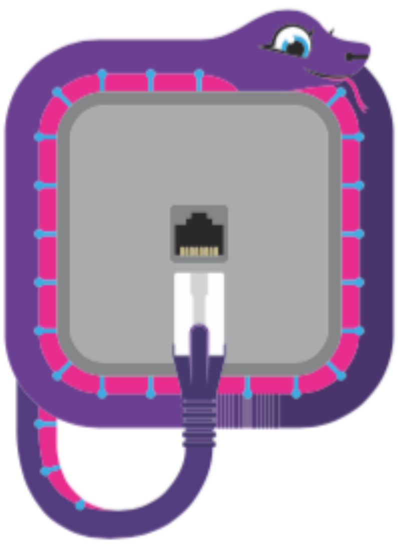
# Ethernet for CircuitPython with Wiznet5K

Created by Brent Rubell

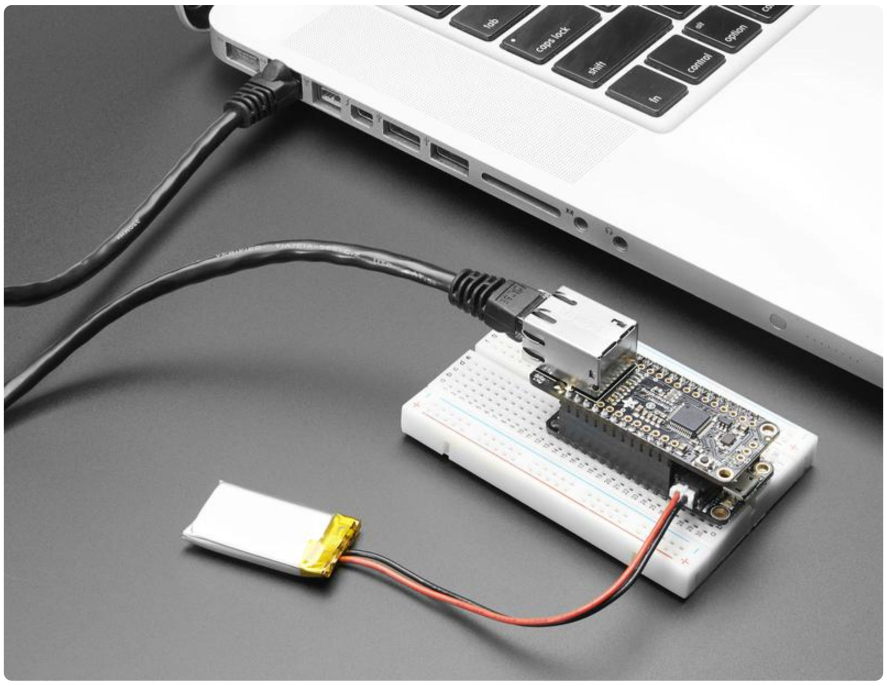


https://learn.adafruit.com/ethernet-for-circuitpython

Last updated on 2022-12-01 03:51:50 PM EST

# Table of Contents

# Overview



Wireless is wonderful, but sometimes you want the strong reliability of a wired connection. If your project is going to be part of a permanent installation, you'll want to add ethernet networking to your project.

Ethernet is incredibly easy to use - there's no network configuration or device pairing. Just plug a standard Ethernet cable into an Ethernet FeatherWing or Ethernet Shield and use the CircuitPython Wiznet5k () library for quick and reliable, networking.

- If you'd like to learn more about the Ethernet standard - check out its section in the All The Internet of Things: Transports guide ().

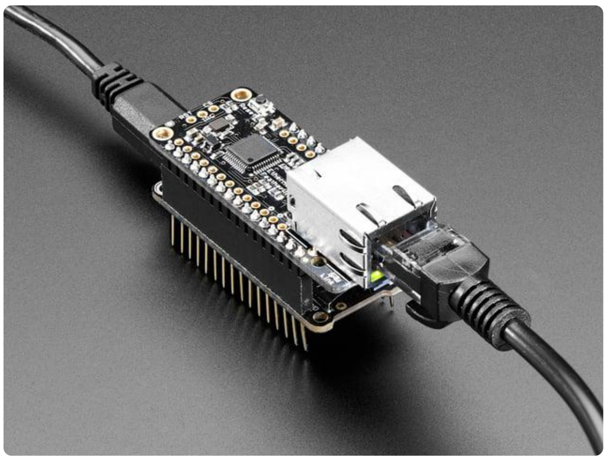
We've built a module for CircuitPython compatible with WIZnet 5k-series TCP/IP Ethernet controllers () to quickly get your projects online. Seconds after connecting, the module will handle performing the DHCP setup for you. You may also supply this module with a statically-assigned network configuration.

