



SpaceX Next Launch Display with Adafruit MagTag

Created by Anne Barela



<https://learn.adafruit.com/spacex-next-launch-display-with-adafruit-magtag>

Last updated on 2024-11-29 10:04:47 AM EST

Table of Contents

| | |
|--------------------------------------------------------------------|----|
| Overview | 3 |
| <hr/> | |
| • Parts | |
| Install CircuitPython | 5 |
| <hr/> | |
| • Set Up CircuitPython | |
| • Option 1 - Load with UF2 Bootloader | |
| • Try Launching UF2 Bootloader | |
| • Option 2 - Use esptool to load BIN file | |
| • Option 3 - Use Chrome Browser To Upload BIN file | |
| CircuitPython Internet Test | 9 |
| <hr/> | |
| • The settings.toml File | |
| • IPv6 Networking | |
| The Data API | 16 |
| <hr/> | |
| Code | 17 |
| <hr/> | |
| • Fonts | |
| • Installing Project Code | |
| • Code Runthrough | |
| Going Further | 21 |
| <hr/> | |
| • Low Power | |
| • Countdown to Launch | |
| • Launchpad | |
| • Other Space APIs | |

Overview

The web API used for this guide has ceased relaying launch information. There are other APIs on the web but this has not been converted yet, so this guide is instructional only.



The Adafruit MagTag is perfect for reminders of upcoming events. This includes space launches. and in this project, launches by SpaceX. SpaceX is launching numerous interesting missions including manned to the ISS, cargo for the ISS, Starlink satellites, and more.

The MagTag fetches launch JSON data via WiFi and displays it on the beautiful eInk display. The mission, when the launch is occurring and a description of the mission (if available) and formatted onto the display.

Place the MagTag on a desk or on a refrigerator - never miss another launch again.

The current version initiates a deep sleep for 24 hours between refreshes, which should make a fully charged battery last a long time. You will still have to charge it up via a USB cable periodically - if it doesn't update, charge it.

This project requires CircuitPython 6.1.0 beta 2 or higher.

Parts



[Adafruit MagTag - 2.9" Grayscale E-Ink WiFi Display](https://www.adafruit.com/product/4800)

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...

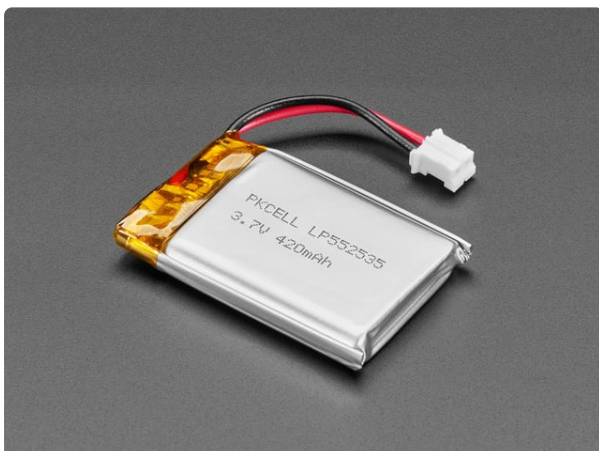
<https://www.adafruit.com/product/4800>



[Adafruit MagTag Starter Kit - ADABOX017 Essentials](https://www.adafruit.com/product/4819)

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>



[Lithium Ion Polymer Battery with Short Cable - 3.7V 420mAh](https://www.adafruit.com/product/4236)

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...

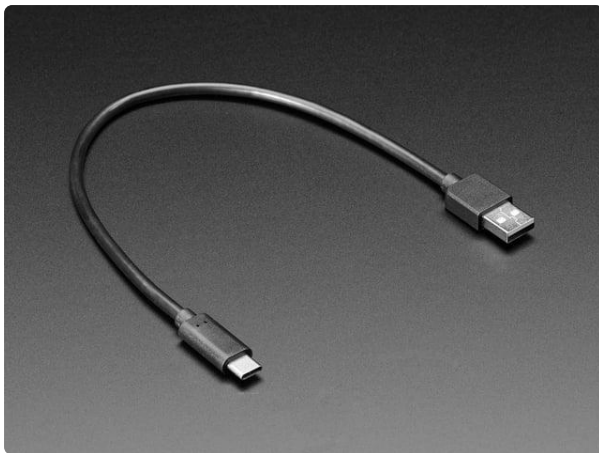
<https://www.adafruit.com/product/4236>



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>



USB Type A to Type C Cable - 1ft - 0.3 meter

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/4473>

Install CircuitPython

[CircuitPython](https://adafru.it/tB7) (<https://adafru.it/tB7>) is a derivative of [MicroPython](https://adafru.it/BeZ) (<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>

CircuitPython 6.1.0-beta.2

This is the latest unstable release of CircuitPython that will work with the MagTag - 2.9" Grayscale E-Ink WiFi Display.

