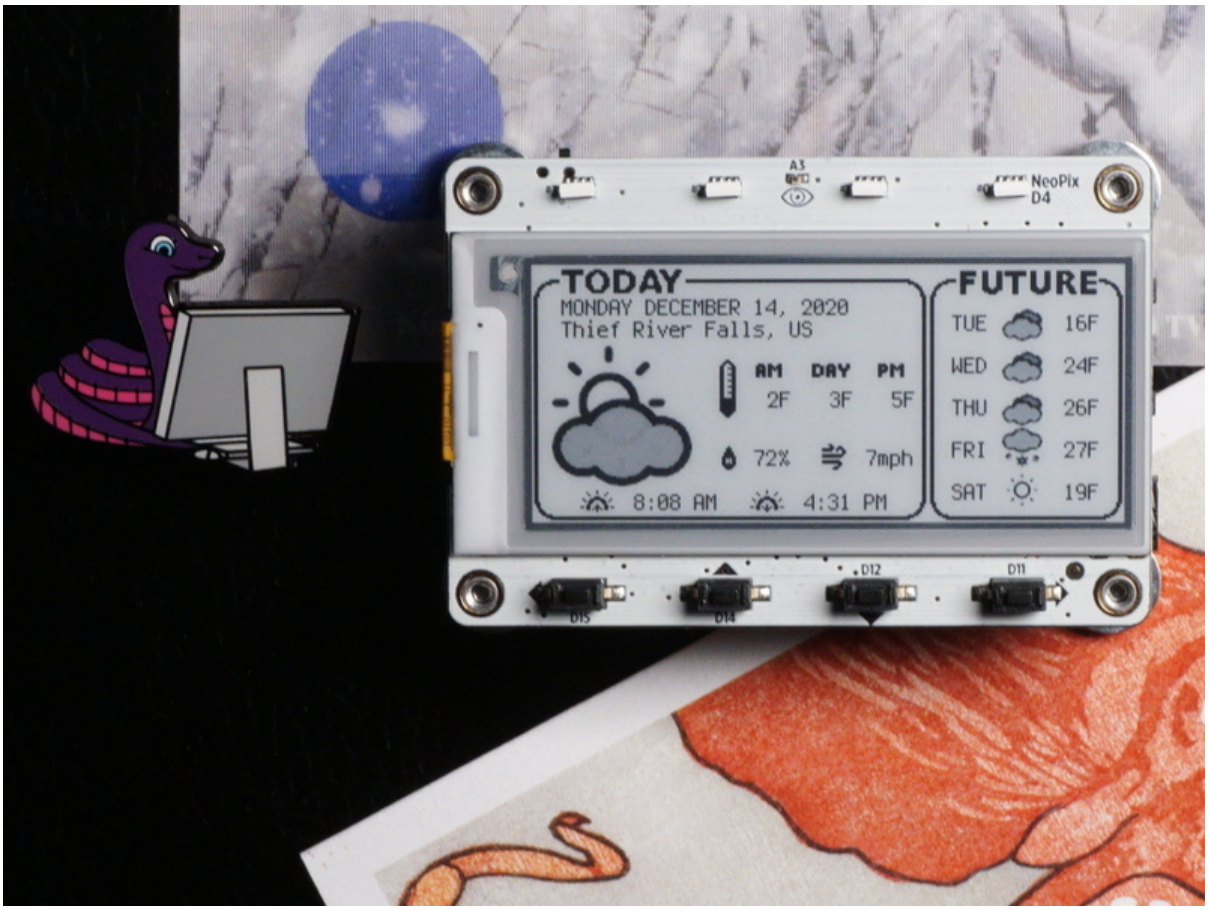
MagTag Daily Weather Forecast Display

Created by Carter Nelson



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

Last updated on 2024-12-02 11:25:41 AM EST

Table of Contents

Overview 3

- Weather API Services
- Parts

Install CircuitPython 5

- 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

- The settings.toml File
- IPv6 Networking

Project Code Install 16

- Weather Icons
- Installing Project Code

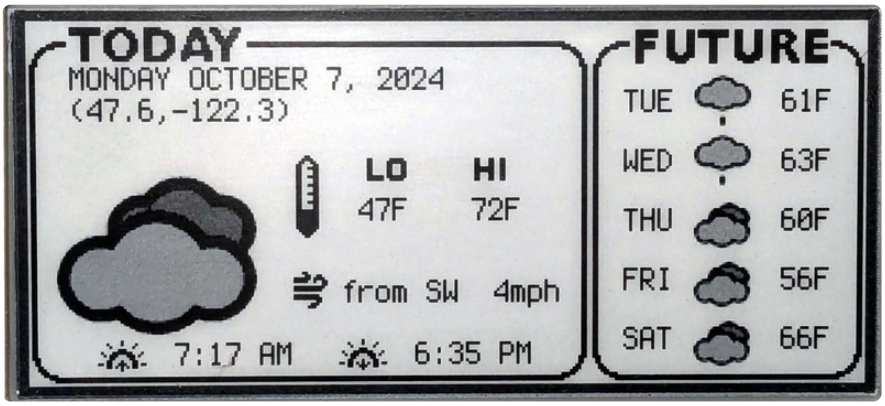
Configuration 21

- Set Location
- Set Timezone
- Set Units

OpenWeatherMap Project Code (OLD) 23

- Weather Icons
- Open Weather Maps API Key
- Set Location
- Celsius or Fahrenheit
- Secrets Summary
- Installing Project Code

Overview



Is the weather outside delightful? Or is it frightful? And what about in the coming days? In this guide we'll show you how to use an Adafruit MagTag (https://adafru.it/Pek) to create a weather display for today and future conditions.

Using the MagTag's WiFi capability, we connect to a weather API service and get weather information. We then parse that and display it up in a nice summary.

Deep sleep is also used to only run the update once a day around midnight, so we can run for many weeks on one charge.

Weather API Services

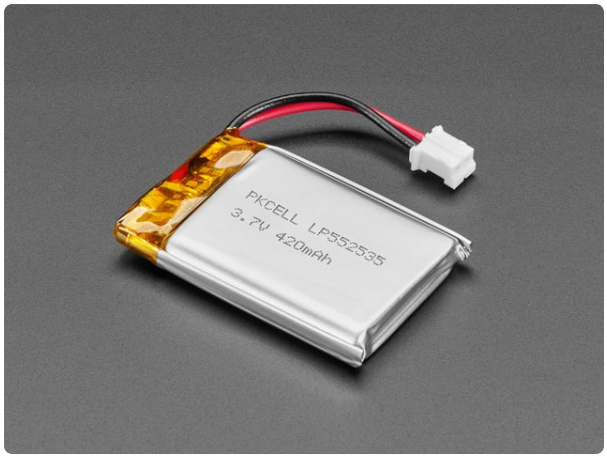
The original code for this guide used the free weather API service (https://adafru.it/Pen) provided by OpenWeatherMap (https://adafru.it/Pel). There have been several updates to the code to keep up with API changes. However, OpenWeather now requires credit card information to obtain an API key - even if only using the free API access. **As a result, this guide no longer uses the OpenWeatherMap API**. The old code is still provided for reference.

The current version of this guide is based on the weather API (https://adafru.it/18lF) provided by Open-Meteo (https://adafru.it/18Aa). This is a truly free weather API which currently does not even require an API key. It is also open source, with the code available in a GitHub repo (https://adafru.it/1a8i). This isn't a 100% drop in replacement in terms of weather forecast information provided. As a result, there are some minor differences relative to the original version.

