MagTag Daily Christmas Countdown

Created by Ruiz Brothers

https://learn.adafruit.com/magtag-daily-christmas-countdown

Last updated on 2024-11-29 10:05:11 AM EST

Table of Contents

Overview

IoT Holiday Countdown

In this project, make a Christmas countdown using Adafruit's MagTag. This grabs the date from the internet and updates each day so you know how many days until Christmas. The little bobbles on the tree are filled in as the days go by so you can quickly see how many days are left.

3D Printed Stand

The stand is designed to be festive and comes in two version: Portrait and Landscape.

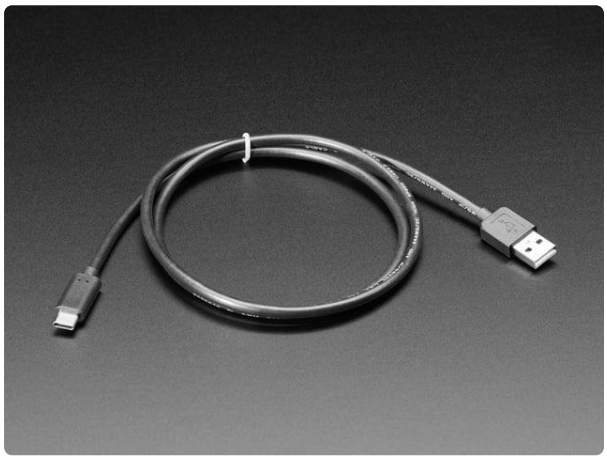
You may use two M3 screws to secure the MagTag to the stand, it's easy peasy!

Parts

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



USB Type A to Type C Cable - approx 1 meter / 3 ft long

As technology changes and adapts, so does Adafruit. This USB Type A to Type C cable will help you with the transition to USB C, even if you're still...

https://www.adafruit.com/product/4474

Install CircuitPython

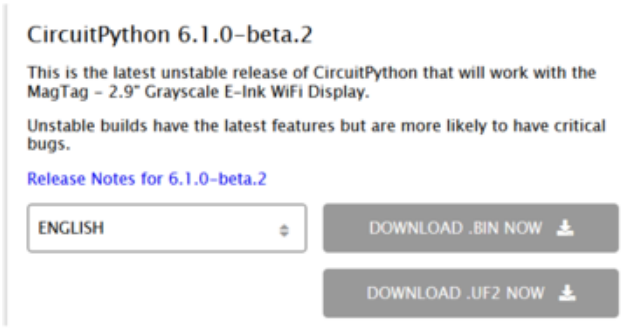
CircuitPython (https://adafru.it/tB7) is a derivative of MicroPython (https://adafru.it/BeZ) designed to simplify experimentation and education on low-cost microcontrollers. It makes it easier than ever to get prototyping by requiring no upfront desktop software downloads. Simply copy and edit files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

Set Up CircuitPython

Follow the steps to get CircuitPython installed on your MagTag.

