



Haiku Viewer for MagTag

Created by Tim C



<https://learn.adafruit.com/haiku-viewer-for-magtag>

Last updated on 2026-01-26 08:19:53 PM UTC

Table of Contents

Overview	3
<ul style="list-style-type: none">• Parts	
Install CircuitPython	4
<ul style="list-style-type: none">• 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	
Create Your settings.toml File	9
<ul style="list-style-type: none">• CircuitPython settings.toml File• settings.toml File Tips• Accessing Your settings.toml Information in code.py	
Code	12
<ul style="list-style-type: none">• Getting the Program's Files• Drive Structure• Code	
Code Explanation	17
<ul style="list-style-type: none">• Loading Haikus• Displayio Visual Elements• Button Debouncers & User Interaction	
Use	19
<ul style="list-style-type: none">• Local haikus.txt file• Adafruit IO Haikus Feed	

Overview



Bring some Zen to your life with this MagTag Haiku Viewer. Load your favorite haikus in a local text file, or use an AdafruitIO feed for easy remote management capabilities. When the MagTag boots up, a random haiku is selected and shown. The right and left buttons cycle through all of the available haikus forward and backwards respectively. The D12 button selects a new haiku to show at random.

Are haikus not your vibe? No problem, the code for this project can be easily repurposed to show jokes, quotes, or anything else you like.

Parts



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

The Adafruit MagTag combines the 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 is...

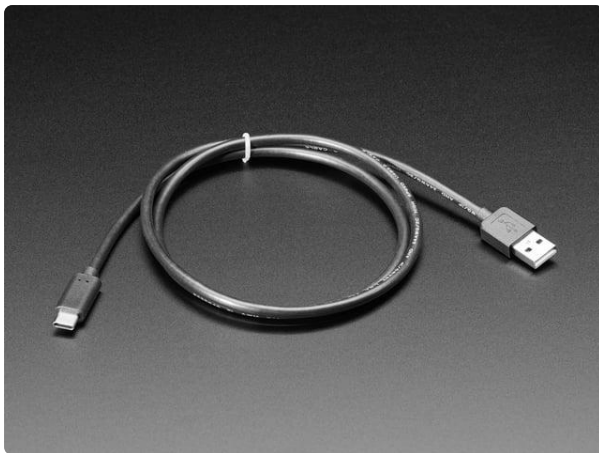
<https://www.adafruit.com/product/4800>



Mini Magnet Feet for RGB LED Matrices (Pack of 4)

Got a glorious RGB Matrix project you want to mount and display in your workspace or home? If you have one of the matrix panels listed below, you'll need a pack of these...

<https://www.adafruit.com/product/4631>



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



Be sure that you [update the TinyUF2 Bootloader](#) before installing CircuitPython!



Adafruit MagTag

By Kattni Rembor

▮ [Update TinyUF2 Bootloader for CircuitPython 10 and Later](#)

<https://learn.adafruit.com/adafruit-magtag/update-tinyuf2-bootloader-for-circuitpython-10-4mb-boards-only>

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

Set Up CircuitPython

Follow the steps to get CircuitPython installed on your MagTag.

<https://adafru.it/OBd>



WARNING: The updated Adafruit MagTag 2025 Edition will not work with CircuitPython 9.2.x or earlier. Make sure you install 10.x.x or later!

CircuitPython 9.2.8

This is the latest **stable** release of CircuitPython that will work with the MagTag - 2.9" Grayscale E-Ink WiFi Display. **Use this release** if you are new to CircuitPython.

WARNING: The updated Adafruit MagTag 2025 Edition will not work with Circuitpython 9.2.x or earlier. Use 10.0.0-beta.1 or later, downloaded from below.

Release Notes for 9.2.8

ENGLISH (US) [dropdown]

DOWNLOAD .UF2 NOW [download icon]

DOWNLOAD .BIN NOW [download icon]

OPEN INSTALLER [external link icon]

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

You can use a 9.x.x release for a pre-2025 MagTag. You **must** use a 10.x.x release for the updated MagTag 2025 Edition.

(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).

CircuitPython 10.0.0-beta.2

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

WARNING: On Espressif ESP32-S2 and ESP32-S3 boards with 4MB flash, CircuitPython 10.0.0-beta.0 and later require TinyUF2 bootloader version 0.33.0 or later. Older TinyUF2 bootloaders don't provide enough room for the firmware and cannot load it. See the [Release Notes](#) for more details, and see [Update UF2 Bootloader](#) below.