Parts

Here are the parts you'll need for this project. You can get them individually:

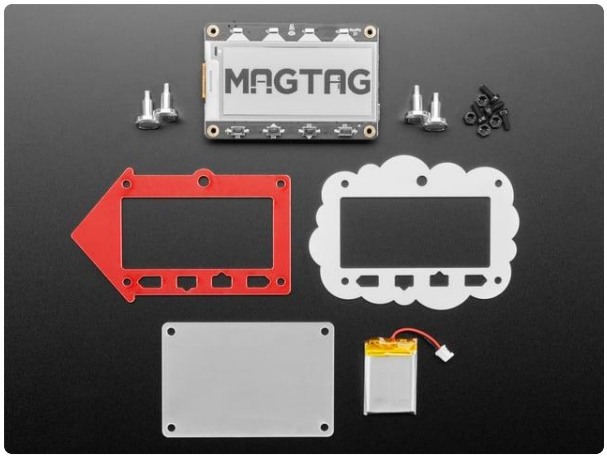
[Adafruit MagTag - 2.9" Grayscale E-Ink WiFi Display]
The Adafruit MagTag combines the new ESP32-S2 wireless module and a 2.9" grayscale E-Ink display to make a low-power IoT display that can show data on its screen even when power...
https://www.adafruit.com/product/4800



[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

[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

OR these items are also available together as a kit:



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

Install CircuitPython

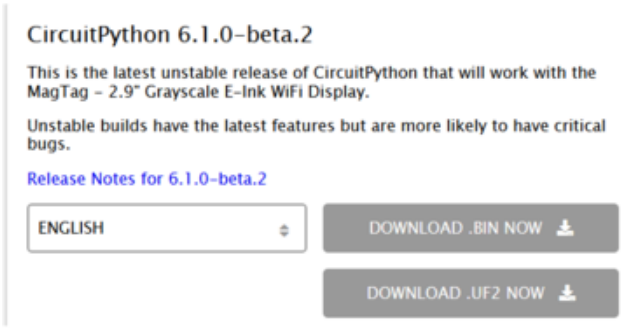
CircuitPython (https://adafru.it/tB7) is a derivative of MicroPython (https://adafru.it/BeZ) designed to simplify experimentation and education on low-cost microcontrollers. It makes it easier than ever to get prototyping by requiring no upfront desktop software downloads. Simply copy and edit files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

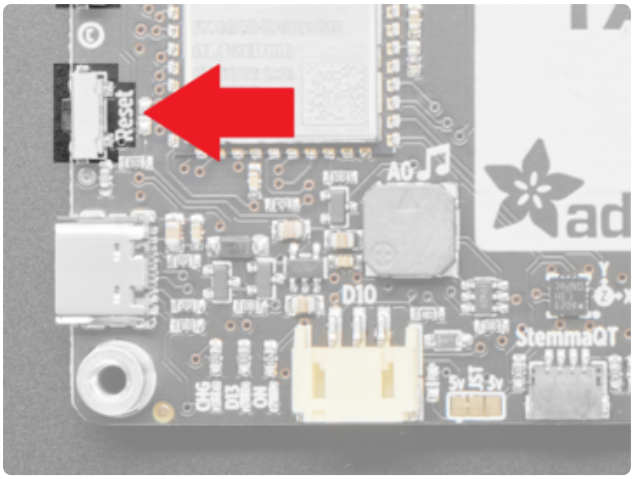
Option 1 - Load with UF2 Bootloader

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.**

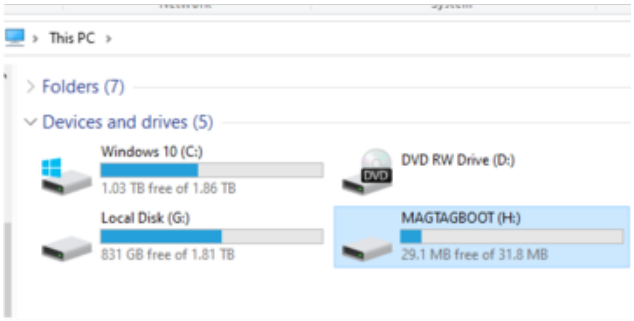
Still, try this first!

Try Launching UF2 Bootloader

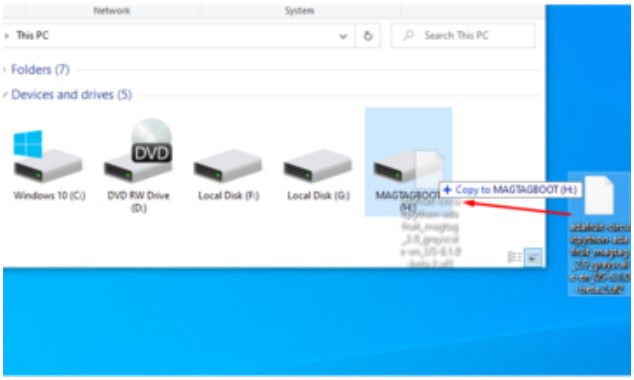
Loading CircuitPython by drag-n-drop UF2 bootloader is the easier way and we recommend it. If you have a MagTag where the front of the board is black, your MagTag came with UF2 already on it.



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.

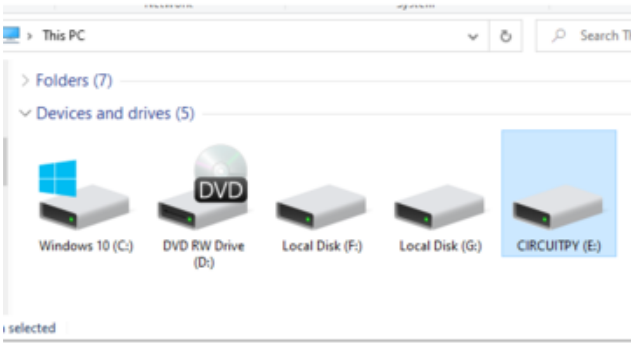


If the UF2 bootloader is installed, you will see a new disk drive appear called **MAGTAGBOOT**



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)

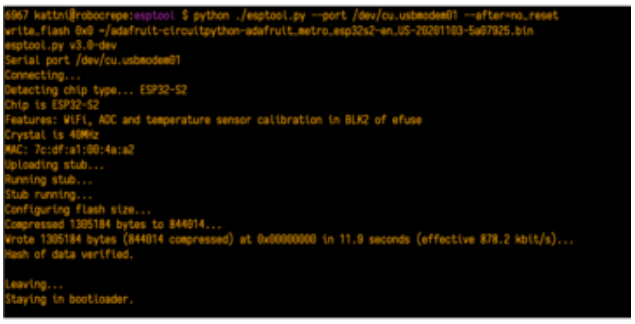


Your board should auto-reset into CircuitPython, or you may need to press reset. A **CIRCUITPY** drive will appear. You're done! Go to the next pages.

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.

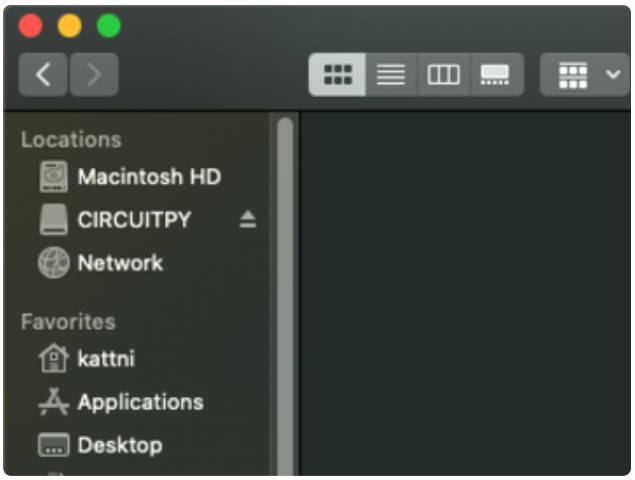
You can upload with **esptool** to the ROM (hardware) bootloader instead!



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!