Download the latest CircuitPython for your board from circuitpython.org

https://adafru.it/OBd



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

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

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

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

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

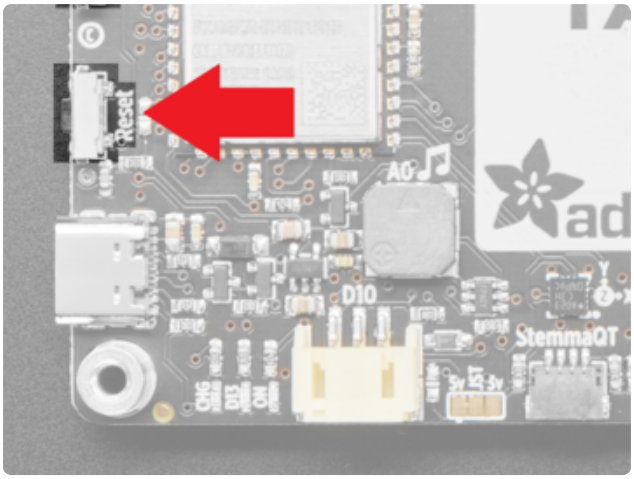
Option 1 - Load with UF2 Bootloader

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

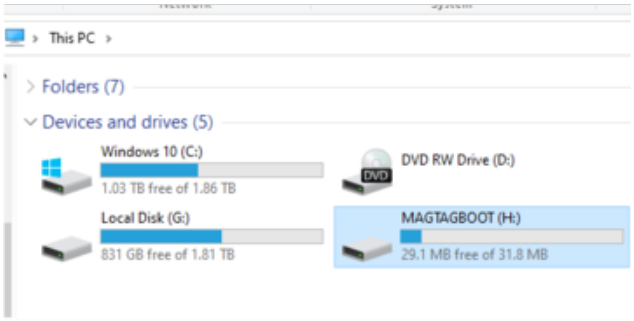
Still, try this first!

Try Launching UF2 Bootloader

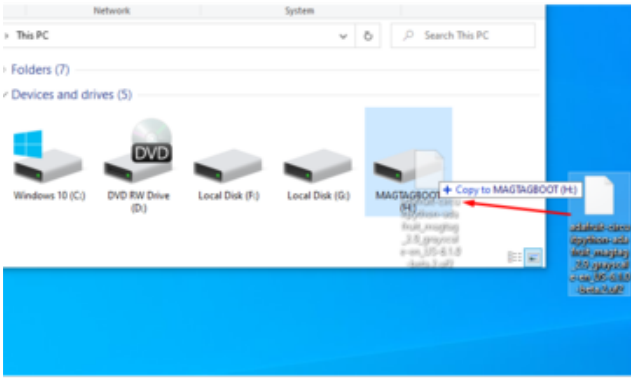
Loading CircuitPython by drag-n-drop UF2 bootloader is the easier way and we recommend it. If you have a MagTag where the front of the board is black, your MagTag came with UF2 already on it.



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

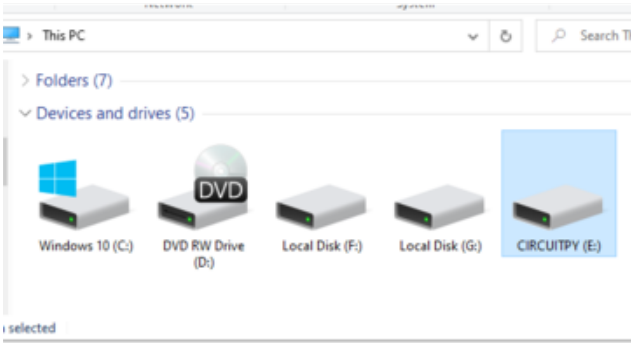


If the UF2 bootloader is installed, you will see a new disk drive appear called **MAGTAGBOOT**



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

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

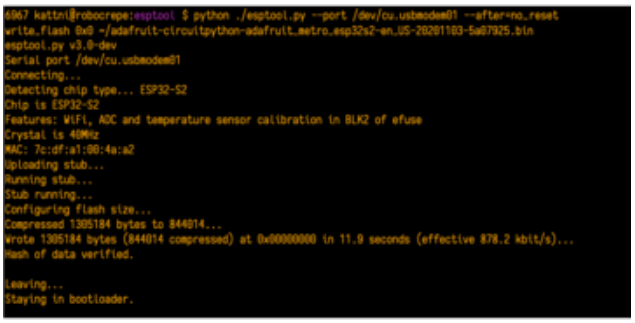


Your board should auto-reset into CircuitPython, or you may need to press reset. A **CIRCUITPY** drive will appear. You're done! Go to the next pages.