Unstable builds have the latest features but are more likely to have critical bugs.

[Release Notes for 6.1.0-beta.2](#)

ENGLISH

DOWNLOAD .BIN NOW

DOWNLOAD .UF2 NOW

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

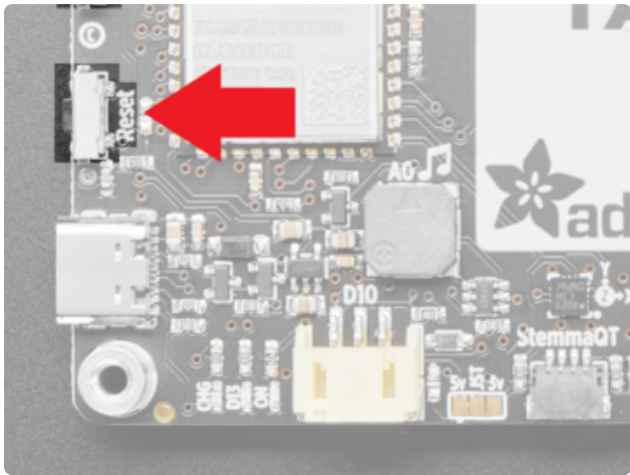
This is by far the easiest way to load CircuitPython. However it requires your board has the UF2 bootloader installed. Some early boards do not (we hadn't written UF2 yet!) - in which case you can load using the built in ROM bootloader.

Still, try this first!

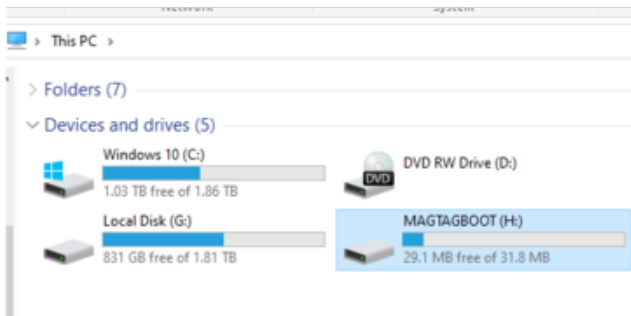


Try Launching UF2 Bootloader

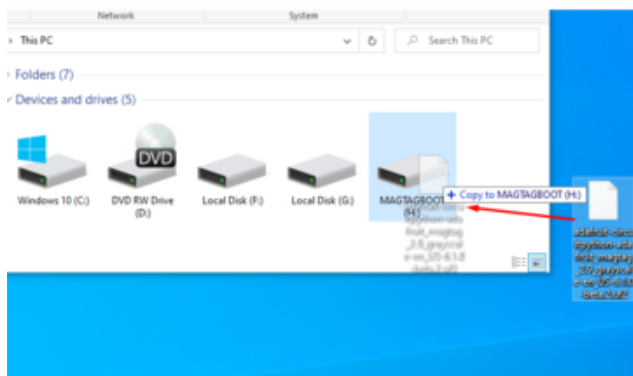
Loading CircuitPython by drag-n-drop UF2 bootloader is the easier way and we recommend it. If you have a MagTag where the front of the board is black, your MagTag came with UF2 already on it.



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.

