MagTag Showerthoughts and Quotes

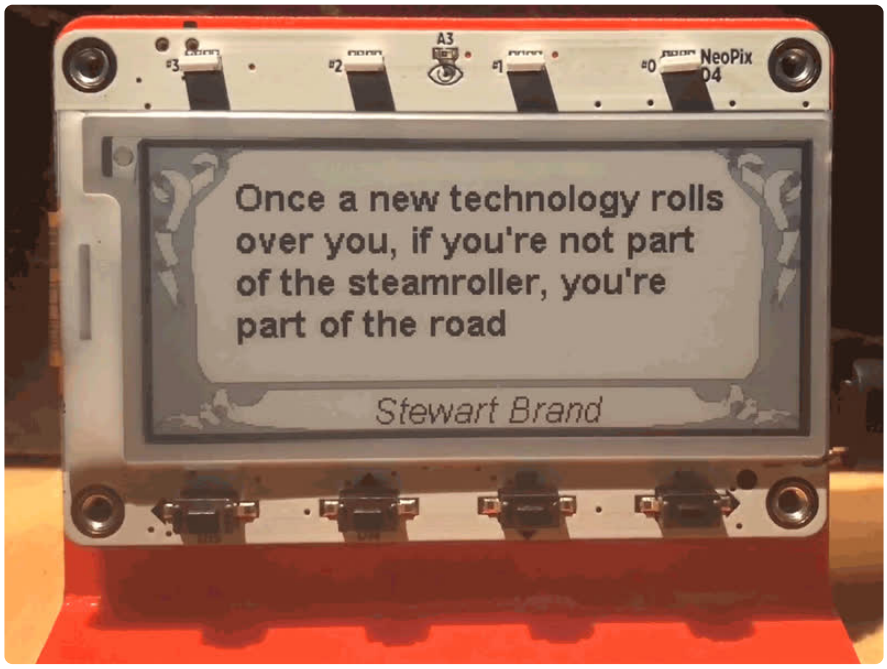
Created by John Park

https://learn.adafruit.com/magtag-showerthoughts

Last updated on 2024-11-29 10:04:25 AM EST

Table of Contents

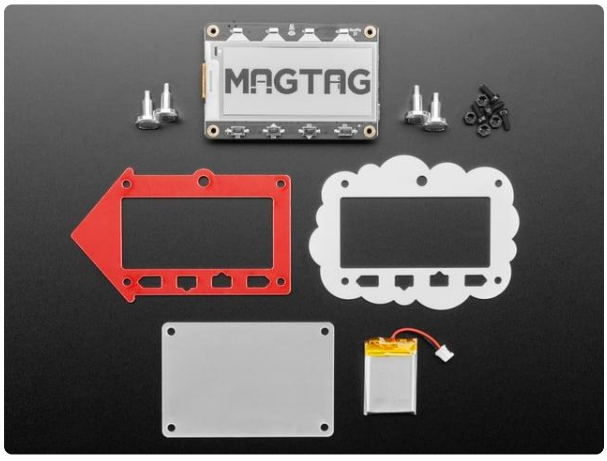
Overview



With the power of the MagTag's ESP32-S2 WiFi enabled processor, we can stream in a continuously updating list of quotes from Adafruit's quotes server. Or, go a bit more to the contemplative/NSFW side with the popular Reddit r/Showerthoughts (https://adafru.it/P9B) display!

By using the built-in deep sleep mode, a battery-powered MagTag can run for days or weeks on a single charge, just waking up every hour to grab a new quote. You can get about 100 refreshes on a battery, so space em out however you like.

Parts



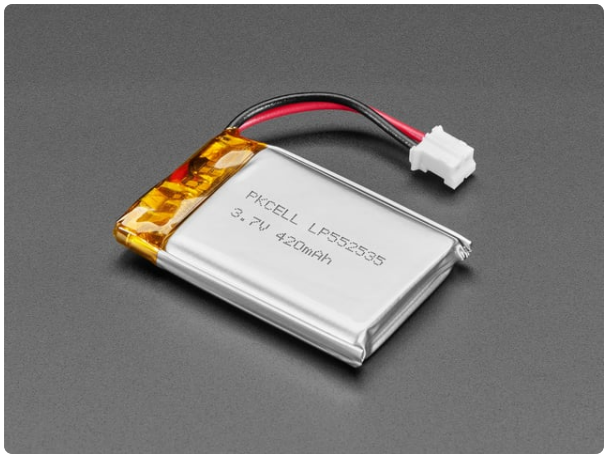
Adafruit MagTag Starter Kit - ADABOX017 Essentials

The Adafruit MagTag combines the new ESP32-S2 wireless module and a 2.9" grayscale E-Ink display to make a low-power IoT display that can show data on its screen...

https://www.adafruit.com/product/4819

[Adafruit MagTag - 2.9" Grayscale E-Ink WiFi Display]
The Adafruit MagTag combines the new ESP32-S2 wireless module and a 2.9" grayscale E-Ink display to make a low-power IoT display that can show data on its screen even when power...
https://www.adafruit.com/product/4800

