MagTag Tides Viewer

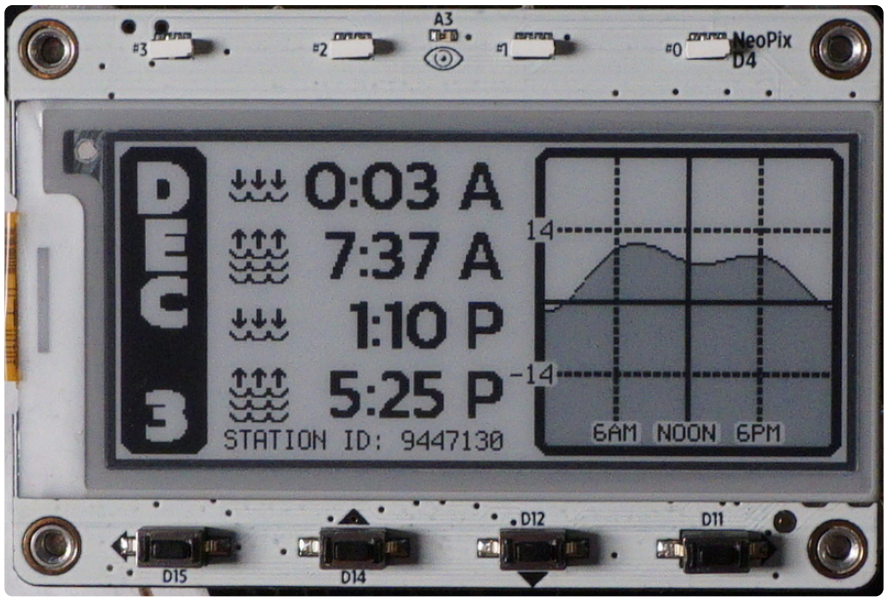
Created by Carter Nelson

https://learn.adafruit.com/magtag-tides-viewer

Last updated on 2024-04-09 11:07:10 AM EDT

Table of Contents

Overview



Tides are a fun thing to track if you live near the ocean. The information is even useful for any activity that is tide dependent, like surfing, boating, or just plain swimming. This guide will show you how to use an Adafruit MagTag (http://adafru.it/4800) to fetch and display daily tide information. It will show the times for the high and low events for the day as well as a neat little plot.

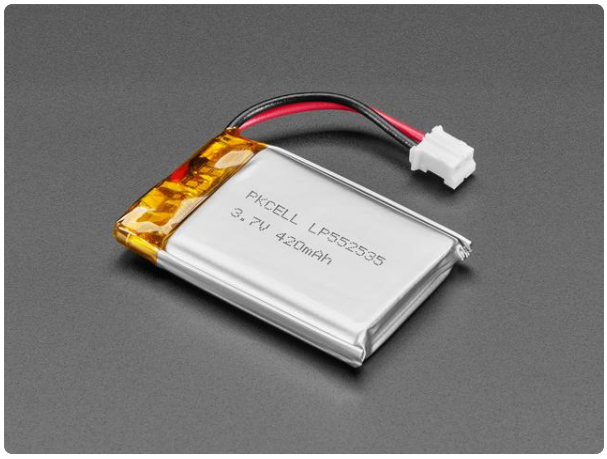
CircuitPython (https://adafru.it/cpy-welcome) is used for the code and the MagTag library takes care of most of the heavy lifting. Just change the code to set your local tide station ID (https://adafru.it/ExQ), and you are good to go.

After updating, the MagTag goes into a deep sleep until 3 am the next morning, so it can stay on your fridge for a few weeks between recharges.

Parts

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

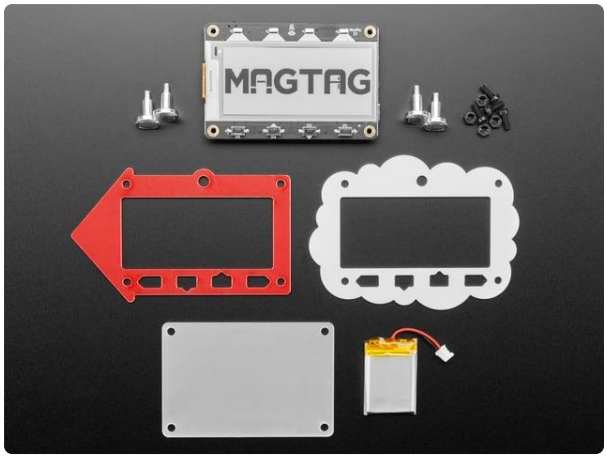
[Adafruit MagTag - 2.9" Grayscale E-Ink WiFi Display]
The Adafruit MagTag combines the new ESP32-S2 wireless module and a 2.9" grayscale E-Ink display to make a low-power IoT display that can show data on its screen even when power...
https://www.adafruit.com/product/4800



[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

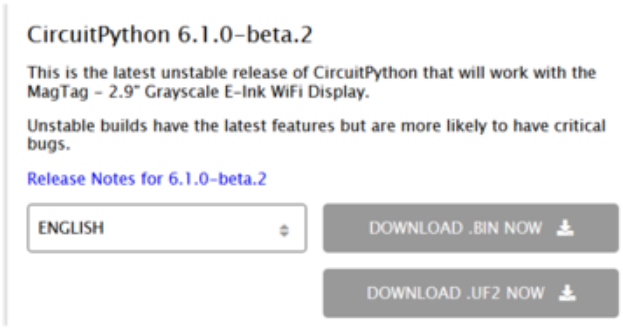
CircuitPython (https://adafru.it/tB7) is a derivative of MicroPython (https://adafru.it/BeZ) designed to simplify experimentation and education on low-cost microcontrollers. It makes it easier than ever to get prototyping by requiring no upfront desktop software downloads. Simply copy and edit files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