Option 2 - Use esptool to load BIN file

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

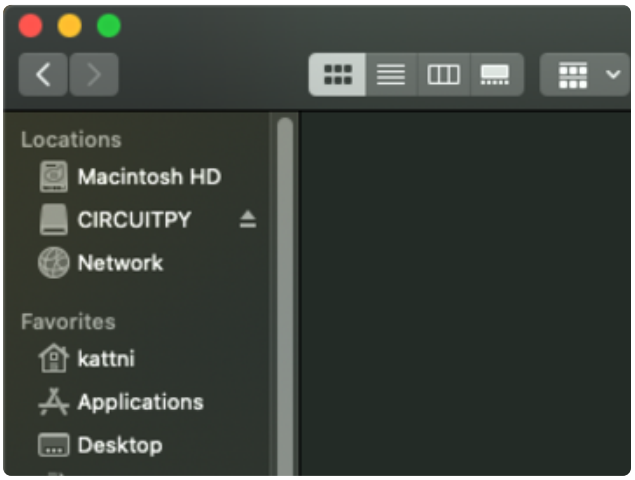
You can upload with **esptool** to the ROM (hardware) bootloader instead!



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

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

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



Press reset to exit the bootloader.

Your **CIRCUITPY** drive should appear!

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

Option 3 - Use Chrome Browser To Upload BIN file

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

CircuitPython Internet Test

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

The first thing you need to do is update your **code.py** to the following. Click the **Download Project Bundle** button below to download the necessary libraries and the **code.py** file in a zip file. Extract the contents of the zip file, and copy the **entire lib folder** and the **code.py** file to your **CIRCUITPY** drive.

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

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

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

print("ESP32-S2 WebClient Test")

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

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

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

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

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

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

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

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

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

print()
```

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

print("Done")
```

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

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

The settings.toml File

We expect people to share tons of projects as they build CircuitPython WiFi widgets. What we want to avoid is people accidentally sharing their passwords or secret tokens and API keys. So, we designed all our examples to use a **settings.toml** file, that is on your **CIRCUITPY** drive, to hold secret/private/custom data. That way you can share your main project without worrying about accidentally sharing private stuff.

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

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

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

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

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

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

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

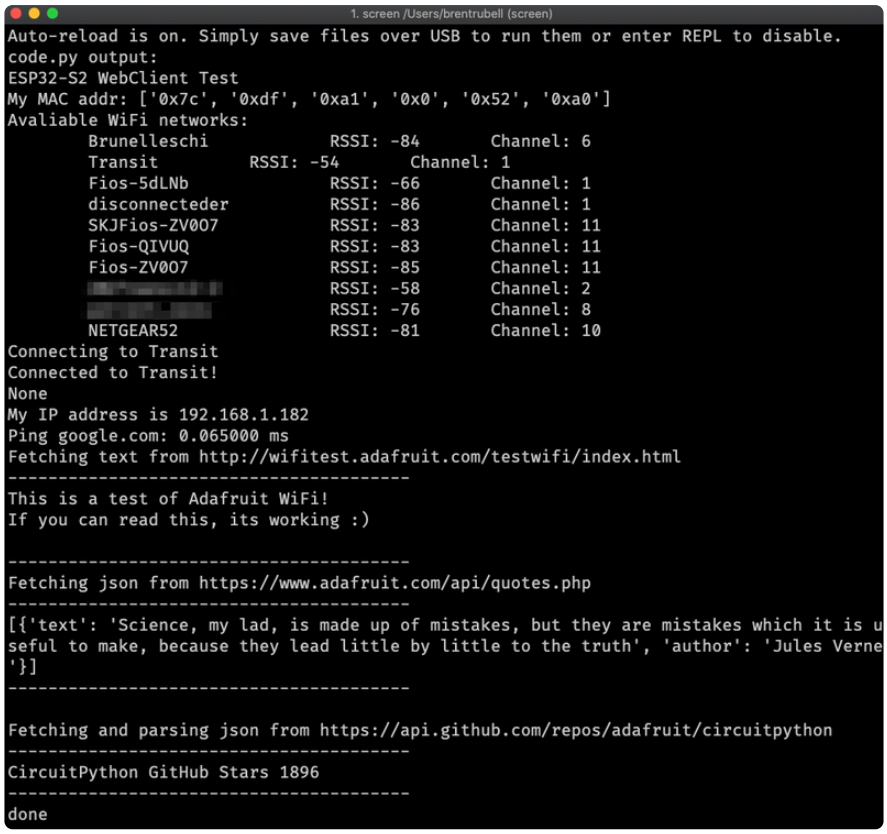
As you make projects you may need more tokens and keys, just add them one line at a time. See for example other tokens such as one for accessing GitHub or the Hackaday API. Other non-secret data like your timezone can also go here.