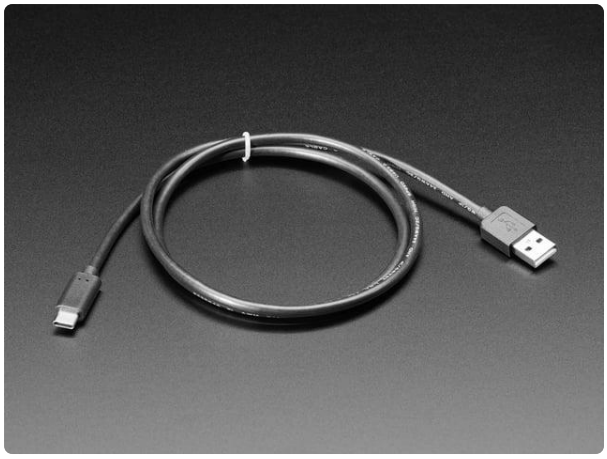
[Mini Magnet Feet for RGB LED Matrices (Pack of 4)]
Got a glorious RGB Matrix project you want to mount and display in your workspace or home? If you have one of the matrix panels listed below, you'll need a pack of these...
https://www.adafruit.com/product/4631



[Lithium Ion Polymer Battery with Short Cable - 3.7V 420mAh]
Lithium-ion polymer (also known as 'lipo' or 'lipoly') batteries are thin, light, and powerful. The output ranges from 4.2V when completely charged to 3.7V. This...
https://www.adafruit.com/product/4236

Depending on which cables you already have on hand, you may also need one of the following items for coding and powering the MagTag.

Micro B USB to USB C Adapter
As technology changes and adapts, so does Adafruit, and speaking of adapting, this adapter has a Micro B USB jack and a USB C...
https://www.adafruit.com/product/4299



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

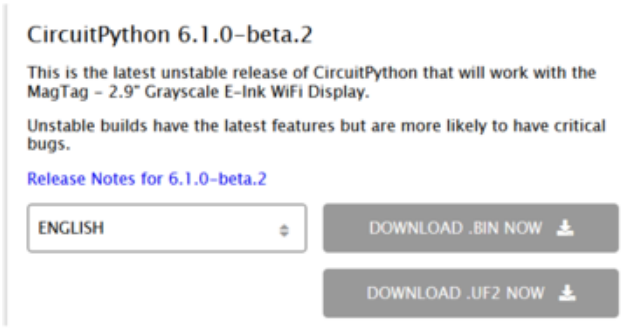
CircuitPython (https://adafru.it/tB7) is a derivative of MicroPython (https://adafru.it/BeZ) designed to simplify experimentation and education on low-cost microcontrollers. It makes it easier than ever to get prototyping by requiring no upfront desktop software downloads. Simply copy and edit files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

Set Up CircuitPython

Follow the steps to get CircuitPython installed on your MagTag.

Download the latest CircuitPython for your board from circuitpython.org

https://adafru.it/OBd



Click the link above and download the latest .BIN and .UF2 file

(depending on how you program the ESP32S2 board you may need one or the other, might as well get both)

Download and save it to your desktop (or wherever is handy).

Plug your MagTag into your computer using a known-good USB cable.

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

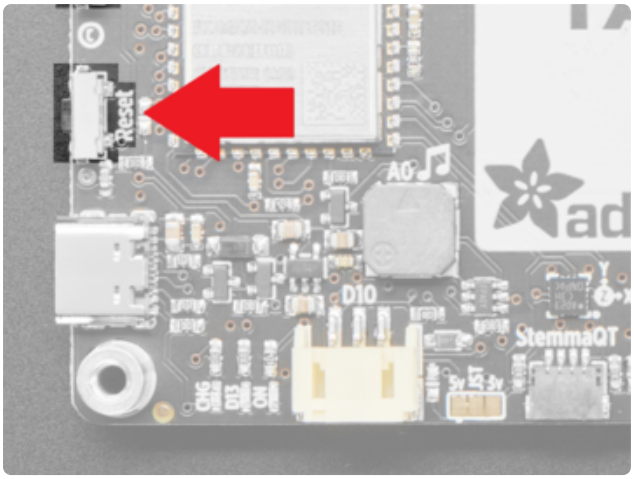
Option 1 - Load with UF2 Bootloader

This is by far the easiest way to load CircuitPython. **However it requires your board has the UF2 bootloader installed. Some early boards do not (we hadn't written UF2 yet!) - in which case you can load using the built in ROM bootloader.**

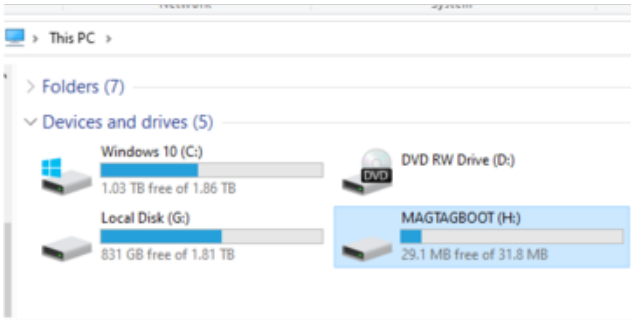
Still, try this first!