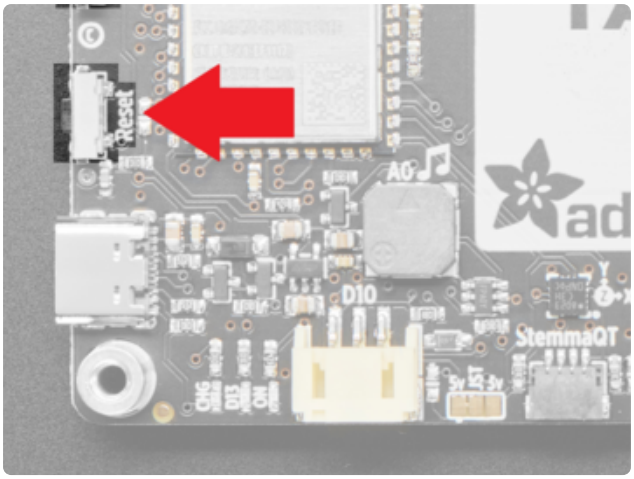
Option 1 - Load with UF2 Bootloader

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.**

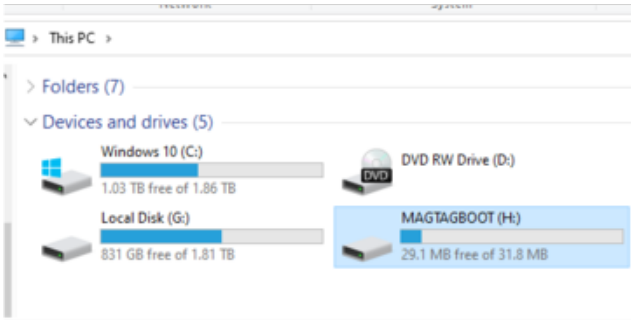
Still, try this first!

Try Launching UF2 Bootloader

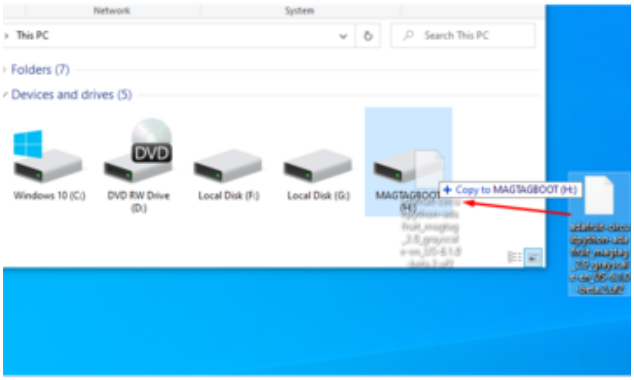
Loading CircuitPython by drag-n-drop UF2 bootloader is the easier way and we recommend it. If you have a MagTag where the front of the board is black, your MagTag came with UF2 already on it.



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.

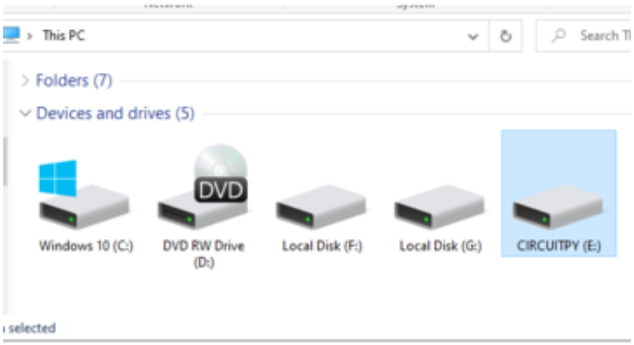


If the UF2 bootloader is installed, you will see a new disk drive appear called **MAGTAGBOOT**



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)

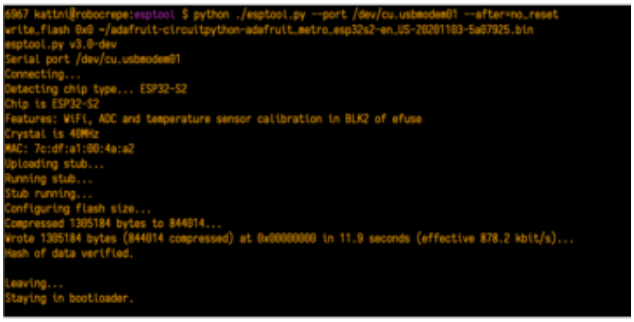


Your board should auto-reset into CircuitPython, or you may need to press reset. A **CIRCUITPY** drive will appear. You're done! Go to the next pages.

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.