In this guide, you will set up your CircuitPython hardware and connect to the internet. We've included code walkthrough portions of this guide, so you can roll your own Ethernet-connected IoT CircuitPython project.

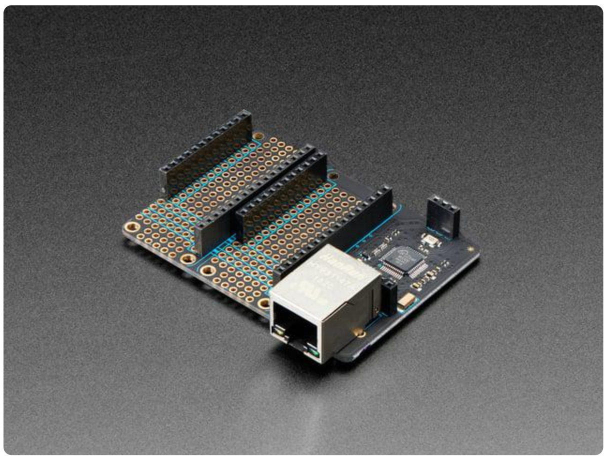
Note: SSL/TLS connections are not supported by the Wiznet5k library at this time. You will only be able to make insecure requests to web servers with this library. ESP32 WiFi-based solutions have integrated TLS so please use that if you need TLS connectivity!

## Parts

The Ethernet FeatherWing is plug-and-play compatible with CircuitPython boards which use the Feather pinout (), such as the Adafruit Feather M4 Express ().

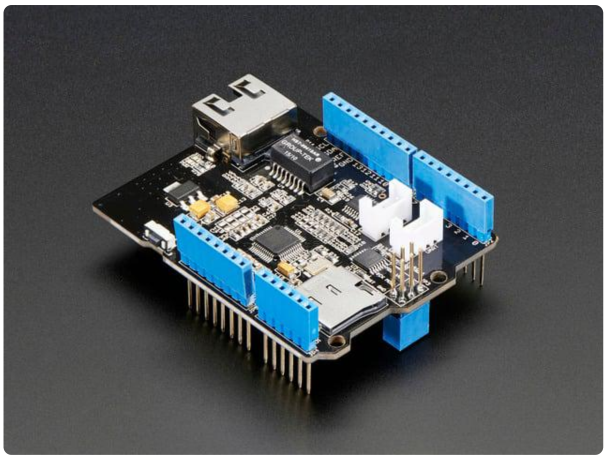


Adafruit Ethernet FeatherWing
Wireless is wonderful, but sometimes you want the strong reliability of a wire. If your Feather board is going to be part of a permanent installation, this Ethernet...
https://www.adafruit.com/product/3201



Particle Ethernet FeatherWing
This three-in-one Particle Ethernet Feather is a mix of our Ethernet 'wing and the
https://www.adafruit.com/product/4003

The Ethernet Shield is compatible with development boards which use the pinout such as the Adafruit Metro M4 () and the Adafruit Grand Central M4 Express (). The shield includes a RJ45 Ethernet port and a microSD card holder for storing and reading data.



Ethernet Shield for Arduino - W5500 Chipset

The W5500 Ethernet Shield for Arduino from Seeed Studio is a great way to set up your projects with internet connectivity with just a single chip.  Similar to the https://www.adafruit.com/product/2971

| | |
|---|---|
| 1 x Ethernet Cable, 5ft<br>Ethernet Cable - 5 ft long | https://www.adafruit.com/product/994 |
| 1 x Ethernet Cable, 10ft<br>Ethernet Cable - 10 ft long | https://www.adafruit.com/product/730 |
| 1 x PoE Splitter with MicroUSB Plug<br>PoE Splitter with MicroUSB Plug - Isolated 12W - 5V 2.4 Amp | https://www.adafruit.com/product/3785 |
| 1 x Passive PoE Injector Cable Set<br>Passive PoE Injector Cable Set | https://www.adafruit.com/product/435 |

# CircuitPython Setup

## CircuitPython Installation

Some CircuitPython compatible boards come with CircuitPython installed. Others are CircuitPython-ready, but need to have it installed. As well, you may want to update the version of CircuitPython already installed on your board. The steps are the same for installing and updating.