Try Launching UF2 Bootloader

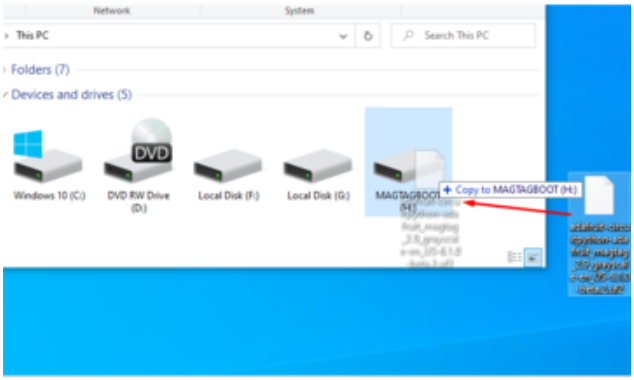
Loading CircuitPython by drag-n-drop UF2 bootloader is the easier way and we recommend it. If you have a MagTag where the front of the board is black, your MagTag came with UF2 already on it.



Launch UF2 by **double-clicking** the Reset button (the one next to the USB C port). You may have to try a few times to get the timing right.

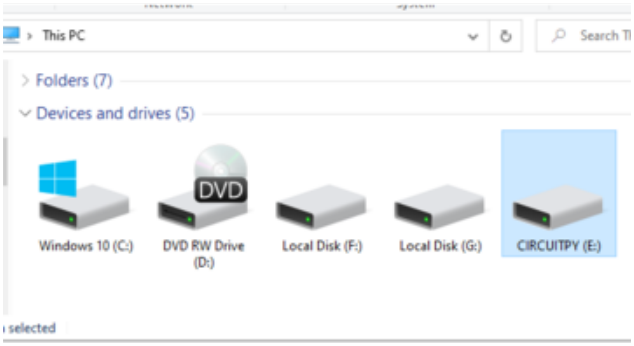


If the UF2 bootloader is installed, you will see a new disk drive appear called **MAGTAGBOOT**



Copy the **UF2** file you downloaded at the first step of this tutorial onto the **MAGTAGBOOT** drive

If you're using Windows and you get an error at the end of the file copy that says **Error from the file copy, Error 0x800701B1: A device which does not exist was specified.** You can ignore this error, the bootloader sometimes disconnects without telling Windows, the install completed just fine and you can continue. If its really annoying, you can also upgrade the bootloader (the latest version of the UF2 bootloader fixes this warning) (https://adafru.it/Pfk)

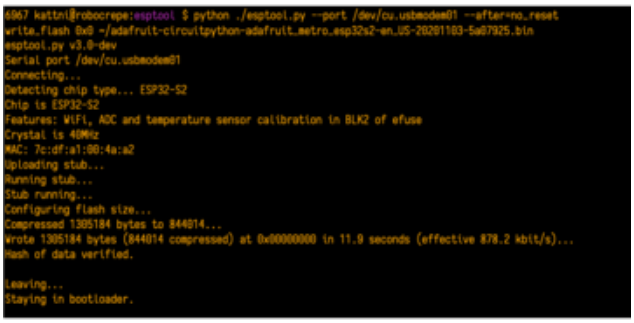


Your board should auto-reset into CircuitPython, or you may need to press reset. A **CIRCUITPY** drive will appear. You're done! Go to the next pages.

Option 2 - Use esptool to load BIN file

If you have an original MagTag with while soldermask on the front, we didn't have UF2 written for the ESP32S2 yet so it will not come with the UF2 bootloader.

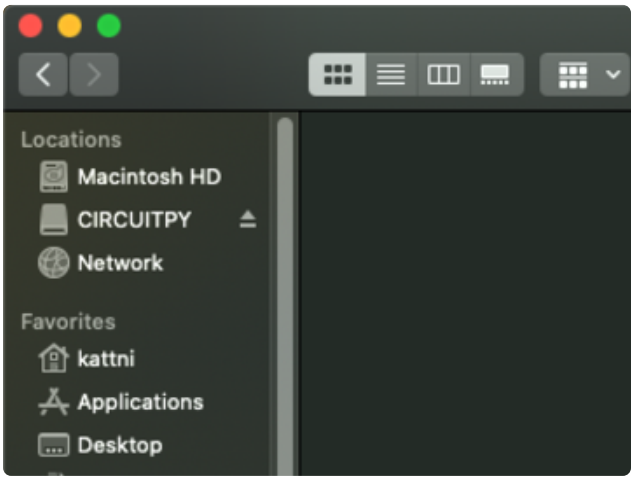
You can upload with **esptool** to the ROM (hardware) bootloader instead!



Follow the initial steps found in the Run esptool and check connection section of the ROM Bootloader page (https://adafru.it/OBc) to verify your environment is set up, your board is successfully connected, and which port it's using.

In the final command to write a binary file to the board, replace the port with your port, and replace "firmware.bin" with the the file you downloaded above.