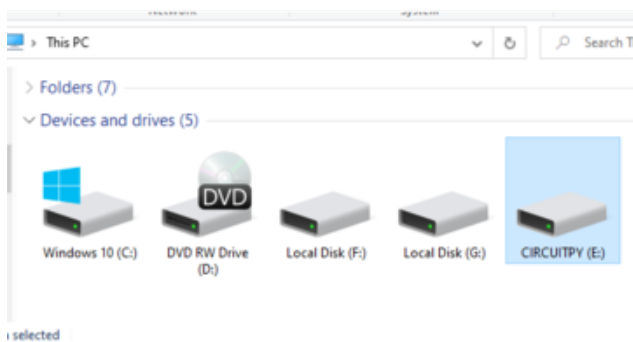


If the UF2 bootloader is installed, you will see a new disk drive appear called **MAGTAGBOOT**



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 \(the latest version of the UF2 bootloader fixes this warning\) \(https://adafru.it/Pfk\)](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 white 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!


```

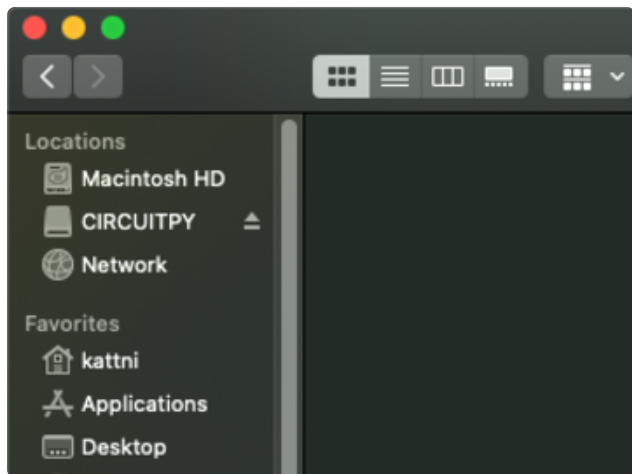
kattni@roborepe:esptool $ python ./esptool.py --port /dev/cu.usbmodem01 --after-no-reset
write_flash 0x0 ~/adafruit-circuitpython-adafruit_metro_esp32-en_US-20201103-5a07925.bin
esptool.py v3.0-dev
Serial port /dev/cu.usbmodem01
Connecting...
Detecting chip type... ESP32-S2
Chip is ESP32-S2
Features: WiFi, ADC and temperature sensor calibration in BLK2 of efuse
Crystal is 40MHz
MAC: 7c:df:a1:00:4a:a2
Uploading stub...
Running stub...
Stub running...
Configuring flash size...
Compressed 1305184 bytes to 844014...
Wrote 1305184 bytes (844014 compressed) at 0x00000000 in 11.9 seconds (effective 878.2 kbit/s)...
Flash of data verified.
Leaving...
Staying in bootloader.

```

Follow the initial steps found in the [Run esptool and check connection section of the ROM Bootloader page \(https://adafruit.it/OBc\)](#) 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 \(https://adafruit.it/Pdq\)](https://adafruit.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)

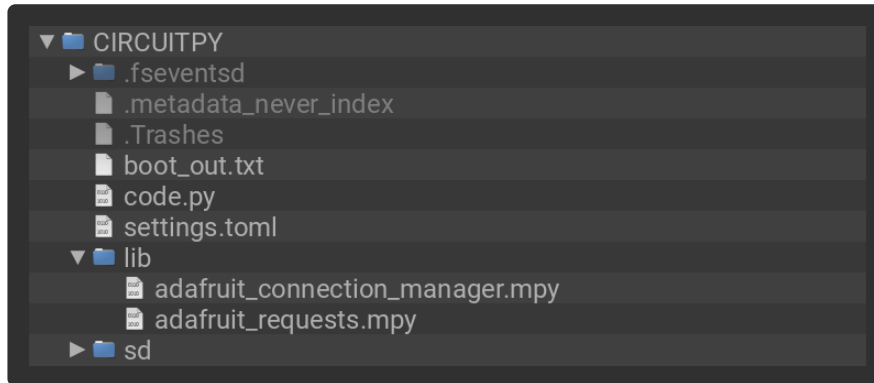
print(f"Fetching json from {JSON_QUOTES_URL}")
response = requests.get(JSON_QUOTES_URL)
print("-" * 40)
print(response.json())
print("-" * 40)