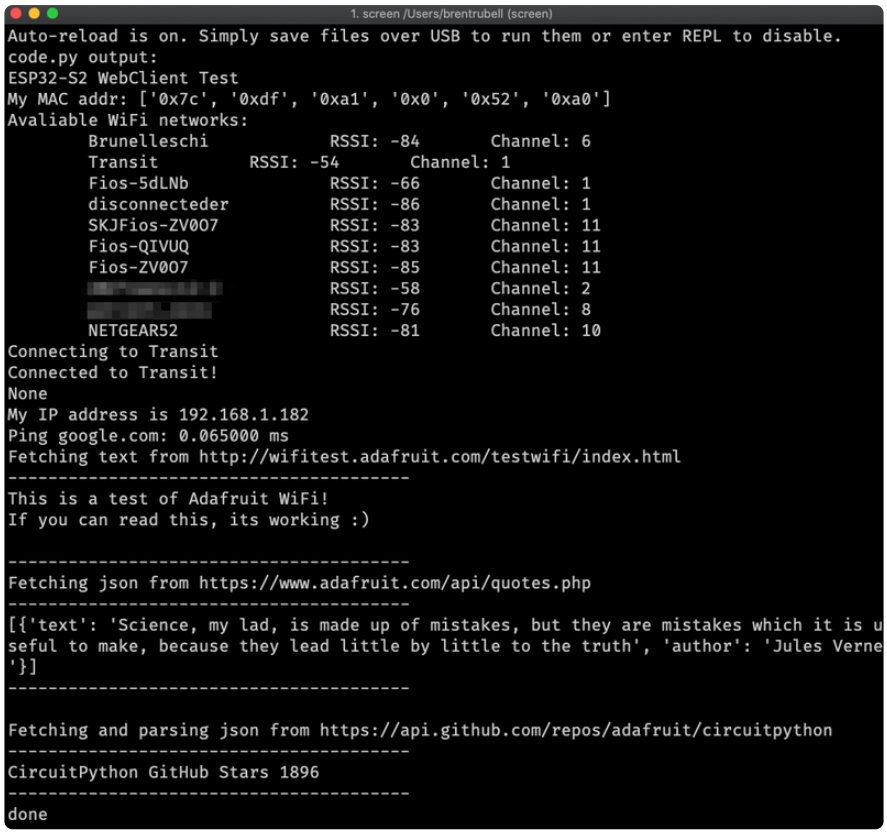
As you make projects you may need more tokens and keys, just add them one line at a time. See for example other tokens such as one for accessing GitHub or the Hackaday API. Other non-secret data like your timezone can also go here.

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')
```

Project Code Install

Here is the project code and other assets. Once installed, see the next guide section for how to configure the code for your location and other settings.

Weather Icons

The icons used in this project were inspired by this other EPD based weather project - eInk / ePaper Weather Station (https://adafru.it/FBM). They are the excellent set of Meteocons (https://adafru.it/Ptd) , which the author (https://adafru.it/Ptd) has made freely available. For this project, we converted them to gray scale, scaled them as needed, and converted to bitmap.

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 code listing below.

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_Weather/** and then click on the directory that matches the version of CircuitPython you're using and copy the contents of that directory to your **CIRCUITPY** drive.

```
# SPDX-FileCopyrightText: 2024 Carter Nelson for Adafruit Industries
#
# SPDX-License-Identifier: MIT
# pylint: disable=redefined-outer-name, eval-used, wrong-import-order,
unsubscriptable-object

import time
import terminalio
import displayio
import adafruit_imageload
from adafruit_display_text import label
from adafruit_magtag.magtag import MagTag

# --| USER CONFIG |-------------------------
LAT = 47.6  # latitude
LON = -122.3  # longitude
TMZ = "America/Los_Angeles"  # https://en.wikipedia.org/wiki/
List_of_tz_database_time_zones
METRIC = False  # set to True for metric units
CITY = None  # optional
# ----------------------------------------

# ----------------------------
# Define various assets
# ----------------------------
BACKGROUND_BMP = "/bmps/weather_bg.bmp"
ICONS_LARGE_FILE = "/bmps/weather_icons_70px.bmp"
ICONS_SMALL_FILE = "/bmps/weather_icons_20px.bmp"
DAYS = ("Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday",
"Sunday")
MONTHS = (
    "January",
    "February",
    "March",
    "April",
    "May",
    "June",
    "July",
    "August",
    "September",
    "October",
    "November",
    "December",
)

# Weather Code Information from https://open-meteo.com/en/docs
# Code  Description
# 0 Clear sky
# 1, 2, 3   Mainly clear, partly cloudy, and overcast
# 45, 48    Fog and depositing rime fog
# 51, 53, 55    Drizzle: Light, moderate, and dense intensity
# 56, 57    Freezing Drizzle: Light and dense intensity
```

```
# 61, 63, 65    Rain: Slight, moderate and heavy intensity
# 66, 67    Freezing Rain: Light and heavy intensity
# 71, 73, 75    Snow fall: Slight, moderate, and heavy intensity
# 77    Snow grains
# 80, 81, 82    Rain showers: Slight, moderate, and violent
# 85, 86    Snow showers slight and heavy
# 95 *  Thunderstorm: Slight or moderate
# 96, 99 *  Thunderstorm with slight and heavy hail

# Map the above WMO codes to index of icon in 3x3 spritesheet
WMO_CODE_TO_ICON = (
    (0,),  # 0 = sunny
    (1,),  # 1 = partly sunny/cloudy
    (2,),  # 2 = cloudy
    (3,),  # 3 = very cloudy
    (61, 63, 65),  # 4 = rain
    (51, 53, 55, 80, 81, 82),  # 5 = showers
    (95, 96, 99),  # 6 = storms
    (56, 57, 66, 67, 71, 73, 75, 77, 85, 86),  # 7 = snow
    (45, 48),  # 8 = fog and stuff
)

magtag = MagTag()

# ----------------------------
# Backgrounnd bitmap
# ----------------------------
magtag.graphics.set_background(BACKGROUND_BMP)

# ----------------------------
# Weather icons sprite sheet
# ----------------------------
icons_large_bmp, icons_large_pal = adafruit_imageload.load(ICONS_LARGE_FILE)
icons_small_bmp, icons_small_pal = adafruit_imageload.load(ICONS_SMALL_FILE)

# ////////////////////////////////////////////////////////////////////////
#  helper functions


def get_forecast():
    URL = f"https://api.open-meteo.com/v1/forecast?latitude={LAT}&longitude={LON}&"
    URL += "daily=weather_code,temperature_2m_max,temperature_2m_min"
    URL += ",sunrise,sunset,wind_speed_10m_max,wind_direction_10m_dominant"
    URL += "&timeformat=unixtime"
    URL += f"&timezone={TMZ}"
    resp = magtag.network.fetch(URL)
    return resp


def make_banner(x=0, y=0):
    """Make a single future forecast info banner group."""
    day_of_week = label.Label(terminalio.FONT, text="DAY", color=0x000000)
    day_of_week.anchor_point = (0, 0.5)
    day_of_week.anchored_position = (0, 10)

    icon = displayio.TileGrid(
        icons_small_bmp,
        pixel_shader=icons_small_pal,
        x=25,
        y=0,
        width=1,
        height=1,
        tile_width=20,
        tile_height=20,
    )

    day_temp = label.Label(terminalio.FONT, text="+100F", color=0x000000)
    day_temp.anchor_point = (0, 0.5)
    day_temp.anchored_position = (50, 10)
```

```
    group = displayio.Group(x=x, y=y)
    group.append(day_of_week)
    group.append(icon)
    group.append(day_temp)

    return group


def temperature_text(tempC):
    if METRIC:
        return "{:3.0f}C".format(tempC)
    else:
        return "{:3.0f}F".format(32.0 + 1.8 * tempC)


def wind_text(speedkmh, direction):
    wind_dir = "N"
    if direction < 337:
        wind_dir = "NW"
    if direction < 293:
        wind_dir = "W"
    if direction < 247:
        wind_dir = "SW"
    if direction < 202:
        wind_dir = "S"
    if direction < 157:
        wind_dir = "SE"
    if direction < 112:
        wind_dir = "E"
    if direction < 67:
        wind_dir = "NE"
    if direction < 22:
        wind_dir = "N"

    wtext = f"from {wind_dir} "

    if METRIC:
        wtext += "{:2.0f}kmh".format(speedkmh)
    else:
        wtext += "{:2.0f}mph".format(0.621371 * speedkmh)
    return wtext


def update_today(data):
    """Update today weather info."""
    # date text
    s = data["daily"]["time"][0] + data["utc_offset_seconds"]
    t = time.localtime(s)
    today_date.text = "{} {} {}, {}".format(
        DAYS[t.tm_wday].upper(), MONTHS[t.tm_mon - 1].upper(), t.tm_mday, t.tm_year
    )
    # weather icon
    w = data["daily"]["weather_code"][0]
    today_icon[0] = next(i for i, t in enumerate(WMO_CODE_TO_ICON) if w in t)
    # temperatures
    today_low_temp.text = temperature_text(data["daily"]["temperature_2m_min"][0])
    today_high_temp.text = temperature_text(data["daily"]["temperature_2m_max"][0])
    # wind
    s = data["daily"]["wind_speed_10m_max"][0]
    d = data["daily"]["wind_direction_10m_dominant"][0]
    today_wind.text = wind_text(s, d)
    # sunrise/set
    sr = time.localtime(data["daily"]["sunrise"][0] + data["utc_offset_seconds"])
    ss = time.localtime(data["daily"]["sunset"][0] + data["utc_offset_seconds"])
    today_sunrise.text = "{:2d}:{:02d} AM".format(sr.tm_hour, sr.tm_min)
    today_sunset.text = "{:2d}:{:02d} PM".format(ss.tm_hour - 12, ss.tm_min)
```