Alpha development releases are early releases. They are unfinished, are likely to have bugs, and the features they provide may change. **Beta** releases may have some bugs and unfinished features, but should be suitable for many uses. A **Release Candidate (rc)** release is considered done and will become the next stable release, assuming no further issues are found.

Please try alpha, beta, and rc releases if you are able. Your testing is invaluable: it helps us uncover and find issues quickly.

Release Notes for 10.0.0-beta.2

ENGLISH (US) [dropdown]

DOWNLOAD .UF2 NOW [download icon]

DOWNLOAD .BIN NOW [download icon]

OPEN INSTALLER [external link icon]



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!

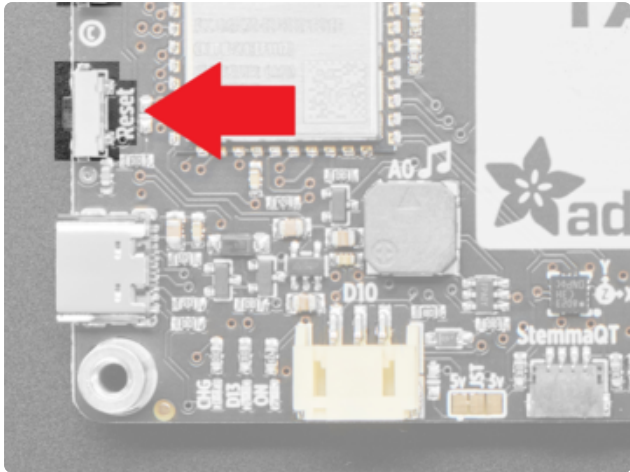


Be sure that you [update the TinyUF2 Bootloader](#) before following these steps for the UF2 bootloader!

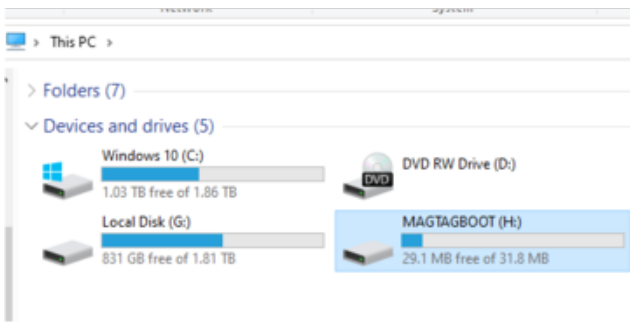


Try Launching UF2 Bootloader

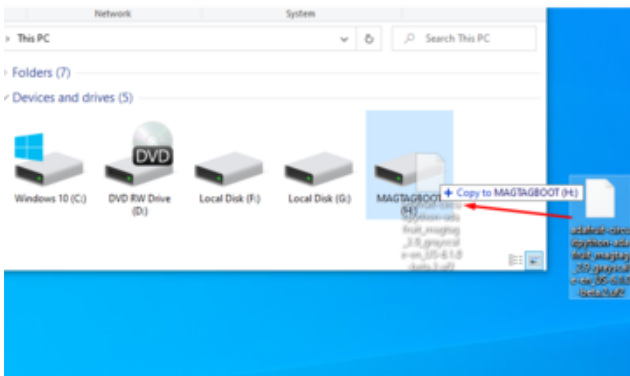
Loading CircuitPython by drag-n-drop UF2 bootloader is the easier way and we recommend it. If you have a MagTag where the front of the board is black, your MagTag came with UF2 already on it.



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.

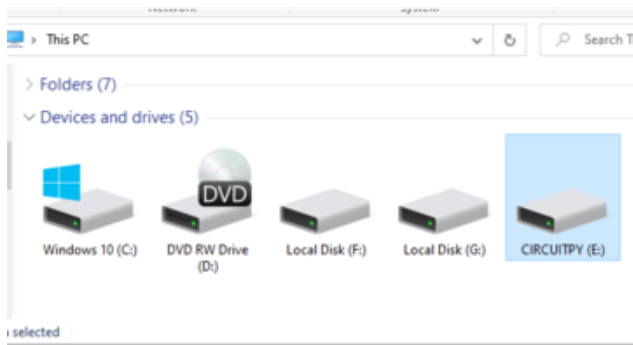


If the UF2 bootloader is installed, you will see a new disk drive appear called **MAGTAGBOOT**



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://adafruit.it/Pfk) (<https://adafruit.it/Pfk>)