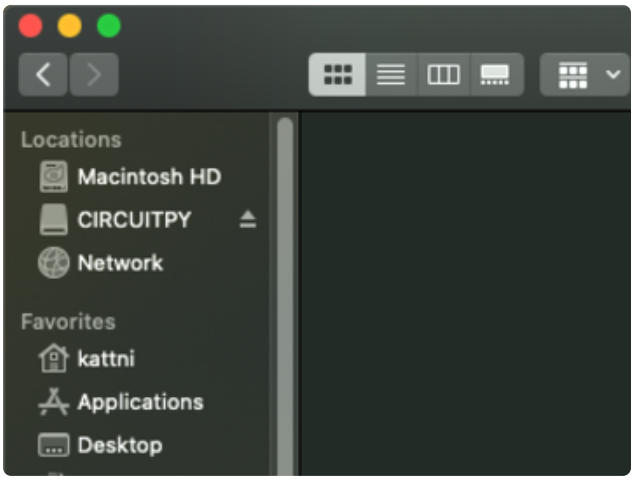
You can upload with **esptool** to the ROM (hardware) bootloader instead!



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!

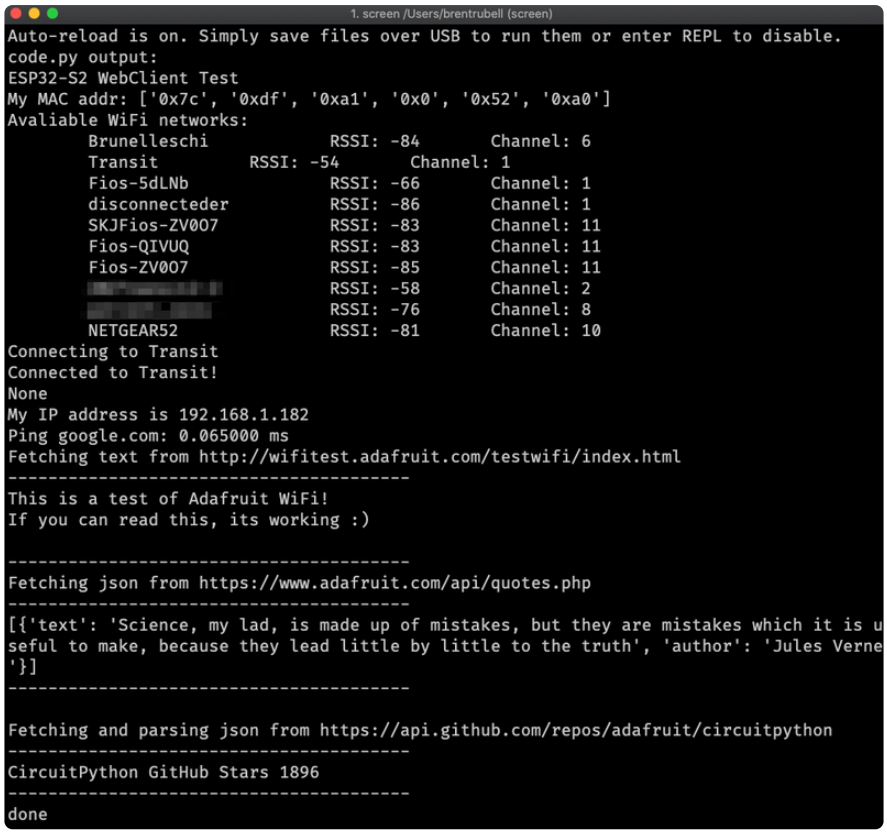
As you make projects you may need more tokens and keys, just add them one line at a time. See for example other tokens such as one for accessing GitHub or the Hackaday API. Other non-secret data like your timezone can also go here.

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!

NOAA Tides Web Service

We will use a web service provided by the National Oceanic and Atmospheric Administration (NOAA (https://adafru.it/ExM)) to get our tide information. One benefit of this is that no API key is needed, since it's your tax dollars at work. However, since this is a part of the US government, the service only covers regions of the United States.

If you live outside the United States, you may have to find a service which sends out tide data as a JSON feed and adapt the code to your new data source. The framework laid out here should provide you a great starting point.

NOAA CO-OPS API

Wow, acronyms! We already covered NOAA above. CO-OPS (https://adafru.it/19kf) stands for Center for Operational Oceanographic Products and Services. Why is there a dash? Don't know, it's just what the government does. But also, it doesn't really matter. If you want to know more, read about it here (https://adafru.it/19kf).

API stands for Application Programming Interface, a general term used a lot in programming. The CO-OPS API is one of many web services that NOAA provides. There's a complete list here (https://adafru.it/ExO).

The main documentation for the web service we will use is here:

CO-OPS API For Data Retrieval

https://adafru.it/ExP

There are a lot of options and types of data that can be returned. We've worked out the magic invocation needed for the PyPortal and you can find it in the code later. The only item you will need to worry about is finding your closest tide monitoring station ID. This is how you will set your location in the code.

So let's see how you can figure that out next.