For the correct time zone string, look at http://worldtimeapi.org/timezones (https://adafru.it/EcP) and remember that if your city is not listed, look for a city in the same time zone, for example Boston, New York, Philadelphia, Washington DC, and Miami are all on the same time as New York.

Of course, don't share your **settings.toml** - keep that out of GitHub, Discord or other project-sharing sites.

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

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



In order, the example code...

Checks the ESP32's MAC address.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

IPv6 Networking

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

IPv6 connectivity & privacy

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

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

Enable IPv6 networking

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

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

Check IP addresses

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

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

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

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

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

Ping v6 networks

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

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

Create & use IPv6 sockets

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

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

Getting The Date & Time

A very common need for projects is to know the current date and time. Especially when you want to deep sleep until an event, or you want to change your display based on what day, time, date, etc. it is

Determining the correct local time is really really hard. There are various time zones, Daylight Savings dates, leap seconds, etc. Trying to get NTP time and then back-calculating what the local time is, is extraordinarily hard on a microcontroller just isn't worth the effort and it will get out of sync as laws change anyways.

For that reason, we have the free adafruit.io time service. **Free for anyone with a free adafruit.io account.** You do need an account because we have to keep accidentally mis-programmed-board from overwhelming adafruit.io and lock them out temporarily. Again, it's free!

There are other services like WorldTimeAPI, but we don't use those for our guides because they are nice people and we don't want to accidentally overload their site. Also, there's a chance it may eventually go down or also require an account.

Step 1) Make an Adafruit account

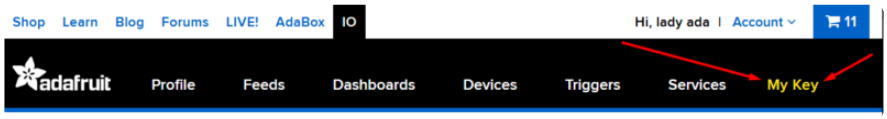
It's free! Visit https://accounts.adafruit.com/ (https://adafru.it/dyy) to register and make an account if you do not already have one

Step 2) Sign into Adafruit IO

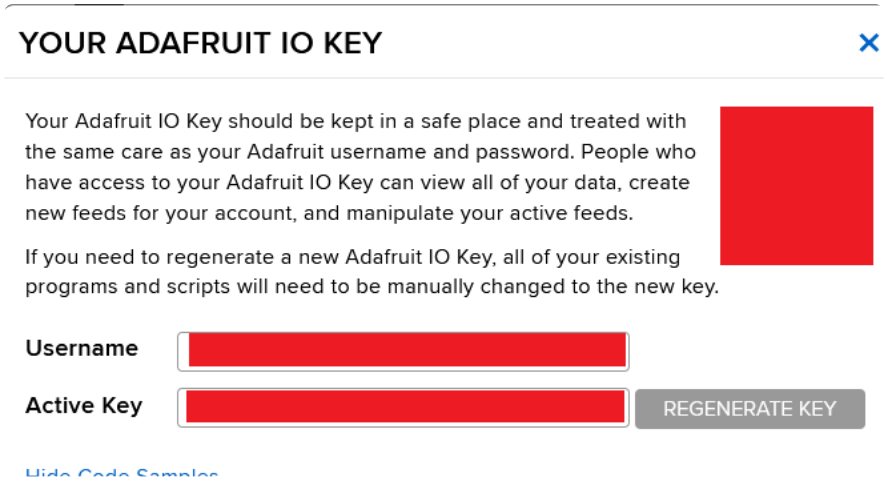
Head over to io.adafruit.com (https://adafru.it/fsU) and click **Sign In** to log into IO using your Adafruit account. It's free and fast to join.