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

Option 2 - Use esptool to load BIN file

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

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

```
root@kali:~/circuitpython# esptool -i python ./esptool.py --port /dev/ou.usbmodem01 --afterno.reset
write_flash 0x0 - /adafruit-circuitpython-adafruit_metro_esp32s2-an_US-20201103-5a87925.bin
esptool.py v3.0-dev
Serial port /dev/ou.usbmodem01
Connecting...
Detecting chip type... ESP32-S2
Chip is ESP32-S2
Features: WiFi, ADC and temperature sensor calibration in BLK2 of efuse
Crystal is 40MHz
MAC: 7c:df:a1:00:4a:a2
Uploading stub...
Running stub...
Stub running...
Configuring flash size...
Compressed 1305184 bytes to 844014...
Wrote 1305184 bytes (844014 compressed) at 0x00000000 in 11.9 seconds (effective 878.2 kbit/s)...
Hash of data verified.
Leaving...
Staying in bootloader.
```

Follow the initial steps found in the [Run esptool and check connection section of the ROM Bootloader page \(https://adafruit.com/OBc\)](https://adafruit.com/blog/2020/11/03/running-esptool-to-load-firmware-into-esp32s2/) 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 \(https://adafru.it/Pdq\)](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.

Create Your settings.toml File

CircuitPython works with WiFi-capable boards to enable you to make projects that have network connectivity. This means working with various passwords and API keys. As of [CircuitPython 8 \(https://adafru.it/Em8\)](https://adafru.it/Em8), there is support for a **settings.toml** file. This is a file that is stored on your **CIRCUITPY** drive, that contains all of your secret network information, such as your SSID, SSID password and any API keys for IoT services. It is designed to separate your sensitive information from your **code.py** file so you are able to share your code without sharing your credentials.

CircuitPython previously used a **secrets.py** file for this purpose. The **settings.toml** file is quite similar.



· **settings.toml** file should be stored in the main directory of your CIRCUITPY drive. It should not be in a folder.

CircuitPython **settings.toml** File

This section will provide a couple of examples of what your **settings.toml** file should look like, specifically for CircuitPython WiFi projects in general.

The most minimal **settings.toml** file must contain your WiFi SSID and password, as that is the minimum required to connect to WiFi. Copy this example, paste it into your **settings.toml**, and update:

- `your_wifi_ssid`
- `your_wifi_password`

```
CIRCUITPY_WIFI_SSID = "your_wifi_ssid"
CIRCUITPY_WIFI_PASSWORD = "your_wifi_password"
```

Many CircuitPython network-connected projects on the Adafruit Learn System involve using Adafruit IO. For these projects, you must also include your Adafruit IO username and key. Copy the following example, paste it into your **settings.toml** file, and update:

- `your_wifi_ssid`
- `your_wifi_password`
- `your_aio_username`
- `your_aio_key`

```
CIRCUITPY_WIFI_SSID = "your_wifi_ssid"
CIRCUITPY_WIFI_PASSWORD = "your_wifi_password"
ADAFRUIT_AIO_USERNAME = "your_aio_username"
ADAFRUIT_AIO_KEY = "your_aio_key"
```

Some projects use different variable names for the entries in the **settings.toml** file. For example, a project might use `ADAFRUIT_AIO_ID` in the place of

`ADAFRUIT_AIO_USERNAME`. If you run into connectivity issues, one of the first things to check is that the names in the `settings.toml` file match the names in the code.



every project uses the same variable name for each entry in the `settings.toml` file! Always verify it matches the code.

settings.toml File Tips

Here is an example `settings.toml` file.

```
# Comments are supported
CIRCUITPY_WIFI_SSID = "guest wifi"
CIRCUITPY_WIFI_PASSWORD = "guessable"
CIRCUITPY_WEB_API_PORT = 80
CIRCUITPY_WEB_API_PASSWORD = "passw0rd"
test_variable = "this is a test"
thumbs_up = "\U0001f44d"
```

In a `settings.toml` file, it's important to keep these factors in mind:

- Strings are wrapped in double quotes; ex: `"your-string-here"`
- Integers are **not** quoted and may be written in decimal with optional sign (`+1`, `-1`, `1000`) or hexadecimal (`0xabcd`).
 - Floats (decimal numbers), octal (`0o567`) and binary (`0b11011`) are not supported.
- Use `\u` escapes for weird characters, `\x` and `\ooo` escapes are not available in `.toml` files
 - Example: `\U0001f44d` for 👍 (thumbs up emoji) and `\u20ac` for € (EUR sign)
- Unicode emoji, and non-ASCII characters, stand for themselves as long as you're careful to save in "UTF-8 without BOM" format



When your **settings.toml** file is ready, you can save it in your text editor with the **.toml** extension.

