Discord and Slack Connected Smart Plant with Adafruit IO Actions

Created by Eva Herrada


Last updated on 2023-08-29 04:13:08 PM EDT
# Table of Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Overview</strong></td>
<td>3</td>
</tr>
<tr>
<td>• Parts</td>
<td></td>
</tr>
<tr>
<td><strong>Install CircuitPython</strong></td>
<td>5</td>
</tr>
<tr>
<td>• CircuitPython Quickstart</td>
<td></td>
</tr>
<tr>
<td>• Safe Mode</td>
<td></td>
</tr>
<tr>
<td>• Flash Resetting UF2</td>
<td></td>
</tr>
<tr>
<td><strong>CircuitPython WiFi</strong></td>
<td>9</td>
</tr>
<tr>
<td>• CircuitPython Microcontroller Pinout</td>
<td></td>
</tr>
<tr>
<td>• CircuitPython Setup</td>
<td></td>
</tr>
<tr>
<td>• CircuitPython Usage</td>
<td></td>
</tr>
<tr>
<td><strong>Internet Connect!</strong></td>
<td>12</td>
</tr>
<tr>
<td>• What’s a secrets file?</td>
<td></td>
</tr>
<tr>
<td>• Connect to WiFi</td>
<td></td>
</tr>
<tr>
<td>• Requests</td>
<td></td>
</tr>
<tr>
<td>• HTTP GET with Requests</td>
<td></td>
</tr>
<tr>
<td>• HTTP POST with Requests</td>
<td></td>
</tr>
<tr>
<td>• Advanced Requests Usage</td>
<td></td>
</tr>
<tr>
<td>• WiFi Manager</td>
<td></td>
</tr>
<tr>
<td><strong>Discord Setup</strong></td>
<td>25</td>
</tr>
<tr>
<td>• Adafruit IO Setup</td>
<td></td>
</tr>
<tr>
<td><strong>Slack Setup</strong></td>
<td>31</td>
</tr>
<tr>
<td>• Adafruit IO Setup</td>
<td></td>
</tr>
<tr>
<td><strong>Code the Smart Plant</strong></td>
<td>36</td>
</tr>
<tr>
<td>• Installing the Project Code</td>
<td></td>
</tr>
<tr>
<td>• Wiring</td>
<td></td>
</tr>
<tr>
<td>• Assembly</td>
<td></td>
</tr>
<tr>
<td>• Code Usage</td>
<td></td>
</tr>
<tr>
<td><strong>Code Run-Through</strong></td>
<td>41</td>
</tr>
</tbody>
</table>
Always on the computer with no time to water your plants? Build an internet-enabled planter to send notifications over a Slack or Discord channel when it needs water.

In this project, you'll build an IoT planter using the Raspberry Pi Pico RP2040 and the STEMMA Soil Sensor. Then, you'll add an AirLift breakout board to connect it to the internet and send sensor data to Adafruit IO. Finally, you'll use Adafruit IO's reactive action features to send webhook notifications about your planter to a Discord or Slack server.

For this project, you can use Slack or Discord. While it is possible to do both, this guide won't go into that.
Parts

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

Adafruit Feather RP2040
A new chip means a new Feather, and the Raspberry Pi RP2040 is no exception. When we saw this chip we thought “this chip is going to be awesome when we give it the Feather...”
https://www.adafruit.com/product/4884

Adafruit AirLift FeatherWing – ESP32 WiFi Co-Processor
Give your Feather project a lift with the Adafruit AirLift FeatherWing - a FeatherWing that lets you use the powerful ESP32 as a WiFi co-processor. You probably have your...
https://www.adafruit.com/product/4264
FeatherWing Doubler - Prototyping Add-on For All Feather Boards
This is the FeatherWing Doubler - a prototyping add-on and more for all Feather boards. This is similar to our https://www.adafruit.com/product/2890

4-pin JST PH to JST SH Cable - STEMMA to QT / Qwiic
Are you a maker in the midst of a STEMMA dilemma? This 200mm long 4-wire...
https://www.adafruit.com/product/4424

USB Type A to Type C Cable - approx 1 meter / 3 ft long
As technology changes and adapts, so does Adafruit. This USB Type A to Type C cable will help you with the transition to USB C, even if you're still...
https://www.adafruit.com/product/4474

Install CircuitPython

CircuitPython () is a derivative of MicroPython () 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

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

Save it wherever is convenient for you.

To enter the bootloader, hold down the BOOT/BOOTSEL button (highlighted in red above), and while continuing to hold it (don't let go!), press and release the reset button (highlighted in blue above). Continue to hold the BOOT/BOOTSEL button until the RPI-RP2 drive appears!