- To install (or update) your CircuitPython board, follow this page and come back here when you've successfully installed (or updated) CircuitPython. ()

## Install the Mu Editor

This guide requires you to edit and interact with CircuitPython code. While you can use any text editor of your choosing, Mu is a simple code editor that works with the

Adafruit CircuitPython boards. It's written in Python and works on Windows, MacOS, Linux and Raspberry Pi. The serial console is built right in, so you get immediate feedback from your board's serial output!

Before proceeding, if you'd like to use Mu, click the button below to install the Mu Editor. There are versions for PC, mac, and Linux.

Install Mu Editor

## CircuitPython Library Installation

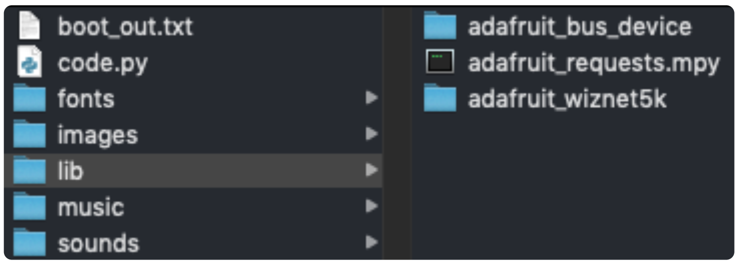
First make sure you are running the latest version of Adafruit CircuitPython () for your board.

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 () matching your version of CircuitPython. The Wiznet5k Library r ()equires at least CircuitPython version 4.0.0.

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

- adafruit_wiznet5k
- adafruit_bus_device
- adafruit_requests.mpy

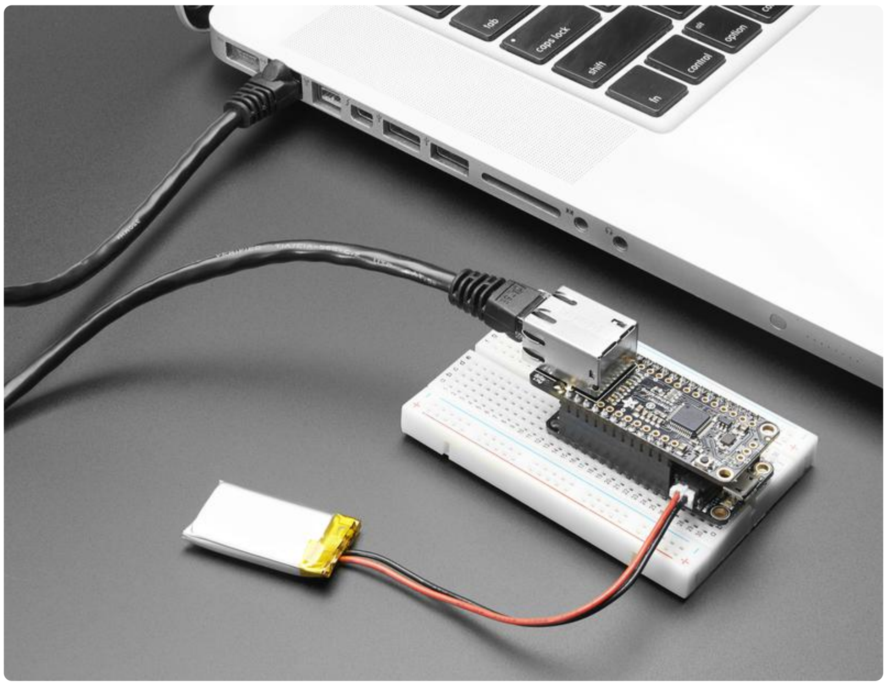
Once all the files are copied, your CIRCUITPY drive should look like the following screenshot:



## Usage

### Connect Ethernet Cable

Make sure you have your Ethernet FeatherWing or Ethernet Shield firmly plugged into your hardware, and an Ethernet cable connected to your router or switch.