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://adafruit.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:

```
1. screen /Users/brentrubell (screen)
Auto-reload is on. Simply save files over USB to run them or enter REPL to disable.
code.py output:
ESP32-S2 WebClient Test
My MAC addr: ['0x7c', '0xdf', '0xa1', '0x0', '0x52', '0xa0']
Available WiFi networks:
  Brunelleschi      RSSI: -84      Channel: 6
  Transit           RSSI: -54      Channel: 1
  Fios-5dLNb        RSSI: -66      Channel: 1
  disconnectededer   RSSI: -86      Channel: 1
  SKJFios-ZV007     RSSI: -83      Channel: 11
  Fios-QIVUQ        RSSI: -83      Channel: 11
  Fios-ZV007        RSSI: -85      Channel: 11
  [REDACTED]         RSSI: -58      Channel: 2
  [REDACTED]         RSSI: -76      Channel: 8
  NETGEAR52         RSSI: -81      Channel: 10
Connecting to Transit
Connected to Transit!
None
My IP address is 192.168.1.182
Ping google.com: 0.065000 ms
Fetching text from http://wifitest.adafruit.com/testwifi/index.html
-----
This is a test of Adafruit WiFi!
If you can read this, its working :)
-----
Fetching json from https://www.adafruit.com/api/quotes.php
-----
[{'text': 'Science, my lad, is made up of mistakes, but they are mistakes which it is u
seful to make, because they lead little by little to the truth', 'author': 'Jules Verne
'}]
-----
Fetching and parsing json from https://api.github.com/repos/adafruit/circuitpython
-----
CircuitPython GitHub Stars 1896
-----
done
```

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 new instance of the [requests](https://adafruit.it/E9o) (<https://adafruit.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 \(https://adafru.it/1a4z\)](https://adafru.it/1a4z). If you're interested in global IPv6 connectivity you can use services like [Hurricane Electric \(https://adafru.it/1a4A\)](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:

```
>>> 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:

```
>>> wifi.radio.dns
('FD5F:3F5C:FE50::1',)
```

```
>>> wifi.radio.dns = ("1.1.1.1",)
>>> wifi.radio.dns
('1.1.1.1',)
```

Ping v6 networks

wifi.radio.ping accepts v6 addresses and names:

```
>>> wifi.radio.ping("google.com")
0.043
>>> 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`, `recvfrom_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:

```
>>> ntp_addr = ("fd5f:3f5c:fe50::20e", 123)
>>> PACKET_SIZE = 48
>>>
>>> buf = bytearray(PACKET_SIZE)
>>> 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\x00\xeaA0n\xeb4\x82-\xeaA0n\xebAU\xb1')
```

The Data API



Projects that use Adafruit displays with WiFi often read local or remote data and display it on the screen. For this project, data from a 3rd party website that provides SpaceX data was used.

The SpaceX-API project is an Open Source REST API for rocket, core, capsule, pad, and launch data. They state "We are not affiliated, associated, authorized, endorsed by, or in any way officially connected with Space Exploration Technologies Inc (SpaceX), or any of its subsidiaries or its affiliates. The names SpaceX as well as related names, marks, emblems and images are registered trademarks of their respective owners."

The url <https://api.spacexdata.com/v4/launches/next> (<https://adafru.it/P2b>) provides the data for the next launch in JSON format. The project is licensed under the Apache 2.0 License.

Details on what data is available can be found at <https://github.com/r-spacex/SpaceX-API> (<https://adafru.it/P2c>).

Three fields which were chosen to display in this project:

1. **name** - the mission name
2. **data_local** - the launch date locally when the launch will occur
3. **details** - details about the launch (when available), a long text string

If other data is desired, the code only need to get the data in the desired fields.

Code



This project requires CircuitPython 6.1.0 beta 2 or higher.

Fonts

To take advantage of the lovely elnk display, 4 different fonts are used. They are obtained when downloading all the project resources using "Download Project Zip" in the code window below.

- Lato-Bold-ltd-25.bdf
- Arial-Bold-12.bdf
- Arial-12.bdf

Note the **Lato-Bold-ltd-25.bdf** font only has the letters to display "Next SpaceX Launch". This is to save space on the limited amount of flash on the MagTag. Lato is a freely licensed version of Arial provided by Google. An additional, built-in font in the `terminalio` library is used for the details.