If the drive does not appear, release all the buttons, and then repeat the process above.

You can also start with your board unplugged from USB, press and hold the BOOTSEL button (highlighted in red above), continue to hold it while plugging it into USB, and wait for the drive to appear before releasing the button.
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 RPI-RP2.

Drag the adafruit_circuitpython_etc.uf2 file to RPI-RP2.

The RPI-RP2 drive will disappear and a new disk drive called CIRCUITPY will appear.

That's it, you're done! :)

Safe Mode

You want to edit your code.py or modify the files on your CIRCUITPY drive, but find that you can't. Perhaps your board has gotten into a state where CIRCUITPY is read-
only. You may have turned off the CIRCUITPY drive altogether. Whatever the reason, safe mode can help.

Safe mode in CircuitPython does not run any user code on startup, and disables auto-reload. This means a few things. First, safe mode bypasses any code in boot.py (where you can set CIRCUITPY read-only or turn it off completely). Second, it does not run the code in code.py. And finally, it does not automatically soft-reload when data is written to the CIRCUITPY drive.

Therefore, whatever you may have done to put your board in a non-interactive state, safe mode gives you the opportunity to correct it without losing all of the data on the CIRCUITPY drive.

Entering Safe Mode

To enter safe mode when using CircuitPython, plug in your board or hit reset (highlighted in red above). Immediately after the board starts up or resets, it waits 1000ms. On some boards, the onboard status LED (highlighted in green above) will blink yellow during that time. If you press reset during that 1000ms, the board will start up in safe mode. It can be difficult to react to the yellow LED, so you may want to think of it simply as a slow double click of the reset button. (Remember, a fast double click of reset enters the bootloader.)

In Safe Mode

If you successfully enter safe mode on CircuitPython, the LED will intermittently blink yellow three times.

If you connect to the serial console, you'll find the following message.

| Auto-reload is off. Run in safe mode! Not running saved code. |
| CircuitPython is in safe mode because you pressed the reset button during boot. Press again to exit safe mode. |
| Press any key to enter the REPL. Use CTRL-D to reload. |

You can now edit the contents of the CIRCUITPY drive. Remember, your code will not run until you press the reset button, or unplug and plug in your board, to get out of safe mode.
Flash Resetting UF2

If your board ever gets into a really weird state and doesn't even show up as a disk drive when installing CircuitPython, try loading this 'nuke' UF2 which will do a 'deep clean' on your Flash Memory. You will lose all the files on the board, but at least you'll be able to revive it! After loading this UF2, follow the steps above to re-install CircuitPython.

Download flash erasing "nuke" UF2

CircuitPython WiFi

It's easy to use the Adafruit AirLift breakout with CircuitPython and the Adafruit CircuitPython ESP32SPI () module. This module allows you to easily add WiFi to your project.

The ESP32SPI library requires a microcontroller with ~128KB of RAM or more. The SAMD21 will not work.

CircuitPython Microcontroller Pinout

Since all CircuitPython-running Feathers follow the same pinout, you do not need to change any of the pins listed below.

To use the ESP32's pins, copy the following lines into your code:

```python
esp32_cs = DigitalInOut(board.D13)
esp32_ready = DigitalInOut(board.D11)
esp32_reset = DigitalInOut(board.D12)
```

If you wish to use the ESP32's GPIO0 pin - solder the jumper on the back of the FeatherWing, highlighted in red.
Then, include the following code to use the pin:

```python
esp32_gpio0 = DigitalInOut(board.D10)
```

## CircuitPython Setup

First make sure you are running the [latest version of Adafruit CircuitPython](https://circuitpython.org) for your board.

Next you'll need to install the necessary libraries to use the hardware. 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, and copy the entire lib folder and the code.py file to your CIRCUITPY drive.

Your CIRCUITPY/lib folder should contain the following folders and files:

- /adafruit_bus_device
- /adafruit_esp32spi
- adafruit_requests.mpy
CircuitPython Usage

Copy the following code to your code.py file on your microcontroller:

```python
import board
import busio
from digitalio import DigitalInOut
from adafruit_esp32spi import adafruit_esp32spi

print("ESP32 SPI hardware test")

esp32_cs = DigitalInOut(board.D13)
esp32_ready = DigitalInOut(board.D11)
esp32_reset = DigitalInOut(board.D12)

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

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	RSSI: %d" % (str(ap['ssid'], 'utf-8'), ap['rssi']))

print("Done!")
```

[Connect to the serial console](#) to see the output. It should look something like the following:
Make sure you see the same output! If you don't, check your wiring. Note that we've changed the pinout in the code example above to reflect the CircuitPython Microcontroller Pinout at the top of this page.