```
def update_future(data):
    """Update the future forecast info."""
    for i, banner in enumerate(future_banners):
        # day of week
        s = data["daily"]["time"][i + 1] + data["utc_offset_seconds"]
        t = time.localtime(s)
        banner[0].text = DAYS[t.tm_wday][:3].upper()
        # weather icon
        w = data["daily"]["weather_code"][i + 1]
        banner[1][0] = next(x for x, t in enumerate(WMO_CODE_TO_ICON) if w in t)
        # temperature
        t = data["daily"]["temperature_2m_max"][i + 1]
        banner[2].text = temperature_text(t)


def go_to_sleep(current_time_secs):
    """Enter deep sleep for time needed."""
    # work in units of seconds
    seconds_in_a_day = 24 * 60 * 60
    three_fifteen = (3 * 60 + 15) * 60
    # wake up 15 minutes after 3am
    seconds_to_sleep = (seconds_in_a_day - current_time_secs) + three_fifteen
    print(
        "Sleeping for {} hours, {} minutes".format(
            seconds_to_sleep // 3600, (seconds_to_sleep // 60) % 60
        )
    )
    magtag.exit_and_deep_sleep(seconds_to_sleep)


# ===========
# U I
# ===========
today_date = label.Label(terminalio.FONT, text="?" * 30, color=0x000000)
today_date.anchor_point = (0, 0)
today_date.anchored_position = (15, 14)

location_name = label.Label(terminalio.FONT, color=0x000000)
if CITY:
    location_name.text = f"{CITY[:16]} ({LAT:.1f},{LON:.1f})"
else:
    location_name.text = f"({LAT},{LON})"

location_name.anchor_point = (0, 0)
location_name.anchored_position = (15, 25)

today_icon = displayio.TileGrid(
    icons_large_bmp,
    pixel_shader=icons_small_pal,
    x=10,
    y=40,
    width=1,
    height=1,
    tile_width=70,
    tile_height=70,
)

today_low_temp = label.Label(terminalio.FONT, text="+100F", color=0x000000)
today_low_temp.anchor_point = (0.5, 0)
today_low_temp.anchored_position = (122, 60)

today_high_temp = label.Label(terminalio.FONT, text="+100F", color=0x000000)
today_high_temp.anchor_point = (0.5, 0)
today_high_temp.anchored_position = (162, 60)

today_wind = label.Label(terminalio.FONT, text="99m/s", color=0x000000)
today_wind.anchor_point = (0, 0.5)
today_wind.anchored_position = (110, 95)
```

```
today_sunrise = label.Label(terminalio.FONT, text="12:12 PM", color=0x000000)
today_sunrise.anchor_point = (0, 0.5)
today_sunrise.anchored_position = (45, 117)

today_sunset = label.Label(terminalio.FONT, text="12:12 PM", color=0x000000)
today_sunset.anchor_point = (0, 0.5)
today_sunset.anchored_position = (130, 117)

today_banner = displayio.Group()
today_banner.append(today_date)
today_banner.append(location_name)
today_banner.append(today_icon)
today_banner.append(today_low_temp)
today_banner.append(today_high_temp)
today_banner.append(today_wind)
today_banner.append(today_sunrise)
today_banner.append(today_sunset)

future_banners = [
    make_banner(x=210, y=18),
    make_banner(x=210, y=39),
    make_banner(x=210, y=60),
    make_banner(x=210, y=81),
    make_banner(x=210, y=102),
]

magtag.splash.append(today_banner)
for future_banner in future_banners:
    magtag.splash.append(future_banner)

# ===========
#  M A I N
# ===========
print("Fetching forecast...")
resp_data = get_forecast()
forecast_data = resp_data.json()

print("Updating...")
update_today(forecast_data)
update_future(forecast_data)

print("Refreshing...")
time.sleep(magtag.display.time_to_refresh + 1)
magtag.display.refresh()
time.sleep(magtag.display.time_to_refresh + 1)

print("Sleeping...")
h, m, s = (int(t) for t in resp_data.headers["date"].split(" ")[4].split(":"))
current_time_secs = (h * 3600) + (m * 60) + (s) +
forecast_data["utc_offset_seconds"]
go_to_sleep(current_time_secs)
#  entire code will run again after deep sleep cycle
#  similar to hitting the reset button
```

Configuration

Once the project code and assets have been installed to the Magtag, edit the **code.py** file to make configuration changes as described below.

All changes are made by editing this small code block found at the top of the code listing:

```
# --| USER CONFIG |--------------------------
LAT = 47.6                  # latitude
LON = -122.3                # longitude
TMZ = "America/Los_Angeles" # https://en.wikipedia.org/wiki/
List_of_tz_database_time_zones
METRIC = False              # set to True for metric units
# ------------------------------------------
```

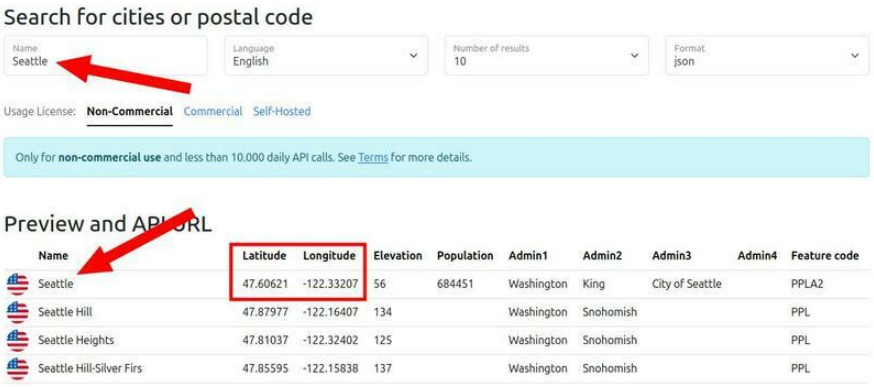
Set Location

The Open-Meteo API currently only works with location specified by latitude and longitude. There's a simple geocoding API accessible here:

Geocoding API

https://adafru.it/1a8p

that can be used to find the latitude and longitude for your location. Search for your location and then find the appropriate latitude and longitude in the results table.



Update these lines of code with the latitude (LAT) and longitude (LON) values:

```
LAT = 47.6                  # latitude
LON = -122.3                # longitude
```

Set Timezone

The timezone for the location also needs to be specified. Use the link below to find the appropriate name to use:

List of Time Zones

https://adafru.it/EgK

Use the text from the **TZ indentifier** column in the timezone list table and update this line of code:

```
TMZ = "America/Los_Angeles" # https://en.wikipedia.org/wiki/
List_of_tz_database_time_zones
```

NOTE: The TMZ value must be set as a string, so don't forget the surrounding quotation marks.

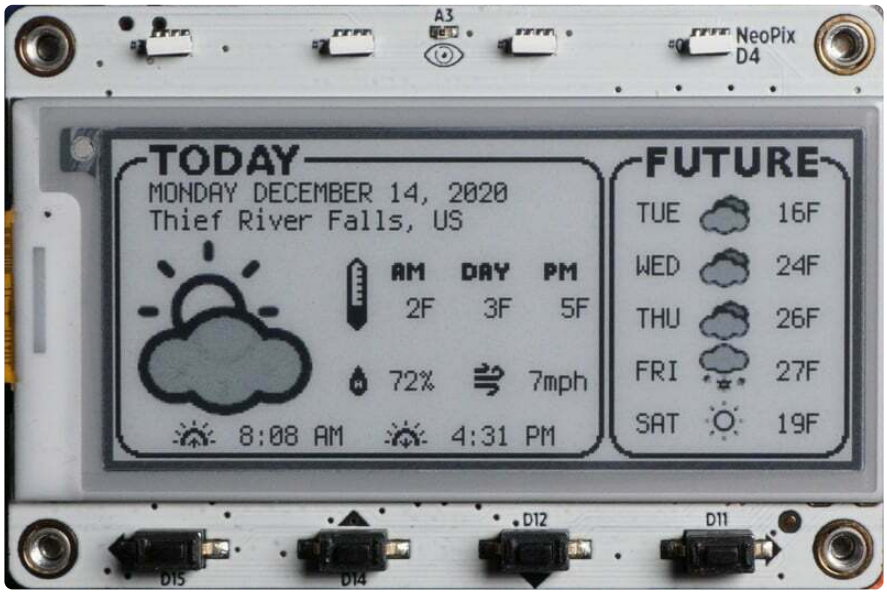
Set Units

The units used for temperature and wind speed can be set to either imperial (deg F, mph) or metric (deg C, kmh). By default, imperial is used. To switch to metric, set the **METRIC** option to **True**:

```
METRIC = True              # set to True for metric units
```

OpenWeatherMap Project Code (OLD)

This code is no longer supported and only here for reference.



Here is the project code along with some additional setup needed.

As a result of changes in OpenWeather's API and pricing, there are two versions of the code. The two versions of the code are provided below in separate guide sections.

- **One Call API** - This is "free" (up to 1000 API calls/day), but requires providing credit card information when setting up.
- **Forecast API** - This is free without providing credit card info.

The code was originally written using the One Call API, before the credit card requirement was added. The second version was written after that requirement was added using the different, but still free, Forecast API. It works mostly the same as the original One Call API version.

Weather Icons

The icons used in this project were inspired by this other EPD based weather project - eInk / ePaper Weather Station (https://adafru.it/FBM). They are the excellent set of Meteocons (https://adafru.it/Ptd) , which the author (https://adafru.it/Ptd) has made freely available. For this project, we converted them to gray scale, scaled them as needed, and converted to bitmap.

Open Weather Maps API Key

We'll be using OpenWeatherMap.org (https://adafru.it/KeU) to retrieve the weather info through its API (https://adafru.it/Pen). In order to do so, you'll need to register for an account and get your API key.

Go to this link (https://adafru.it/EeH) and register for a free account. Once registered, you'll get an email containing your API key, also known as the "openweather token".

Copy and paste this key into your **secrets.py** file that is on the root level of your **CIRCUITPY** drive. Name the entry `openweather_token`, ex:

```
'openweather_token' : 'my_openweather_token',
```

It'll look something like this

```
'openweather_token' : 'b218dde228d7be12f16bf4640208b9f5',
```

this is not a valid token, you have to get your own!

It may take some time for your Open Weather Maps API key to become active - tens of minutes or even hours.

Set Location

We will also use **secrets.py** to set your location. To do so, add new entry called `openweather_location` and specify **either** your city name and country code:

```
'openweather_location' : 'Thief River Falls, US',
```

or by using a two tuple of latitude and longitude:

```
'openweather_location' : (47.6062, -122.3321),
```

Note that negative longitude values are use for West.

You can verify your town or city is valid by visiting https://openweathermap.org/find (https://adafru.it/Peo) and typing it into the box, if you get a weather report, then that's a known city! However, if you run into a situation where the city name is ambiguous (multiple same name cities), then use the latitude/longitude value to specify location.

Celsius or Fahrenheit

The units used for the temperature readings can be set to either Celsius or Fahrenheit. The units are set to Fahrenheit by default. To change to Celsius, look for this line of code:

```
METRIC = False  # set to True for metric units
```

and change it to:

```
METRIC = True  # set to True for metric units
```

In addition to changing temperature readings to Celsius, this will also change the wind readings to meters per second.

Secrets Summary

Your resulting **secrets.py** file should looking something like the example below, but with your specific values. It's OK if there are more entries - for other services, etc. But you need at least the ones shown here.

```
secrets = {
    'ssid' : 'myssid',
    'password' : 'mypassword',
    'openweather_token' : 'my_openweather_token',
    'openweather_location' : 'Thief River Falls, US',
}
```

Don't forget to end each line with a comma.

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 code listing below.

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_Weather/** 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.

Note that there are two versions of the code located in subfolders:

- **onecall** - Requires non-free account/API key.
- **forecast** - Works with free account/API key.

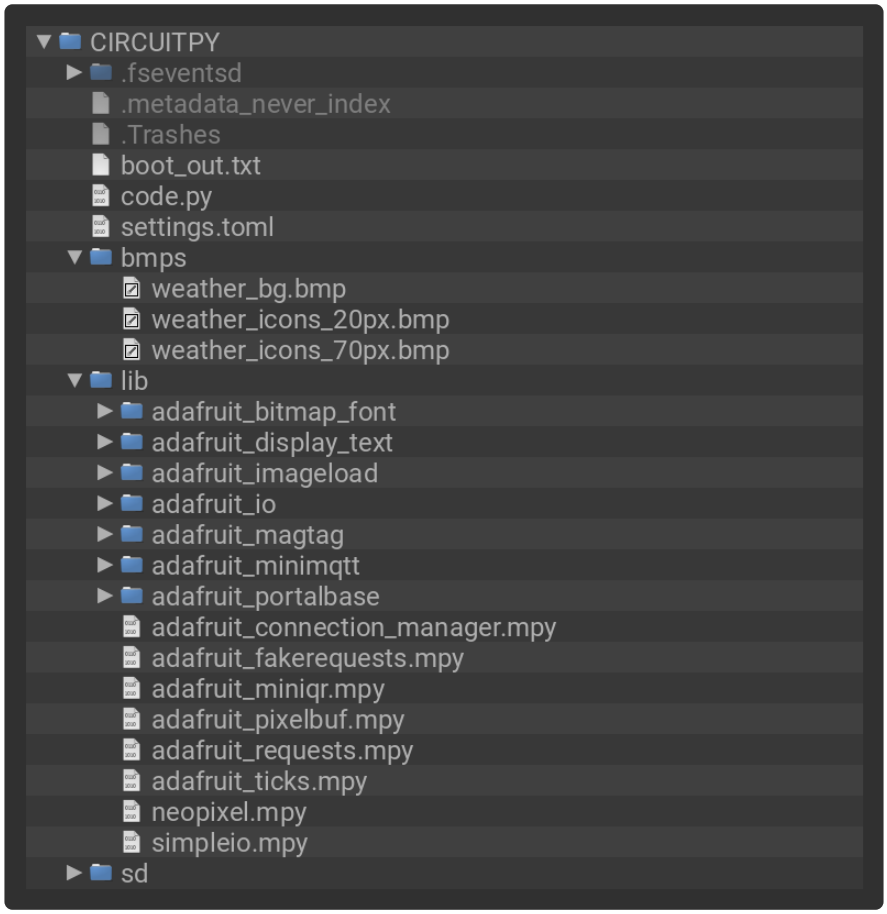
See the code specific sections below for more detail. Only one version should be used at a time.

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

One Call API (non free)

This is the original version of the code, written when access to the One Call API was available with a free API key. The One Call API now requires the API key be associated with an Openweathermap (https://adafru.it/11fH) account that's been set up for payments.

Older account/API keys **may** still work. For any new accounts/API keys that want a free version, the Forecast API version of the code below will likely be needed.




