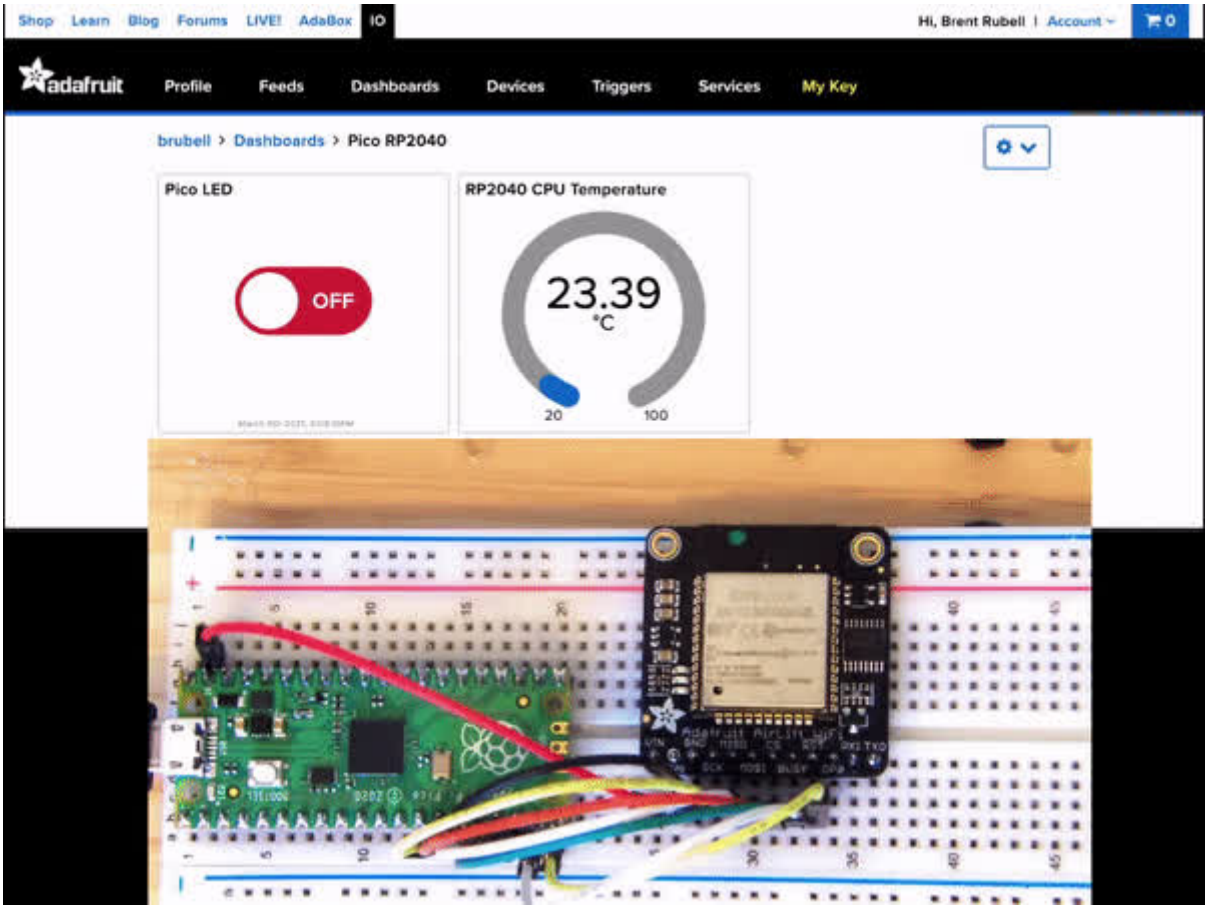
Quickstart IoT - Raspberry Pi Pico RP2040 with WiFi

Created by Brent Rubell

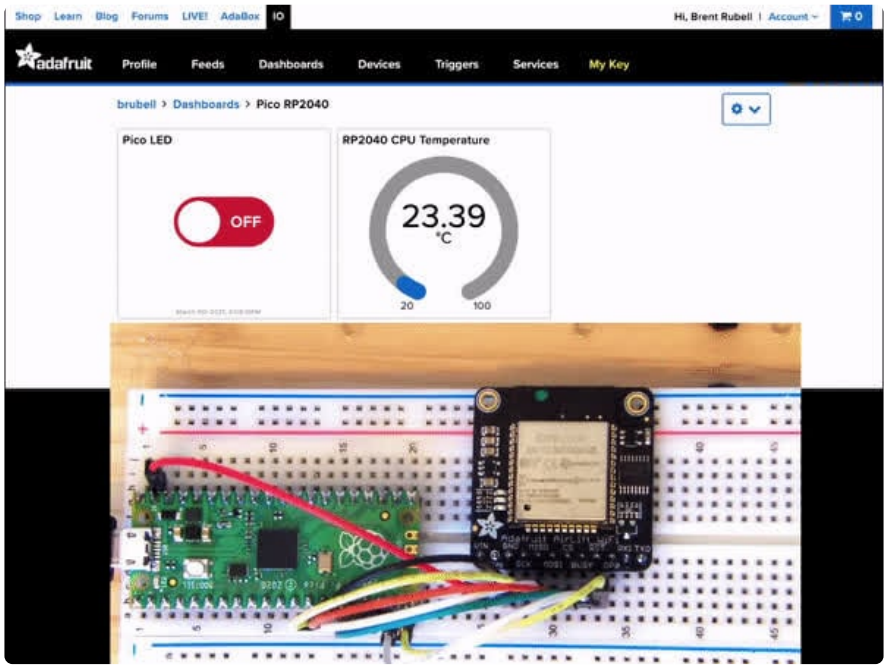


https://learn.adafruit.com/quickstart-rp2040-pico-with-wifi-and-circuitpython

Last updated on 2026-02-03 10:56:11 PM UTC

Table of Contents

Overview



Connect your Raspberry Pi Pico CircuitPython project to the internet by adding an AirLift breakout board. The Adafruit AirLift is a breakout board that lets you use the ESP32 as a WiFi co-processor for a Pico.

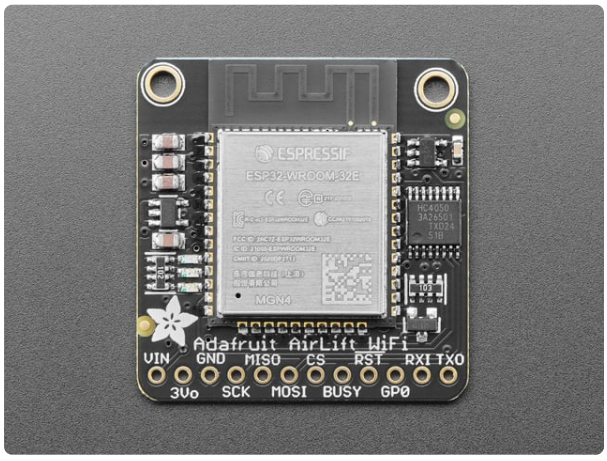
In this guide, you will wire up a Pico to an AirLift breakout and connect to the internet. Then, you'll learn how to fetch JSON and raw text data from the internet. Finally, an example is included for connecting your Pico to Adafruit IO, the Adafruit internet-of-things service (https://adafru.it/fH9), so you can interact with and visualize your project's data.

Why use AirLift?