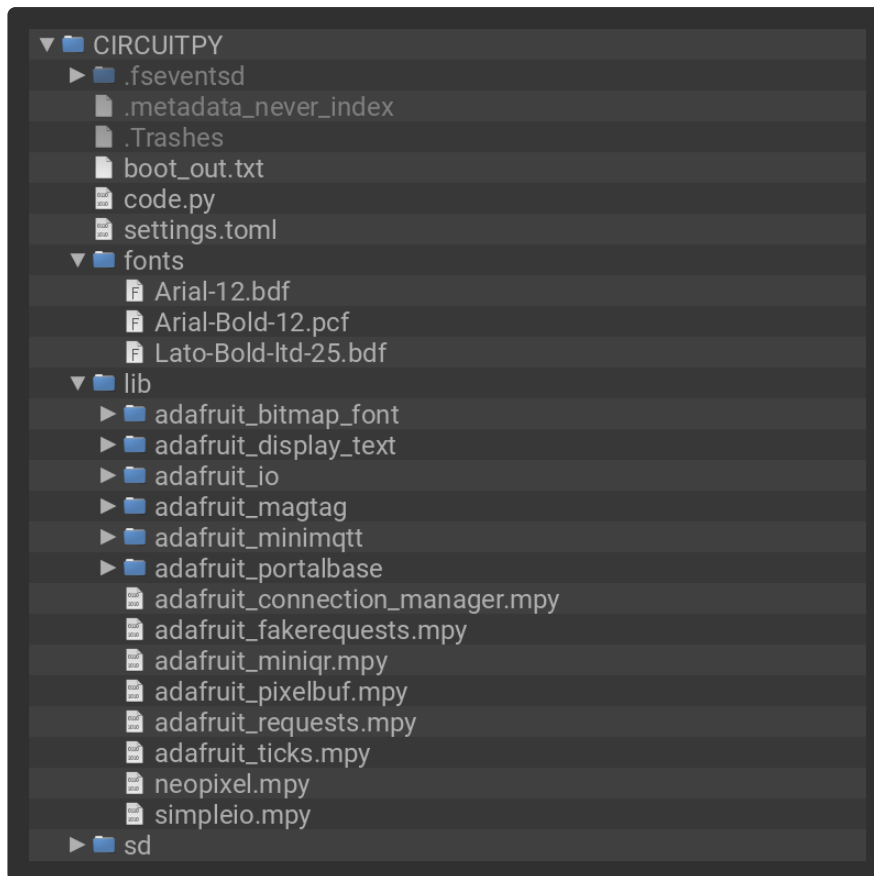
If you want to look at using your own fonts, see the excellent guide [Custom Fonts for CircuitPython Displays \(https://adafru.it/E7E\)](https://adafru.it/E7E).

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_SpaceX/** 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:



If you're having difficulty running this example, it could be because your MagTag CircuitPython firmware or library needs to be upgraded! Please be sure to follow <https://learn.adafruit.com/adafruit-magtag/circuitpython> to install the latest CircuitPython firmware and then also replace/update ALL the MagTag-specific libraries mentioned here <https://learn.adafruit.com/adafruit-magtag/circuitpython-libraries-2>

```
# SPDX-FileCopyrightText: 2020 Anne Barela for Adafruit Industries
#
# SPDX-License-Identifier: MIT

# SpaceX Launch Display, by Anne Barela November 2020
# MIT License - for Adafruit Industries LLC
# See https://github.com/r-spacex/SpaceX-API for API info

import time
import terminalio
from adafruit_magtag.magtag import MagTag

months = ["January", "February", "March", "April", "May", "June", "July",
          "August", "September", "October", "November", "December"]
USE_24HR_TIME = True
# in seconds, we can refresh about 100 times on a battery
TIME_BETWEEN_REFRESHES = 24 * 60 * 60 # once a day delay

# Set up data location and fields
DATA_SOURCE = "https://api.spacexdata.com/v4/launches/next"
DETAIL_LOCATION = ['details']
NAME_LOCATION = ['name']
DATE_LOCATION = ['date_local']
```

```

# These functions take the JSON data keys and does checks to determine
# how to display the data. They're used in the add_text blocks below

def mission_transform(val):
    if val == None:
        val = "Unavailable"
    return "Mission: " + val

def time_transform(val2):
    if val2 == None:
        return "When: Unavailable"
    month = int(val2[5:7])
    day = int(val2[8:10])
    hour = int(val2[11:13])
    min = int(val2[14:16])

    if USE_24HR_TIME:
        timestring = "%d:%02d" % (hour, min)
    elif hour > 12:
        timestring = "%d:%02d pm" % (hour-12, min)
    else:
        timestring = "%d:%02d am" % (hour, min)

    return "%s %d, at %s" % (months[month-1], day, timestring)

def details_transform(val3):
    if val3 == None or not len(val3):
        return "Details: To Be Determined"
    return "Details: " + val3[0:166] + "..."

# Set up the MagTag with the JSON data parameters
magtag = MagTag(
    url=DATA_SOURCE,
    json_path=(NAME_LOCATION, DATE_LOCATION, DETAIL_LOCATION)
)

magtag.add_text(
    text_font="/fonts/Lato-Bold-ltd-25.bdf",
    text_position=(10, 15),
    is_data=False
)

# Display heading text below with formatting above
magtag.set_text("Next SpaceX Launch")

# Formatting for the mission text
magtag.add_text(
    text_font="/fonts/Arial-Bold-12.pcf",
    text_position=(10, 38),
    text_transform=mission_transform
)

# Formatting for the launch time text
magtag.add_text(
    text_font="/fonts/Arial-12.bdf",
    text_position=(10, 60),
    text_transform=time_transform
)

# Formatting for the details text
magtag.add_text(
    text_font=terminalio.FONT,
    text_position=(10, 94),
    line_spacing=0.8,
    text_wrap=47,      # wrap text at this count
    text_transform=details_transform
)

try:
    # Have the MagTag connect to the internet