```
# SPDX-FileCopyrightText: 2020 Carter Nelson for Adafruit Industries
#
# SPDX-License-Identifier: MIT
# pylint: disable=redefined-outer-name, eval-used, wrong-import-order

import time
import terminalio
import displayio
import adafruit_imageload
from adafruit_display_text import label
from adafruit_magtag.magtag import MagTag
from secrets import secrets

# --| USER CONFIG |-------------------------
METRIC = False  # set to True for metric units
# ------------------------------------------

# ----------------------------
# Define various assets
# ----------------------------
```

```
BACKGROUND_BMP = "/bmps/weather_bg.bmp"
ICONS_LARGE_FILE = "/bmps/weather_icons_70px.bmp"
ICONS_SMALL_FILE = "/bmps/weather_icons_20px.bmp"
ICON_MAP = ("01", "02", "03", "04", "09", "10", "11", "13", "50")
DAYS = ("Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday",
"Sunday")
MONTHS = (
    "January",
    "February",
    "March",
    "April",
    "May",
    "June",
    "July",
    "August",
    "September",
    "October",
    "November",
    "December",
)
magtag = MagTag()

# ---------------------------
# Backgrounnd bitmap
# ---------------------------
magtag.graphics.set_background(BACKGROUND_BMP)

# ---------------------------
# Weather icons sprite sheet
# ---------------------------
icons_large_bmp, icons_large_pal = adafruit_imageload.load(ICONS_LARGE_FILE)
icons_small_bmp, icons_small_pal = adafruit_imageload.load(ICONS_SMALL_FILE)

# ////////////////////////////////////////////////////////////////////////


def get_data_source_url(api="onecall", location=None):
    """Build and return the URL for the OpenWeather API."""
    if api.upper() == "GEO":
        URL = "https://api.openweathermap.org/geo/1.0/direct?q="
        URL += location
    elif api.upper() == "GEOREV":
        URL = "https://api.openweathermap.org/geo/1.0/reverse?limit=1"
        URL += "&lat={}".format(location[0])
        URL += "&lon={}".format(location[1])
    elif api.upper() == "ONECALL":
        URL = "https://api.openweathermap.org/data/2.5/onecall?
exclude=minutely,hourly,alerts"
        URL += "&lat={}".format(location[0])
        URL += "&lon={}".format(location[1])
    else:
        raise ValueError("Unknown API type: " + api)
    return URL + "&appid=" + secrets["openweather_token"]


def get_latlon(city_name):
    """Use the Geolocation API to determine lat/lon for given city."""
    magtag.url = get_data_source_url(api="geo", location=city_name)
    raw_data = eval(magtag.fetch())[0]
    return raw_data["lat"], raw_data["lon"]


def get_city(latlon_location):
    """Use the Geolocation API to determine city for given lat/lon."""
    magtag.url = get_data_source_url(api="georev", location=latlon_location)
    raw_data = eval(magtag.fetch())[0]
    return raw_data["name"] + ", " + raw_data["country"]
```

```
def get_forecast(location):
    """Use OneCall API to fetch forecast and timezone data."""
    resp = magtag.network.fetch(get_data_source_url(api="onecall",
location=location))
    json_data = resp.json()
    return json_data["daily"], json_data["current"]["dt"],
json_data["timezone_offset"]


def make_banner(x=0, y=0):
    """Make a single future forecast info banner group."""
    day_of_week = label.Label(terminalio.FONT, text="DAY", color=0x000000)
    day_of_week.anchor_point = (0, 0.5)
    day_of_week.anchored_position = (0, 10)

    icon = displayio.TileGrid(
        icons_small_bmp,
        pixel_shader=icons_small_pal,
        x=25,
        y=0,
        width=1,
        height=1,
        tile_width=20,
        tile_height=20,
    )

    day_temp = label.Label(terminalio.FONT, text="+100F", color=0x000000)
    day_temp.anchor_point = (0, 0.5)
    day_temp.anchored_position = (50, 10)

    group = displayio.Group(x=x, y=y)
    group.append(day_of_week)
    group.append(icon)
    group.append(day_temp)

    return group


def temperature_text(tempK):
    if METRIC:
        return "{:3.0f}C".format(tempK - 273.15)
    else:
        return "{:3.0f}F".format(32.0 + 1.8 * (tempK - 273.15))


def wind_text(speedms):
    if METRIC:
        return "{:3.0f}m/s".format(speedms)
    else:
        return "{:3.0f}mph".format(2.23694 * speedms)


def update_banner(banner, data):
    """Update supplied forecast banner with supplied data."""
    banner[0].text = DAYS[time.localtime(data["dt"]).tm_wday][:3].upper()
    banner[1][0] = ICON_MAP.index(data["weather"][0]["icon"][:2])
    banner[2].text = temperature_text(data["temp"]["day"])


def update_today(data, tz_offset=0):
    """Update today info banner."""
    date = time.localtime(data["dt"])
    sunrise = time.localtime(data["sunrise"] + tz_offset)
    sunset = time.localtime(data["sunset"] + tz_offset)

    today_date.text = "{} {} {}, {}".format(
        DAYS[date.tm_wday].upper(),
        MONTHS[date.tm_mon - 1].upper(),
        date.tm_mday,
```

```
        date.tm_year,
    )
    today_icon[0] = ICON_MAP.index(data["weather"][0]["icon"][:2])
    today_morn_temp.text = temperature_text(data["temp"]["morn"])
    today_day_temp.text = temperature_text(data["temp"]["day"])
    today_night_temp.text = temperature_text(data["temp"]["night"])
    today_humidity.text = "{:3d}%".format(data["humidity"])
    today_wind.text = wind_text(data["wind_speed"])
    today_sunrise.text = "{:2d}:{:02d} AM".format(sunrise.tm_hour, sunrise.tm_min)
    today_sunset.text = "{:2d}:{:02d} PM".format(sunset.tm_hour - 12, sunset.tm_min)


def go_to_sleep(current_time):
    """Enter deep sleep for time needed."""
    # compute current time offset in seconds
    hour, minutes, seconds = time.localtime(current_time)[3:6]
    seconds_since_midnight = 60 * (hour * 60 + minutes) + seconds
    three_fifteen = (3 * 60 + 15) * 60
    # wake up 15 minutes after 3am
    seconds_to_sleep = (24 * 60 * 60 - seconds_since_midnight) + three_fifteen
    print(
        "Sleeping for {} hours, {} minutes".format(
            seconds_to_sleep // 3600, (seconds_to_sleep // 60) % 60
        )
    )
    magtag.exit_and_deep_sleep(seconds_to_sleep)


# ===========
# Location
# ===========
if isinstance(secrets["openweather_location"], str):
    # Get lat/lon using city name
    city = secrets["openweather_location"]
    print("Getting lat/lon for city:", city)
    latlon = get_latlon(city)
elif isinstance(secrets["openweather_location"], tuple):
    # Get city name using lat/lon
    latlon = secrets["openweather_location"]
    print("Getting city name for lat/lon:", latlon)
    city = get_city(latlon)
else:
    raise ValueError("Unknown location:", secrets["openweather_location"])

print("City =", city)
print("Lat/Lon = ", latlon)

# ===========
# U I
# ===========
today_date = label.Label(terminalio.FONT, text="?" * 30, color=0x000000)
today_date.anchor_point = (0, 0)
today_date.anchored_position = (15, 13)

city_name = label.Label(terminalio.FONT, text=city, color=0x000000)
city_name.anchor_point = (0, 0)
city_name.anchored_position = (15, 24)

today_icon = displayio.TileGrid(
    icons_large_bmp,
    pixel_shader=icons_small_pal,
    x=10,
    y=40,
    width=1,
    height=1,
    tile_width=70,
    tile_height=70,
)
```

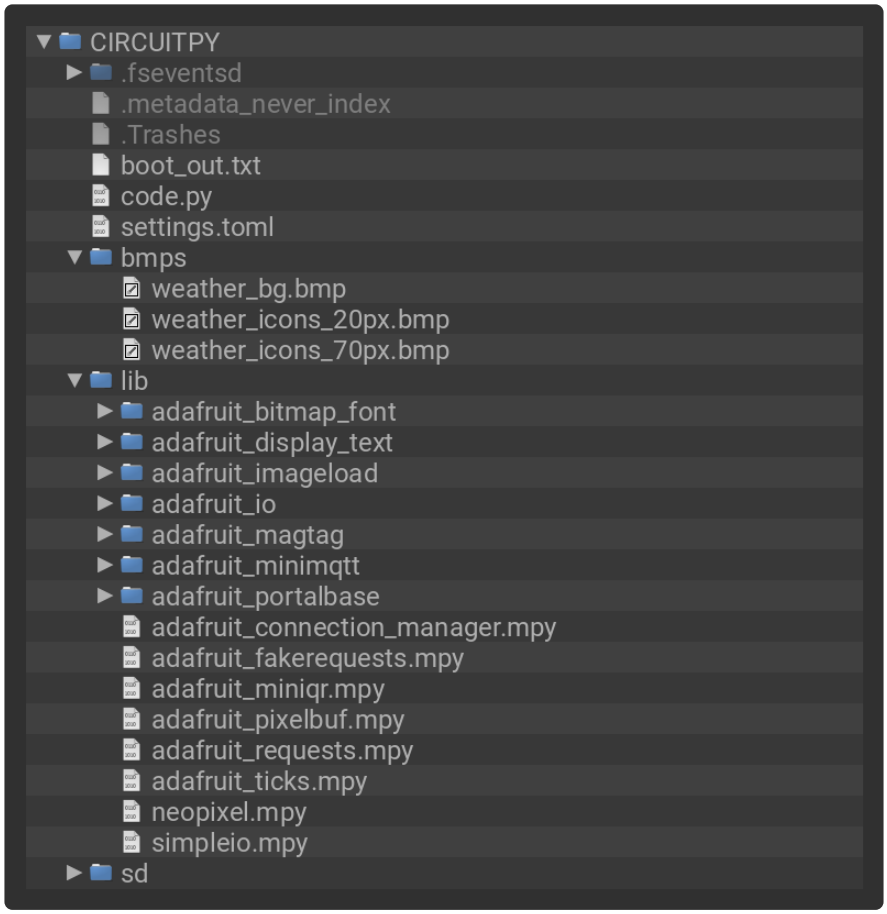
```
today_morn_temp = label.Label(terminalio.FONT, text="+100F", color=0x000000)
today_morn_temp.anchor_point = (0.5, 0)
today_morn_temp.anchored_position = (118, 59)