Having WiFi managed by a separate chip means your code is simpler, you don't have to cache socket data, or compile in & debug an SSL library. With AirLift, you can send basic but powerful socket-based commands over 8MHz SPI for high speed data transfer. The ESP32 can handle all the heavy lifting of connecting to a WiFi network and transferring data from a site, and using the latest TLS/SSL encryption (it has root certificates pre-burned in).

The firmware on board is a slight variant of the Arduino WiFiNINA core, which works great, (https://adafru.it/E7O) and **our Adafruit IO Libraries for** CircuitPython **(https://adafru.it/Ean) support AirLift!**

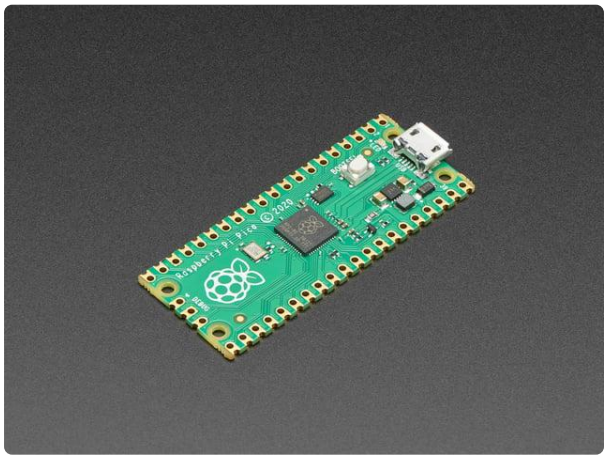
Parts



Adafruit AirLift – ESP32 WiFi Co-Processor Breakout Board

Give your plain ol' microcontroller project a lift with the Adafruit AirLift - a breakout board that lets you use the powerful ESP32 as a WiFi co-processor. You probably...

https://www.adafruit.com/product/4201



Raspberry Pi Pico RP2040

The Raspberry Pi foundation changed single-board computing when they released the Raspberry Pi computer, now they're ready to...

https://www.adafruit.com/product/4864

1 x USB Cable — https://www.adafruit.com/product/592

USB cable - USB A to Micro-B - 3 foot long

---

1 x Breadboard — https://www.adafruit.com/product/239

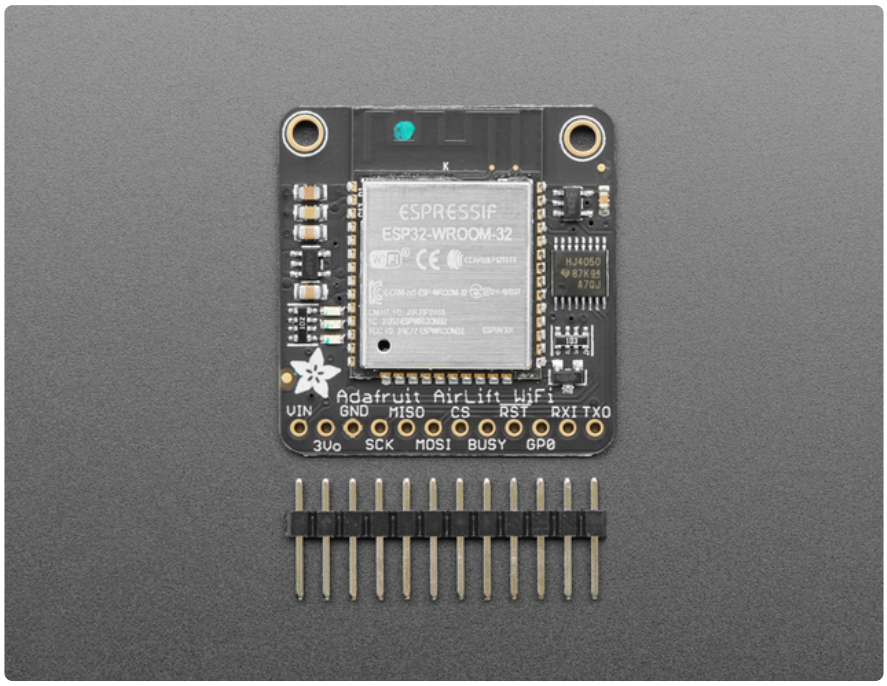
Full sized breadboard

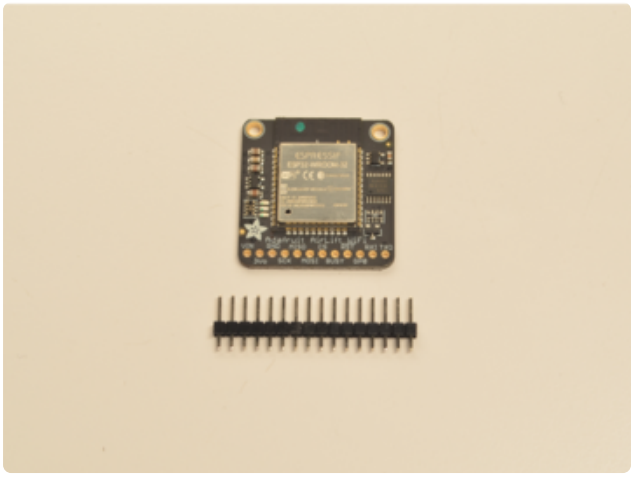
---

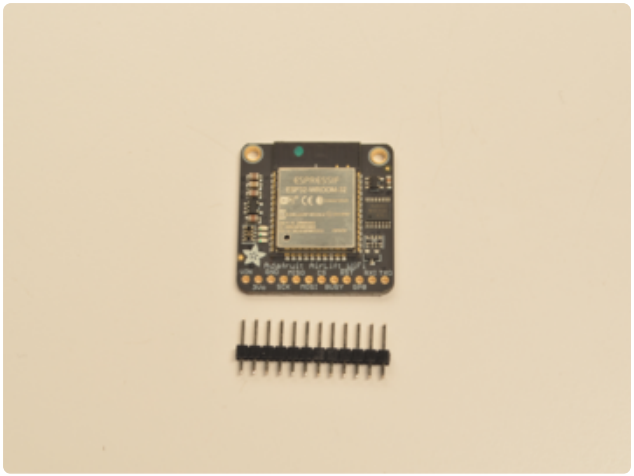
1 x Breadboarding Wires https://www.adafruit.com/product/153