The output should look something like the output in the image.



Press reset to exit the bootloader.

Your **CIRCUITPY** drive should appear!

You're all set! Go to the next pages.

Option 3 - Use Chrome Browser To Upload BIN file

If for some reason you cannot get esptool to run, you can always try using the Chrome-browser version of esptool we have written. This is handy if you don't have Python on your computer, or something is really weird with your setup that makes esptool not run (which happens sometimes and isn't worth debugging!) You can follow along on the Web Serial ESPTool (https://adafru.it/Pdq) page and either load the UF2 bootloader and then come back to Option 1 on this page, or you can download the CircuitPython BIN file directly using the tool in the same manner as the bootloader.

CircuitPython Internet Test

One of the great things about the ESP32 is the built-in WiFi capabilities. This page covers the basics of getting connected using CircuitPython.

The first thing you need to do is update your **code.py** to the following. 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.

```
# SPDX-FileCopyrightText: 2020 Brent Rubell for Adafruit Industries
#
# SPDX-License-Identifier: MIT

import os
import ipaddress
import ssl
import wifi
import socketpool
import adafruit_requests

# URLs to fetch from
TEXT_URL = "http://wifitest.adafruit.com/testwifi/index.html"
JSON_QUOTES_URL = "https://www.adafruit.com/api/quotes.php"
JSON_STARS_URL = "https://api.github.com/repos/adafruit/circuitpython"

print("ESP32-S2 WebClient Test")

print(f"My MAC address: {[hex(i) for i in wifi.radio.mac_address]}")

print("Available WiFi networks:")
for network in wifi.radio.start_scanning_networks():
    print("\t%s\t\tRSSI: %d\tChannel: %d" % (str(network.ssid, "utf-8"),
                                              network.rssi, network.channel))
wifi.radio.stop_scanning_networks()

print(f"Connecting to {os.getenv('CIRCUITPY_WIFI_SSID')}")
wifi.radio.connect(os.getenv("CIRCUITPY_WIFI_SSID"),
os.getenv("CIRCUITPY_WIFI_PASSWORD"))
print(f"Connected to {os.getenv('CIRCUITPY_WIFI_SSID')}")
print(f"My IP address: {wifi.radio.ipv4_address}")

ping_ip = ipaddress.IPv4Address("8.8.8.8")
ping = wifi.radio.ping(ip=ping_ip)

# retry once if timed out
if ping is None:
    ping = wifi.radio.ping(ip=ping_ip)

if ping is None:
    print("Couldn't ping 'google.com' successfully")
else:
    # convert s to ms
    print(f"Pinging 'google.com' took: {ping * 1000} ms")

pool = socketpool.SocketPool(wifi.radio)
requests = adafruit_requests.Session(pool, ssl.create_default_context())

print(f"Fetching text from {TEXT_URL}")
response = requests.get(TEXT_URL)
print("-" * 40)
print(response.text)
print("-" * 40)

print(f"Fetching json from {JSON_QUOTES_URL}")
response = requests.get(JSON_QUOTES_URL)
print("-" * 40)
print(response.json())
print("-" * 40)

print()
```

```
print(f"Fetching and parsing json from {JSON_STARS_URL}")
response = requests.get(JSON_STARS_URL)
print("-" * 40)
print(f"CircuitPython GitHub Stars: {response.json()['stargazers_count']}")
print("-" * 40)

print("Done")
```

Your **CIRCUITPY** drive should resemble the following.

To get connected, the next thing you need to do is update the **settings.toml** file.

The settings.toml 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 **settings.toml** file, that is on your **CIRCUITPY** drive, to hold secret/private/custom data. That way you can share your main project without worrying about accidentally sharing private stuff.

If you have a fresh install of CircuitPython on your board, the initial **settings.toml** file on your **CIRCUITPY** drive is empty.

To get started, you can update the **settings.toml** on your **CIRCUITPY** drive to contain the following code.

```
# SPDX-FileCopyrightText: 2023 Adafruit Industries
#
# SPDX-License-Identifier: MIT

# This is where you store the credentials necessary for your code.
# The associated demo only requires WiFi, but you can include any
# credentials here, such as Adafruit IO username and key, etc.
CIRCUITPY_WIFI_SSID = "your-wifi-ssid"
CIRCUITPY_WIFI_PASSWORD = "your-wifi-password"
```

This file should contain a series of Python variables, each assigned to a string. Each variable should describe what it represents (say `wifi_ssid`), followed by an

= (equals sign), followed by the data in the form of a Python string (such as `"my-wifi-password"` including the quote marks).

At a minimum you'll need to add/update your WiFi SSID and WiFi password, so do that now!