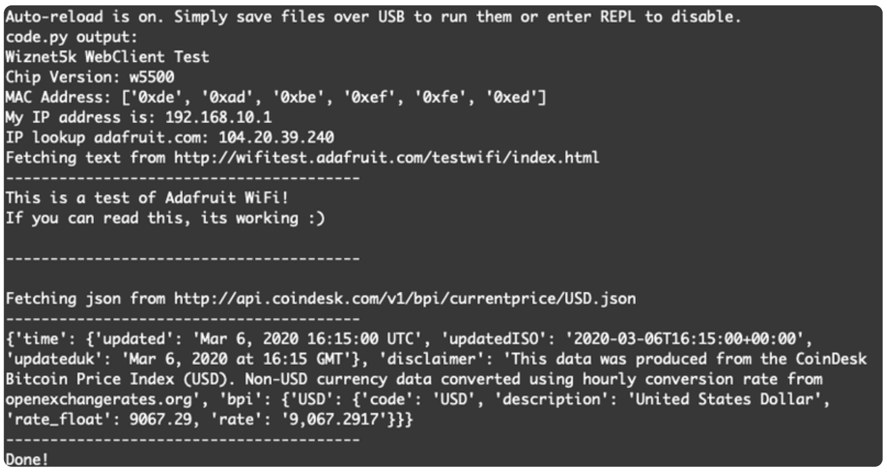
### Code Usage

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

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

import board
import busio
import digitalio
import adafruit_requests as requests
from adafruit_wiznet5k.adafruit_wiznet5k import WIZNET5K
import adafruit_wiznet5k.adafruit_wiznet5k_socket as socket

print("Wiznet5k WebClient Test")

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

# For Adafruit Ethernet FeatherWing
cs = digitalio.DigitalInOut(board.D10)
# For Particle Ethernet FeatherWing
# cs = digitalio.DigitalInOut(board.D5)
spi_bus = busio.SPI(board.SCK, MOSI=board.MOSI, MISO=board.MISO)
```

```
# Initialize ethernet interface with DHCP
eth = WIZNET5K(spi_bus, cs)

# Initialize a requests object with a socket and ethernet interface
requests.set_socket(socket, eth)

print("Chip Version:", eth.chip)
print("MAC Address:", [hex(i) for i in eth.mac_address])
print("My IP address is:", eth.pretty_ip(eth.ip_address))
print(
    "IP lookup adafruit.com: %s" %
eth.pretty_ip(eth.get_host_by_name("adafruit.com"))
)


# eth._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!")
```

Save the code.py file and open the REPL.



```
Auto-reload is on. Simply save files over USB to run them or enter REPL to disable.
code.py output:
Wiznet5k WebClient Test
Chip Version: w5500
MAC Address: ['0xde', '0xad', '0xbe', '0xef', '0xfe', '0xed']
My IP address is: 192.168.10.1
IP lookup adafruit.com: 104.20.39.240
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://api.coindesk.com/v1/bpi/currentprice/USD.json
----------------------------------------
{'time': {'updated': 'Mar 6, 2020 16:15:00 UTC', 'updatedISO': '2020-03-06T16:15:00+00:00',
'updateduk': 'Mar 6, 2020 at 16:15 GMT'}, 'disclaimer': 'This data was produced from the CoinDesk
Bitcoin Price Index (USD). Non-USD currency data converted using hourly conversion rate from
openexchangerates.org', 'bpi': {'USD': {'code': 'USD', 'description': 'United States Dollar',
'rate_float': 9067.29, 'rate': '9,067.2917'}}}
----------------------------------------
Done!
```

If you don't get an IP address, check you have a green link light, and that your Ethernet is going out to an internet connected router. You may also have to set up the MAC address to allow it access, check with your system admin if you're not sure.

In order, the example code:

Initializes the Ethernet chipset over SPI using the SPI port and the CS pin.

```
cs = digitalio.DigitalInOut(board.D10)
spi_bus = busio.SPI(board.SCK, MOSI=board.MOSI, MISO=board.MISO)

eth = WIZNET5K(spi_bus, cs)
```

Tells the `requests` library the socket, and interface we'll be using. The interface is set to an `eth` object. This is a little bit of a hack, but it lets us use `requests` like CPython does.

```
requests.set_socket(socket, eth)
```

Verifies the Ethernet hardware was found, checks the chip version and MAC address.

```
print("Chip Version:", eth.chip)
print("MAC Address:", [hex(i) for i in eth.mac_address])
```

Prints out the local IP and attempts to perform an IP address lookup for adafruit.com.