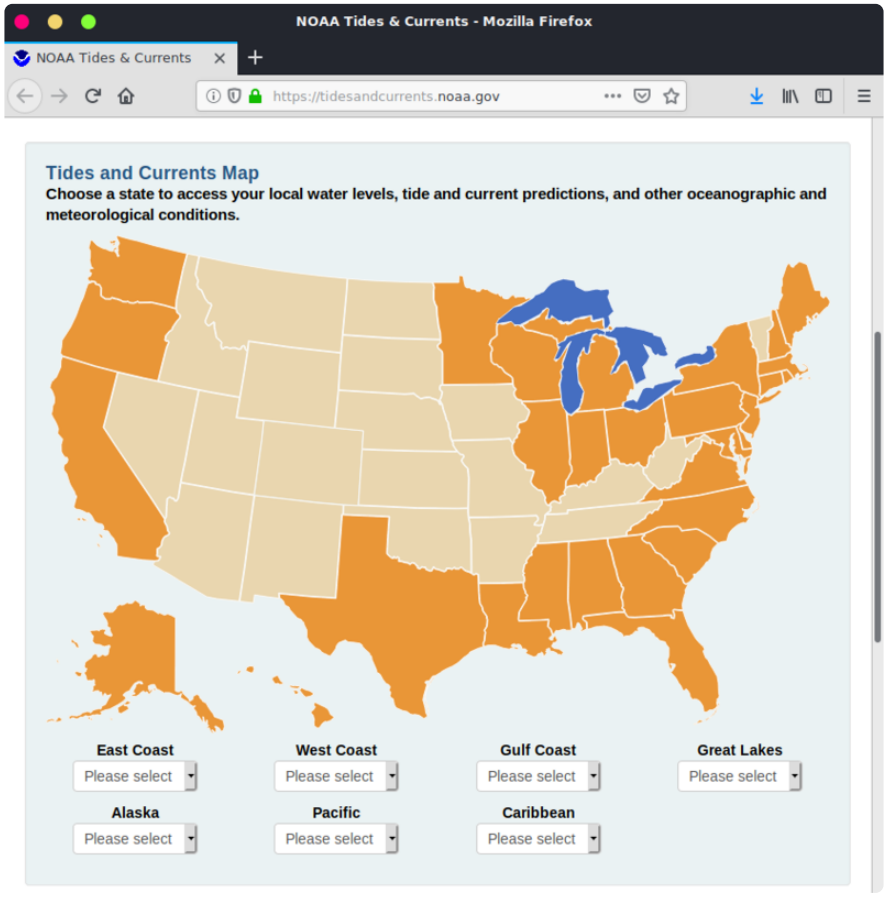
Find Your Tide Station ID

To find you station ID, start by going to this webpage:

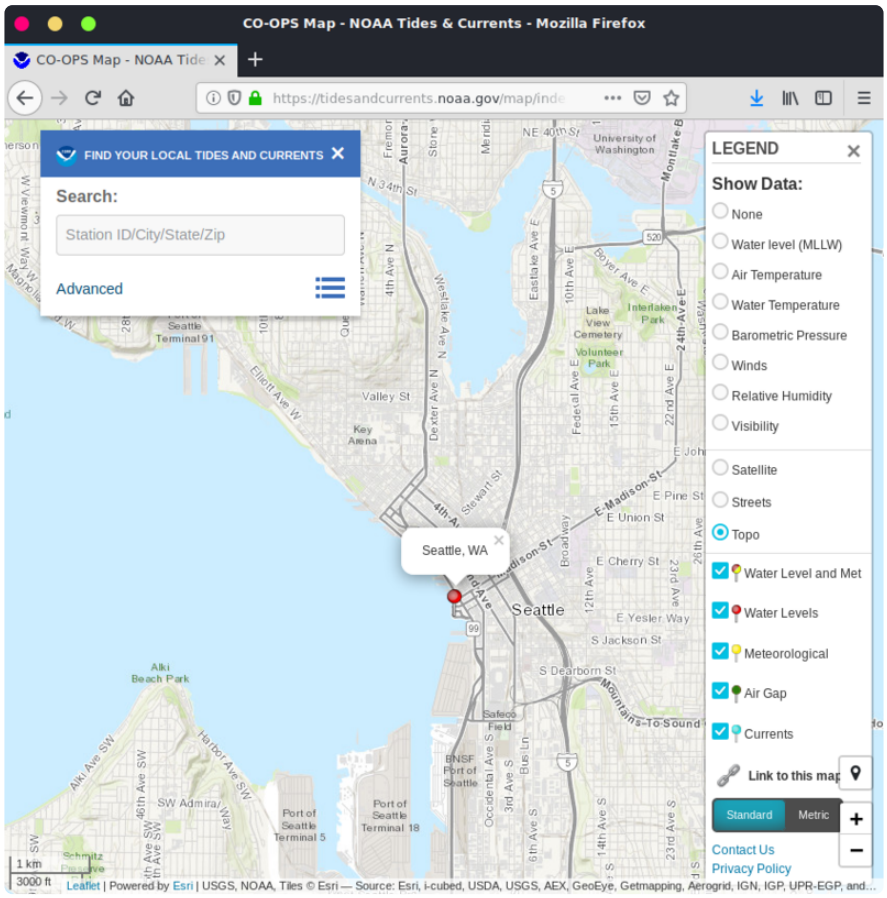
Tides and Currents Map

https://adafru.it/ExQ

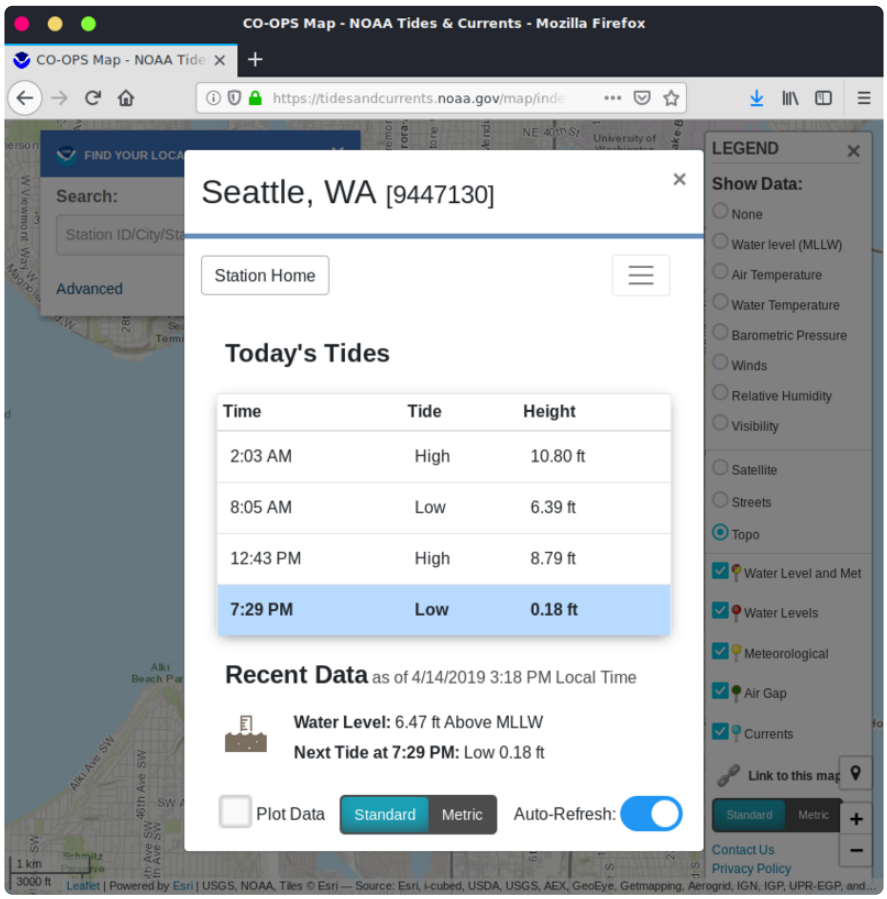
Scroll down a bit and you'll see a map. You can simply click on the state of interest. Note that some states are not clickable. That's because they don't have tides :(



Once you've clicked on a state you'll get a familiar map like interface that you can drag and zoom around. Use that to zoom in and find the station marker that seems like it will work best for your location.



Click on the marker and it will bring up information about that marker.



The station ID will be the number shown near the top. This is what you will enter into your code.

Basic Tide Time Info