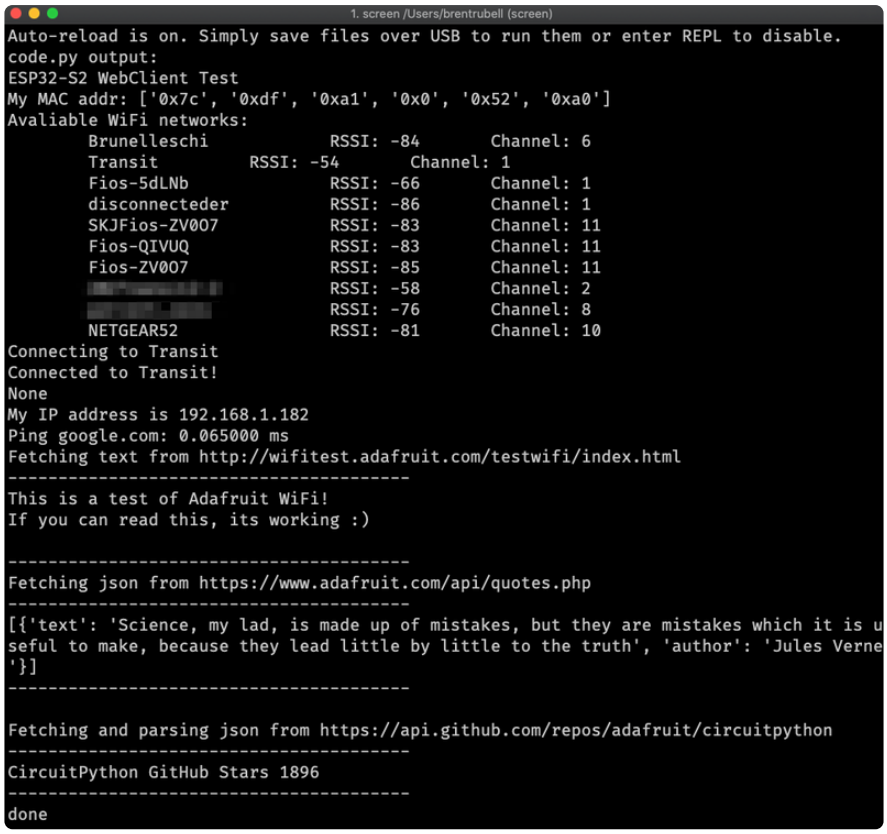
As you make projects you may need more tokens and keys, just add them one line at a time. See for example other tokens such as one for accessing GitHub or the Hackaday API. Other non-secret data like your timezone can also go here.

For the correct time zone string, look at http://worldtimeapi.org/timezones (https://adafru.it/EcP) 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 **settings.toml** - keep that out of GitHub, Discord or other project-sharing sites.

> Don't share your settings.toml file! It has your passwords and API keys in it!

If you connect to the serial console, you should see something like the following:



In order, the example code...

Checks the ESP32's MAC address.

```
print(f"My MAC address: {[hex(i) for i in wifi.radio.mac_address]}")
```

Performs a scan of all access points and prints out the access point's name (SSID), signal strength (RSSI), and channel.

```
print("Available WiFi networks:")
for network in wifi.radio.start_scanning_networks():
    print("\t%s\t\tRSSI: %d\tChannel: %d" % (str(network.ssid, "utf-8"),
                                              network.rssi, network.channel))
wifi.radio.stop_scanning_networks()
```

Connects to the access point you defined in the **settings.toml** file, and prints out its local IP address.

```
print(f"Connecting to {os.getenv('WIFI_SSID')}")
wifi.radio.connect(os.getenv("WIFI_SSID"), os.getenv("WIFI_PASSWORD"))
print(f"Connected to {os.getenv('WIFI_SSID')}")
print(f"My IP address: {wifi.radio.ipv4_address}")
```

Attempts to ping a Google DNS server to test connectivity. If a ping fails, it returns `None`. Initial pings can sometimes fail for various reasons. So, if the initial ping is successful (`is not None`), it will print the echo speed in ms. If the initial ping fails, it

will try one more time to ping, and then print the returned value. If the second ping fails, it will result in `"Ping google.com: None ms"` being printed to the serial console. Failure to ping does not always indicate a lack of connectivity, so the code will continue to run.

```
ping_ip = ipaddress.IPv4Address("8.8.8.8")
ping = wifi.radio.ping(ip=ping_ip) * 1000
if ping is not None:
    print(f"Ping google.com: {ping} ms")
else:
    ping = wifi.radio.ping(ip=ping_ip)
    print(f"Ping google.com: {ping} ms")
```

The code creates a socketpool using the wifi radio's available sockets. This is performed so we don't need to re-use sockets. Then, it initializes a a new instance of the requests (https://adafru.it/E9o) interface - which makes getting data from the internet really really easy.

```
pool = socketpool.SocketPool(wifi.radio)
requests = adafruit_requests.Session(pool, ssl.create_default_context())
```

To read in plain-text from a web URL, call `requests.get` - you may pass in either a http, or a https url for SSL connectivity.

```
print(f"Fetching text from {TEXT_URL}")
response = requests.get(TEXT_URL)
print("-" * 40)
print(response.text)
print("-" * 40)
```

Requests can also display a JSON-formatted response from a web URL using a call to `requests.get`.

```
print(f"Fetching json from {JSON_QUOTES_URL}")
response = requests.get(JSON_QUOTES_URL)
print("-" * 40)
print(response.json())
print("-" * 40)
```

Finally, you can fetch and parse a JSON URL using `requests.get`. This code snippet obtains the `stargazers_count` field from a call to the GitHub API.

```
print(f"Fetching and parsing json from {JSON_STARS_URL}")
response = requests.get(JSON_STARS_URL)
print("-" * 40)
print(f"CircuitPython GitHub Stars: {response.json()['stargazers_count']}")
print("-" * 40)
```

OK you now have your ESP32 board set up with a proper **settings.toml** file and can connect over the Internet. If not, check that your **settings.toml** file has the right SSID and password and retrace your steps until you get the Internet connectivity working!