Once you've succeeded, continue onto the next page!

If you can read the Firmware and MAC address but fails on scanning SSIDs, check your power supply, you may be running out of juice to the ESP32 and it’s resetting

Internet Connect!

Once you have CircuitPython setup and libraries installed we can get your board connected to the Internet.

To get connected, you will need to start by creating a secrets file.

What's a secrets file?

We expect people to share tons of projects as they build CircuitPython WiFi widgets. What we want to avoid is people accidentally sharing their passwords or secret tokens and API keys. So, we designed all our examples to use a secrets.py file, that is in your CIRCUITPY drive, to hold secret/private/custom data. That way you can share your main project without worrying about accidentally sharing private stuff.
Your secrets.py file should look like this:

```python
# This file is where you keep secret settings, passwords, and tokens!
# If you put them in the code you risk committing that info or sharing it
secrets = {
    'ssid' : 'home ssid',
    'password' : 'my password',
    'timezone' : "America/New_York", # http://worldtimeapi.org/timezones
    'github_token' : 'fawfj23rakjnfwiefa',
    'hackaday_token' : 'h4xx0rs3kret',
}
```

Inside is a python dictionary named secrets with a line for each entry. Each entry has an entry name (say 'ssid') and then a colon to separate it from the entry key 'home ssid' and finally a comma ,

At a minimum you'll need the ssid and password for your local WiFi setup. As you make projects you may need more tokens and keys, just add them one line at a time. See for example other tokens such as one for accessing github or the hackaday API. Other non-secret data like your timezone can also go here, just cause its called secrets doesn't mean you can't have general customization data in there!

For the correct time zone string, look at http://worldtimeapi.org/timezones () and remember that if your city is not listed, look for a city in the same time zone, for example Boston, New York, Philadelphia, Washington DC, and Miami are all on the same time as New York.

Of course, don't share your secrets.py - keep that out of GitHub, Discord or other project-sharing sites.

**Connect to WiFi**

OK now you have your secrets setup - you can connect to the Internet using the ESP32SPI and the Requests modules.

First make sure you are running the latest version of Adafruit CircuitPython () 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 (). Our introduction guide has a great page on how to install the library bundle () for both express and non-express boards.
Remember for non-express boards like the Feather M0, you'll need to manually install the necessary libraries from the bundle:

- adafruit_bus_device
- adafruit_esp32_spi
- adafruit_requests
- neopixel

Before continuing make sure your board's lib folder or root filesystem has the above files copied over.

Next connect to the board's serial REPL () so you are at the CircuitPython >>> prompt.

Into your lib folder. Once that's done, load up the following example using Mu or your favorite editor:

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

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

# Get wifi details and more from a secrets.py file
try:
    from secrets import secrets
except ImportError:
    print("WiFi secrets are kept in secrets.py, please add them there!")
    raise

print("ESP32 SPI webclient test")

TEXT_URL = "http://wifitest.adafruit.com/testwifi/index.html"
JSON_URL = "http://api.coindesk.com/v1/bpi/currentprice/USD.json"

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

# If you have an AirLift Shield:
# esp32_cs = DigitalInOut(board.D10)
# esp32_ready = DigitalInOut(board.D7)
# esp32_reset = DigitalInOut(board.D5)

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

# If you have an externally connected ESP32:
# NOTE: You may need to change the pins to reflect your wiring
# esp32_cs = DigitalInOut(board.D9)
```

©Adafruit Industries
# esp32_ready = DigitalInOut(board.D10)
# esp32_reset = DigitalInOut(board.D5)

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

requests.set_socket(socket, esp)

if esp.status == 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("%s		RSSI: %d" % (ap["ssid"], "utf-8"), ap["rssi"], (hex(i) for i in esp.MAC_address))

print("Connecting to AP...")
while not esp.is_connected:
    try:
        esp.connect_AP(secrets["ssid"], secrets["password"])
    except OSError as e:
        print("could not connect to AP, retrying: ", e)
        continue
    print("Connected to", str(esp."
ss", "utf-8"), "tRSSI:", esp.rssi)
    print("My IP address is", esp.pretty_ip(esp.ip_address))
    print("IP lookup adafruit.com: %s" % esp.pretty_ip(esp.get_host_by_name("adafruit.com")))
    print("Ping google.com: %d ms" % esp.ping("google.com"))

    esp._debug = True
    print("Fetching text from", TEXT_URL)
    r = requests.get(TEXT_URL)
    print(".-" * 40)
    print(r.text)
    print(".-" * 40)
    r.close()

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

    print("Done!")

And save it to your board, with the name code.py.

This first connection example doesn't use a secrets file - you'll hand-enter your SSID/password to verify connectivity first!

Then go down to this line