today_day_temp = label.Label(terminalio.FONT, text="+100F", color=0x000000)
today_day_temp.anchor_point = (0.5, 0)
today_day_temp.anchored_position = (149, 59)

today_night_temp = label.Label(terminalio.FONT, text="+100F", color=0x000000)
today_night_temp.anchor_point = (0.5, 0)
today_night_temp.anchored_position = (180, 59)

today_humidity = label.Label(terminalio.FONT, text="100%", color=0x000000)
today_humidity.anchor_point = (0, 0.5)
today_humidity.anchored_position = (105, 95)

today_wind = label.Label(terminalio.FONT, text="99m/s", color=0x000000)
today_wind.anchor_point = (0, 0.5)
today_wind.anchored_position = (155, 95)

today_sunrise = label.Label(terminalio.FONT, text="12:12 PM", color=0x000000)
today_sunrise.anchor_point = (0, 0.5)
today_sunrise.anchored_position = (45, 117)

today_sunset = label.Label(terminalio.FONT, text="12:12 PM", color=0x000000)
today_sunset.anchor_point = (0, 0.5)
today_sunset.anchored_position = (130, 117)

today_banner = displayio.Group()
today_banner.append(today_date)
today_banner.append(city_name)
today_banner.append(today_icon)
today_banner.append(today_morn_temp)
today_banner.append(today_day_temp)
today_banner.append(today_night_temp)
today_banner.append(today_humidity)
today_banner.append(today_wind)
today_banner.append(today_sunrise)
today_banner.append(today_sunset)

future_banners = [
    make_banner(x=210, y=18),
    make_banner(x=210, y=39),
    make_banner(x=210, y=60),
    make_banner(x=210, y=81),
    make_banner(x=210, y=102),
]

magtag.splash.append(today_banner)
for future_banner in future_banners:
    magtag.splash.append(future_banner)

# ===========
#  M A I N
# ===========
print("Fetching forecast...")
forecast_data, utc_time, local_tz_offset = get_forecast(latlon)

print("Updating...")
update_today(forecast_data[0], local_tz_offset)
for day, forecast in enumerate(forecast_data[1:6]):
    update_banner(future_banners[day], forecast)

print("Refreshing...")
time.sleep(magtag.display.time_to_refresh + 1)
magtag.display.refresh()
time.sleep(magtag.display.time_to_refresh + 1)

print("Sleeping...")
```

```
go_to_sleep(utc_time + local_tz_offset)
#  entire code will run again after deep sleep cycle
#  similar to hitting the reset button
```

Forecast API (free)

This is a newer version of the code that uses the still free 5 Day / 3 Hour Forecast API. It attempts to provide the same informational display as the original version of the code. However, since there are some differences in the nature of the data returned by the two API's, there are some caveats.

- The forecast is for only four days, not five. That's because one of the "5 days" is the current day.
- Weather conditions forecasted for noon local time are used as "daily forecast". There is no "daily" summary forecast available as there is with the One Call API.




```
# SPDX-FileCopyrightText: 2020 Carter Nelson for Adafruit Industries
#
# SPDX-License-Identifier: MIT
# pylint: disable=redefined-outer-name, eval-used, wrong-import-order

import time
import terminalio
import displayio
import adafruit_imageload
from adafruit_display_text import label
from adafruit_magtag.magtag import MagTag
```

```
from secrets import secrets

# --| USER CONFIG |-------------------------
METRIC = False  # set to True for metric units
# ----------------------------------------

# ----------------------------
# Define various assets
# ----------------------------
BACKGROUND_BMP = "/bmps/weather_bg.bmp"
ICONS_LARGE_FILE = "/bmps/weather_icons_70px.bmp"
ICONS_SMALL_FILE = "/bmps/weather_icons_20px.bmp"
ICON_MAP = ("01", "02", "03", "04", "09", "10", "11", "13", "50")
DAYS = ("Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday",
"Sunday")
MONTHS = (
    "January",
    "February",
    "March",
    "April",
    "May",
    "June",
    "July",
    "August",
    "September",
    "October",
    "November",
    "December",
)
magtag = MagTag()

# ----------------------------
# Backgrounnd bitmap
# ----------------------------
magtag.graphics.set_background(BACKGROUND_BMP)

# ----------------------------
# Weather icons sprite sheet
# ----------------------------
icons_large_bmp, icons_large_pal = adafruit_imageload.load(ICONS_LARGE_FILE)
icons_small_bmp, icons_small_pal = adafruit_imageload.load(ICONS_SMALL_FILE)

# ////////////////////////////////////////////////////////////////////////


def get_data_source_url(api="forecast", location=None):
    """Build and return the URL for the OpenWeather API."""
    if location is None:
        raise ValueError("Must specify location.")
    if api.upper() == "GEO":
        URL = "https://api.openweathermap.org/geo/1.0/direct?q="
        URL += location
    elif api.upper() == "GEOREV":
        URL = "https://api.openweathermap.org/geo/1.0/reverse?limit=1"
        URL += "&lat={}".format(location[0])
        URL += "&lon={}".format(location[1])
    elif api.upper() == "FORECAST":
        URL = "https://api.openweathermap.org/data/2.5/forecast?"
        URL += "&lat={}".format(location[0])
        URL += "&lon={}".format(location[1])
    elif api.upper() == "CURRENT":
        URL = "https://api.openweathermap.org/data/2.5/weather?"
        URL += "&lat={}".format(location[0])
        URL += "&lon={}".format(location[1])
    else:
        raise ValueError("Unknown API type: " + api)
    return URL + "&appid=" + secrets["openweather_token"]
```

```
def get_latlon(city_name):
    """Use the Geolocation API to determine lat/lon for given city."""
    magtag.url = get_data_source_url(api="geo", location=city_name)
    raw_data = eval(magtag.fetch())[0]
    return raw_data["lat"], raw_data["lon"]


def get_city(latlon_location):
    """Use the Geolocation API to determine city for given lat/lon."""
    magtag.url = get_data_source_url(api="georev", location=latlon_location)
    raw_data = eval(magtag.fetch())[0]
    return raw_data["name"] + ", " + raw_data["country"]


def get_approx_time(latlon_location):
    magtag.url = get_data_source_url(api="current", location=latlon_location)
    raw_data = eval(magtag.fetch())
    return raw_data["dt"] + raw_data["timezone"]


def get_forecast(location):
    """Use Forecast API to fetch weather data and return a "daily" forecast.
    NOTE: The data query is assumed to have been done at ~2AM local time so that
    the forecast data results are (also approx):
                0  3AM
                1  6AM
                2  9AM
                3 12PM
                4  3PM
                5  6PM
                6  9PM
                7 12AM
                :   :
                (etc.)
    """
    resp = magtag.network.fetch(get_data_source_url(api="forecast",
location=location))
    json_data = resp.json()
    # build "daily" forecast data (similar to Onecall API)
    # it is assumed the first entry in the list is for ~3AM local
    # and the list is 40 entries at 3 hours intervals
    #     index  local_time
    #       0        3AM
    #       1        6AM
    #       2        9AM
    #       3       12PM
    #       4        3PM
    #       5        6PM
    #       6        9PM
    #       7       12AM
    #      etc.     etc.
    if json_data["cnt"] != 40:
        raise RuntimeError("Unexpected forecast response length.")
    timezone_offset = json_data["city"]["timezone"]
    daily_data = []
    # use the 12PM values from each day, access via direct indexing (slice)
    for data in json_data["list"][3::8]:
        daily_data.append(
            {
                "dt": data["dt"] + timezone_offset,
                "weather": data["weather"],
                "temp": {"day": data["main"]["temp"]},
            }
        )
    # add extra data for day 0 (current day)
    daily_data[0]["sunrise"] = json_data["city"]["sunrise"] + timezone_offset
    daily_data[0]["sunset"] = json_data["city"]["sunset"] + timezone_offset
    daily_data[0]["temp"] = {
        "morn": json_data["list"][2]["main"]["temp"],
```