Here's the URL that gets the high and low times for the current day at the station location. The station ID is the very last thing in the URL, so you can change it for your location if you want.

```
https://tidesandcurrents.noaa.gov/api/datagetter?
date=today&amp;product=predictions&amp;datum=mllw&amp;interval=hilo&amp;format=json&amp;units=
```

If you put that address in your web browser you'll get something that looks like this:

Not a very pretty web page. That's because it's just JSON data. The key **predictions** is the root for all the data. Then there are several entries that look like:

```
{"t":"2019-04-15 02:55", "v":"3.401", "type":"H"}
```

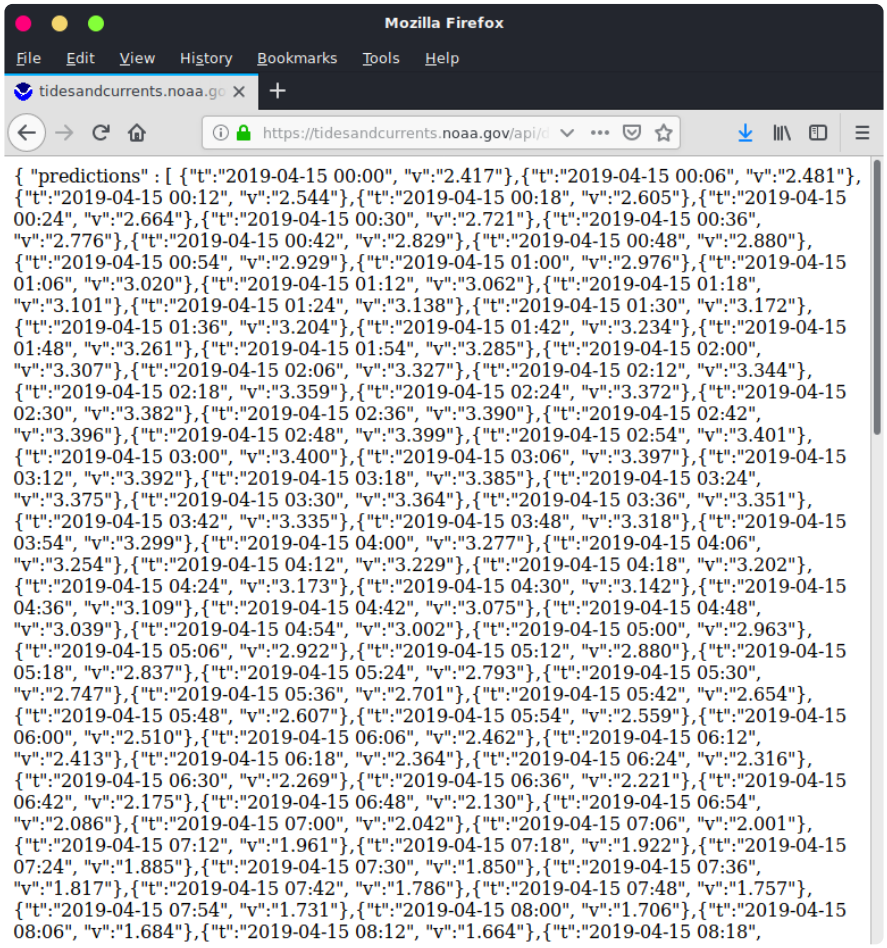
The **t** entry gives us the date and time, the **v** entry gives us the tide level, and the **type** is **H** for high tide or **L** for low tide.