Accessing Your **settings.toml** Information in **code.py**

In your **code.py** file, you'll need to **import** the **os** library to access the **settings.toml** file. Your settings are accessed with the **os.getenv()** function. You'll pass your settings entry to the function to import it into the **code.py** file.

```
import os
print(os.getenv("test_variable"))
```

```
CircuitPython REPL
code.py output:
this is a test

Code done running.

Press any key to enter the REPL. Use CTRL-D to reload.
```

In the upcoming CircuitPython WiFi examples, you'll see how the **settings.toml** file is used for connecting to your SSID and accessing your API keys.

Code

Getting the Program's Files

To use the application, you need to obtain **code.py** with the program, the **bamboo.bmp** image file, and optionally the **haikus.txt** data file to place on the MagTag **CIRCUITPY** drive.

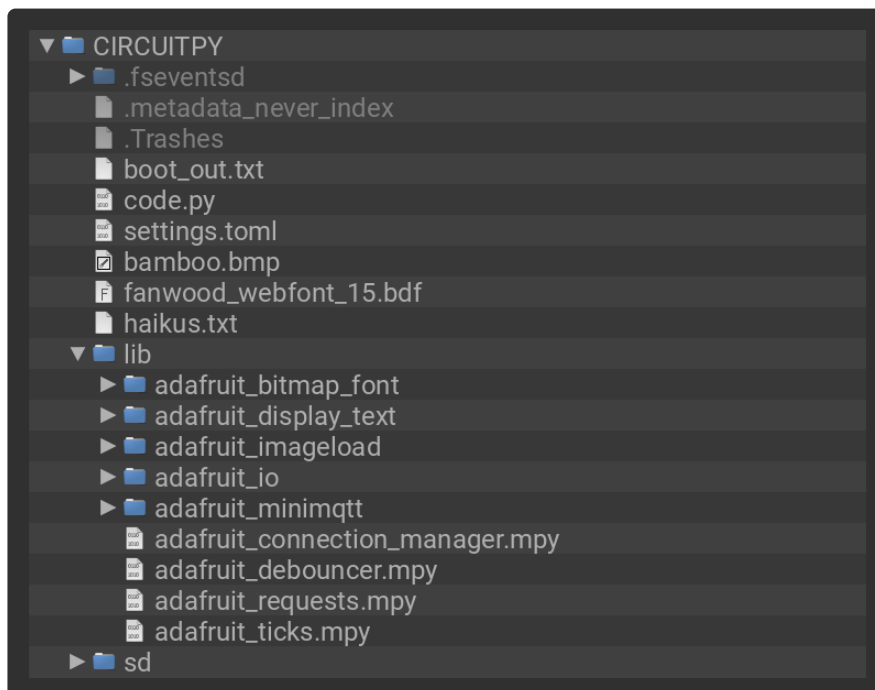
Thankfully, this can be done in one go. In the example below, click the **Download Project Bundle** button below to download the necessary libraries and the `code.py` file in a zip file.

Connect your board to your computer via a known good data+power USB cable. The board should show up in your File Explorer/Finder (depending on your operating system) as a flash drive named **CIRCUITPY**.

Extract the contents of the zip file, copy the **lib** directory files to **CIRCUITPY/lib**. Copy the **code.py**, **haikus.txt**, and **bamboo.bmp** file to the **CIRCUITPY** drive on the MagTag via your computer. The program should self start.

Drive Structure

After copying the files, your drive should look like the listing below. It can contain other files as well, but must contain these at a minimum.



Code

The `code.py` for the Haiku Viewer app is shown below.

```
# SPDX-FileCopyrightText: 2025 Tim C, written for Adafruit Industries
#
# SPDX-License-Identifier: MIT
"""
MagTag Haiku Viewer
```