IPv6 Networking

Starting in CircuitPython 9.2, IPv6 networking is available on most Espressif wifi boards. Socket-using libraries like **adafruit_requests** and **adafruit_ntp** will need to be updated to use the new APIs and for now can only connect to services on IPv4.

IPv6 connectivity & privacy

IPv6 addresses are divided into many special kinds, and many of those kinds (like those starting with **FC**, **FD**, **FE**) are private or local; Addresses starting with other prefixes like **2002:** and **2001:** are globally routable. In 2024, far from all ISPs and home networks support IPv6 internet connectivity. For more info consult resources like Wikipedia (https://adafru.it/1a4z). If you're interested in global IPv6 connectivity you can use services like Hurricane Electric (https://adafru.it/1a4A) to create an "IPv6 tunnel" (free as of 2024, but requires expertise and a compatible router or host computer to set up)

It's also important to be aware that, as currently implemented by Espressif, there are privacy concerns especially when these devices operate on the global IPv6 network: The device's unique identifier (its EUI-64 or MAC address) is used by default as part of its IPv6 address. This means that the device identity can be tracked across multiple networks by any service it connects to.

Enable IPv6 networking

Due to the privacy consideration, IPv6 networking is not automatically enabled. Instead, it must be explicitly enabled by a call to `start_dhcp_client` with the `ipv6=True` argument specified:

```
wifi.start_dhcp_client(ipv6=True)
```

Check IP addresses

The read-only `addresses` property of the `wifi.radio` object holds all addresses, including IPv4 and IPv6 addresses:

```
&gt;&gt;&gt; wifi.radio.addresses
('FE80::7EDF:A1FF:FE00:518C', 'FD5F:3F5C:FE50:0:7EDF:A1FF:FE00:518C', '10.0.3.96')
```

The `wifi.radio.dns` servers can be IPv4 or IPv6:

```
&gt;&gt;&gt; wifi.radio.dns
('FD5F:3F5C:FE50::1',)
```

```
&gt;&gt;&gt; wifi.radio.dns = ("1.1.1.1",)
&gt;&gt;&gt; wifi.radio.dns
('1.1.1.1',)
```

Ping v6 networks

`wifi.radio.ping` accepts v6 addresses and names:

```
&gt;&gt;&gt; wifi.radio.ping("google.com")
0.043
&gt;&gt;&gt; wifi.radio.ping("ipv6.google.com")
0.048
```

Create & use IPv6 sockets

Use the address family `socket.AF_INET6`. After the socket is created, use methods like `connect`, `send`, `recfrom_into`, etc just like for IPv4 sockets. This code snippet shows communicating with a private-network NTP server; this IPv6 address will not work on your network:

```
&gt;&gt;&gt; ntp_addr = ("fd5f:3f5c:fe50::20e", 123)
&gt;&gt;&gt; PACKET_SIZE = 48
&gt;&gt;&gt;
&gt;&gt;&gt; buf = bytearray(PACKET_SIZE)
&gt;&gt;&gt; with socket.socket(socket.AF_INET6, socket.SOCK_DGRAM) as s:
...     s.settimeout(1)
...     buf[0] = 0b0010_0011
...     s.sendto(buf, ntp_addr)
...     print(s.recvfrom_into(buf))
...     print(buf)
...
48
(48, ('fd5f:3f5c:fe50::20e', 123))
bytearray(b'$\x01\x03\xeb\x00\x00\x00\x00\x00\x00\x00GGPS\x00\xeaA0h\x07s;
\xc0\x00\x00\x00\x00\x00\x00\x00\xeaA0n\xeb4\x82-\xeaA0n\xebAU\xb1')
```

Code the MagTag Showerthoughts Viewer

Text Editor

Adafruit recommends using the Mu editor for editing your CircuitPython code. You can get more info in this guide (https://adafru.it/ANO).

Alternatively, you can use any text editor that saves simple text files.

Follow this guide (https://adafru.it/Pla) for info on getting your AIO credentials for the **secrets.py** file.

Installing Project Code

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 **MagTag_Showerthoughts/** 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:

CIRCUITPY

```
Temporarily unable to load content:
```

How It Works

Libraries

We import a few libraries to take care of the heavy lifting, in this case:

```
import time
import random
from adafruit_magtag.magtag import MagTag
```

The `time` library allows us to do some pausing between steps.

We'll use the `random` library to randomize posts.