So we have what we need - the time for each of the high and low tides for the given day. You'll see how this is parsed out later in the code.

Tide Levels Throughout the Day

With a slight modification of the URL, we can get predicted water levels in 6 minute increments over the period of the current day. Here's the URL for that:

```
https://tidesandcurrents.noaa.gov/api/datagetter?
date=today&amp;product=predictions&amp;datum=mllw&amp;format=json&amp;units=metric&amp;time_zo
```



More data like before, but now much more of it!

With data every 6 minutes, that's 10 per hour, or 240 for the entire day. And now each entry contains simply **t** and **v**. So for the given time **t**, we get the predicted water level **v**. We can use that to make a neat little plot of water level vs. time for the day. This will give us a graphical representation of the tidal activity.

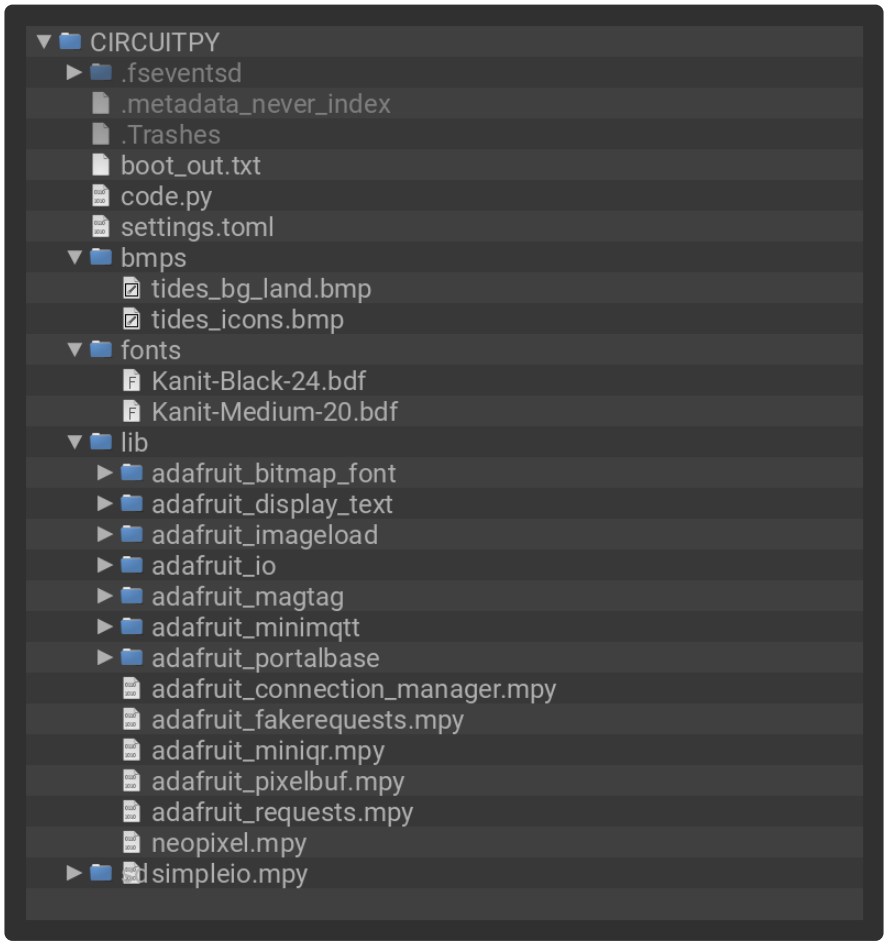
Tide Viewer Code

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_Tides/** 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:




```
# SPDX-FileCopyrightText: 2020 Carter Nelson for Adafruit Industries
#
# SPDX-License-Identifier: MIT

import time
import terminalio
import displayio
```

```
import adafruit_imageload
from adafruit_display_text import label
from adafruit_bitmap_font import bitmap_font
from adafruit_magtag.magtag import MagTag

# --| USER CONFIG |-------------------------
STATION_ID = (
    "9447130"  # tide location, find yours here: https://tidesandcurrents.noaa.gov/
)
METRIC = False  # set to True for metric units
VSCALE = 2  # pixels per ft or m
DAILY_UPDATE_HOUR = 3  # 24 hour format
DST_ON = True  # Day Light Saving currently active?
# ------------------------------------------

# don't change these
PLOT_WIDTH = 116
PLOT_HEIGHT = 116
PLOT_X = 174
PLOT_Y = 6
PLOT_Y_SCALE = round(PLOT_HEIGHT / (4 * VSCALE))
DATE_FONT = bitmap_font.load_font("/fonts/Kanit-Black-24.bdf")
TIME_FONT = bitmap_font.load_font("/fonts/Kanit-Medium-20.bdf")

# our MagTag
magtag = MagTag()
magtag.json_path = ["predictions"]

# ----------------------------
# Grid overlay for plot
# ----------------------------
grid_bmp, grid_pal = adafruit_imageload.load("/bmps/tides_bg_land.bmp")
grid_pal.make_transparent(1)
grid_overlay = displayio.TileGrid(grid_bmp, pixel_shader=grid_pal)

# ----------------------------
# Tide plot (bitmap, palette, tilegrid)
# ----------------------------
tide_plot = displayio.Bitmap(PLOT_WIDTH, PLOT_HEIGHT, 4)

tide_pal = displayio.Palette(4)
tide_pal[0] = 0x000000  # black
tide_pal[1] = 0x555555  # dark gray
tide_pal[2] = 0xAAAAAA  # light gray
tide_pal[3] = 0xFFFFFF  # white
tide_pal.make_transparent(3)

tide_tg = displayio.TileGrid(tide_plot, pixel_shader=tide_pal, x=PLOT_X, y=PLOT_Y)

# ----------------------------
# Plot scale labels
# ----------------------------
plot_y_pos = label.Label(terminalio.FONT, text="+99", color=0x000000)
plot_y_pos.text = "{:>3}".format(PLOT_Y_SCALE)
plot_y_pos.anchor_point = (1.0, 0.5)
plot_y_pos.anchored_position = (178, 34)

plot_y_neg = label.Label(terminalio.FONT, text="-99", color=0x000000)
plot_y_neg.text = "{:>3}".format(-1 * PLOT_Y_SCALE)
plot_y_neg.anchor_point = (1.0, 0.5)
plot_y_neg.anchored_position = (178, 92)

plot_y_labels = displayio.Group()
plot_y_labels.append(plot_y_pos)
plot_y_labels.append(plot_y_neg)

# ----------------------------
# Date label
# ----------------------------
```

```
date_label = displayio.Group()
date_text = [label.Label(DATE_FONT, text="A", color=0xFFFFFF) for _ in range(5)]
y_offset = 8
for text in date_text:
    date_label.append(text)
    text.anchor_point = (0.5, 0)
    text.anchored_position = (20, y_offset)
    y_offset += 23

# ----------------------------
# HiLo Times and Icons
# ----------------------------
tide_info = displayio.Group()

hilo_times = [label.Label(TIME_FONT, text="12:34 P", color=0x000000) for _ in
range(4)]
y_offset = 18
for hilo in hilo_times:
    tide_info.append(hilo)
    hilo.hidden = True
    hilo.anchor_point = (1, 0.5)
    hilo.anchored_position = (158, y_offset)
    y_offset += 28

icon_bmp, icon_pal = adafruit_imageload.load("/bmps/tides_icons.bmp")
icon_pal.make_transparent(1)
hilo_icons = [
    displayio.TileGrid(
        icon_bmp,
        pixel_shader=icon_pal,
        width=1,
        height=1,
        tile_width=24,
        tile_height=24,
    )
    for _ in range(4)
]
y_offset = 6
for icon in hilo_icons:
    tide_info.append(icon)
    icon.hidden = True
    icon.x = 46
    icon.y = y_offset
    y_offset += 28

# ----------------------------
# Station ID
# ----------------------------
station_info = label.Label(
    terminalio.FONT, text="STATION ID: " + STATION_ID, color=0x000000
)
station_info.anchor_point = (1, 1)
station_info.anchored_position = (158, 126)

# ----------------------------
# Add all the graphic layers
# ----------------------------
magtag.splash.append(tide_tg)
magtag.splash.append(grid_overlay)
magtag.splash.append(plot_y_labels)
magtag.splash.append(tide_info)
magtag.splash.append(date_label)
magtag.splash.append(station_info)

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


def get_data_source_url(station=STATION_ID, metric=METRIC, hilo_only=True):
    """Build and return the URL for the tides API."""
```