Displays haikus loaded from the haikus.txt file and/or the AdafruitIO feed 'haikus'. Press left and right buttons to cycle through the available haikus. Press the D12 button to change to a new random haiku.

```
"""
import os
import random
import time

import bitmaptools
import board
import digitalio
import displayio
import supervisor
import wifi

from adafruit_io.adafruit_io import IO_HTTP, AdafruitIO_RequestError
from adafruit_bitmap_font import bitmap_font
import adafruit_connection_manager
from adafruit_debouncer import Debouncer
from adafruit_display_text.bitmap_label import Label
import adafruit_imageload
import adafruit_requests

# AdafruitIO connection. Will stay None if no WIFI or AIO credentials in
settings.toml
io = None

# list of all haikus collected from the local file and
# ones pulled from 'haikus' feed on Adafruit IO
all_haikus = []

# If the WIFI is auto connected from web workflow
if wifi.radio.connected:
    # Check for AdafruitIO credentials in settings.toml
    aio_username = os.getenv("ADAFRUIT_AIO_USERNAME")
    aio_key = os.getenv("ADAFRUIT_AIO_KEY")

    # if there are AIO credentials
    if None not in {aio_username, aio_key}:
        # Initialize connection_manager and requests
        pool = adafruit_connection_manager.get_radio_socketpool(wifi.radio)
        ssl_context = adafruit_connection_manager.get_radio_ssl_context(wifi.radio)
        requests = adafruit_requests.Session(pool, ssl_context)
        # Initialize an Adafruit IO HTTP API object
        io = IO_HTTP(aio_username, aio_key, requests)

# if the AdafruitIO connection is active
if io is not None:
    try:
        # Connect to the haikus IO feed
        haiku_feed = io.get_feed("haikus")
        # Retrieve data value from the feed
        print("Retrieving data from haiku feed...")
        haikus_data_from_io = io.receive_all_data(haiku_feed["key"])

        # add the fetched haikus to the list
        for haiku_data in haikus_data_from_io:
            # replace escaped double slash newlines with single slash newline
            all_haikus.append(haiku_data["value"].replace("\\n", "\n"))
    except AdafruitIO_RequestError as re:
        print("No haikus feed found on AdafruitIO. Using local haikus.txt file
only.")

# attempt to load hakus from local file
try:
    with open("haikus.txt", "r") as f:
```

```

    haiku_txt_file_content = f.read()

    # add all the haikus from the file to the list
    for haiku in haiku_txt_file_content.split("\n\n"):
        all_haikus.append(haiku)
except OSError as e:
    print("No haikus.txt file found.")

# make sure there is at least one haiku to show
if len(all_haikus) == 0:
    raise ValueError(
        "\nNo haikus were found.\nPlease add haikus to IO\nor the haikus file."
    )

print(f"Choosing from {len(all_haikus)} haikus")
# index of haiku to show. start on a random one
current_index = random.randint(0, len(all_haikus) - 1)

# get the selected haiku from the list
haiku = all_haikus[current_index]

# variable to hold the built-in eInk display reference
display = supervisor.runtime.display

# main group to hold all other visual elements
main_group = displayio.Group()

# background group used for white background behind the quote
# scale 8x to save memory on the Bitmap
bg_group = displayio.Group(scale=8)

# Create & append Bitmap for the white background
bg_bmp = displayio.Bitmap(display.width // 8, display.height // 8, 1)
bg_palette = displayio.Palette(1)
bg_palette[0] = 0xFFFFFF
bg_tg = displayio.TileGrid(bg_bmp, pixel_shader=bg_palette)
bg_group.append(bg_tg)
main_group.append(bg_group)

# Bamboo image element across the bottom
bamboo_bmp, bamboo_palette = adafruit_imageload.load("bamboo.bmp")
bamboo_tg = displayio.TileGrid(bitmap=bamboo_bmp, pixel_shader=bamboo_palette)
bamboo_tg.y = display.height - bamboo_tg.tile_height - 5
main_group.append(bamboo_tg)

# Grey border around the edges of the display
border_group = displayio.Group(scale=4)
border_bmp = displayio.Bitmap(display.width // 4, display.height // 4, 1)
border_palette = displayio.Palette(2)
border_palette[0] = 0x999999
border_palette[1] = 0xFFFFFF
border_palette.make_transparent(1)
bitmaptools.fill_region(
    border_bmp, 1, 1, border_bmp.width - 1, border_bmp.height - 1, 1
)
border_tg = displayio.TileGrid(border_bmp, pixel_shader=border_palette)
border_group.append(border_tg)
main_group.append(border_group)

# load the custom font & initialize Label to show haiku
FONT = bitmap_font.load_font("fanwood_webfont_15.bdf", displayio.Bitmap)
haiku_lbl = Label(FONT, text=haiku, scale=1, color=0x333333, line_spacing=1.0)
haiku_lbl.anchor_point = (0, 0)
haiku_lbl.anchored_position = (8, 0)
main_group.append(haiku_lbl)

# set up left button pin and debouncer
left_btn_pin = digitalio.DigitalInOut(board.D15)