Step 3) Get your Adafruit IO Key

Click on **My Key** in the top bar



You will get a popup with your **Username** and **Key** (In this screenshot, we've covered it with red blocks)



Go to the **settings.toml** file on your CIRCUITPY drive and add three lines for `AIO_USERNAME`, `ADAFRUIT_AIO_KEY` and `TIMEZONE` so you get something like the following:

```
# This file is where you keep secret settings, passwords, and tokens!
# If you put them in the code you risk committing that info or sharing it

CIRCUITPY_WIFI_SSID = "your-wifi-ssid"
```

```
CIRCUITPY_WIFI_PASSWORD = "your-wifi-password"
ADAFRUIT_AIO_USERNAME = "your-adafruit-io-username"
ADAFRUIT_AIO_KEY = "your-adafruit-io-key"
# Timezone names from http://worldtimeapi.org/timezones
TIMEZONE="America/New_York"
```

The timezone is optional, if you don't have that entry, adafruit.io will guess your timezone based on geographic IP address lookup. You can visit http://worldtimeapi.org/timezones (https://adafru.it/EcP) to see all the time zones available (even though we do not use Worldtime for time-keeping, we do use the same time zone table).

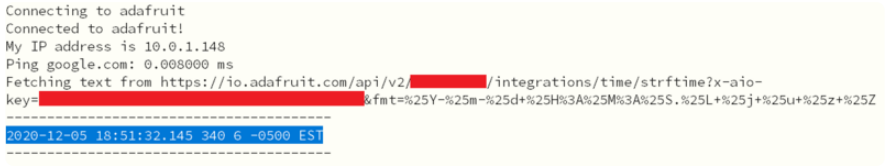
Step 4) Upload Test Python Code

This code is like the Internet Test code from before, but this time it will connect to adafruit.io and get the local time

```
import ipaddress
import os
import ssl
import wifi
import socketpool
import adafruit_requests
import secrets


# Get our username, key and desired timezone
ssid = os.getenv("CIRCUITPY_WIFI_SSID")
password = os.getenv("CIRCUITPY_WIFI_PASSWORD")
aio_username = os.getenv("ADAFRUIT_AIO_USERNAME")
aio_key = os.getenv("ADAFRUIT_AIO_KEY")
timezone = os.getenv("TIMEZONE")
TIME_URL = f"https://io.adafruit.com/api/v2/{aio_username}/integrations/time/
strftime?x-aio-key={aio_key}&amp;tz={timezone}"
TIME_URL += "&amp;fmt=%25Y-%25m-%25d+%25H%3A%25M%3A%25S.%25L+%25j+%25u+%25z+%25Z"

print("ESP32-S2 Adafruit IO Time test")

print("My MAC addr:", [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("Connecting to", ssid)
wifi.radio.connect(ssid, password)
print(f"Connected to {ssid}!")
print("My IP address is", wifi.radio.ipv4_address)

ipv4 = ipaddress.ip_address("8.8.4.4")
print("Ping google.com:", wifi.radio.ping(ipv4), "ms")

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

print("Fetching text from", TIME_URL)
response = requests.get(TIME_URL)
print("-" * 40)
```

```
print(response.text)
print("-" * 40)
```

After running this, you will see something like the below text. We have blocked out the part with the secret username and key data!



Note at the end you will get the date, time, and your timezone! If so, you have correctly configured your **settings.toml** and can continue to the next steps!

