent:.1f} %"
if battery_percent &gt;= 100 and send_io:
    io.send_data(battery_feed["key"], battery_percent)
bat_clock = ticks_add(bat_clock, bat_timer)
```

Adafruit IO Trigger



Create New Action

You can receive an email when the battery has been fully charged.

Go to https://io.adafruit.com (https://adafru.it/eZ8)and click on **Actions**
Click the **New Actions** button.
Choose **Yes, use Blockly!**

Reactive Trigger Setup

Click on the **Triggers** menu, then click on **Reactive**

Drag the **Reactive** block and snap it to the **Trigger** section in the **Action Root** block

Click on the **Values** menu, then click on the **Feed** block

Drag the **Feed** block and snap it to the Feed section in the **Reactive** block. Then, select the **battery-monitor** feed from the dropdown menu

Click on the **Values** menu, then click the **A number, whole or decimal** block.

Drag the **A number, whole or decimal** block and snap it to the **Feed or Value** section of the Reactive block. Then, type "100" into the **Feed or Value** block.

Under the **Operator** section, select ">=" from the dropdown menu.

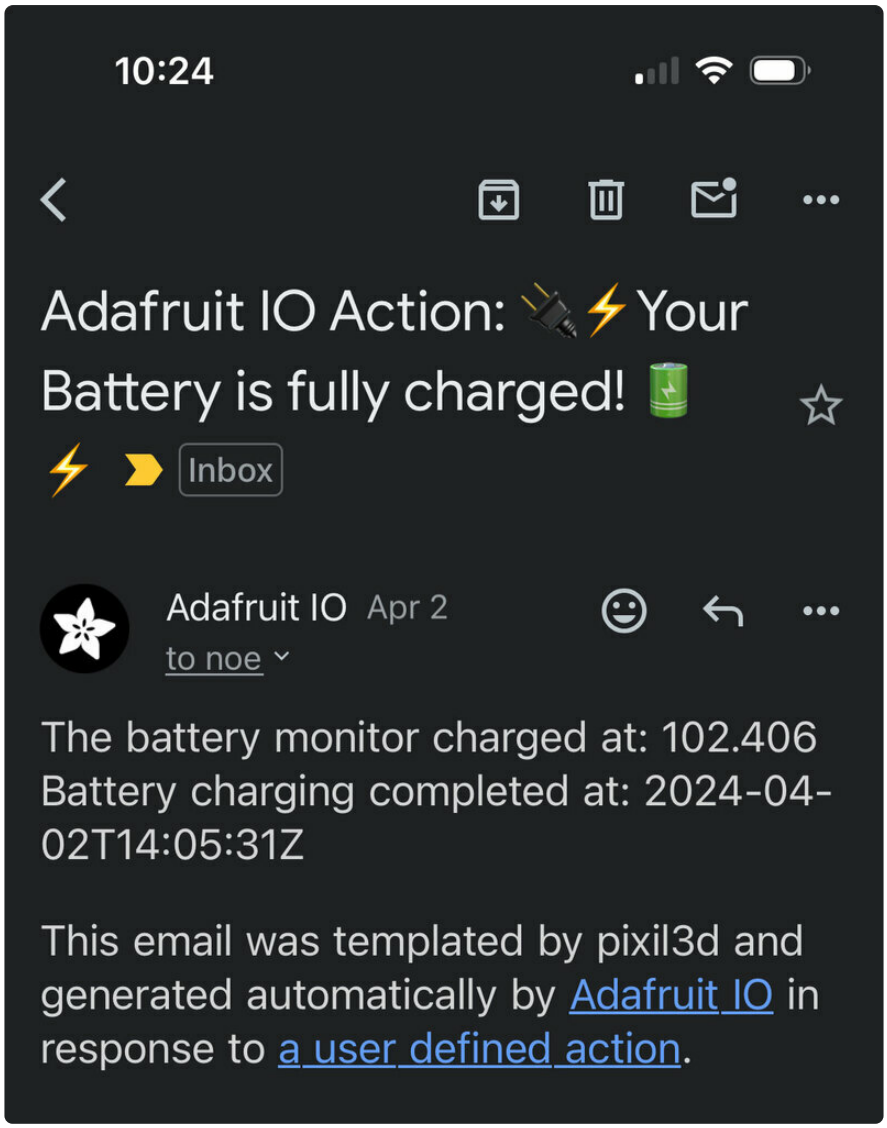
Action Setup

Click on Actions menu, then click on **Email**
Drag the **Email** block and snap it to the **Act ion** section in the Action Root block.
Type your preferred text into the **subject** and **body** in the **Email** block.
Click on Values menu, then click on a **Feed** block.