```



```

magtag.network.connect()
# This statement gets the JSON data and displays it automagically
value = magtag.fetch()
print("Response is", value)
except (ValueError, RuntimeError, ConnectionError, OSError) as 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 Runthrough

The `adafruit_magtag` library handles much of the internet data fetching and data display for guides like this.

Three `magtag.add_text` transform functions are defined. They each specify how three different JSON text fields are displayed. `time_transform` has two different ways to display the time. One in month, day, year, time format and one in year, month, day, time format. Select the one you want and comment out the other (or create your own, the data returned is formatted as follows:

2020-12-05T11:39:00-05:00

Error checking is used if fields are blank to note that status on the display.

The MagTag is then invoked with the JSON data.

Each `magtag.add_text` block defines the data to display. There are four, the first one for the title (`is_data = False` to set so the function knows this is not a JSON text field). Each text display as a different font in this example (although you can use the same font for multiple fields). The `text_position` varies for each to spread text over the display. On the last field, there is a `text_wrap` property to automatically wrap long lines of text, perfect for the long `description`.

Finally a 2 second delay allows the screen to refresh. Then the board deep sleeps for the specified delay in seconds (set to 24 hours by default).

Going Further

This example was kept fairly basic to illustrate how to do JSON data fetching and display the data on the MagTag.

A number of additions could be useful. Note, more elaborate display techniques might be beyond the capabilities of the `adafruit_magtag` library but could be implemented in more discrete code.

Low Power

It is likely one charge of the battery might last a month or so with this change and checking the data once a day. The deep sleep code is still beta, additional savings may be seen. If the display doesn't update after the launch, be sure to charge the battery for awhile.

Countdown to Launch

Use Adafruit IO to get the correct local time and make a display of the days, hours, and minutes until launch. See the following code examples for time manipulation (warning: more intermediate skills may be needed):

- [MagTag_CountdownCelebration \(https://adafru.it/P3d\)](https://adafru.it/P3d)
- [MagTag_SimpleClock \(https://adafru.it/P3e\)](https://adafru.it/P3e)

The NeoPixels could light up to let the user know a launch is imminent or a sound could be made. This will draw more power and shorten battery life. This isn't a worry if always on USB power.

Launchpad

The launchpad data was a text field in the previous version of the API. Often `details` contains the launchpad data too. Either have a function to convert the new launchpad number to a known value or parse it from the details. An example in `details` is "LC-39A, Kennedy Space Center".

Other Space APIs

I chose SpaceX as I live near Cape Canaveral and I like to watch the launches. There are other space launch data feeds that could be used with changes to the URL, fields used, and formatting. For example, see:

- RocketLaunch.Live - [Rocket Launch API \(https://adafru.it/P3f\)](https://adafru.it/P3f)