```
    esp.connect_AP(b'MY_SSID_NAME', b'MY_SSID_PASSWORD')
```

and change MY_SSID_NAME and MY_SSID_PASSWORD to your access point name and password, keeping them within the " quotes. (This example doesn't use the secrets'
file, but it's also very stand-alone so if other things seem to not work you can always re-load this. You should get something like the following:

![Screenshot of COM1 PuTTY window with ESP32 console output]

In order, the example code...

Initializes the ESP32 over SPI using the SPI port and 3 control pins:

```python
esp32_cs = DigitalInOut(board.ESP_CS)
esp32_ready = DigitalInOut(board.ESP_BUSY)
esp32_reset = DigitalInOut(board.ESP_RESET)
spi = busio.SPI(board.SCK, board.MOSI, board.MISO)
esp = adafruit_esp32spi.ESP_SPIcontrol(spi, esp32_cs, esp32_ready, esp32_reset)
```

To use the AirLift FeatherWing’s pins, replace the following lines into your code:

```python
esp32_cs = DigitalInOut(board.D13)
esp32_ready = DigitalInOut(board.D11)
esp32_reset = DigitalInOut(board.D12)
```

Tells our requests library the type of socket we're using (socket type varies by connectivity type - we'll be using the adafruit_esp32spi_socket for this example). We'll also set the interface to an esp object. This is a little bit of a hack, but it lets us use requests like CPython does.
requests.set_socket(socket, esp)

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

```python
if esp.status == adafruit_esp32spi.WL_IDLE_STATUS:
    print("ESP32 found and in idle mode")
    print("Firmware vers.", esp.firmware_version)
    print("MAC addr:", [hex(i) for i in esp.MAC_address])
```

Performs a scan of all access points it can see and prints out the name and signal strength:

```python
for ap in esp.scan_networks():
    print("%s		RSSI: %d" % (str(ap['ssid'], 'utf-8'), ap['rssi']))
```

Connects to the AP we've defined here, then prints out the local IP address, attempts to do a domain name lookup and ping google.com to check network connectivity (note sometimes the ping fails or takes a while, this isn't a big deal)

```python
print("Connecting to AP...")
esp.connect_AP(b'MY_SSID_NAME', b'MY_SSID_PASSWORD')
print("My IP address is", esp.pretty_ip(esp.ip_address))
print("IP lookup adafruit.com: %s" % esp.pretty_ip(esp.get_host_by_name("adafruit.com")))
print("Ping google.com: %d ms" % esp.ping("google.com"))
```

OK now we're getting to the really interesting part. With a SAMD51 or other large-RAM (well, over 32 KB) device, we can do a lot of neat tricks. Like for example we can implement an interface a lot like `requests()` - which makes getting data really really easy

To read in all the text from a web URL call `requests.get()` - you can pass in `https` URLs for SSL connectivity