```
print("My IP address is:", eth.pretty_ip(eth.ip_address))
print("IP lookup adafruit.com: %s"
%eth.pretty_ip(eth.get_host_by_name("adafruit.com")))
```

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

To read in all the text from a web URL, call `requests.get` -

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

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

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

## Manual Network Configuration

The Wiznet5k library automatically takes care of DHCP and DNS configuration, so you can get your project online quickly. However, there are cases where a network administrator will provide you with an IP address or you may need to manually configure the interface.

This code performs the same Ethernet simpletest as the above section, but allows you to configure your network interface.

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

import board
import busio
import digitalio
import adafruit_requests as requests
from adafruit_wiznet5k.adafruit_wiznet5k import WIZNET5K
import adafruit_wiznet5k.adafruit_wiznet5k_socket as socket

TEXT_URL = "http://wifitest.adafruit.com/testwifi/index.html"

# Setup your network configuration below
IP_ADDRESS = (192, 168, 10, 1)
SUBNET_MASK = (255, 255, 0, 0)
GATEWAY_ADDRESS = (192, 168, 0, 1)
DNS_SERVER = (8, 8, 8, 8)

print("Wiznet5k WebClient Test (no DHCP)")

cs = digitalio.DigitalInOut(board.D10)
spi_bus = busio.SPI(board.SCK, MOSI=board.MOSI, MISO=board.MISO)

# Initialize ethernet interface without DHCP
eth = WIZNET5K(spi_bus, cs, is_dhcp=False)

# Set network configuration
eth.ifconfig = (IP_ADDRESS, SUBNET_MASK, GATEWAY_ADDRESS, DNS_SERVER)

# Initialize a requests object with a socket and ethernet interface
requests.set_socket(socket, eth)

print("Chip Version:", eth.chip)
print("MAC Address:", [hex(i) for i in eth.mac_address])
print("My IP address is:", eth.pretty_ip(eth.ip_address))
print(
    "IP lookup adafruit.com: %s" %
eth.pretty_ip(eth.get_host_by_name("adafruit.com"))
)

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

print()
```

You will need to manually set your network configuration:

- Set `IP_ADDRESS` to your desired IP address.
- Set `SUBNET_MASK` to the router's subnet mask
- Set `GATEWAY_ADDRESS` to the router's IP address.
- Set `DNS_SERVER` to the DNS server you'd like to use. We're using 8.8.8.8 which is Google's Public DNS.

```
# Setup your network configuration below
IP_ADDRESS = (192, 168, 10, 1)
SUBNET_MASK = (255, 255, 0, 0)
GATEWAY_ADDRESS = (192, 168, 0, 1)
DNS_SERVER = (8, 8, 8, 8)
```

When the Ethernet interface is initialized, we'll disable the automatic DHCP process.

```
# Initialize ethernet interface without DHCP
eth = WIZNET5K(spi_bus, cs, is_dhcp=False)
```

Then, we'll set up the network configuration by passing the `ifconfig` property a tuple containing the IP address, subnet mask, gateway address and DNS server.

```
# Set network configuration
eth.ifconfig = (IP_ADDRESS, SUBNET_MASK, GATEWAY_ADDRESS, DNS_SERVER)
```

## Usage with Requests

Note: SSL/TLS connections are not supported by CircuitPython at this time. You will only be able to make insecure requests to web servers with this library.

We've written a requests-like () library for web interfacing named Adafruit_CircuitPython_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.

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

import board
import busio
from digitalio import DigitalInOut
from adafruit_wiznet5k.adafruit_wiznet5k import WIZNET5K
import adafruit_wiznet5k.adafruit_wiznet5k_socket as socket
import adafruit_requests as requests

cs = DigitalInOut(board.D10)
spi_bus = busio.SPI(board.SCK, MOSI=board.MOSI, MISO=board.MISO)
```

```
# Initialize ethernet interface with DHCP
eth = WIZNET5K(spi_bus, cs)

# Initialize a requests object with a socket and ethernet interface
requests.set_socket(socket, eth)

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

attempts = 3  # Number of attempts to retry each request
failure_count = 0
response = None

print("Fetching text from %s" % TEXT_URL)
while not response:
    try:
        response = requests.get(TEXT_URL)
        failure_count = 0
    except AssertionError as error:
        print("Request failed, retrying...\n", error)
        failure_count += 1
        if failure_count >= attempts:
            raise AssertionError(
                "Failed to resolve hostname, \
                                  please check your router's DNS configuration."
            ) from error
        continue
print("-" * 40)

print("Text Response: ", response.text)
print("-" * 40)
response.close()
response = None

print("Fetching JSON data from %s" % JSON_GET_URL)
while not response:
    try:
        response = requests.get(JSON_GET_URL)
        failure_count = 0
    except AssertionError as error:
        print("Request failed, retrying...\n", error)
        failure_count += 1
        if failure_count >= attempts:
            raise AssertionError(
                "Failed to resolve hostname, \
                                  please check your router's DNS configuration."
            ) from error
        continue
print("-" * 40)