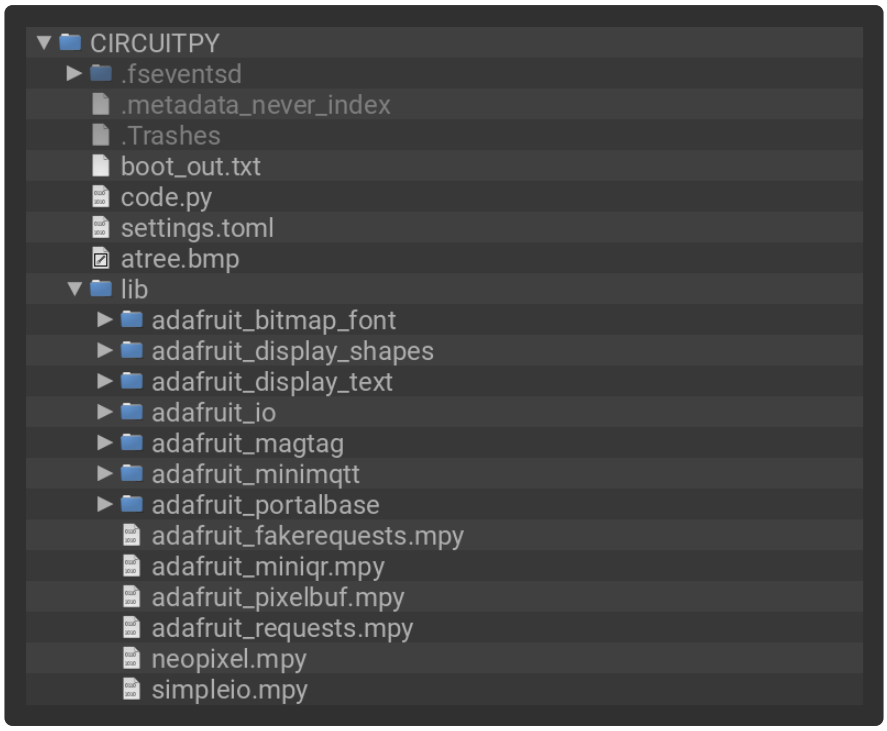
Coding the MagTag Christmas Countdown

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_Christmas_Countdown/** 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 Liz Clark for Adafruit Industries
#
# SPDX-License-Identifier: MIT

import time
import displayio
from adafruit_magtag.magtag import MagTag
from adafruit_display_shapes.circle import Circle

#  create MagTag and connect to network
try:
    magtag = MagTag()
    magtag.network.connect()
except (ConnectionError, ValueError, RuntimeError) as e:
    print("*** MagTag(), Some error occured, retrying! -", e)
    # Exit program and restart in 1 seconds.
    magtag.exit_and_deep_sleep(1)



#  displayio groups
group = displayio.Group()
tree_group = displayio.Group()
circle_group = displayio.Group()

#  import tree bitmap
filename = "/atree.bmp"

# CircuitPython 6 & 7 compatible
tree = displayio.OnDiskBitmap(open(filename, "rb"))
tree_grid = displayio.TileGrid(
    tree, pixel_shader=getattr(tree, 'pixel_shader', displayio.ColorConverter())
)

# # CircuitPython 7+ compatible
# tree = displayio.OnDiskBitmap(filename)
# tree_grid = displayio.TileGrid(tree, pixel_shader=tree.pixel_shader)

#  add bitmap to its group
tree_group.append(tree_grid)
#  add tree group to the main group
```

```
group.append(tree_group)

#  list of circle positions
spots = (
    (246, 53),
    (246, 75),
    (206, 42),
    (206, 64),
    (206, 86),
    (176, 31),
    (176, 53),
    (176, 75),
    (176, 97),
    (136, 42),
    (136, 64),
    (136, 86),
    (106, 31),
    (106, 53),
    (106, 75),
    (106, 97),
    (66, 31),
    (66, 53),
    (66, 75),
    (66, 97),
    (36, 20),
    (36, 42),
    (36, 64),
    (36, 86),
    (36, 108)
    )

#  circles to cover-up bitmap's number ornaments

ball_color = [0x555555, 0xaaaaaa, 0xFFFFFF] # All colors except black (0x000000)
ball_index = 0

#  creating the circles & pulling in positions from spots
for spot in spots:
    circle = Circle(x0=spot[0], y0=spot[1], r=11, fill=ball_color[ball_index]) #
Each ball has a color
    ball_index += 1
    ball_index %= len(ball_color)