```python
TEXT_URL = "http://wifitest.adafruit.com/testwifi/index.html"
print("Fetching text from", TEXT_URL)
r = requests.get(TEXT_URL)
print(r.text)
r.close()
```

Or, if the data is in structured JSON, you can get the json pre-parsed into a Python dictionary that can be easily queried or traversed. (Again, only for nRF52840, M4 and other high-RAM boards)
Requests

We've written a requests-like () library for web interfacing named Adafruit_CircuitPython on_Requests (). This library allows you to send HTTP/1.1 requests without "crafting" them and provides helpful methods for parsing the response from the server.

Here's an example of using Requests to perform GET and POST requests to a server.

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

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

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

# If you are using a board with pre-defined ESP32 Pins:
estp32_cs = DigitalInOut(board.ESP_CS)
estp32_ready = DigitalInOut(board.ESP_BUSY)
estp32_reset = DigitalInOut(board.ESP_RESET)

# If you have an externally connected ESP32:
estp32_cs = DigitalInOut(board.D9)
estp32_ready = DigitalInOut(board.D10)
estp32_reset = DigitalInOut(board.D5)

# If you have an AirLift Featherwing or ItsyBitsy Airlift:
estp32_cs = DigitalInOut(board.D13)
estp32_ready = DigitalInOut(board.D11)
estp32_reset = DigitalInOut(board.D12)

spi = busio.SPI(board.SCK, board.MOSI, board.MISO)
estp = adafruit_esp32spi.ESP_SPIcontrol(spi, estp32_cs, estp32_ready, estp32_reset)

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

# Initialize a requests object with a socket and esp32spi interface
socket.set_interface(esp)
requests.set_socket(socket, esp)

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

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

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

data = "31F"
print("POSTing data to {0}: {1}".format(JSON_POST_URL, data))
response = requests.post(JSON_POST_URL, data=data)
print("-" * 40)

json_resp = response.json()
# Parse out the 'data' key from json_resp dict.
print("Data received from server:", json_resp["data"])
print("-" * 40)
response.close()

json_data = {"Date": "July 25, 2019"}
print("POSTing data to {0}: {1}".format(JSON_POST_URL, json_data))
response = requests.post(JSON_POST_URL, json=json_data)
print("-" * 40)

json_resp = response.json()
# Parse out the 'json' key from json_resp dict.
print("JSON Data received from server:", json_resp["json"])
print("-" * 40)
response.close()

The code first sets up the ESP32SPI interface. Then, it initializes a request object using an ESP32 socket and the esp object.
spi = busio.SPI(board.SCK, board.MOSI, board.MISO)
esp = adafruit_esp32spi.ESP_SPIcontrol(spi, esp32_cs, esp32_ready, esp32_reset)

print("Connecting to AP...")
while not esp.is_connected:
    try:
        esp.connect_AP(b'MY_SSID_NAME', b'MY_SSID_PASSWORD')
    except RuntimeError as e:
        print("could not connect to AP, retrying ", e)
        continue
print("Connected to ", str(esp.ssid, 'utf-8'), "\tRSSI: ", esp.rssi)

# Initialize a requests object with a socket and esp32spi interface
requests.set_socket(socket, esp)

Make sure to set the ESP32 pinout to match your AirLift breakout's connection:

```
esp32_cs = DigitalInOut(board.D9)
esp32_ready = DigitalInOut(board.D10)
esp32_reset = DigitalInOut(board.D5)
```

**HTTP GET with Requests**

The code makes a HTTP GET request to Adafruit's WiFi testing website - [http://wifitest.adafruit.com/testwifi/index.html](http://wifitest.adafruit.com/testwifi/index.html).

To do this, we'll pass the URL into `requests.get()`. We're also going to save the response from the server into a variable named `response`.

While we requested data from the server, we'd what the server responded with. Since we already saved the server's `response`, we can read it back. Luckily for us, requests automatically decodes the server's response into human-readable text, you can read it back by calling `response.text`.

Lastly, we'll perform a bit of cleanup by calling `response.close()`. This closes, deletes, and collect's the response's data.

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

While some servers respond with text, some respond with json-formatted data consisting of attribute–value pairs.
CircuitPython_Requests can convert a JSON-formatted response from a server into a CPython `dict` object.

We can also fetch and parse json data. We'll send a HTTP get to a url we know returns a json-formatted response (instead of text data).

Then, the code calls `response.json()` to convert the response to a CPython `dict`.

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

**HTTP POST with Requests**

Requests can also POST data to a server by calling the `requests.post` method, passing it a `data` value.

```python
data = '31F'
print("POSTing data to {0}: {1}".format(JSON_POST_URL, data))
response = requests.post(JSON_POST_URL, data=data)
print(''.*40)
json_resp = response.json()
# Parse out the 'data' key from json_resp dict.
print("Data received from server:", json_resp['data'])
print(''.*40)
response.close()
```

You can also post json-formatted data to a server by passing `json` data into the `requests.post` method.

```python
json_data = \"{"Date" : "July 25, 2019\"\}\\nprint("POSTing data to {0}: {1}".format(JSON_POST_URL, json_data))
response = requests.post(JSON_POST_URL, json=json_data)
print(''.*40)
json_resp = response.json()
# Parse out the 'json' key from json_resp dict.
print("JSON Data received from server:", json_resp['json'])
print(''.*40)
response.close()
```
Advanced Requests Usage

Want to send custom HTTP headers, parse the response as raw bytes, or handle a response's http status code in your CircuitPython code?

We've written an example to show advanced usage of the requests module below.

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

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

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

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

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

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

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

# Initialize a requests object with a socket and esp32spi interface
socket.set_interface(esp)
requests.set_socket(socket, esp)

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

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

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

That simplest example works but it's a little finicky - you need to constantly check WiFi status and have many loops to manage connections and disconnections. For more advanced uses, we recommend using the WiFiManager object. It will wrap the connection/status/requests loop for you - reconnecting if WiFi drops, resetting the ESP32 if it gets into a bad state, etc.

Here's a more advanced example that shows the WiFi manager and also how to POST data with some extra headers:

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

import time
import board
import busio
from digitalio import DigitalInOut
import neopixel
from adafruit_esp32spi import adafruit_esp32spi
from adafruit_esp32spi import adafruit_esp32spi_wifimanager

print("ESP32 SPI webclient test")

# Get wifi details and more from a secrets.py file
try:
    from secrets import secrets
except ImportError:
   print("WiFi secrets are kept in secrets.py, please add them there!")
   raise

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

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

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

""" Use below for Most Boards"
status_light = neopixel.NeoPixel(board.NEOPIXEL, 1, brightness=0.2
```
Next, set up an Adafruit IO feed named test