```
        "day": json_data["list"][4]["main"]["temp"],
        "night": json_data["list"][6]["main"]["temp"],
    }
    daily_data[0]["humidity"] = json_data["list"][3]["main"]["humidity"]
    daily_data[0]["wind_speed"] = json_data["list"][3]["wind"]["speed"]

    return daily_data


def make_banner(x=0, y=0):
    """Make a single future forecast info banner group."""
    day_of_week = label.Label(terminalio.FONT, text="DAY", color=0x000000)
    day_of_week.anchor_point = (0, 0.5)
    day_of_week.anchored_position = (0, 10)

    icon = displayio.TileGrid(
        icons_small_bmp,
        pixel_shader=icons_small_pal,
        x=25,
        y=0,
        width=1,
        height=1,
        tile_width=20,
        tile_height=20,
    )

    day_temp = label.Label(terminalio.FONT, text="+100F", color=0x000000)
    day_temp.anchor_point = (0, 0.5)
    day_temp.anchored_position = (50, 10)

    group = displayio.Group(x=x, y=y)
    group.append(day_of_week)
    group.append(icon)
    group.append(day_temp)

    return group


def temperature_text(tempK):
    if METRIC:
        return "{:3.0f}C".format(tempK - 273.15)
    else:
        return "{:3.0f}F".format(32.0 + 1.8 * (tempK - 273.15))


def wind_text(speedms):
    if METRIC:
        return "{:3.0f}m/s".format(speedms)
    else:
        return "{:3.0f}mph".format(2.23694 * speedms)


def update_banner(banner, data):
    """Update supplied forecast banner with supplied data."""
    banner[0].text = DAYS[time.localtime(data["dt"]).tm_wday][:3].upper()
    banner[1][0] = ICON_MAP.index(data["weather"][0]["icon"][:2])
    banner[2].text = temperature_text(data["temp"]["day"])


def update_today(data):
    """Update today info banner."""
    date = time.localtime(data["dt"])
    sunrise = time.localtime(data["sunrise"])
    sunset = time.localtime(data["sunset"])

    today_date.text = "{} {} {}, {}".format(
        DAYS[date.tm_wday].upper(),
        MONTHS[date.tm_mon - 1].upper(),
        date.tm_mday,
```

```
        date.tm_year,
    )
    today_icon[0] = ICON_MAP.index(data["weather"][0]["icon"][:2])
    today_morn_temp.text = temperature_text(data["temp"]["morn"])
    today_day_temp.text = temperature_text(data["temp"]["day"])
    today_night_temp.text = temperature_text(data["temp"]["night"])
    today_humidity.text = "{:3d}%".format(data["humidity"])
    today_wind.text = wind_text(data["wind_speed"])
    today_sunrise.text = "{:2d}:{:02d} AM".format(sunrise.tm_hour, sunrise.tm_min)
    today_sunset.text = "{:2d}:{:02d} PM".format(sunset.tm_hour - 12, sunset.tm_min)


def go_to_sleep(current_time):
    """Enter deep sleep for time needed."""
    # compute current time offset in seconds
    hour, minutes, seconds = time.localtime(current_time)[3:6]
    seconds_since_midnight = 60 * (hour * 60 + minutes) + seconds
    two_fifteen = (2 * 60 + 15) * 60
    # wake up 15 minutes after 2am
    seconds_to_sleep = (24 * 60 * 60 - seconds_since_midnight) + two_fifteen
    print(
        "Sleeping for {} hours, {} minutes".format(
            seconds_to_sleep // 3600, (seconds_to_sleep // 60) % 60
        )
    )
    magtag.exit_and_deep_sleep(seconds_to_sleep)


# ===========
# Location
# ===========
if isinstance(secrets["openweather_location"], str):
    # Get lat/lon using city name
    city = secrets["openweather_location"]
    print("Getting lat/lon for city:", city)
    latlon = get_latlon(city)
elif isinstance(secrets["openweather_location"], tuple):
    # Get city name using lat/lon
    latlon = secrets["openweather_location"]
    print("Getting city name for lat/lon:", latlon)
    city = get_city(latlon)
else:
    raise ValueError("Unknown location:", secrets["openweather_location"])

print("City =", city)
print("Lat/Lon = ", latlon)

# ===========
# U I
# ===========
today_date = label.Label(terminalio.FONT, text="?" * 30, color=0x000000)
today_date.anchor_point = (0, 0)
today_date.anchored_position = (15, 13)

city_name = label.Label(terminalio.FONT, text=city, color=0x000000)
city_name.anchor_point = (0, 0)
city_name.anchored_position = (15, 24)

today_icon = displayio.TileGrid(
    icons_large_bmp,
    pixel_shader=icons_small_pal,
    x=10,
    y=40,
    width=1,
    height=1,
    tile_width=70,
    tile_height=70,
)
```

```
today_morn_temp = label.Label(terminalio.FONT, text="+100F", color=0x000000)
today_morn_temp.anchor_point = (0.5, 0)
today_morn_temp.anchored_position = (118, 59)

today_day_temp = label.Label(terminalio.FONT, text="+100F", color=0x000000)
today_day_temp.anchor_point = (0.5, 0)
today_day_temp.anchored_position = (149, 59)

today_night_temp = label.Label(terminalio.FONT, text="+100F", color=0x000000)
today_night_temp.anchor_point = (0.5, 0)
today_night_temp.anchored_position = (180, 59)

today_humidity = label.Label(terminalio.FONT, text="100%", color=0x000000)
today_humidity.anchor_point = (0, 0.5)
today_humidity.anchored_position = (105, 95)

today_wind = label.Label(terminalio.FONT, text="99m/s", color=0x000000)
today_wind.anchor_point = (0, 0.5)
today_wind.anchored_position = (155, 95)

today_sunrise = label.Label(terminalio.FONT, text="12:12 PM", color=0x000000)
today_sunrise.anchor_point = (0, 0.5)
today_sunrise.anchored_position = (45, 117)

today_sunset = label.Label(terminalio.FONT, text="12:12 PM", color=0x000000)
today_sunset.anchor_point = (0, 0.5)
today_sunset.anchored_position = (130, 117)

today_banner = displayio.Group()
today_banner.append(today_date)
today_banner.append(city_name)
today_banner.append(today_icon)
today_banner.append(today_morn_temp)
today_banner.append(today_day_temp)
today_banner.append(today_night_temp)
today_banner.append(today_humidity)
today_banner.append(today_wind)
today_banner.append(today_sunrise)
today_banner.append(today_sunset)

future_banners = [
    make_banner(x=210, y=18),
    make_banner(x=210, y=39),
    make_banner(x=210, y=60),
    make_banner(x=210, y=81),
]

magtag.splash.append(today_banner)
for future_banner in future_banners:
    magtag.splash.append(future_banner)

# ===========
#  M A I N
# ===========
print("Fetching forecast...")
forecast_data = get_forecast(latlon)

print("Updating...")
update_today(forecast_data[0])
for day, forecast in enumerate(forecast_data[1:5]):
    update_banner(future_banners[day], forecast)

print("Refreshing...")
time.sleep(magtag.display.time_to_refresh + 1)
magtag.display.refresh()
time.sleep(magtag.display.time_to_refresh + 1)

print("Sleeping...")
go_to_sleep(get_approx_time(latlon))
```

```
#  entire code will run again after deep sleep cycle
#  similar to hitting the reset button
```