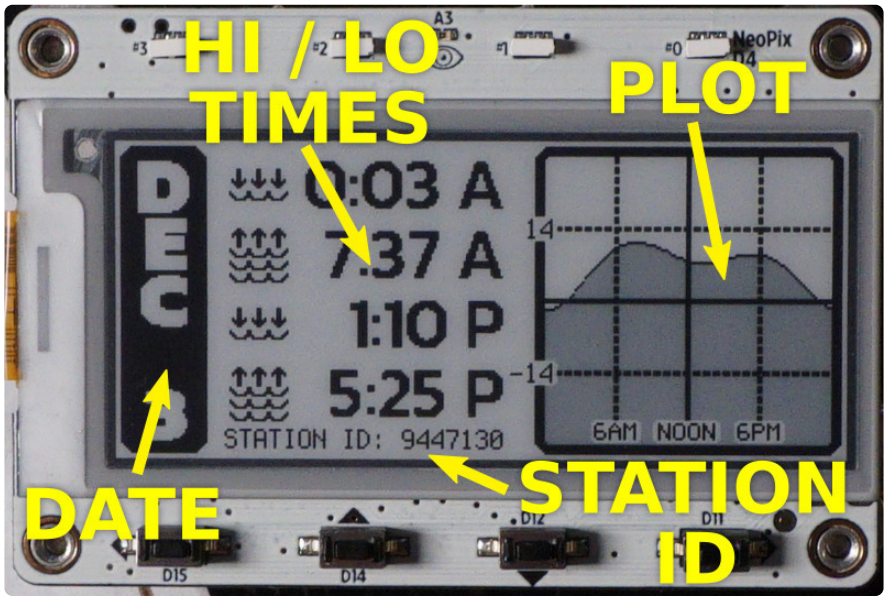
```
    date = "{}{:02}{:02}".format(now.tm_year, now.tm_mon, now.tm_mday)

    URL = "https://api.tidesandcurrents.noaa.gov/api/prod/datagetter?format=json"
    URL += "&product=predictions"
    URL += "&interval=hilo" if hilo_only else ""
    URL += "&datum=mllw"  # MLLW = "tides"
    URL += "&units=metric" if metric else "&units=english"
    URL += "&time_zone=lst_ldt" if DST_ON else "&time_zone=lst"
    URL += "&begin_date=" + date
    URL += "&end_date=" + date
    URL += "&station=" + station

    return URL


def get_tide_data():
    """Fetch JSON tide data and return parsed results in a list."""
    # Get raw JSON data
    magtag.url = get_data_source_url(hilo_only=False)
    raw_data = magtag.fetch()

    # Results will be stored in a list that is PLOT_WIDTH long
    new_tide_data = [PLOT_HEIGHT - 1] * PLOT_WIDTH

    # Convert raw data to display coordinates
    for data in raw_data:
        _, t = data["t"].split(" ")  # date and time
        h, m = t.split(":")  # hours and minutes
        v = data["v"]  # water level
        x = round((PLOT_WIDTH - 1) * (60 * float(h) + float(m)) / 1434)
        y = (PLOT_HEIGHT // 2) - round(VSCALE * float(v))
        y = 0 if y < 0 else y
        y = PLOT_HEIGHT - 1 if y >= PLOT_HEIGHT else y
        new_tide_data[x] = y

    return new_tide_data


def get_hilo_data():
    """Get high / low times."""
    # Get raw JSON data
    magtag.url = get_data_source_url(hilo_only=True)

    return magtag.fetch()


def show_today():
    """Display month and day."""
    month_text = (
        "JAN",
        "FEB",
        "MAR",
        "APR",
        "MAY",
        "JUN",
        "JUL",
        "AUG",
        "SEP",
        "OCT",
        "NOV",
        "DEC",
    )[now.tm_mon - 1]
    day_text = "{:2}".format(now.tm_mday)

    date_label[0].text = month_text[0]
    date_label[1].text = month_text[1]
    date_label[2].text = month_text[2]
    date_label[3].text = day_text[0]
    date_label[4].text = day_text[1]
```

```
def plot_tides():
    """Graphical plot of water level."""
    tide_plot.fill(3)
    for x in range(PLOT_WIDTH):
        y = tide_data[x]
        for yfill in range(y, PLOT_HEIGHT):
            try:
                tide_plot[x, yfill] = 2
            except IndexError:
                pass
        tide_plot[x, y] = 0


def show_hilo():
    """Show high / low times."""
    for i in hilo_icons:
        i.hidden = True
    for t in hilo_times:
        t.hidden = True
    for i, data in enumerate(hilo_data):
        # make it visible
        hilo_icons[i].hidden = False
        hilo_times[i].hidden = False
        # icon
        hilo_icons[i][0] = 0 if data["type"] == "H" else 1
        # time
        h, m = data["t"].split(" ")[1].split(":")
        m = int(m)
        h = int(h)
        ampm = "A" if h < 12 else "P"
        h = h if h < 13 else h - 12
        hilo_times[i].text = "{:>2}:{:02} {}".format(h, m, ampm)


def time_to_sleep():
    """Compute amount of time to sleep."""
    # daily event time
    event_time = time.struct_time(
        (now[0], now[1], now[2], DAILY_UPDATE_HOUR, 0, 0, -1, -1, now[8])
    )
    # how long is that from now?
    remaining = time.mktime(event_time) - time.mktime(now)
    # is that today or tomorrow?
    if remaining < 0:  # ah its aready happened today...
        remaining += 24 * 60 * 60  # wrap around to the next day
    # return it
    return remaining


# ===========
#  M A I N
# ===========
# get current time
magtag.get_local_time()
now = time.localtime()

# show today's date
show_today()

# get and plot tide levels
tide_data = get_tide_data()
plot_tides()

# get and show hilo tide times
hilo_data = get_hilo_data()
show_hilo()
```

```
# refresh display
time.sleep(magtag.display.time_to_refresh + 1)
magtag.display.refresh()
time.sleep(magtag.display.time_to_refresh + 1)

# ZZZZZZzzzzzzzzz
now = time.localtime()
magtag.exit_and_deep_sleep(time_to_sleep())
#
# code.py runs again when board wakes up
#
```

Tide Information Display

Once the code runs, the MagTag display should update to show the information below.



Customizing

Simple Stuff

At the top of the code there is a section with some user changeable settings. It looks like this:

```
# --| USER CONFIG |-------------------------
STATION_ID = (
    "9447130"  # tide location, find yours here: https://tidesandcurrents.noaa.gov/
)
METRIC = False  # set to True for metric units
VSCALE = 2  # pixels per ft or m
DAILY_UPDATE_HOUR = 3  # 24 hour format
# ----------------------------------------
```

The most important is `STATION_ID`, which sets the location. If you want the plot to use meters instead of feet, you can set `METRIC` to `True`. The plot does not auto adjust the scale, so if you find plot data getting squished or going off the grid, you can change `VSCALE` as needed. And finally, `DAILY_UPDATE_HOUR` is the hour in 24 hour format at which time the tide information and display are updated.

Not So Simple Stuff

The plot grid and banner behind the date label were created as a bitmap image. This is then loaded and added in with the other various graphical elements. So to actually change either of those, you would need to alter the source bitmap.

Here is the SVG file used to create the bitmap.

magtag_land.svg

https://adafru.it/P9c

There were some additional refinements done using image editor software (gimp) after exporting the SVG to a PNG. Then the PNG was converted to a BMP for use in CircuitPython. If you want to alter this graphic, starting with the SVG file is the best way.