- If you do not know how to set up a feed, follow this page and come back when you've set up a feed named test.

You’ll note here we use a secrets.py file to manage our SSID info. The wifimanager is given the ESP32 object, secrets and a neopixel for status indication.

Note, you’ll need to add a some additional information to your secrets file so that the code can query the Adafruit IO API:

- `aio_username`
- `aio_key`

You can go to your adafruit.io View AIO Key link to get those two values and add them to the secrets file, which will now look something like this:
```python
# This file is where you keep secret settings, passwords, and tokens!
# If you put them in the code you risk committing that info or sharing it

secrets = {
    'ssid': '_your_ssid_',
    'password': '_your_wifi_password_',
    'timezone': "America/Los_Angeles", # http://worldtimeapi.org/timezones
    'aio_username': '_your_aio_username_',
    'aio_key': '_your_aio_key_',
}
```

We can then have a simple loop for posting data to Adafruit IO without having to deal with connecting or initializing the hardware!

Take a look at your test feed on Adafruit.io and you'll see the value increase each time the CircuitPython board posts data to it!

---

**Discord Setup**

In this section, I'll guide you through the process of setting up your Discord server to handle the webhooks being sent from Adafruit IO.
First, click the Add a Server button on the left under the servers you’re in.

Then, click the Create my Own button. Or, if you’d like to use a different template, press one of the other buttons.

Now, click For me and my Friends. Like the last one, you can choose a different option, but I won't be going over how to set them up (although the webhook setup should be exactly the same regardless of what type of server you have).
Finally, name your server and press Create.

Next, click on the icon for the server you just created and it should look something like this.

Now that you're in the server, press on the name in the upper left-hand corner and press the Server Settings button.
Click on the Integrations tab, then press Create Webhook.

Rename your webhook and add a profile picture if you'd like to, then set the channel you'd like it to post in and click Copy Webhook URL and paste it somewhere. Then press Save Changes.

Adafruit IO Setup

The first step is to make a feed that the device will write to.

Navigate to the Feeds tab and click New Feed. In this screenshot, I'm creating it in a group I made for this guide, but it's probably easier to just create it in the Default group.
Now, make your feed. I called mine plant.

After you've created it, you should see it on your Feeds page.
Now that you've made the feed, go to the Actions tab and press New Action.

Then, select Reactive Action.

Then, select your feed in the dropdown next to if.

In the dropdown next to is, select less than.

In the text box below that, write the moisture value you want it to notify you at. I used 500, but that was really just a guess. My advice would be to wait until you normally would water your plants and then get the moisture value at that time.

Keep in mind that the moisture value is very dependant on a number of things other than the actual amount of water in the soil, so if you move it you will probably have to set a new moisture value to account for that.
In the dropdown next to Then select send a webhook message. Paste the link you got from Discord in the box under that.

When you've done that, select your feed in the dropdown to the left of value and time.

Set the final dropdown to Discord Template, fill in the username you want for the bot on the first line in the set of quotes after the colon, and set the second line to the message you want it to send you.

If you want it to ping you, put your discord ID in the message with this formatting `<@discord_id>`.

---

**Slack Setup**

In this section, you'll go through the process of adding a webhook to a Slack workspace that you're already a part of.
First click on the name of the workspace, then Settings & administration (or just Administration if you're not an admin) and then Manage apps. If you're not an administrator, you should see less options on the last menu but the one you need should still be there.

Now click on Custom integrations.
After that, type Incoming WebHooks into the search bar and click on the first result.

Then, click Add to Slack.

You should now see a screen that looks something like this. Here you'll decide which channel you want the webhook to post into. For this example, I'm using a direct message to myself, so I'll select my name from the list.
Finally, you should see this screen. Copy the Webhook URL (the one I've put a red rectangle around) and paste it somewhere so you have it for the next section.