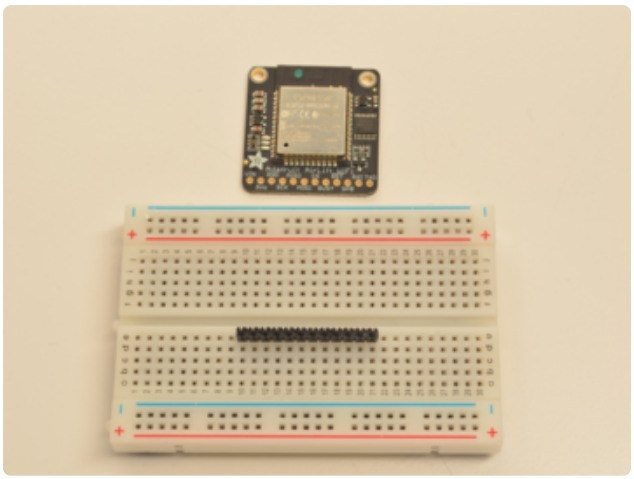
Breadboarding wire bundle

Assembly



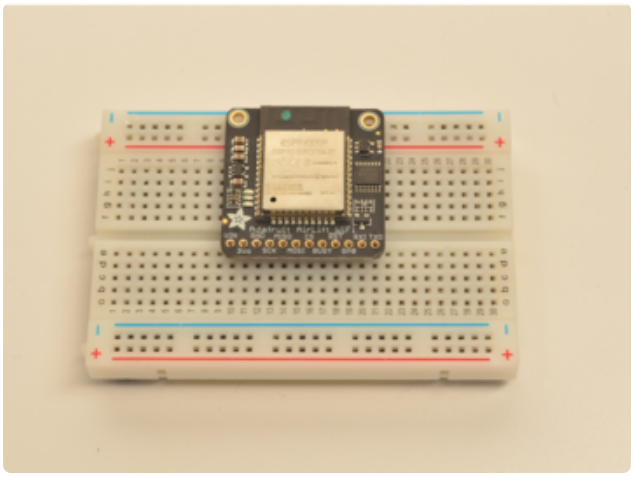






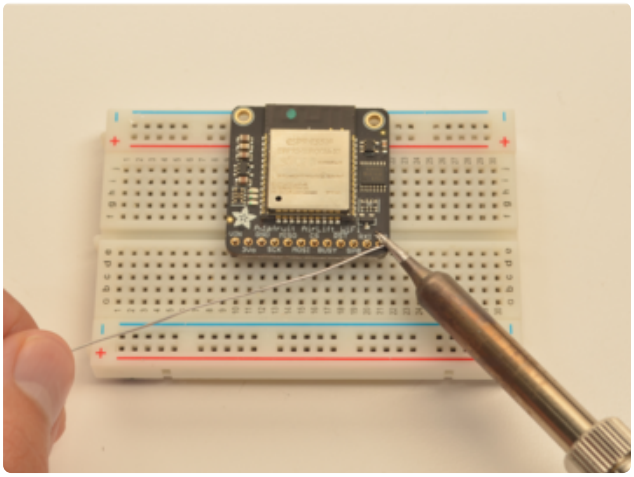
Prepare the header strip:

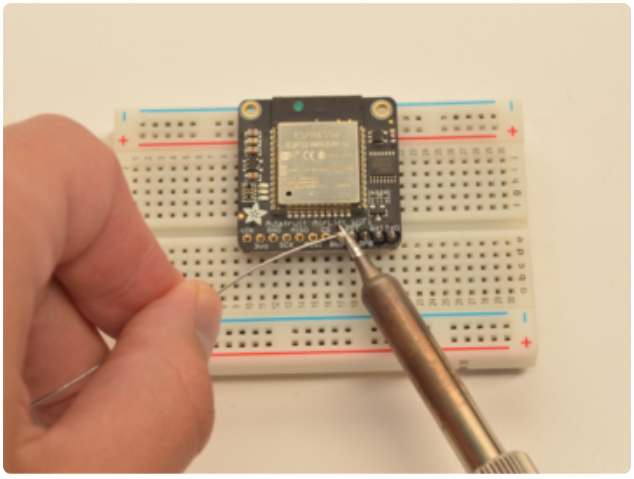
Cut the strip to length if necessary. It will be easier to solder if you insert it into a breadboard - **long pins down**

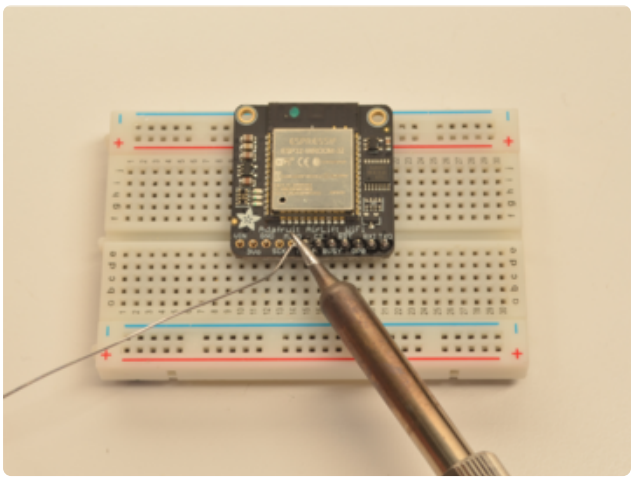


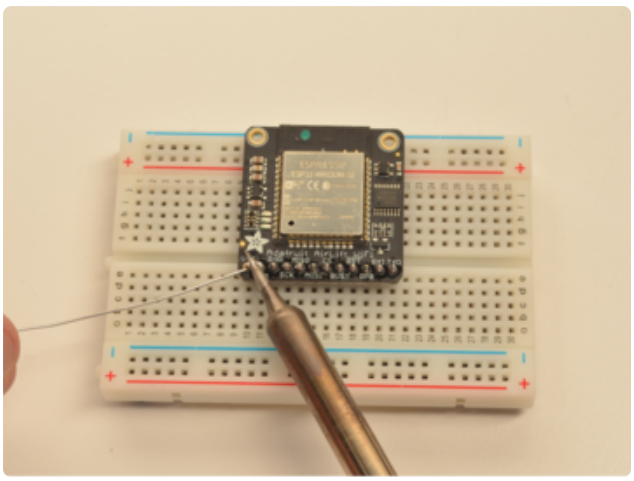
Add the breakout board:

Place the breakout board over the pins so that the short pins poke through the breakout pads





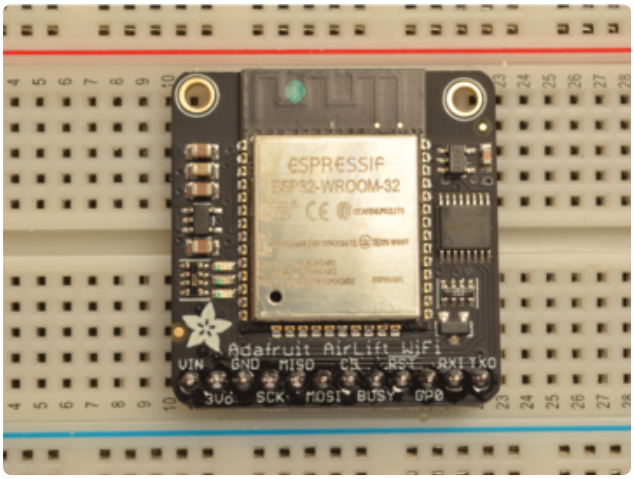




And Solder!

Be sure to solder all 12 pins for reliable electrical contact.

(For tips on soldering, be sure to check out our Guide to Excellent Soldering (https://adafru.it/aTk)).



You're done! Check your solder joints visually and continue onto the next steps

Installing the Mu Editor

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!

Mu is our recommended editor - please use it (unless you are an experienced er with a favorite editor already!). While it has been announced end of life 026, it still works fine.

are free to use whatever text editor you wish along with a terminal program to connect to the CircuitPython REPL. Thonny is one such editor.