Drag the Feed block into the using section in the **Email** block. Then, select **battery-monitor** from the **dropdown** menu

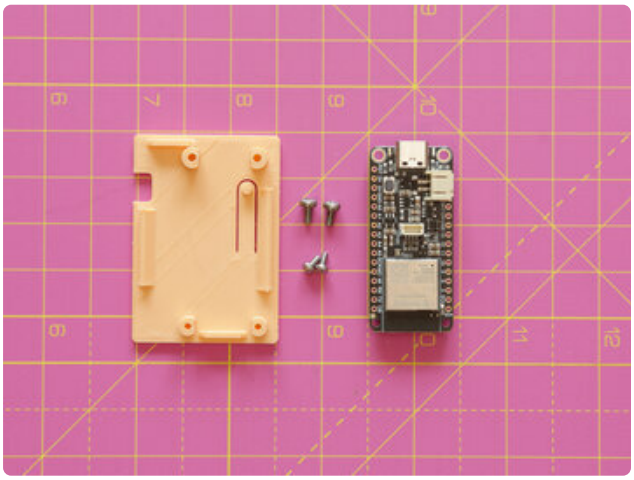
Click the **Save** button when you are finished.

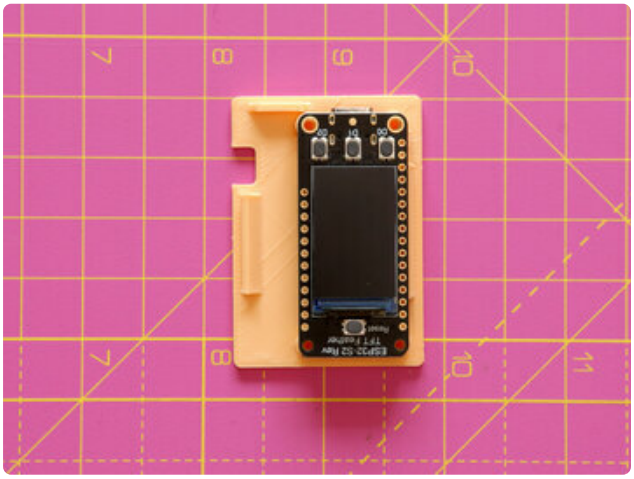
New Email

You'll now receive an email when the battery is fully charged.


10:24
Adafruit IO Action: 🔌⚡Your Battery is fully charged! 🔋
⚡ Inbox
Adafruit IO Apr 2
to noe
The battery monitor charged at: 102.406
Battery charging completed at: 2024-04-02T14:05:31Z
This email was templated by pixil3d and generated automatically by Adafruit IO in response to a user defined action.

Assembly





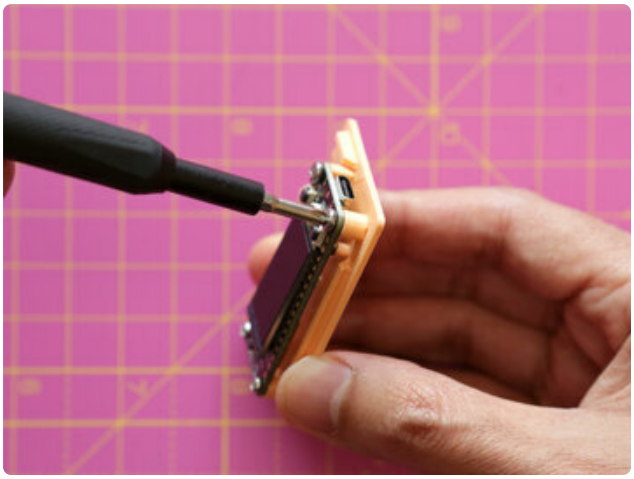
Installing Feather

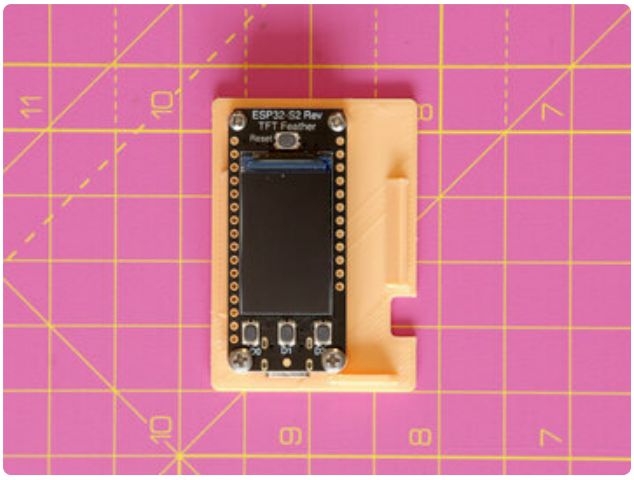
Get the 3D printed back cover, hardware screws, and Feather ready to assemble.

Place the Feather over the back cover with the mounting holes lined up and the TFT screen facing up.

Use the following hardware to secure the Feather to the back cover.