The `adafruit_magtag` library makes it very simple to set up the MagTag's display, get online via WiFi, and to request and parse .json files.

Showerthought Variables

We'll set up some variables for the sources of our data. This represent the Reddit URL as well as the .json traversal needed to grab the quote and author name.

```
DATA_SOURCE = "https://www.reddit.com/r/showerthoughts/hot.json?limit=10"
quote_num = random.randint(0, 9)  # we get 10 possibilities, pick one of them
QUOTE_LOCATION = ["data", "children", quote_num, "data", "title"]
AUTHOR_LOCATION = ["data", "children", quote_num, "data", "author"]
```

Sleepy Time

The MagTag uses a deep sleep mode to conserve battery power between updates. We'll set the quotes to refresh every hour, which means we can get days of use on a single charge of a LiPo battery!

```
TIME_BETWEEN_REFRESHES = 1 * 60 * 60  # one hour delay
```

Setup

There are a number of setup steps we'll take next. First, we create the `MagTag()` object named `magtag` (all lower-case is easier to type anyway!).

Then we'll set the background graphic to the on-disk bitmap file.

```
magtag = MagTag(
    url=DATA_SOURCE,
    json_path=(QUOTE_LOCATION, AUTHOR_LOCATION),
)

magtag.graphics.set_background("/bmps/magtag_shower_bg.bmp")
```

Text

Next up, we'll use the magtag library's text commands to create the two text objects, one for the quote and one for the author.

The `magtag.add_text()` command has arguments for the font, wrap width, maximum length of text glyphs, position, line spacing, and anchor point for the text.

```
magtag.add_text(
    text_font="/fonts/Arial-Bold-12.bdf",
    text_wrap=28,
    text_maxlen=120,
    text_position=(
        (magtag.graphics.display.width // 2),
        (magtag.graphics.display.height // 2) - 10,
    ),
    line_spacing=0.75,
    text_anchor_point=(0.5, 0.5),  # center the text on x &amp;amp; y
)

# author in italic text, no wrapping
magtag.add_text(
    text_font="/fonts/Arial-Italic-12.bdf",
    text_position=(magtag.graphics.display.width // 2, 118),
    text_anchor_point=(0.5, 0.5),  # left justify this line
)
```

Connect

Next we'll get online, using the authentication details in the **secrets.py** file to connect to the network.

Then, we'll use the `magtag.fetch()` command which will go to the specified `DATA_SOURCE` url, grab the .json file, and parse it for the `QUOTE_LOCATION` text and `AUTHOR_LOCATION` text. (We include some error checking just in case. That's Internet!)

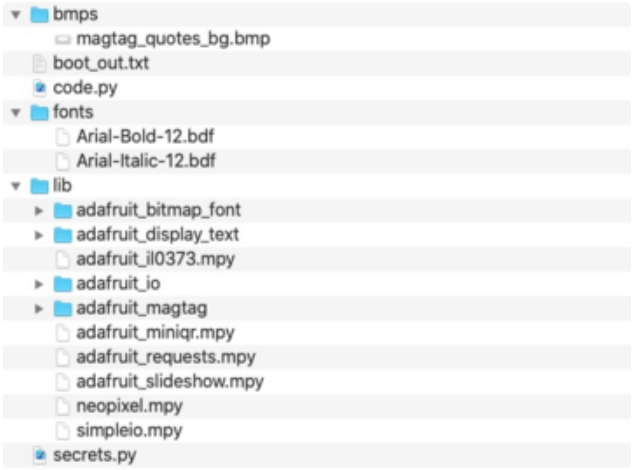
Once retrieved and parsed, the text is displayed on the MagTag.

Next, the MagTag takes an hour long nap, then wakes up to do it all over again. This then repeats until the end of time, keeping you happy with fresh quotes!

```
try:
    magtag.network.connect()
    value = magtag.fetch()
    print("Response is", value)
except (ValueError, RuntimeError) as e:
    magtag.set_text(e)
    print("Some error occured, retrying! -", e)

# wait 2 seconds for display to complete
time.sleep(2)
magtag.exit_and_deep_sleep(TIME_BETWEEN_REFRESHES)
```

Code the MagTag Quotes Board



Libraries

First, make sure you've installed the fundamental MagTag libraries as shown on the MagTag CircuitPython Setup page.

Text Editor

Adafruit recommends using the Mu editor for editing your CircuitPython code. You can get more info in this guide (https://adafru.it/ANO).

Alternatively, you can use any text editor that saves simple text files.

Code

Click the Download Project Bundle button below in the code window to get a zip file with all the files needed for the project. Copy **code.py** from the zip file and place on the **CIRCUITPY** drive.

Also copy the whole **/bmps** directory from the zip file and place and its contents it on the **CIRCUITYPY** drive. These are some sample images you can start with.