```

```

left_btn_pin.direction = digitalio.Direction.INPUT
left_btn_pin.pull = digitalio.Pull.UP
left_btn = Debouncer(left_btn_pin)

# set up right button pin and debouncer
right_btn_pin = digitalio.DigitalInOut(board.D11)
right_btn_pin.direction = digitalio.Direction.INPUT
right_btn_pin.pull = digitalio.Pull.UP
right_btn = Debouncer(right_btn_pin)

# setup D12 button and debouncer
random_btn_pin = digitalio.DigitalInOut(board.D12)
random_btn_pin.direction = digitalio.Direction.INPUT
random_btn_pin.pull = digitalio.Pull.UP
random_btn = Debouncer(random_btn_pin)

# set main_group as root_group to show on the display
display.root_group = main_group

def refresh_display():
    """
    Wait until the display is ready to refresh,
    and then call refresh() on it.
    :return:
    """
    time.sleep(display.time_to_refresh)
    display.refresh()

# initial refresh to show the first haiku
refresh_display()

# main loop
while True:
    # update the button debouncers
    right_btn.update()
    left_btn.update()
    random_btn.update()

    # if the right button was pressed
    if right_btn.fell:
        print("right button pressed")
        # increment the current index
        current_index += 1
        # wrap around if past len of the list
        if current_index >= len(all_haikus):
            current_index = 0
        # get the new haiku
        haiku = all_haikus[current_index % len(all_haikus)]
        # set the new haiku text on the Label
        haiku_lbl.text = haiku
        refresh_display()

    # if the left button was pressed
    if left_btn.fell:
        print("left button pressed")
        # decrement the current index
        current_index -= 1
        # wrap around if below 0
        if current_index < 0:
            current_index = len(all_haikus) - 1
        # get the new haiku
        haiku = all_haikus[current_index % len(all_haikus)]
        # set the new haiku text on the Label
        haiku_lbl.text = haiku
        refresh_display()

    # if the (D12) random button was pressed

```

```

if random_btn.fell:
    # ensure at least 2 haikus for random function
    if len(all_haikus) > 1:
        # choose a random haiku, different from currently showing one
        old_index = current_index
        while current_index == old_index:
            current_index = random.randint(0, len(all_haikus) - 1)

        # get the new haiku
        haiku = all_haikus[current_index % len(all_haikus)]
        # set the new haiku text on the Label
        haiku_lbl.text = haiku
        refresh_display()

```

Code Explanation

The code for the project is contained in a single **code.py** file. The code has sections for high level tasks. Each task is covered in more detail below.

- Loading haikus from the local file and Adafruit IO
- Set up displayio visual elements
- Set up debouncers for the relevant buttons
- Respond to user interaction when the buttons are pressed

Loading Haikus

The code first checks whether the device is connected to a WiFi network. The connection itself happens automatically as long as **settings.toml** contains valid network credential values in the **CIRCUITPY_WIFI_SSID** and **CIRCUITPY_WIFI_PASSWORD** keys. Next it ensures that there are values for **ADAFRUIT_AIO_USERNAME** and **ADAFRUIT_AIO_KEY**, if those values exist it initializes an HTTP connection to Adafruit IO. The Adafruit IO connection is then used to fetch the "haikus" feed.

The entries within the feed have an extra slash inserted into their newline escape sequences automatically by the Adafruit IO server, before adding them to the local list of haikus to show the double slash escape **"\\n"** is changed to a normal single slash newline **"\n"**. When set as text on the **Label**, the newline escapes will be where visual line-breaks appear.

The local text file **haikus.txt** is opened and the contents are **read()**. The content is split on double newlines **"\n\n"** to separate all of the haikus contained in the file. Each one is added to the local list of haikus to show.

The `current_index` stores the index of the haiku that is currently displayed on the elnk display. It gets set to a random index initially and updated in response to user interaction.

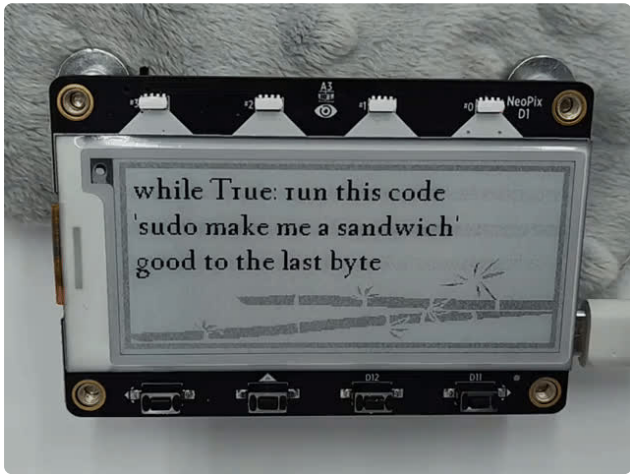
Displayio Visual Elements



The visual interface is comprised of 4 main components:

- The white background which uses `8x scale` in order to save memory by using a smaller Bitmap. It is placed behind all other visual elements.
- The grey bamboo image shown along the bottom of the display. It's loaded with the [adafruit_imageload](https://adafru.it/1ayF) (<https://adafru.it/1ayF>) library.
- The grey border drawn around the edges of the display. The [bitmaptools.fill_region\(\)](https://adafru.it/1ayG) (<https://adafru.it/1ayG>) function is used along with transparent color indexes within a `Palette` to "cut out" a rectangle from a full grey Bitmap leaving just the border visible.
- The `BitmapLabel` used to show the haiku. It uses a custom font loaded from the file `fanwood_webfont_15.bdf`. The font is one of the ones included in the [circuitpython_fonts bundle](https://adafru.it/1a1q) (<https://adafru.it/1a1q>).

Button Debouncers & User Interaction

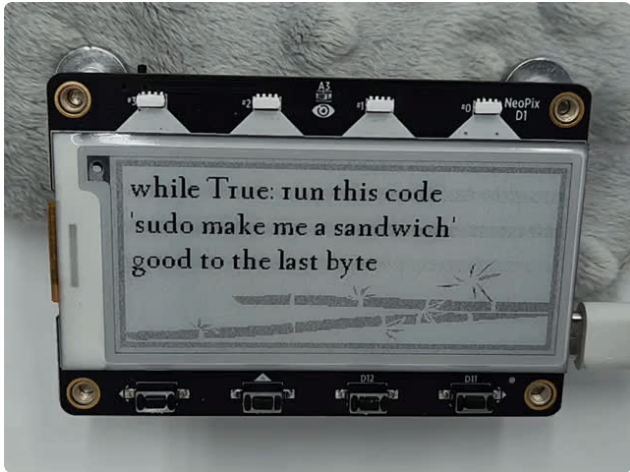


The [adafruit_debouncer](https://adafru.it/1ayH) (<https://adafru.it/1ayH>) library is used with the **D11**, **D12**, and **D15** button pins to determine when the user presses the buttons. During the main loop each debouncer instance is updated and then checked for the falling edge that indicates the button is pressed. The right and left buttons, pins **D11** and **D15**, will increment or decrement the current index and show the new haiku from the list. The down button on pin **D12** will select a different random haiku from the list to show, updating the current index to the randomly chosen one.

Use

There are two ways to load haikus on your MagTag. Write them in a local text file **haikus.txt** and put the file on the **CIRCUITPY** drive, or create a "**haikus**" feed on Adafruit IO and add them to feed. Depending on where you display your MagTag and how you power it, one or the other may be more convenient.

The code compiles the full list of haikus from both places so you can use either one or mix it up with some of each.



The right and left buttons, labeled **D11** and **D15**, will cycle through the list of available haikus moving to the next or previous one to show it on the display.



The down button, labeled **D12**, will select a different random haiku from the list and change to it.

Local haikus.txt file

The first way to load haikus is to put them in a text file named **haikus.txt** in the root of the **CIRCUITPY** drive. If you downloaded the project bundle includes a **haikus.txt** with a handful of electronics and coding themed haikus. The format of the file is fairly simple with each 3-line haiku being separated by a full blank line, see the truncated example below.

```
hot iron, cool bench
everything is organized
nirvana achieved

continuity?
the multimeter goes beep
testing continues

rainbow LEDs
blink and chase and undulate
example code.py
```

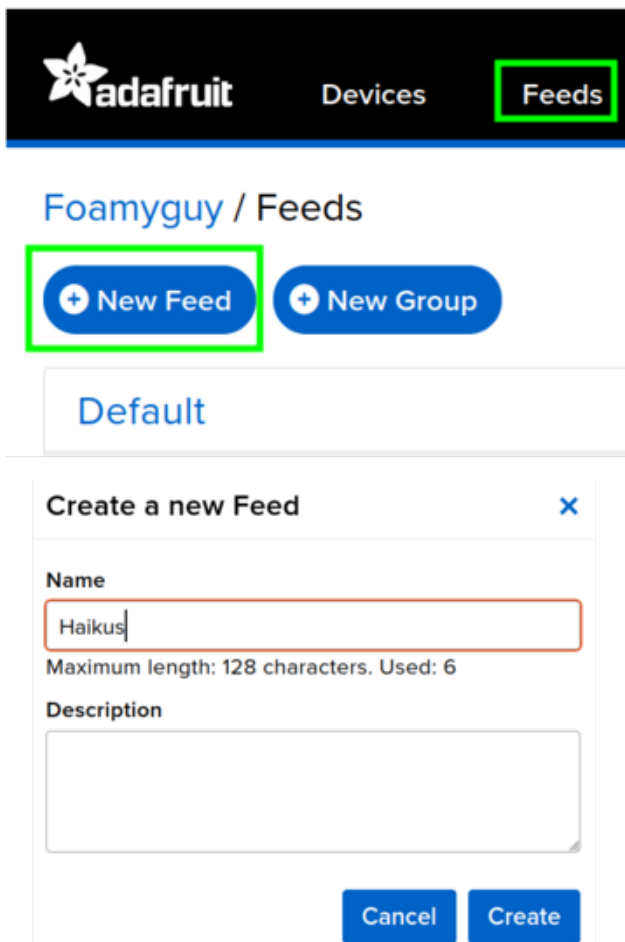
Add your favorite haikus to the file by making new lines at the bottom or inserted between the existing ones. Or you can start from a blank file and write your haikus