Adafruit IO Setup

The first step is to make a feed that the device will write to.

Navigate to the Feeds tab and click New Feed. In this screenshot, I'm creating it in a group I made for this guide, but it's probably easier to just create it in the Default group.

Now, make your feed. I called mine plant.
After you've created it, you should see it on your Feeds page.

Now that you've made the feed, go to the Actions tab and press New Action.

Then, select Reactive Action.
Then, select your feed in the dropdown next to if.

In the dropdown next to is, select less than.

In the text box below that, write the moisture value you want it to notify you at. I used 500, but that was really just a guess. My advice would be to wait until you normally would water your plants and then get the moisture value at that time.

Keep in mind that the moisture value is very dependant on a number of things other than the actual amount of water in the soil, so if you move it you will probably have to set a new moisture value to account for that.

In the dropdown next to Then select send a webhook message. Paste the link you got from Slack in the box under that.

When you've done that, select your feed in the dropdown to the left of value and time.

Set the final dropdown to Slack Template, and fill in the message you'd like it to send you.

---

**Code the Smart Plant**

**Installing the Project Code**

Download a zip of the project by clicking Download Project Bundle below.

After unzipping the file, copy code.py to the CIRCUITPY drive which appears when the Feather is connected to your computer via a USB cable.

```python
# SPDX-FileCopyrightText: 2021 Eva Herrada for Adafruit Industries
# SPDX-License-Identifier: MIT
import time
```
import board
from digitalio import DigitalInOut
from adafruit_esp32spi import adafruit_esp32spi
from adafruit_esp32spi import adafruit_esp32spi_wifimanager
import adafruit_esp32spi.adafruit_esp32spi_socket as socket
import neopixel
import adafruit_minimqtt.adafruit_minimqtt as MQTT
from adafruit_io.adafruit_io import IO_MQTT
from adafruit_seesaw.seesaw import Seesaw
import busio

# Used to make sure that the Adafruit IO Trigger is only run once when the moisture value is below
# the desired threshold, set in MIN
LOW = False
# The minimum moisture value. If the value is below this number, it will activate the Adafruit IO
# trigger. This number should match the number you set in your Adafruit IO trigger.
# Feel free
# to mess around and try out different moisture values as how wet this actually is can vary a lot
# depending on where the sensor is and the soil in the pot.
MIN = 500

# Set up moisture sensor with seesaw
i2c = board.I2C()  # uses board.SCL and board.SDA
seesaw = Seesaw(i2c, addr=0x36)

# Get wifi details and more from a secrets.py file
try:
    from secrets import secrets
except ImportError:
    print("WiFi secrets are kept in secrets.py, please add them there!")
    raise

# Connect to WiFi
print("Connecting to WiFi...")
wifi = adafruit_esp32spi_wifimanager.ESPSPI_WiFiManager(esp, secrets, status_light)

# Define callback functions which will be called when certain events happen.
# pylint: disable=unused-argument
def connected(client):
    # This method is called when the client connects to Adafruit IO
    client.subscribe("plant")

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

def message(client, feed_id, payload):
    # This method is called when a feed receives a new message
    print("Feed {0} received new value: {1}".format(feed_id, payload))

# Connect to WiFi
print("Connected!")
# Initialize MQTT interface with the esp interface
MQTT.set_socket(socket, esp)

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

# 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_subscribe = subscribe
io.on_message = message

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

START = 0
while True:
    # read moisture level through capacitive touch pad
touch = seesaw.moisture_read()

    # read temperature from the temperature sensor
temp = seesaw.get_temp()

    if touch < MIN:
        if not LOW:
            io.publish(plant_feed, touch)
            print("published")
            LOW = True
    elif touch >= MIN and time.time() - START > 10:
        io.publish(plant_feed, touch)
        print("published to Adafruit IO")
        START = time.time()
        LOW = False

    print("temp: " + str(temp) + " moisture: " + str(touch))
    time.sleep(1)

There are also a few libraries you'll need to copy over to the Feather:

- adafruit_esp32spi/
- adafruit_requests/
- adafruit_minimqtt/
- adafruit_io/
- adafruit_seesaw/
- adafruit_requests.mpy
- neopixel.mpy
Your secrets.py file should at the very least have values for the fields `ssid`, `password`, `timezone`, `aio_username`, and `aio_key`. Make sure to copy it over to CIRCUITPY as well.

After you've done all that, this is what your CIRCUITPY drive should look like.