    #  adding circles to their display group
    circle_group.append(circle)

#  adding circles group to main display group
group.append(circle_group)

#  grabs time from network
magtag.get_local_time()
#  parses time into month, date, etc
now = time.localtime()
month = now[1]
day = now[2]
(hour, minutes, seconds) = now[3:6]
seconds_since_midnight = 60 * (hour*60 + minutes)+seconds
print( f"day is {day}, ({seconds_since_midnight} seconds since midnight)")


#  sets colors of circles to transparent to reveal dates that have passed & current
date
for i in range(day):
    circle_group[i].fill = None
    time.sleep(0.1)

#  updates display with bitmap and current circle colors
magtag.display.root_group = group
magtag.display.refresh()
```

```
time.sleep(5)

#  goes into deep sleep till a 'stroke' past midnight
print("entering deep sleep")
seconds_to_sleep = 24*60*60 - seconds_since_midnight + 10
print( f"sleeping for {seconds_to_sleep} seconds")
magtag.exit_and_deep_sleep(seconds_to_sleep)

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

Bitmap File

Copy **atree.bmp** from within the zip file downloaded in the last step to the **CIRCUITPY** main (root) directory.

Review

Make sure you've followed these steps:

- Created a **secrets.py** file with your network WiFi info and Adafruit IO info and copied the file to the **CIRCUITPY** main (root) directory.
- Loaded all the required library files and directories into the **CIRCUITPY /lib** directory
- Copied **atree.bmp** to the main (root) directory of the **CIRCUITPY** drive
- Copied **code.py** to the main (root) directory of the **CIRCUITPY** drive

---

CircuitPython Code Walkthrough

The code begins by importing the libraries.

```
import time
import displayio
from adafruit_magtag.magtag import MagTag
from adafruit_display_shapes.circle import Circle
```

A `magtag` object is created to access the **adafruit_magtag** library. Your MagTag connects to your network with `magtag.network.connect()`.

```
#  create MagTag and connect to network
magtag = MagTag()
magtag.network.connect()
```

Three **displayio** groups are used. `tree_group` holds the tree bitmap image. `circle_group` holds the circles that will either hide or show the numbers on the tree bitmap. `group` is the main group that will allow for `tree_group` and `circle_group` to be shown at the same time.

```
#  displayio groups
group = displayio.Group()
tree_group = displayio.Group()
circle_group = displayio.Group()
```

The tree bitmap is brought in and added to the `tree_group`. `tree_group` is added to `group`.

```
# import tree bitmap
filename = "/atree.bmp"

# CircuitPython 6 &amp;amp; 7 compatible
tree = displayio.OnDiskBitmap(open(filename, "rb"))
tree_grid = displayio.TileGrid(
    tree, pixel_shader=getattr(tree, 'pixel_shader', displayio.ColorConverter())
)

# # CircuitPython 7+ compatible
# tree = displayio.OnDiskBitmap(filename)
# tree_grid = displayio.TileGrid(tree, pixel_shader=tree.pixel_shader)
```

Each circle has a specific coordinate that corresponds with the numbers on the tree bitmap. These coordinates are brought in via a list called `spots`.

```
#  list of circle positions
spots = (
    (246, 53),
    (246, 75),
    (206, 42),
    (206, 64),
    (206, 86),
    (176, 31),
    (176, 53),
    (176, 75),
    (176, 97),
    (136, 42),
    (136, 64),
    (136, 86),
    (106, 31),
    (106, 53),
    (106, 75),
    (106, 97),
    (66, 31),
    (66, 53),
    (66, 75),
    (66, 97),
    (36, 20),
    (36, 42),
    (36, 64),
    (36, 86),
    (36, 108)
    )
```

The 25 circles are created using a `for` statement. This allows the list of coordinates to be iterated through so that each circle will be in the correct position. All of the circles have a radius of `11` and `0xFF00FF` as the default color, causing them to appear as a dark grey.