Download and Install Mu

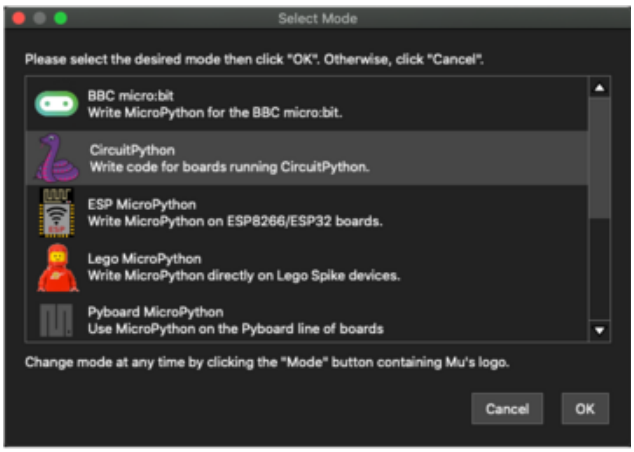
Download Mu from [https://codewith.mu](https://adafru.it/Be6).

Click the **Download** link for downloads and installation instructions.

Click **Start Here** to find a wealth of other information, including extensive tutorials and and how-to's.

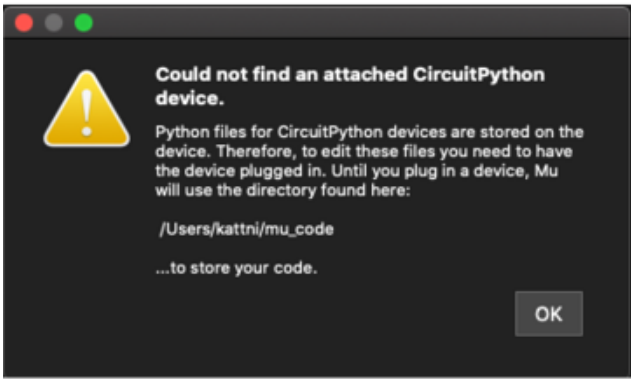
dows users: due to the nature of MSI installers, please remove old ions of Mu before installing the latest version.

Starting Up Mu



The first time you start Mu, you will be prompted to select your 'mode' - you can always change your mind later. For now please select **CircuitPython**!

The current mode is displayed in the lower right corner of the window, next to the "gear" icon. If the mode says "Microbit" or something else, click the **Mode** button in the upper left, and then choose "CircuitPython" in the dialog box that appears.



Mu attempts to auto-detect your board on startup, so if you do not have a CircuitPython board plugged in with a **CIRCUITPY** drive available, Mu will inform you where it will store any code you save until you plug in a board.

To avoid this warning, plug in a board and ensure that the **CIRCUITPY** drive is mounted before starting Mu.

Using Mu

You can now explore Mu! The three main sections of the window are labeled below; the button bar, the text editor, and the serial console / REPL.

Now you're ready to code! Let's keep going...

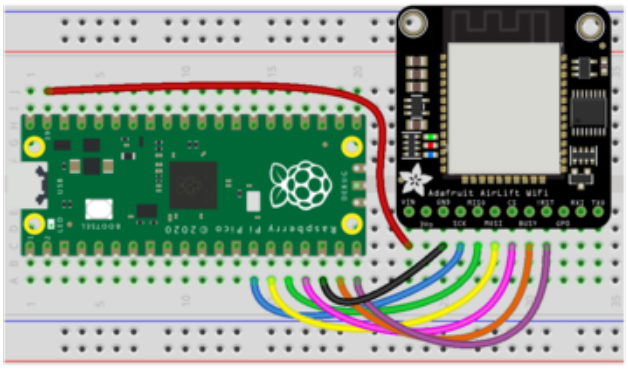
CircuitPython WiFi

guide requires CircuitPython to be installed on your Pico

If you have not yet installed CircuitPython on your Raspberry Pi Pico, visit our Getting Started product page for the Pico and come back here when you've installed CircuitPython (https://adafru.it/QaQ).

Pico Wiring

MUST use the Pico's VSYS pin for powering the AirLift Breakout.



Pico **VSYS** to **AirLift Vin**
Pico GND to **AirLift GND**
Pico GP10 (SPI1 SCK) to **AirLift SCK**
Pico GP11 (SPI1 TX) to **AirLift MOSI**
Pico GP12 (SPI1 RX) to **AirLift MISO**
Pico GP13 (SPI1 CSn) to **AirLift CS**
Pico GP14 to **AirLift BUSY**
Pico GP15 to **AirLift !RST**

For more information about the SPI peripherals and pinouts of the Pico, check out this guide (https://adafru.it/Qsd).

CircuitPython Installation of ESP32SPI Library

You'll need to install the Adafruit CircuitPython ESP32SPI (https://adafru.it/DWV) library on your CircuitPython board.

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

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

You can manually install the necessary libraries from the bundle:

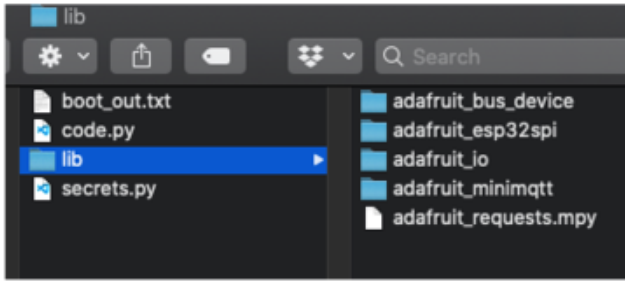
- **adafruit_bus_device**
- **adafruit_minimqtt**
- **adafruit_io**
- **adafruit_esp32_spi**
- **adafruit_requests**

CircuitPython Usage

Connect to the board's serial REPL (https://adafru.it/Awz) via Mu so you are at the CircuitPython >>> prompt. Once that's done, load up the following example in Mu and save it to the Pico as **code.py**.

```
import board
import busio
from digitalio import DigitalInOut

from adafruit_esp32spi import adafruit_esp32spi
import adafruit_requests as requests

print("Raspberry Pi RP2040 - ESP32SPI hardware test")

esp32_cs = DigitalInOut(board.GP13)
esp32_ready = DigitalInOut(board.GP14)
esp32_reset = DigitalInOut(board.GP15)

spi = busio.SPI(board.GP10, board.GP11, board.GP12)
esp = adafruit_esp32spi.ESP_SPIcontrol(spi, esp32_cs, esp32_ready, esp32_reset)

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])

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

```
print("Done!")
```



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

The output to the serial monitor should look something like the following:

```
Raspberry Pi RP2040 - ESP32SPI hardware test
ESP32 found and in idle mode
Firmware vers. bytearray(b'1.3.0\x00')
MAC addr: ['0xbd', '0xb0', '0xe', '0x33', '0x4f', '0xc4']
                                  RSSI: -57
                          RSSI: -66
        Fios-5dLNb                RSSI: -72
                                  RSSI: -85
                          RSSI: -86
        Fios-hZg3C                RSSI: -87
                                  RSSI: -88
                          RSSI: -88
                                  RSSI: -91
                                  RSSI: -91
Done!