2x M2 x 6mm long machine screws
2x M2.5 x 6mm long machine screws

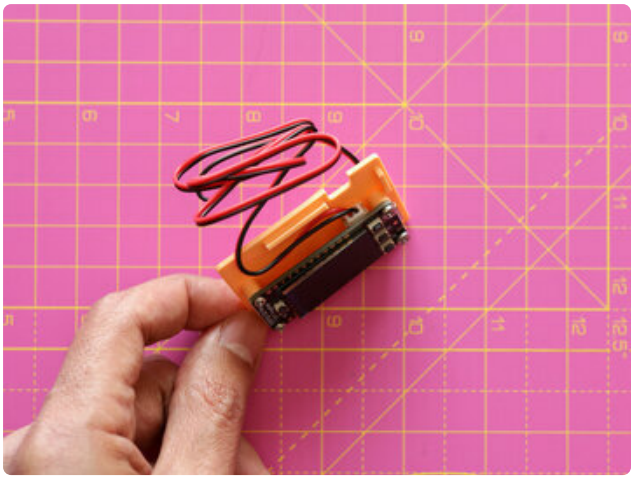




Secure Feather

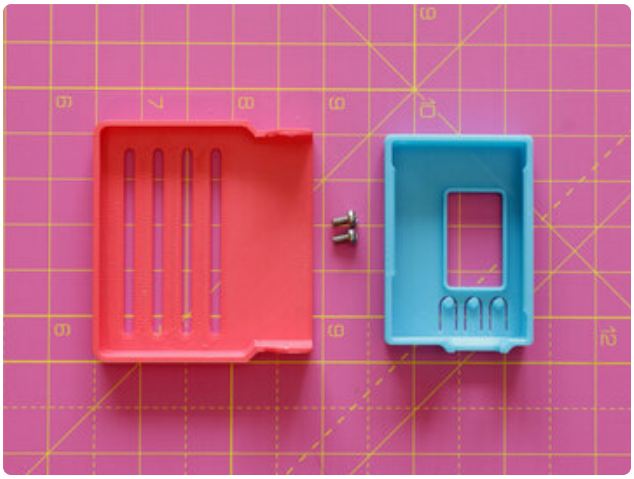
Install and fasten the hardware screws to secure the Feather ESP32-S2 to the 3D printed back cover.

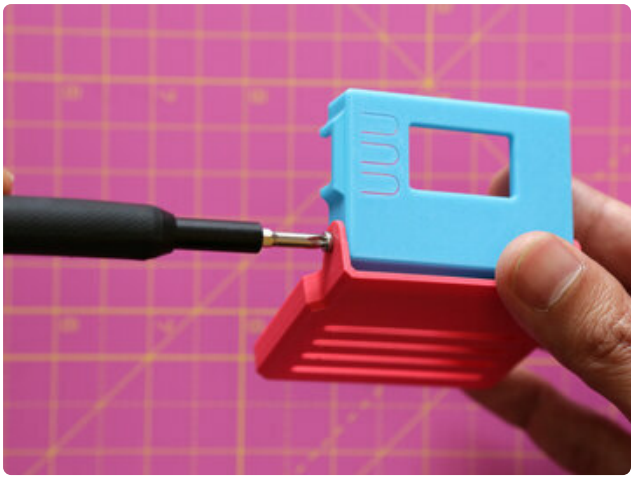
Ensure the Feather has been secured in the correct orientation.



Connect JST Extension

Plug in the 2-pin JST extension cable to the battery port on-board the Feather ESP32-S2.





Install Case & Battery Tray

Slide the 3D printed case in between the battery tray's two mounting tabs.

Line up the mounting holes, then insert and fasten two M2.5 x 6mm long machine screws to secure the two parts together.

Assemble Enclosure

Orient the back cover with the front case and begin to fit the JST extension cable inside the enclosure.

Fit the JST extensions cables socket connector through the notch in the back cover.

Press the front case and back cover together to close them shut.

Install Battery

Connect the battery to the JST extension cable.

Place the battery into the 3D printed tray.

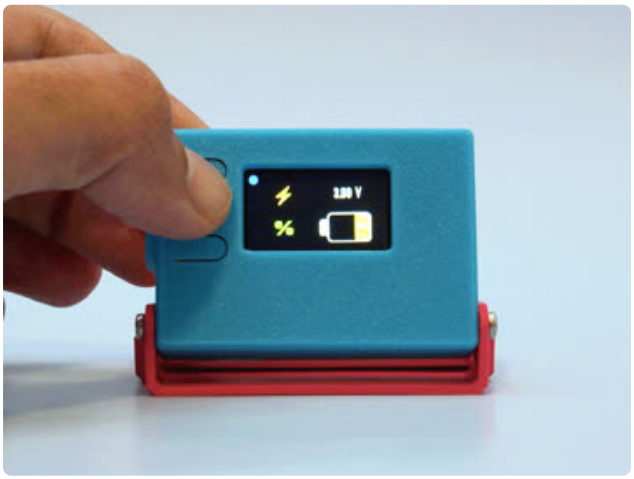
Final Build

Congratulations on your build!

The Feather ESP32-S2 will power on immediately after connecting the battery. Plug in a USB cable to your Feather to begin charging the battery.

Allow the Feather to connect to your WiFi and establish a voltage reading. This may take a minute or two to output to the TFT display.

Usage



Button D0

Press the **D0 button (top)** to enable sending data to Adafruit IO. The blue circle indicates data is sending to Adafruit IO.

Button D1

Press the **D1 button (middle)** to toggle between icon view and plain text view.

Button D2

Press the **D2 button (bottom)** to toggle between the battery icon and battery percentage text.

USB Charging

Use a 5V 1A power supply and USB-C type cable to charge the battery.