print("JSON Response: ", response.json())
print("-" * 40)
response.close()
response = None

data = "31F"
print("POSTing data to {0}: {1}".format(JSON_POST_URL, data))
while not response:
    try:
        response = requests.post(JSON_POST_URL, data=data)
        failure_count = 0
    except AssertionError as error:
        print("Request failed, retrying...\n", error)
        failure_count += 1
        if failure_count >= attempts:
            raise AssertionError(
                "Failed to resolve hostname, \
                                           please check your router's DNS configuration."
                ) from error
            continue
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()
response = None

json_data = {"Date": "July 25, 2019"}
print("POSTing data to {0}: {1}".format(JSON_POST_URL, json_data))
while not response:
    try:
        response = requests.post(JSON_POST_URL, json=json_data)
        failure_count = 0
    except AssertionError as error:
        print("Request failed, retrying...\n", error)
        failure_count += 1
        if failure_count >= attempts:
            raise AssertionError(
                "Failed to resolve hostname, \
                                  please check your router's DNS configuration."
            ) from error
        continue
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 ethernet interface ( `eth` ). Then, it initializes a request object using a `socket` and the `esp` interface.

```
import board
import busio
from digitalio import DigitalInOut
from adafruit_wiznet5k.adafruit_wiznet5k import WIZNET5K
import adafruit_wiznet5k.adafruit_wiznet5k_socket as socket
import adafruit_requests as requests

cs = DigitalInOut(board.D10)
spi_bus = busio.SPI(board.SCK, MOSI=board.MOSI, MISO=board.MISO)

# Initialize ethernet interface with DHCP
eth = WIZNET5K(spi_bus, cs)

# Initialize a requests object with a socket and ethernet interface
requests.set_socket(socket, eth)
```

## HTTP GET with Requests

The code makes a HTTP GET request to Adafruit's WiFi testing website - 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`. The call to `requests.get()` is wrapped in a loop which retries 3 times in the event of a failed request.

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

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

```
print("Fetching text from %s"%TEXT_URL)
while not response:
    try:
        response = requests.get(TEXT_URL)
        failure_count = 0
    except AssertionError as error:
        print("Request failed, retrying...\n", error)
        failure_count += 1
        if failure_count &gt;= attempts:
            raise AssertionError("Failed to resolve hostname, \
                                  please check your router's DNS configuration.")
        continue
print('-'*40)

print("Text Response: ", response.text)
print('-'*40)
response.close()
```

While some servers respond with text, some respond with json-formatted data consisting of attribute–value pairs.

CircuitPython_Requests can convert a JSON-formatted response from a server into a CPython **`dict`** object.

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

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

```
print("Fetching JSON data from %s"%JSON_GET_URL)
while not response:
    try:
        response = requests.get(JSON_GET_URL)
        failure_count = 0
    except AssertionError as error:
        print("Request failed, retrying...\n", error)
        failure_count += 1
        if failure_count &gt;= attempts:
            raise AssertionError("Failed to resolve hostname, \
                                  please check your router's DNS configuration.")
        continue
print('-'*40)

print("JSON Response: ", response.json())
print('-'*40)
response.close()
```

## HTTP POST with Requests

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

```
data = '31F'
print("POSTing data to {0}: {1}".format(JSON_POST_URL, data))
while not response:
    try:
        response = requests.post(JSON_POST_URL, data=data)
        failure_count = 0
    except AssertionError as error:
        print("Request failed, retrying...\n", error)
        failure_count += 1
        if failure_count &gt;= attempts:
            raise AssertionError("Failed to resolve hostname, \
                                  please check your router's DNS configuration.")
        continue
print('-'*40)

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

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

```
json_data = {"Date" : "July 25, 2019"}
print("POSTing data to {0}: {1}".format(JSON_POST_URL, json_data))
while not response:
    try:
        response = requests.post(JSON_POST_URL, json=json_data)
        failure_count = 0
    except AssertionError as error:
        print("Request failed, retrying...\n", error)
        failure_count += 1
        if failure_count &gt;= attempts:
            raise AssertionError("Failed to resolve hostname, \
                                  please check your router's DNS configuration.")
        continue
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()
```

## Usage with MiniMQTT

Usage with MiniMQTT ()

## Python Docs

Python Docs ()