Copy the **/fonts** directory from the zip file and place and its contents it on the **CIRCUITYPY** drive.

```
# SPDX-FileCopyrightText: 2020 Limor Fried for Adafruit Industries
#
# SPDX-License-Identifier: MIT

# MagTag Shower Thoughts
# Be sure to put WiFi access point info in secrets.py file to connect

import time
import random
from adafruit_magtag.magtag import MagTag


# Set up where we'll be fetching data from
DATA_SOURCE = "https://www.reddit.com/r/showerthoughts/hot.json?limit=10"
quote_num = random.randint(0, 9)  # we get 10 possibilities, pick one of them
QUOTE_LOCATION = ["data", "children", quote_num, "data", "title"]
AUTHOR_LOCATION = ["data", "children", quote_num, "data", "author"]

# in seconds, we can refresh about 100 times on a battery
TIME_BETWEEN_REFRESHES = 1 * 60 * 60  # one hour delay

magtag = MagTag(
    url=DATA_SOURCE,
    json_path=(QUOTE_LOCATION, AUTHOR_LOCATION),
)

magtag.graphics.set_background("/bmps/magtag_shower_bg.bmp")

# quote in bold text, with text wrapping
magtag.add_text(
    text_font="/fonts/Arial-Bold-12.pcf",
    text_wrap=28,
    text_maxlen=120,
    text_position=(
        (magtag.graphics.display.width // 2),
        (magtag.graphics.display.height // 2) - 10,
    ),
    line_spacing=0.75,
    text_anchor_point=(0.5, 0.5),  # center the text on x & y
)

# author in italic text, no wrapping
magtag.add_text(
    text_font="/fonts/Arial-Italic-12.bdf",
    text_position=(magtag.graphics.display.width // 2, 118),
    text_anchor_point=(0.5, 0.5),  # left justify this line
)

try:
    magtag.network.connect()
    value = magtag.fetch()
    print("Response is", value)
except (ValueError, RuntimeError, ConnectionError, OSError) as e:
    magtag.set_text(e)
    print("Some error occured, retrying! -", e)

# wait 2 seconds for display to complete
time.sleep(2)
magtag.exit_and_deep_sleep(TIME_BETWEEN_REFRESHES)
```

How It Works
Libraries

We import a few libraries to take care of the heavy lifting, in this case:

```
import time
from adafruit_magtag.magtag import MagTag
```

The `time` library allows us to do some pausing between steps.

The `adafruit_magtag` library makes it very simple to set up the MagTag's display, get online via WiFi, and to request and parse .json files.

Quote Variables

We'll set up some variables for the sources of our quote data.

```
DATA_SOURCE = "https://www.adafruit.com/api/quotes.php"
QUOTE_LOCATION = [0, "text"]
AUTHOR_LOCATION = [0, "author"]
```

Sleepy Time

The MagTag uses a deep sleep mode to conserve battery power between updates. We'll set the quotes to refresh every hour, which means we can get days of use on a single charge of a LiPo battery!

```
TIME_BETWEEN_REFRESHES = 1 * 60 * 60  # one hour delay
```

Setup

There are a number of setup steps we'll take next. First, we create the `MagTag()` object named `magtag` (all lower-case is easier to type anyway!).

Then we'll set the background graphic to the on-disk bitmap file.

```
magtag = MagTag(
    url=DATA_SOURCE,
    json_path=(QUOTE_LOCATION, AUTHOR_LOCATION),
)
magtag.graphics.set_background("/bmps/magtag_quotes_bg.bmp")
```

Text

Next up, we'll use the magtag library's text commands to create the two text objects, one for the quote and one for the author.

The `magtag.add_text()` command has arguments for the font, wrap width, maximum length of text glyphs, position, line spacing, and anchor point for the text.

```
magtag.add_text(
    text_font="/fonts/Arial-Bold-12.bdf",
    text_wrap=28,
    text_maxlen=120,
    text_position=(
```

```
        (magtag.graphics.display.width // 2),
        (magtag.graphics.display.height // 2) - 10,
    ),
    line_spacing=0.75,
    text_anchor_point=(0.5, 0.5),  # center the text on x &amp;amp; y
)

# author in italic text, no wrapping
magtag.add_text(
    text_font="/fonts/Arial-Italic-12.bdf",
    text_position=(magtag.graphics.display.width // 2, 118),
    text_anchor_point=(0.5, 0.5),  # center it in the nice scrolly thing
)
```

Connect

Next we'll get online, using the authentication details in the **secrets.py** file to connect to the network.

Then, we'll use the `magtag.fetch()` command which will go to the specified `DATA_SOURCE` url, grab the .json file, and parse it for the `QUOTE_LOCATION` text and `AUTHOR_LOCATION` text. (We include some error checking just in case. That's Internet!)

Once retrieved and parsed, the text is displayed on the MagTag.

Next, the MagTag takes an hour long nap, then wakes up to do it all over again. This then repeats until the end of time, keeping you happy with fresh quotes!

```
try:
    magtag.network.connect()
    value = magtag.fetch()
    print("Response is", value)
except (ValueError, RuntimeError) as e:
    magtag.set_text(e)
    print("Some error occured, retrying later -", e)
# wait 2 seconds for display to complete
time.sleep(2)
magtag.exit_and_deep_sleep(TIME_BETWEEN_REFRESHES)
```