Code done running.
```

Make sure you see the same output! If you don't, check your wiring. Once you've succeeded, continue onto the next page!

u can read the Firmware and MAC address but fails on scanning SSIDs, ck that your AirLift is wired to the Pico's VSYS pin.

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.

settings.toml file should be stored in the main directory of your **CUITPY** 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.**

every project uses the same variable name for each entry in the **ngs.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"`

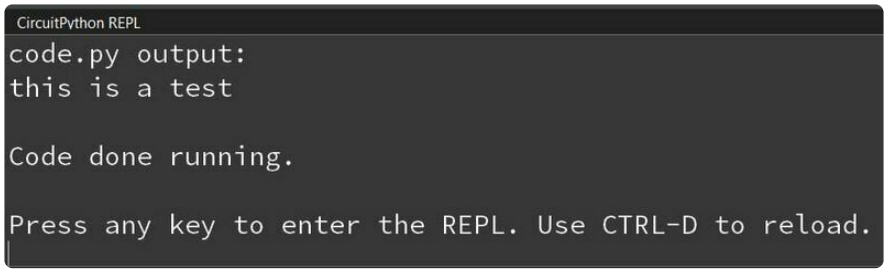
- Integers are **not** quoted and may be written in decimal with optional sign (`+1`, `-1`, `1000`) or hexadecimal (`0xabcd`).
    - Floats (decimal numbers), octal (`0o567`) and binary (`0b11011`) are not supported.
- Use `\u` escapes for weird characters, `\x` and `\ooo` escapes are not available in **.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!

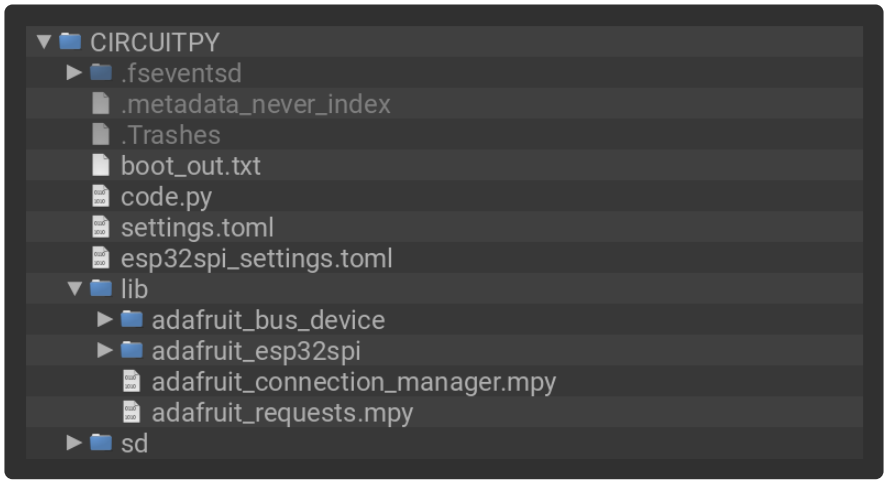
Connect to WiFi

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:



ate to CircuitPython 9.2.x or later to use this example.

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

```
from os import getenv

import adafruit_connection_manager
import adafruit_requests
import board
import busio
from digitalio import DigitalInOut

# Use this import for adafruit_esp32spi version 11.0.0 and up.
# Note that frozen libraries may not be up to date.
# import adafruit_esp32spi
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
ssid = getenv("CIRCUITPY_WIFI_SSID")
password = getenv("CIRCUITPY_WIFI_PASSWORD")

print("ESP32 SPI webclient test")

TEXT_URL = "http://wifitest.adafruit.com/testwifi/index.html"
JSON_URL = "http://wifitest.adafruit.com/testwifi/sample.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)
print("MAC addr:", ":".join("%02X" % byte for byte in esp.MAC_address))

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

print("Connecting to AP...")
while not esp.is_connected:
    try:
        esp.connect_AP(ssid, password)
    except OSError as e:
```

```
        print("could not connect to AP, retrying: ", e)
        continue
print("Connected to", esp.ap_info.ssid, "\tRSSI:", esp.ap_info.rssi)
print("My IP address is", esp.ipv4_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) (https://adafru.it/Bec).

```
&gt;&gt;&gt; import wifitest
ESP32 SPI webclient test
ESP32 found and in idle mode
Firmware vers. 1.7.5
MAC addr: 24:C9:DC:BD:0F:3F
    HomeNetwork             RSSI: -46
    HomeNetwork             RSSI: -76
    Fios-12345              RSSI: -92
    FiOS-AB123              RSSI: -92
    NETGEAR53               RSSI: -93
Connecting to AP...
Connected to HomeNetwork    RSSI: -45
My IP address is 192.168.1.245
IP lookup adafruit.com: 104.20.39.240
Ping google.com: 30 ms
Fetching text from http://wifitest.adafruit.com/testwifi/index.html
----------------------------------------
This is a test of Adafruit WiFi!
If you can read this, its working :)
----------------------------------------

Fetching json from http://wifitest.adafruit.com/testwifi/sample.json
----------------------------------------
{'fun': True, 'company': 'Adafruit', 'founded': 2005, 'primes': [2, 3, 5], 'pi':
3.14, 'mixed': [False, None, 3, True, 2.7, 'cheese']}
```

```
----------------------------------------
Done!
```

Going over the example above, here's a breakdown of what the program is doing:

- Initialize 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)

#...

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

- Get the socket pool and the SSL context, and then tell the `adafruit_requests` library about them.

```
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)
```

- Verify 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:", ":".join("%02X" % byte for byte in esp.MAC_address))
```

- Perform a scan of all access points it can see and print out the name and signal strength.

```
for ap in esp.scan_networks():
    print("\t%-23s RSSI: %d" % (ap.ssid, ap.rssi))
```

- Connect to the AP we've defined here, then print out the local IP address. Then attempt 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(ssid, password)
    except OSError as e:
        print("could not connect to AP, retrying: ", e)
        continue
print("Connected to", esp.ap_info.ssid, "\tRSSI:", esp.ap_info.rssi)
print("My IP address is", esp.ipv4_address)
print(
    "IP lookup adafruit.com: %s" %
esp.pretty_ip(esp.get_host_by_name("adafruit.com"))
)
```

Now we're getting to the really interesting part of the example program. We've written a library for web fetching web data, named adafruit_requests (https://adafru.it/FpW). It is a lot like the regular Python library named requests (https://adafru.it/1af4). This library allows you to send HTTP and HTTPS requests easily and provides helpful methods for parsing the response from the server.

- Here is the part of the example program is fetching text data from a URL.

```
TEXT_URL = "http://wifitest.adafruit.com/testwifi/index.html"  # Further up in the
program

# ...

print("Fetching text from", TEXT_URL)
r = requests.get(TEXT_URL)
print('-' * 40)
print(r.text)
print('-' * 40)
r.close()
```

- Finally, here the program is fetching some JSON data. The `adafruit_requests` library will parse the JSON into a Python dictionary whose structure is the same as the structure of the JSON.

```
JSON_URL = "http://wifitest.adafruit.com/testwifi/sample.json"   # Further up in the
program

# ...

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

```
print('-' * 40)
r.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.

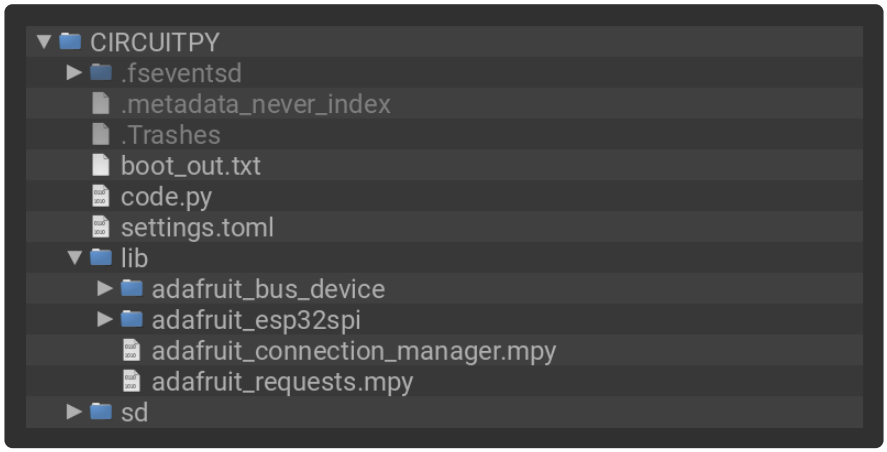
To use with CircuitPython, you need to first install a few libraries, into the lib folder on your **CIRCUITPY** drive. Then you need to update **code.py** with the example script.