**Wiring**

Luckily there isn't really any wiring for this project. Put the Feather RP2040 and AirLift FeatherWing on the FeatherWing Doubler, then plug the STEMMA to STEMMA QT cable into both the Feather RP2040 and the moisture sensor.
Assembly

Now that you've put the code on the Feather, plug the STEMMA cable in and put the sensor in your plant. I've found it works better if you put one side of the sensor against the edge of the pot.

Code Usage

This project is rather easy to use. After you've set it up, put the sensor into the pot against the side, and plugged it in, it will send you a message if the moisture level is under 500, or whatever value you selected. If it is under that value, it won't send any more messages to Adafruit IO so that you don't get pinged every second until you water your plant.

Then, if the moisture value is above your value, it will send a message to Adafruit IO with the moisture value which shouldn't make the action send a webhook, and if the number dips below 500 again it will send another message that will make the action go off.
First, the code imports the required libraries.

```python
import time
import board
from digitalio import DigitalInOut
from adafruit_esp32spi import adafruit_esp32spi
from adafruit_esp32spi import adafruit_esp32spi_wifimanager
import adafruit_esp32spi.adafruit_esp32spi_socket as socket
import neopixel
import adafruit_minimqtt.adafruit_minimqtt as MQTT
from adafruit_seesaw.seesaw import Seesaw
import busio
```

Next, the values we'll need in the main loop are created. The first variable, `low`, makes sure that you don't get a message every second when your plant needs to be watered, and `min` should match the value you are comparing the feed to in the Adafruit IO Action.

```python
LOW = False
MIN = 500
```

Then, the moisture sensor gets initialized.

```python
i2c_bus = board.I2C()
seesaw = Seesaw(i2c_bus, addr=0x36)
```
After that, the code checks to see if secrets.py exists and then sets up the AirLift FeatherWing.

```python
# Get wifi details and more from a secrets.py file
try:
    from secrets import secrets
except ImportError:
    print("WiFi secrets are kept in secrets.py, please add them there!")
    raise

# Set up WiFi
esp32_cs = DigitalInOut(board.D13)
esp32_ready = DigitalInOut(board.D11)
esp32_reset = DigitalInOut(board.D12)

spi = busio.SPI(board.SCK, board.MOSI, board.MISO)
esp = adafruit_esp32spi.ESP_SPIcontrol(spi, esp32_cs, esp32_ready, esp32_reset)
status_light = neopixel.NeoPixel(board.NEOPIXEL, 1, brightness=0.2)
wifi = adafruit_esp32spi_wifimanager.ESP8266_AT_WiFiManager(esp, secrets, status_light)
```

The callback functions are then defined which will be run when the device establishes a connection with Adafruit IO, when a new feed is subscribed to, and when a new message is received, respectively.

```python
# Define callback functions which will be called when certain events happen.
# pylint: disable=unused-argument
def connected(client):
    # This method is called when the client connects to Adafruit IO
    client.subscribe("plant")

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

def message(client, feed_id, payload):
    # This method is called when a feed receives a new message
    print("Feed {0} received new value: {1}".format(feed_id, payload))
```

After the callback functions, the code connects to the internet and then to Adafruit IO.

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

# Initialize MQTT interface with the esp interface
MQTT.set_socket(socket, esp)

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

# 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_subscribe = subscribe
```
io.on_message = message

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

Before the main loop, the code defines a variable, `start`. This variable is used to make sure that the feed isn't published to more often than desired.

```
START = 0
```

The first thing that happens in the main loop is the sensor's temperature and moisture are read. The temperature isn't used in this code, but I left it in there just in case someone wants to add temperature to their project.

Then the code checks if the value from the sensor is below the specified threshold. It then checks to see if this is the first time since it was last above the threshold that it has published a value. This is done so that when the sensor is below the threshold, the webhook isn't activated all the time. It then publishes the value.

In the next if block, the code checks to see if the sensor value is above the threshold and then checks to see if it's been 10 seconds since the last value was published and if it has been ten seconds it publishes the value.

Finally, the code prints the sensor values and the loop starts over again.

```
while True:
    # read moisture level through capacitive touch pad
    touch = seesaw.moisture_read()

    # read temperature from the temperature sensor
    temp = seesaw.get_temp()

    if touch < MIN:
        if not LOW:
            io.publish("plant", touch)
            print("published")
        LOW = True

    elif touch >= MIN and time.time() - START > 10:
        io.publish("plant", touch)
        print("published to Adafruit IO")
        START = time.time()
        LOW = False

    print("temp: " + str(temp) + " moisture: " + str(touch))
    time.sleep(1)
```