using the same syntax if you don't want to use the ones that come with the project bundle.

Adafruit IO Haikus Feed

The other option is to set up an Adafruit IO Feed named "Haikus" and then add each haiku as an entry in the feed.

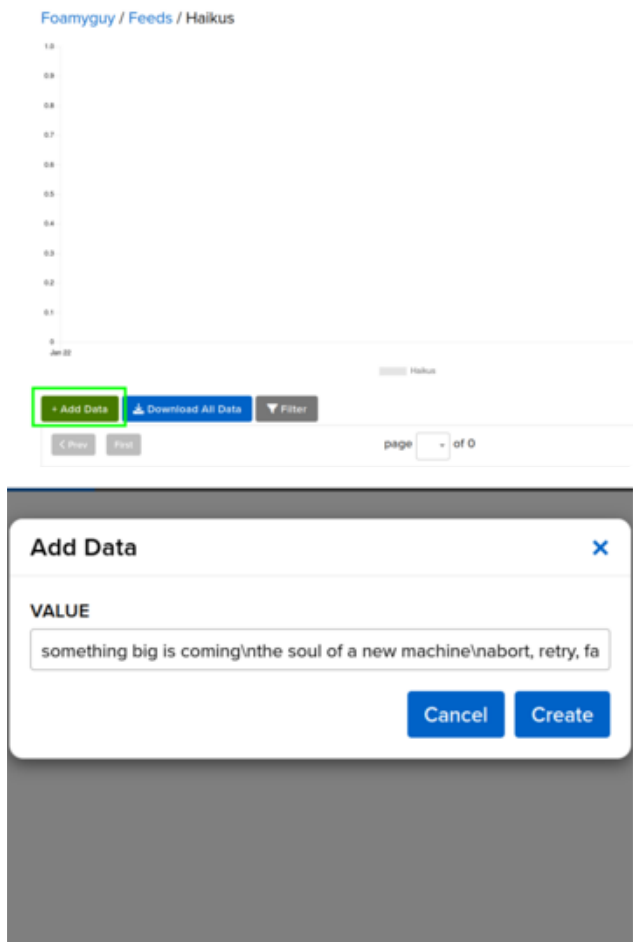
In order to load haikus from Adafruit IO, you must follow the steps on the [Create Your settings.toml page](#). You need to have all of: `ADAFRUIT_WIFI_SSID`, `ADAFRUIT_WIFI_PASSWORD`, `ADAFRUIT_AIO_USERNAME`, and `ADAFRUIT_AIO_KEY` in `settings.toml` with appropriate values for your WiFi network and AIO access.



Log in to [Adafruit IO \(https://adafru.it/fsU\)](https://adafru.it/fsU), click on "Feeds" in the navigation links then click on the "New Feed" button.

Enter the name "Haikus" for the new feed then press the **Create button**. The project code uses the feed name to fetch data so it is important to use the exact name "Haikus" or else it will not find the correct feed.

After the feed is created, **click on its name** in the list of feeds to go to the page for the new feed.



Click on the "Add Data" button to create a new entry in the feed.

Type or paste the lines of the haiku with `\n` in between each line. You must include the newline escape sequence `\n` in the haiku in order to get the lines to wrap at the appropriate points when displayed on the MagTag. For example:

```
The keyboard goes clack\nwriting code at 3AM\n time for espresso!
```

Once a haiku with appropriate newline escapes is entered, press the "Create" button.

Repeat the process to add as many haikus as you want.

The project code will fetch the data from this feed and add the haikus from it to the list to be shown by the app.