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.

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

import os

import adafruit_connection_manager
import board
import busio
from adafruit_esp32spi import adafruit_esp32spi
from digitalio import DigitalInOut

import adafruit_requests

# Get WiFi details, ensure these are setup in settings.toml
ssid = os.getenv("CIRCUITPY_WIFI_SSID")
password = os.getenv("CIRCUITPY_WIFI_PASSWORD")

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

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

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

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

print("Connecting to AP...")
while not radio.is_connected:
    try:
```

```
        radio.connect_AP(ssid, password)
    except RuntimeError as e:
        print("could not connect to AP, retrying: ", e)
        continue
print("Connected to", str(radio.ap_info.ssid, "utf-8"), "\tRSSI:",
radio.ap_info.rssi)

# Initialize a requests session
pool = adafruit_connection_manager.get_radio_socketpool(radio)
ssl_context = adafruit_connection_manager.get_radio_ssl_context(radio)
requests = adafruit_requests.Session(pool, ssl_context)

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

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

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

    json_data = response.json()
    headers = json_data["headers"]
    print("Response's Custom User-Agent Header: {0}".format(headers["User-Agent"]))
    print("-" * 60)

    # Read Response's HTTP status code
    print("Response HTTP Status Code: ", response.status_code)
    print("-" * 60)
```

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



WiFi Manager

The way the examples above connect to WiFi works but it's a little finicky. Since WiFi is not necessarily so reliable, you may have disconnects and need to reconnect. For more advanced uses, we recommend using the `WiFiManager` class. 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.

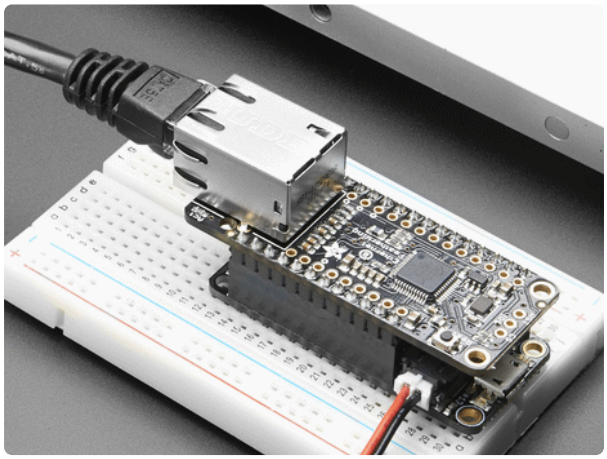
Here's a more advanced example that shows using the `WiFiManager` and also how to fetch the current time from a web source.

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

import time
from os import getenv

import board
import busio
import neopixel
import rtc
from digitalio import DigitalInOut

# Use these imports for adafruit_esp32spi version 11.0.0 and up.
# Note that frozen libraries may not be up to date.
# import adafruit_esp32spi
# from adafruit_esp32spi.wifimanager import WiFiManager
from adafruit_esp32spi import adafruit_esp32spi
from adafruit_esp32spi.adafruit_esp32spi_wifimanager import WiFiManager

# Get wifi details and more from a settings.toml file
# tokens used by this Demo: CIRCUITPY_WIFI_SSID, CIRCUITPY_WIFI_PASSWORD
ssid = getenv("CIRCUITPY_WIFI_SSID")
password = getenv("CIRCUITPY_WIFI_PASSWORD")

print("ESP32 local time")

TIME_API = "http://worldtimeapi.org/api/ip"

# 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_pixel = neopixel.NeoPixel(board.NEOPIXEL, 1, brightness=0.2)
"""Uncomment below for ItsyBitsy M4"""
# status_pixel = 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_pixel = adafruit_rgbled.RGBLED(RED_LED, BLUE_LED, GREEN_LED)

wifi = WiFiManager(esp, ssid, password, status_pixel=status_pixel)

the_rtc = rtc.RTC()

response = None
```

```
while True:
    try:
        print("Fetching json from", TIME_API)
        response = wifi.get(TIME_API)
        break
    except OSError as e:
        print("Failed to get data, retrying\n", e)
        continue

json = response.json()
current_time = json["datetime"]
the_date, the_time = current_time.split("T")
year, month, mday = (int(x) for x in the_date.split("-"))
the_time = the_time.split(".")[0]
hours, minutes, seconds = (int(x) for x in the_time.split(":"))

# We can also fill in these extra nice things
year_day = json["day_of_year"]
week_day = json["day_of_week"]
is_dst = json["dst"]

now = time.struct_time((year, month, mday, hours, minutes, seconds, week_day,
year_day, is_dst))
print(now)
the_rtc.datetime = now

while True:
    print(time.localtime())
    time.sleep(1)
```

Further Information

For more information on the basics of doing networking in CircuitPython, see this guide:



Networking in CircuitPython
By Anne Barela

https://learn.adafruit.com/networking-in-circuitpython

Usage with Adafruit IO

Adafruit IO is an internet of things platform (https://adafru.it/BRB) (designed by Adafruit (https://adafru.it/R5C)!) to help connect your project to the internet. You're going to send the Pico's CPU temperature to Adafruit IO and display it on a

dashboard page. You'll add a toggle block to turn on or off the Pico's LED from the internet.

Secrets File Setup for Adafruit IO

To connect to Adafruit IO, you will need to start by modifying your secrets file to include your Adafruit IO credentials.

- **If you do not already have a secrets file**, visit this page and come back here when you've created a secrets file (https://adafru.it/EF-)

Navigate to io.adafruit.com (https://adafru.it/fsU) and click My Key to obtain your Adafruit IO username and Active Key. Add these credentials into the secrets file, which will now look something like the following:

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

secrets = {
    'ssid' : 'home ssid',
    'password' : 'my password',
    'timezone' : "America/New_York", # http://worldtimeapi.org/timezones
    'aio_username': 'MY_ADAFRUIT_IO_USERNAME',
    'aio_key': 'MY_ADAFRUIT_IO_PASSWORD'
    }
```

Create Adafruit IO Feeds

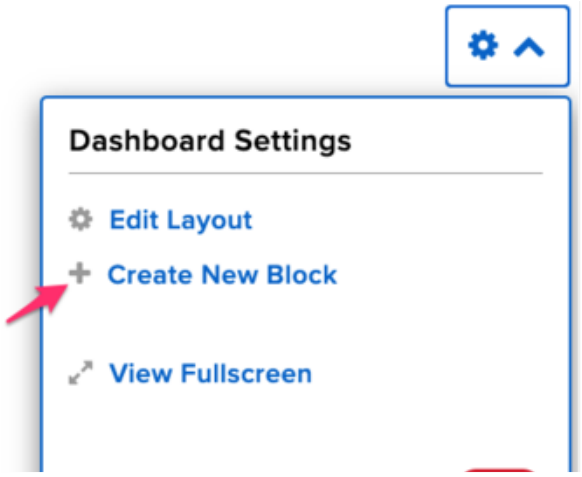
Next, set up two new Adafruit IO feeds - `temperature` and `led`. The temperature feed will store the value of the Pico's CPU temperature and the LED feed will store the state of a toggle switch block on an Adafruit IO dashboard.

- If you do not know how to set up a feed on Adafruit IO, follow this page and come back when you've created the two feeds listed above (https://adafru.it/f5k).

Create an Adafruit IO Dashboard

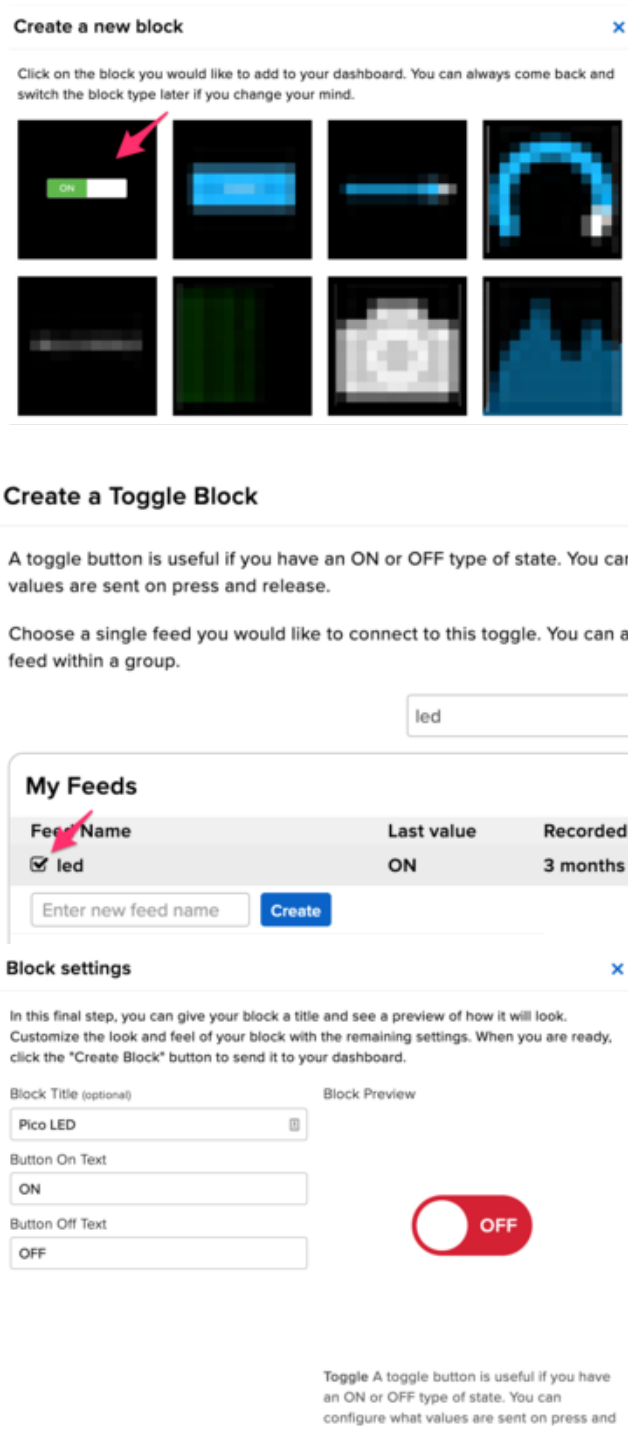
Navigate to the Adafruit IO Dashboards page (https://adafru.it/eIS) and create a new dashboard named **Pico RP2040**.

- If you do not know how to create a dashboard on Adafruit IO, follow this page and come back when you've created a new dashboard (https://adafru.it/Fm7).



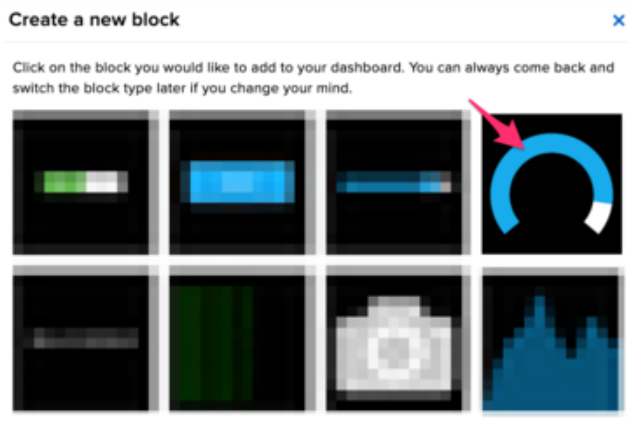
Click the cog in the upper right hand corner of the dashboard to bring up the Dashboard Settings.

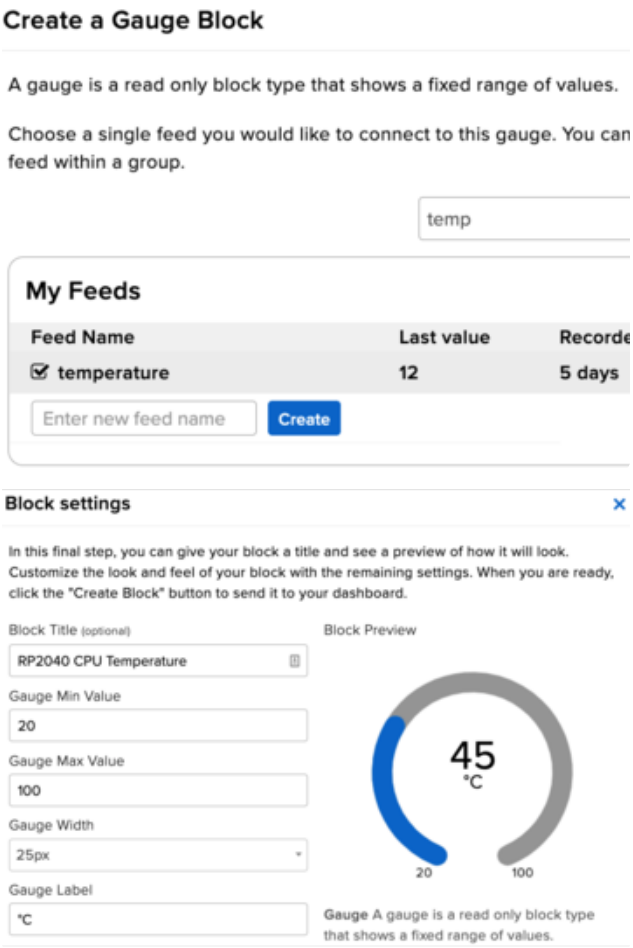
From the Dashboard settings, **click Create New Block**



Add a toggle block to turn the Pico's LED on or off.

From the Create a New Block screen, select the Toggle Block.
Connect the led feed to the block by ticking the checkbox.
In the final screen, click **Create Block**.

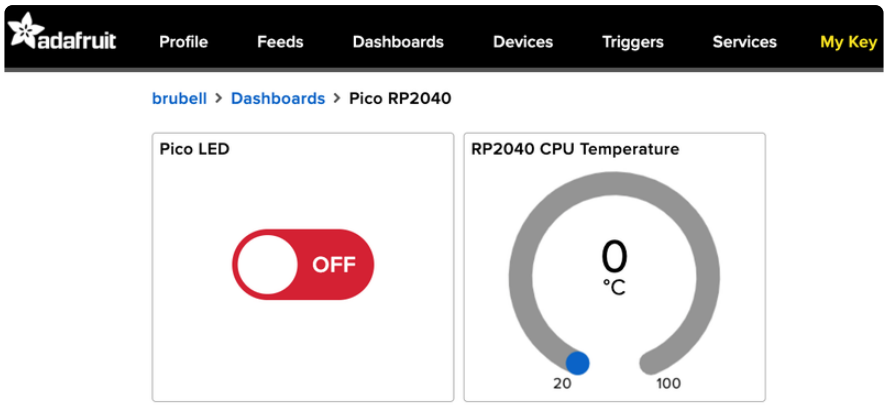




Next, create a gauge block to display the Pico's CPU temperature.

Create a new gauge block.
Connect the temperature feed to the block by ticking the checkbox.
Under Block Settings, you may change the block's look and feel.
When you've finished customizing the block, click Create Block.

Once you've finished creating both blocks, your dashboard should look something like the following image:



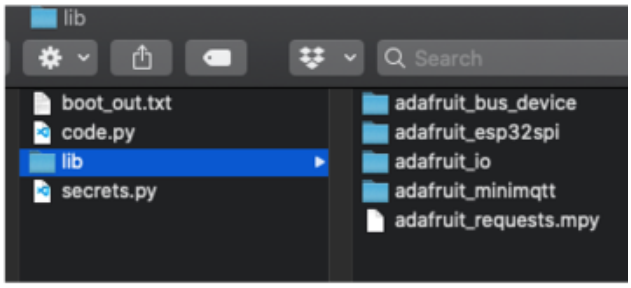
Code

Copy the code below into Mu and save as a file named **code.py** on your **CIRCUITPY** drive.

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

import time
from os import getenv

import adafruit_connection_manager
import adafruit_minimqtt.adafruit_minimqtt as MQTT
import board
import busio
from adafruit_esp32spi import adafruit_esp32spi, adafruit_esp32spi_wifimanager
from digitalio import DigitalInOut
from microcontroller import cpu

from adafruit_io.adafruit_io import IO_MQTT

# Get WiFi details and Adafruit IO keys, ensure these are setup in settings.toml
# (visit io.adafruit.com if you need to create an account, or if you need your
Adafruit IO key.)
ssid = getenv("CIRCUITPY_WIFI_SSID")
password = getenv("CIRCUITPY_WIFI_PASSWORD")
aio_username = getenv("ADAFRUIT_AIO_USERNAME")
aio_key = getenv("ADAFRUIT_AIO_KEY")

### WiFi ###

# Raspberry Pi RP2040
esp32_cs = DigitalInOut(board.GP13)
esp32_ready = DigitalInOut(board.GP14)
esp32_reset = DigitalInOut(board.GP15)

spi = busio.SPI(board.GP10, board.GP11, board.GP12)
esp = adafruit_esp32spi.ESP_SPIcontrol(spi, esp32_cs, esp32_ready, esp32_reset)

wifi = adafruit_esp32spi_wifimanager.WiFiManager(esp, ssid, password)

# Configure the RP2040 Pico LED Pin as an output
led_pin = DigitalInOut(board.LED)
led_pin.switch_to_output()


# Define callback functions which will be called when certain events happen.
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(f"Subscribed to {topic} with QOS level {granted_qos}")


def publish(client, userdata, topic, pid):
    # This method is called when the client publishes data to a feed.
    print(f"Published to {topic} with PID {pid}")
    if userdata is not None:
        print("Published User data: ", end="")
        print(userdata)


def disconnected(client):
    # Disconnected function will be called when the client disconnects.
    print("Disconnected from Adafruit IO!")


def on_led_msg(client, topic, message):
    # Method called whenever user/feeds/led has a new value
    print(f"New message on topic {topic}: {message} ")
    if message == "ON":
        led_pin.value = True
    elif message == "OFF":
        led_pin.value = False
    else:
        print("Unexpected message on LED feed.")


# Connect to WiFi
print("Connecting to WiFi...")
wifi.connect()
print("Connected!")

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",
    port=1883,
    username=aio_username,
    password=aio_key,
    socket_pool=pool,
    ssl_context=ssl_context,
)

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

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

# Set up a callback for the led feed
io.add_feed_callback("led", on_led_msg)

# Connect to Adafruit IO
print("Connecting to Adafruit IO...")
io.connect()

# Subscribe to all messages on the led feed
io.subscribe("led")
```

```
prv_refresh_time = 0.0
while True:
    # Poll for incoming messages
    try:
        io.loop()
    except (ValueError, RuntimeError) as e:
        print("Failed to get data, retrying\n", e)
        wifi.reset()
        wifi.connect()
        io.reconnect()
        continue
    # Send a new temperature reading to IO every 30 seconds
    if (time.monotonic() - prv_refresh_time) > 30:
        # take the cpu's temperature
        cpu_temp = cpu.temperature
        # truncate to two decimal points
        cpu_temp = str(cpu_temp)[:5]
        print("CPU temperature is %s degrees C" % cpu_temp)
        # publish it to io
        print("Publishing %s to temperature feed..." % cpu_temp)
        io.publish("temperature", cpu_temp)
        print("Published!")
        prv_refresh_time = time.monotonic()
```