Each circle is added to the `circle_group` as they are created. After all 25 circles are created, the `circle_group` is added to `group`, joining `tree_group`.

```
#  circles to cover-up bitmap's number ornaments

#  creating the circles &amp;amp; pulling in positions from spots
for spot in spots:
    circle = Circle(x0=spot[0], y0=spot[1],
                    r=11,
                    fill=0xFF00FF)
    #  adding circles to their display group
    circle_group.append(circle)

#  adding circles group to main display group
group.append(circle_group)
```

The MagTag gathers the current time with the function `magtag.get_local_time()`. `time.localtime()` is called to parse the time data, including the month and date, into an array that can be accessed.

```
#  grabs time from network
magtag.get_local_time()
#  parses time into month, date, etc
now = time.localtime()
month = now[1]
day = now[2]
```

Additionally, the time is pulled down as well and put into an equation to figure out how many seconds it has been since midnight. This will be used to calculate how long the MagTag should go into deep sleep for.

```
(hour, minutes, seconds) = now[3:6]
seconds_since_midnight = 60 * (hour*60 + minutes)+seconds
print( f"day is {day}, ({seconds_since_midnight} seconds since midnight)")
```

The countdown's progress is revealed on the tree with a `for` statement. The `circle_group` is iterated through with the date number acting as the `range`. The circles that fall in that `range` have their `fill` set to `None`. As a result, the numbers on the bitmap tree are shown for the dates that have passed and the current date.

```
#  sets colors of circles to transparent to reveal dates that have passed &amp;amp;
current date
for i in range(day):
    circle_group[i].fill = None
    time.sleep(0.1)
```

The MagTag display's `root_group` is set to to show `group`, which includes the tree bitmap and the circles. This is followed by a refresh, which is necessary when updating an E-Ink display.

```
#  updates display with bitmap and current circle colors
magtag.display.root_group = group
```

```
magtag.display.refresh()
time.sleep(5)
```

Finally, the MagTag enters a deep sleep until midnight after calculating how many seconds have passed since the previous midnight. With deep sleep, you can keep this project running on a battery for a long time since the power draw is minimal. Additionally, there is no need for a loop since the entire code will run again from the beginning when the MagTag awakens.

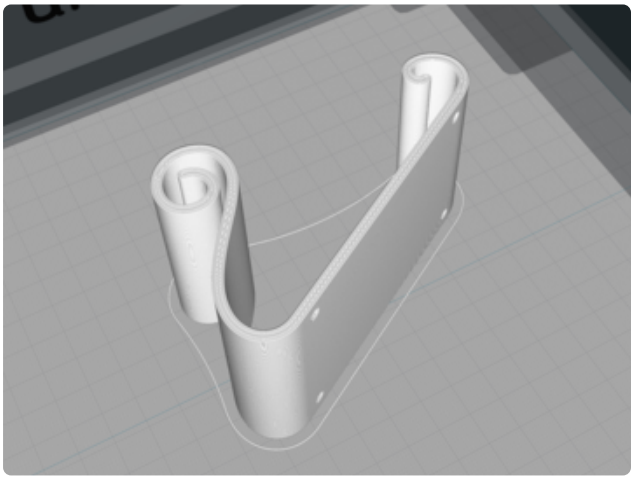
```
print("entering deep sleep")
seconds_to_sleep = 24*60*60 - seconds_since_midnight + 10
print( f"sleeping for {seconds_to_sleep} seconds")
magtag.exit_and_deep_sleep(seconds_to_sleep)

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

3D Printing

Parts List

STL files for 3D printing are oriented to print "as-is" on FDM style machines. Parts are designed to 3D print without any support material. Original design source may be downloaded using the links below.

fancy-portrait.stl
fancy-landscape.stl



Slicing Parts

Slice with setting for PLA material. The parts were sliced using CURA using the slice settings below.

PLA filament
215c extruder
0.2 layer height
10% gyroid infill
60mm/s print speed
60c heated bed

Design Source Files

The project assembly was designed in Fusion 360. This can be downloaded in different formats like STEP, STL and more. Electronic components like Adafruit's board, displays, connectors and more can be downloaded from the Adafruit CAD parts GitHub Repo (https://adafru.it/AW8).

Download STLs

https://adafru.it/PbE

Download CAD files

https://adafru.it/PbF

Attach the MagTag to the Stand

Use two M3 x 8mm screws to secure the MagTag to the stand.