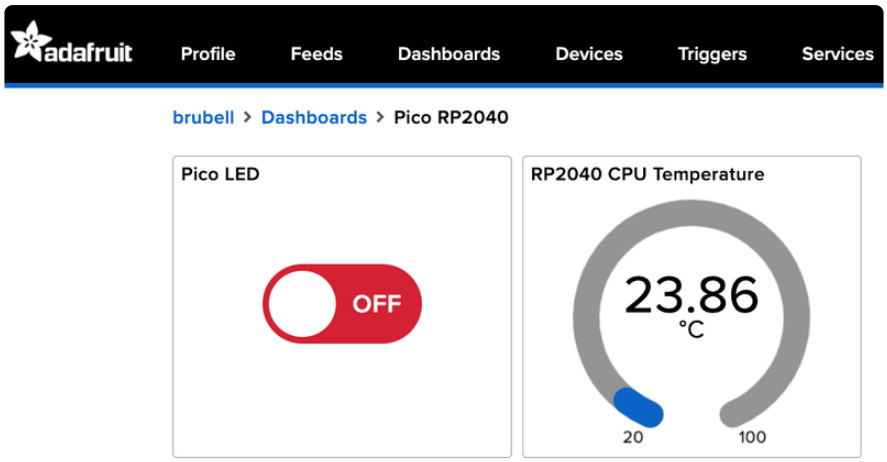
Once all the files are copied from your computer to the Pico, you should have the following files on your **CIRCUITPY** drive.

Code Usage

The code should connect to your wireless network and Adafruit IO.

```
Connecting to WiFi...
Connected!
Connecting to Adafruit IO...
Connected to Adafruit IO!
Subscribed to brubell/f/led with QOS level 0
CPU temperature is 24.32 degrees C
Publishing 24.32 to temperature feed...
Published!
```

Navigate to your Adafruit IO Dashboard and you should see a new value in the gauge block. Every 30 seconds, the Pico reads its internal CPU temperature and sends it to Adafruit IO.



Toggle the switch block to turn the Pico's LED on or off: