Raspberry Pi E-Ink Weather Station using Python

Created by Melissa LeBlanc-Williams

https://learn.adafruit.com/raspberry-pi-e-ink-weather-station-using-python
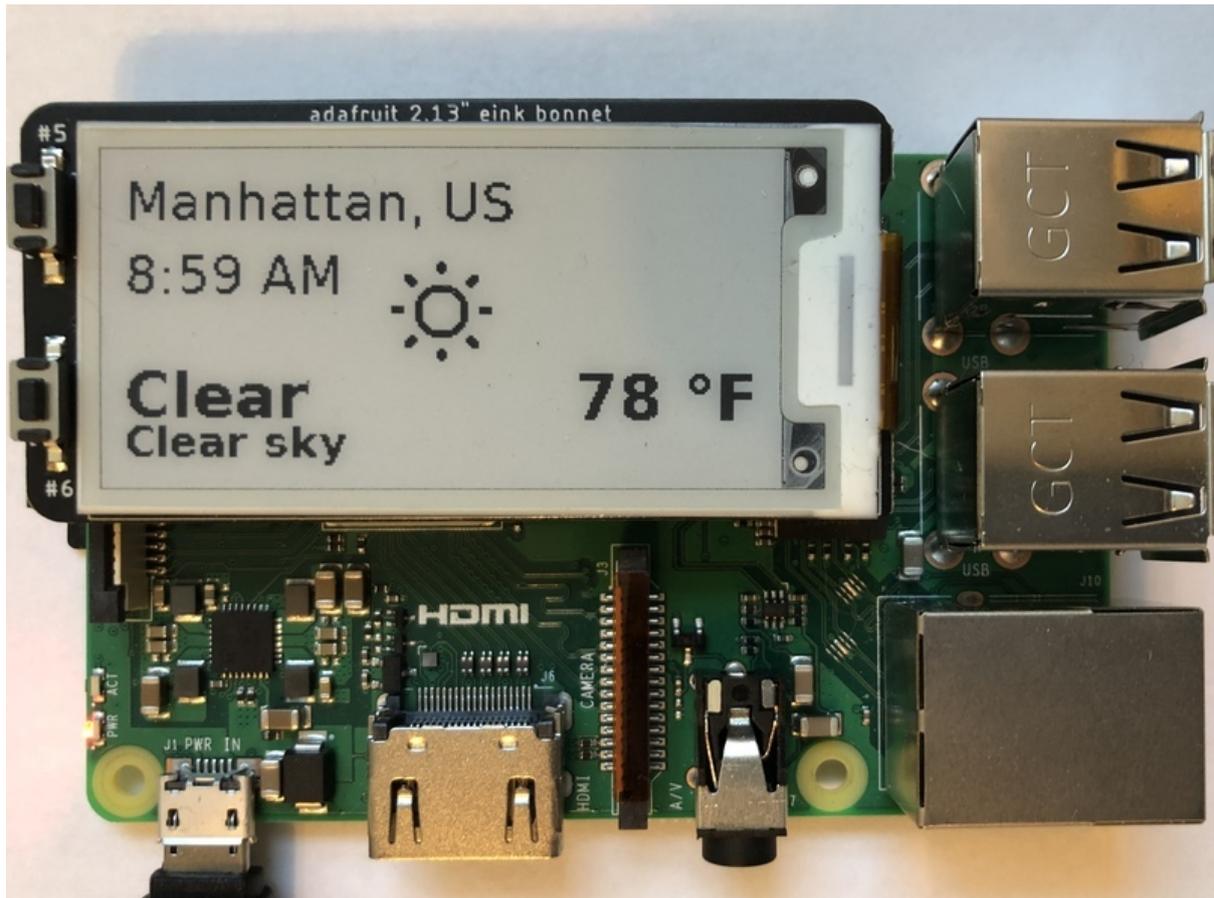
Last updated on 2024-11-01 02:02:18 PM EDT

Table of Contents

Overview
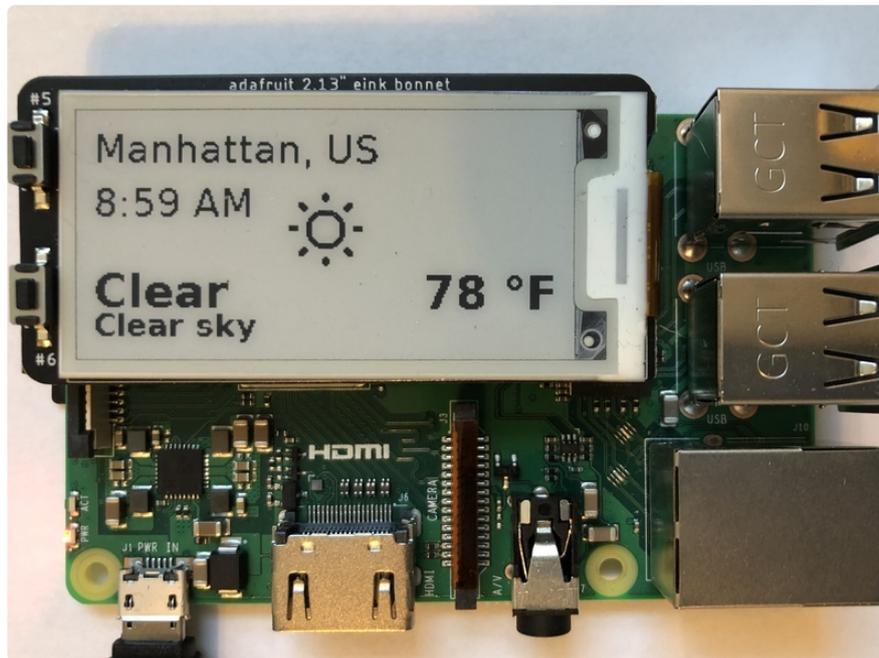
With this project, you will always know what the weather is. The eInk Bonnet or Breakout will always let you know what the weather is. If you lose power to your Raspberry Pi or other single board computer, you will still be able to see what the latest weather update is because the ePaper display still shows the last thing written to it!

Using Python, this project queries the Open Weather Maps site API to find out the current weather for your location and displays it along with an informative icon.

This project makes use of the meteocons True Type Font to display the icon, so there are no actual images used in this project.

Parts

To run this, you will need a Single Board Computer such as the Raspberry Pi.

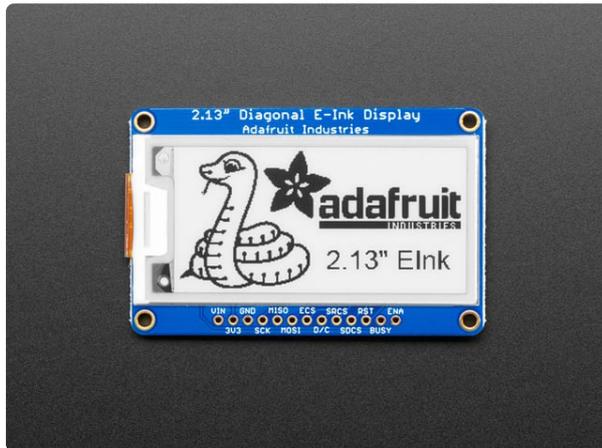Raspberry Pi 4 Model B - 4 GB RAM
The Raspberry Pi 4 Model B is the newest Raspberry Pi computer made, and the Pi Foundation knows you can always make a good thing better! And what could make the Pi 4 better...
https://www.adafruit.com/product/4296

You will need a 2.13" Monochrome eInk display such as the eInk Bonnet or the eInk Breakout.

Adafruit 2.13" Monochrome eInk / ePaper Display with SRAM
Easy e-paper finally comes to microcontrollers, with this breakout that's designed to make it a breeze to add a monochromatic eInk display. Chances are you've seen one of those...
https://www.adafruit.com/product/4197

If you use a breakout board, you will need a few additional parts. First you will need some tactile switches for buttons:

Tactile Button switch (6mm) x 20 pack
Little clicky switches are standard input "buttons" on electronic projects. These work best in a PCB but
https://www.adafruit.com/product/367

You will also needs a couple of 100K Resistors:
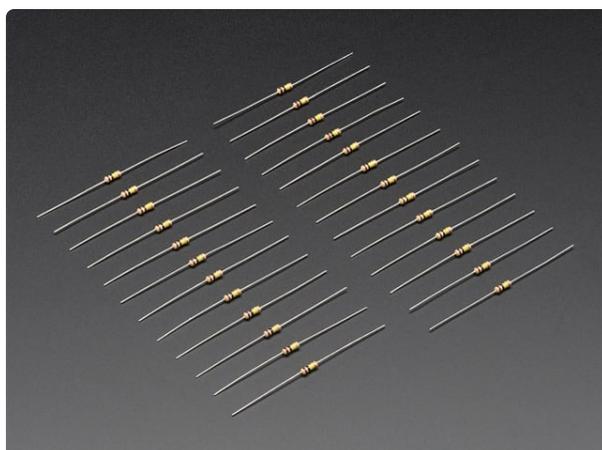
Through-Hole Resistors - 100K ohm 5% 1/4W - Pack of 25
ΩMG! You're not going to be able to resist these handy resistor packs! Well, axially, they do all of the resisting for you!This is a 25 Pack of 100K...
https://www.adafruit.com/product/2787

1 x Full sized breadboard   https://www.adafruit.com/product/239
Breadboard for assembling parts

(150mm)

Handy for making wire harnesses or jumpering between
headers on PCB's.

1 x Premium Male/Male Jumper Wires - 40 x 6"
(150mm)   https://www.adafruit.com/product/758

Handy for making wire harnesses or jumpering between
headers on PCB's.

---

1 x Stacking Header for Pi A+/B+/Pi 2/Pi 3 - 2x20
Extra Tall Header   https://www.adafruit.com/product/1979

Header for adding bonnets without interfering with Pi
cooling

Python Setup

Wiring

It's easy to use eInk breakouts and bonnets with Python and the Adafruit
CircuitPython EPD (https://adafru.it/BTd) module. This module allows you to easily
write Python code to control the display.

Using the eInk Bonnet

Since the eInk Bonnet comes preassembled, all you need to do is place it onto the
GPIO pins.

Since there's dozens of Linux computers/boards you can use we will show wiring for
Raspberry Pi. For other platforms, please visit the guide for CircuitPython on Linux to
see whether your platform is supported (https://adafru.it/BSN).

Connect the display as shown below to your Raspberry Pi.
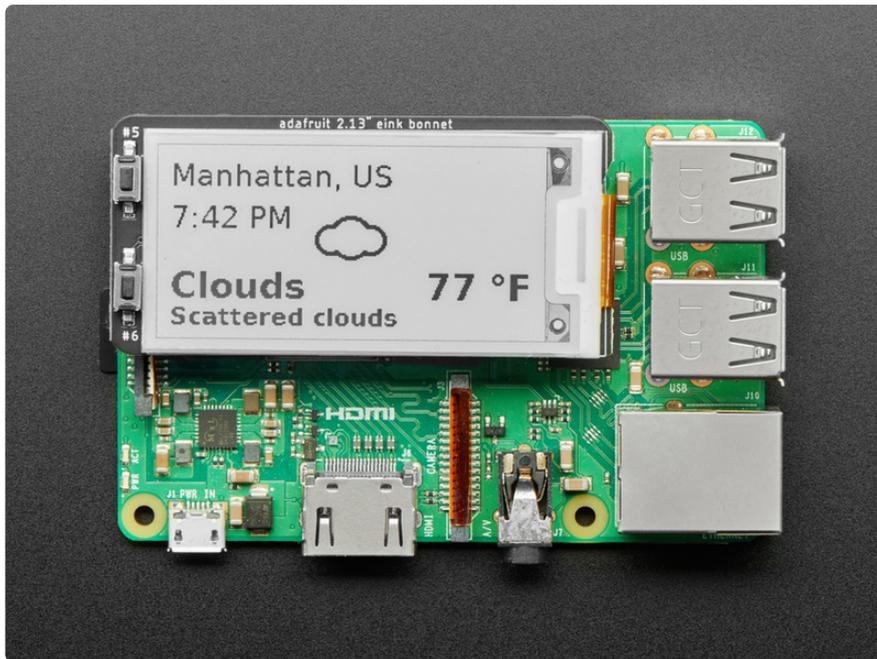
Using an eInk Breakout

Alternatively, if you have a 2.13" Monochrome eInk Display available, you can wire it up along with a couple of buttons and resistors. There's a lot of wires, which is why we recommend using the bonnet.
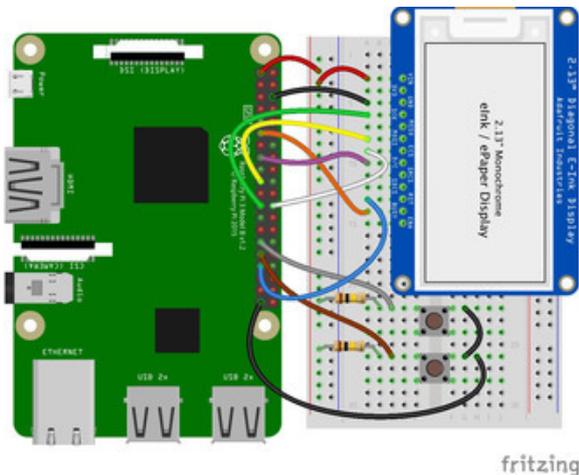
3V Rail connects to the Pi's **3V** pin

eInk **VIN** connects to the **3V Rail**

eInk **GND** connects to the Pi's **ground**

eInk **CLK** connects to SPI clock. On the Pi, thats **SLCK**

eInk **MOSI** connects to SPI MOSI. On the Pi, thats also **MOSI**

eInk **ECS** connects to our SPI Chip Select pin. We'll be using **CE0**

eInk **D/C** connects to our SPI Chip Select pin. We'll be using **GPIO 22**.

eInk **RST** connects to our Reset pin. We'll be using **GPIO 13**.

GPIO 5 connects to one side of a tactile switch.

GPIO 6 connects to one side of the other tactile switch.

Place a **100KΩ** resistor between **GPIO5** and the **+3V rail**.

Place a **100KΩ** resistor between **GPIO6** and the **+3V rail**.

Connect the other side of each tactile switch to the Pi's **ground**.

Note this is not a kernel driver that will let you have the console appear on the TFT. However, this is handy when you can't install an fbtft driver, and want to use the TFT purely from 'user Python' code!

You can only use this technique with Linux/computer devices that have hardware SPI support, and not all single board computers have an SPI device, so check before continuing

Software Setup

You'll need to install the Adafruit_Blinka library that provides the CircuitPython support in Python. This may also require enabling SPI on your platform and verifying

you are running Python 3. [Since each platform is a little different, and Linux changes often, please visit the CircuitPython on Linux guide to get your computer ready](https://adafru.it/BSN) (https://adafru.it/BSN)!

If you have already installed the kernel module, you will need to remove it by running the installer and choosing uninstall.

Python Installation of EPD Library

Once that's done, from your command line run the following command:

- `sudo pip3 install adafruit-circuitpython-epd`

If your default Python is version 3 you may need to run 'pip' instead. Just make sure you aren't trying to use CircuitPython on Python 2.x, it isn't supported!

If that complains about pip3 not being installed, then run this first to install it:

- `sudo apt-get install python3-pip`

DejaVu TTF Font

Raspberry Pi usually comes with the DejaVu font already installed, but in case it didn't, you can run the following to install it:

- `sudo apt-get install fonts-dejavu`

This package was previously calls **ttf-dejavu**, so if you are running an older version of Raspberry Pi OS, it may be called that.

Pillow Library

We also need PIL, the Python Imaging Library, to allow graphics and using text with custom fonts. There are several system libraries that PIL relies on, so installing via a package manager is the easiest way to bring in everything:

- `sudo apt-get install python3-pil`

That's it. You should be ready to go!

Weather Station Code

Open Weather Map API Key

We'll be using OpenWeatherMap.org to retrieve the weather info through its API. In order to do so, you'll need to register for an account and get your API key.

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

Python Code

The code for this project was based off the code available in the PyPortal Weather Station (https://adafru.it/LRF) guide, though the graphics portion of this project works quite differently. Go ahead and download the project and we'll go over the code.

Download Weather Station Project

https://adafru.it/Obv

To start with, look for this line in the code and be sure to enter your `OPEN_WEATHER_T OKEN` and set `LOCATION` variable to your location:

```
# You'll need to get a token from openweathermap.org, looks like:
# 'b6907d289e10d714a6e88b30761fae22'
OPEN_WEATHER_TOKEN = ""
```

Next, look for this line in the code and set the `LOCATION` variable to your location:

```
# Use cityname, country code where countrycode is ISO3166 format.
# E.g. "New York, US" or "London, GB"
LOCATION = "Manhattan, US"
```

To run the example, upload **code.py**, **weather_graphics.py**, and **meteocons.ttf** to the same directory and use the following command:

`python3 code.py`

You should see an output similar to this:
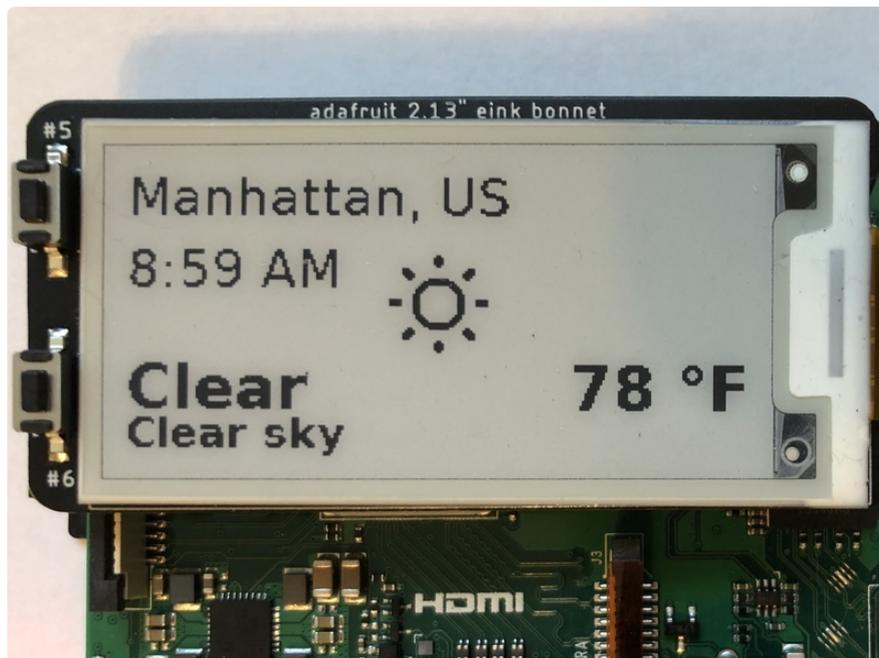
Automation Changes

If you would like to run the weather script in a Cron job or to run on start up, you may find it easier to move the **meteocons.ttf** file to a more central place.

A good place to put it would be to create a folder named **meteocons** inside of in **/usr/ share/fonts/truetype** and then move the **meteocons.ttf** file inside of that folder.

You'll also need to change the location of the file inside of **weather_graphics.py**.

How It Works

There are two files that are used. Let's start by going over the main file, which is the one you will use to to run the example.

Main File

First we start by loading any libraries that are used. The notable libraries are **urllib**, **adafruit_epd**, **weather_graphics**. The library called **urllib** is a built-in library used for retrieving and parsing data from the internet. The **adafruit_epd** library is used to initialize and write data to the ePaper Display, and the **weather_graphics** is the portion of code that will handle any graphics, which we'll go over next.

```
import time
import urllib.request
import urllib.parse
import digitalio
import busio
import board
from adafruit_epd.ssd1680 import Adafruit_SSD1680Z
from adafruit_epd.ssd1675 import Adafruit_SSD1675
```

```
from adafruit_epd.ssd1680 import Adafruit_SSD1680
from weather_graphics import Weather_Graphics
```

Next, we setup SPI and any pins used. If you are using the EInk bonnet or have wired it up like in the setup page, you shouldn't need to change anything unless you are using a different board than the Raspberry Pi.

```
spi = busio.SPI(board.SCK, MOSI=board.MOSI, MISO=board.MISO)
ecs = digitalio.DigitalInOut(board.CE0)
dc = digitalio.DigitalInOut(board.D22)
rst = digitalio.DigitalInOut(board.D27)
busy = digitalio.DigitalInOut(board.D17)
```

The next section contains the parameters used for connecting to Open Weather. You should have already filled in the token and location, and if not, take a look at the beginning of this page.

```
# You'll need to get a token from openweathermap.org, looks like:
# 'b6907d289e10d714a6e88b30761fae22'
OPEN_WEATHER_TOKEN = ""

# Use cityname, country code where countrycode is ISO3166 format.
# E.g. "New York, US" or "London, GB"
LOCATION = "Manhattan, US"
DATA_SOURCE_URL = "http://api.openweathermap.org/data/2.5/weather"
```

The next section just checks to make sure you've added the token and assembles the URL with the parameters.

```
if len(OPEN_WEATHER_TOKEN) == 0:
    raise RuntimeError(
        "You need to set your token first. If you don't already have one, you can
register for a free account at https://home.openweathermap.org/users/sign_up"
    )

# Set up where we'll be fetching data from
params = {"q": LOCATION, "appid": OPEN_WEATHER_TOKEN}
data_source = DATA_SOURCE_URL + "?" + urllib.parse.urlencode(params)
```

After that, we setup the ePaper display and set the rotation. If you are using an older eInk Bonnet, you will need to comment out the line for the newer bonnet and uncomment the one for the older bonnet.

```
# Initialize the Display
display = Adafruit_SSD1680Z(      # New Bonnet ssd1680z [GDEY0213B74]
#display = Adafruit_SSD1680(      # Old eInk Bonnet ssd1680
#display = Adafruit_SSD1675(    # Older eInk Bonnet ssd1675
#    122, 250, spi, cs_pin=ecs, dc_pin=dc, sramcs_pin=None, rst_pin=rst,
busy_pin=busy,
    120, 250, spi, cs_pin=ecs, dc_pin=dc, sramcs_pin=None, rst_pin=rst,
busy_pin=busy,

)
```

```
display.rotation = 1
```

Next we initialize the Graphics library and initialize the `weather_refresh` variable with a default value. The `weather_refresh` variable is used to determine the last time that the weather data was refreshed so that we can have 2 different timing loops for weather and time.

```
gfx = Weather_Graphics(display, am_pm=True, celsius=False)
weather_refresh = None
```

Finally, we have the main code loop. It checks to see if the weather data has been retrieved at all or that at least 600 seconds (10 minutes) have passed since the last time it was refreshed. If either condition is met, it uses `urllib` to get the data from the Open Weather URL. It then updates the time (as well as the display) using the weather graphics and waits for 300 seconds (5 minutes) so that the display isn't refreshed too often, which can cause the eInk displays to fail prematurely.

```
while True:
    # only query the weather every 10 minutes (and on first run)
    if (not weather_refresh) or (time.monotonic() - weather_refresh) &gt; 600:
        response = urllib.request.urlopen(data_source)
        if response.getcode() == 200:
            value = response.read()
            print("Response is", value)
            gfx.display_weather(value)
            weather_refresh = time.monotonic()
        else:
            print("Unable to retrieve data at {}".format(url))

    gfx.update_time()
    time.sleep(300)  # wait 5 minutes before updating anything again
```

Weather Graphics Library

Next we'll go over the graphics file. First we start by importing any libraries that we'll be using. The notable library this time is **PIL** or the Python Imaging library, which is used to handle all the font rendering.

```
from datetime import datetime
import json
from PIL import Image, ImageDraw, ImageFont
from adafruit_epd.epd import Adafruit_EPD
```

Next, we set up the fonts. If you would like to change them, you can do so here. Each font is a combination of the font file and the font size in points.

```
small_font = ImageFont.truetype(
    "/usr/share/fonts/truetype/dejavu/DejaVuSans-Bold.ttf", 16
)
medium_font = ImageFont.truetype("/usr/share/fonts/truetype/dejavu/DejaVuSans.ttf",
20)
```

```
large_font = ImageFont.truetype(
    "/usr/share/fonts/truetype/dejavu/DejaVuSans-Bold.ttf", 24
)
icon_font = ImageFont.truetype("./meteocons.ttf", 48)
```

Next we create an icon map, which will map the specified OpenWeather icon code (https://adafru.it/EcD) to the font character that the meteocons font (https://adafru.it/fNi) uses. If you wanted to change the icons, this would be the place to make changes.

```
# Map the OpenWeatherMap icon code to the appropriate font character
# See http://www.alessioatzeni.com/meteocons/ for icons
ICON_MAP = {
    "01d": "B",
    "01n": "C",
    "02d": "H",
    "02n": "I",
    "03d": "N",
    "03n": "N",
    "04d": "Y",
    "04n": "Y",
    "09d": "Q",
    "09n": "Q",
    "10d": "R",
    "10n": "R",
    "11d": "Z",
    "11n": "Z",
    "13d": "W",
    "13n": "W",
    "50d": "J",
    "50n": "K",
}
```

Next we define a couple of colors to make the code more readable:

```
# RGB Colors
WHITE = (255, 255, 255)
BLACK = (0, 0, 0)
```

Now we have the actual Weather_Graphics class:

```
class Weather_Graphics:
```

The first function in the Weather Graphics is code to initialize any variables that we'll be storing to default values including setting the fonts that we set up earlier.

```
def __init__(self, display, *, am_pm=True, celsius=True):
    self.am_pm = am_pm
    self.celsius = celsius

    self.small_font = small_font
    self.medium_font = medium_font
    self.large_font = large_font

    self.display = display

    self._weather_icon = None
```

```
        self._city_name = None
        self._main_text = None
        self._temperature = None
        self._description = None
        self._time_text = None
```

Next we have the `display_weather` function, which all it does is format the data that is passed in into something more readable to humans. As it does, it prints the value to the console to make debugging easier. Once everything is set, it calls the function to update the time.

```python
def display_weather(self, weather):
    weather = json.loads(weather.decode("utf-8"))

    # set the icon/background
    self._weather_icon = ICON_MAP[weather["weather"][0]["icon"]]

    city_name = weather["name"] + ", " + weather["sys"]["country"]
    print(city_name)
    self._city_name = city_name

    main = weather["weather"][0]["main"]
    print(main)
    self._main_text = main

    temperature = weather["main"]["temp"] - 273.15  # its...in kelvin
    print(temperature)
    if self.celsius:
        self._temperature = "%d °C" % temperature
    else:
        self._temperature = "%d °F" % ((temperature * 9 / 5) + 32)

    description = weather["weather"][0]["description"]
    description = description[0].upper() + description[1:]
    print(description)
    self._description = description
    # "thunderstorm with heavy drizzle"

    self.update_time()
```

The `update_time` function is pretty short. It just retrieves the time from the linux operating system using the **datetime** library, then it formats and sets the value to one of the class variables. Once that is done, it calls the `update_display` function.

```python
def update_time(self):
    now = datetime.now()
    self._time_text = now.strftime("%I:%M %p").lstrip("0").replace(" 0", " ")
    self.update_display()
```

The last function in the library is the `update_display` function, which is used to take the values of each of the class variables and draw them to a canvas. PIL's `Image.new()` function is used to create a canvas the size of the display and then the `draw.text()` function is used to draw the text to the canvas in various locations. Once all the text is drawn, the canvas is passed into the EPD library with the `self.display.image()` function and then the display is refreshed with the `self.display.display()` function.

```python
def update_display(self):
    self.display.fill(Adafruit_EPD.WHITE)
    image = Image.new("RGB", (self.display.width, self.display.height), color=WHITE)
    draw = ImageDraw.Draw(image)

    # Draw the Icon
    (font_width, font_height) = icon_font.getsize(self._weather_icon)
    draw.text(
        (
            self.display.width // 2 - font_width // 2,
            self.display.height // 2 - font_height // 2 - 5,
        ),
        self._weather_icon,
        font=icon_font,
        fill=BLACK,
    )

    # Draw the city
    draw.text(
        (5, 5), self._city_name, font=self.medium_font, fill=BLACK,
    )

    # Draw the time
    (font_width, font_height) = medium_font.getsize(self._time_text)
    draw.text(
        (5, font_height * 2 - 5),
        self._time_text,
        font=self.medium_font,
        fill=BLACK,
    )

    # Draw the main text
    (font_width, font_height) = large_font.getsize(self._main_text)
    draw.text(
        (5, self.display.height - font_height * 2),
        self._main_text,
        font=self.large_font,
        fill=BLACK,
    )

    # Draw the description
    (font_width, font_height) = small_font.getsize(self._description)
    draw.text(
        (5, self.display.height - font_height - 5),
        self._description,
        font=self.small_font,
        fill=BLACK,
    )

    # Draw the temperature
    (font_width, font_height) = large_font.getsize(self._temperature)
    draw.text(
        (
            self.display.width - font_width - 5,
            self.display.height - font_height * 2,
        ),
        self._temperature,
        font=self.large_font,
        fill=BLACK,
    )

    self.display.image(image)
    self.display.display()
```

Full Example Code

The first file is the main file:

```python
# SPDX-FileCopyrightText: 2020 Melissa LeBlanc-Williams for Adafruit Industries
#
# SPDX-License-Identifier: MIT

"""
This example queries the Open Weather Maps site API to find out the current
weather for your location... and display it on a eInk Bonnet!
"""

import time
import urllib.request
import urllib.parse
import digitalio
import busio
import board
from adafruit_epd.ssd1675 import Adafruit_SSD1675
from adafruit_epd.ssd1680 import Adafruit_SSD1680
from weather_graphics import Weather_Graphics

spi = busio.SPI(board.SCK, MOSI=board.MOSI, MISO=board.MISO)
ecs = digitalio.DigitalInOut(board.CE0)
dc = digitalio.DigitalInOut(board.D22)
rst = digitalio.DigitalInOut(board.D27)
busy = digitalio.DigitalInOut(board.D17)

# You'll need to get a token from openweathermap.org, looks like:
# 'b6907d289e10d714a6e88b30761fae22'
OPEN_WEATHER_TOKEN = ""

# Use cityname, country code where countrycode is ISO3166 format.
# E.g. "New York, US" or "London, GB"
LOCATION = "Manhattan, US"
DATA_SOURCE_URL = "http://api.openweathermap.org/data/2.5/weather"

if len(OPEN_WEATHER_TOKEN) == 0:
    raise RuntimeError(
        "You need to set your token first. If you don't already have one, you can
register for a free account at https://home.openweathermap.org/users/sign_up"
    )

# Set up where we'll be fetching data from
params = {"q": LOCATION, "appid": OPEN_WEATHER_TOKEN}
data_source = DATA_SOURCE_URL + "?" + urllib.parse.urlencode(params)

# Initialize the Display
display = Adafruit_SSD1680(      # Newer eInk Bonnet
# display = Adafruit_SSD1675(    # Older eInk Bonnet
    122, 250, spi, cs_pin=ecs, dc_pin=dc, sramcs_pin=None, rst_pin=rst,
busy_pin=busy,
)

display.rotation = 1

gfx = Weather_Graphics(display, am_pm=True, celsius=False)
weather_refresh = None

while True:
    # only query the weather every 10 minutes (and on first run)
    if (not weather_refresh) or (time.monotonic() - weather_refresh) > 600:
        response = urllib.request.urlopen(data_source)
        if response.getcode() == 200:
            value = response.read()
            print("Response is", value)
```

```
            gfx.display_weather(value)
            weather_refresh = time.monotonic()
        else:
            print("Unable to retrieve data at {}".format(url))

    gfx.update_time()
    time.sleep(300)  # wait 5 minutes before updating anything again
```

The other file that you will need is the graphics file:

```
# SPDX-FileCopyrightText: 2020 Melissa LeBlanc-Williams for Adafruit Industries
#
# SPDX-License-Identifier: MIT

from datetime import datetime
import json
from PIL import Image, ImageDraw, ImageFont
from adafruit_epd.epd import Adafruit_EPD

small_font = ImageFont.truetype(
    "/usr/share/fonts/truetype/dejavu/DejaVuSans-Bold.ttf", 16
)
medium_font = ImageFont.truetype("/usr/share/fonts/truetype/dejavu/DejaVuSans.ttf",
20)
large_font = ImageFont.truetype(
    "/usr/share/fonts/truetype/dejavu/DejaVuSans-Bold.ttf", 24
)
icon_font = ImageFont.truetype("./meteocons.ttf", 48)

# Map the OpenWeatherMap icon code to the appropriate font character
# See http://www.alessioatzeni.com/meteocons/ for icons
ICON_MAP = {
    "01d": "B",
    "01n": "C",
    "02d": "H",
    "02n": "I",
    "03d": "N",
    "03n": "N",
    "04d": "Y",
    "04n": "Y",
    "09d": "Q",
    "09n": "Q",
    "10d": "R",
    "10n": "R",
    "11d": "Z",
    "11n": "Z",
    "13d": "W",
    "13n": "W",
    "50d": "J",
    "50n": "K",
}

# RGB Colors
WHITE = (255, 255, 255)
BLACK = (0, 0, 0)


class Weather_Graphics:
    def __init__(self, display, *, am_pm=True, celsius=True):
        self.am_pm = am_pm
        self.celsius = celsius

        self.small_font = small_font
        self.medium_font = medium_font
        self.large_font = large_font

        self.display = display
```

```python
        self._weather_icon = None
        self._city_name = None
        self._main_text = None
        self._temperature = None
        self._description = None
        self._time_text = None

    def display_weather(self, weather):
        weather = json.loads(weather.decode("utf-8"))

        # set the icon/background
        self._weather_icon = ICON_MAP[weather["weather"][0]["icon"]]

        city_name = weather["name"] + ", " + weather["sys"]["country"]
        print(city_name)
        self._city_name = city_name

        main = weather["weather"][0]["main"]
        print(main)
        self._main_text = main

        temperature = weather["main"]["temp"] - 273.15  # its...in kelvin
        print(temperature)
        if self.celsius:
            self._temperature = "%d °C" % temperature
        else:
            self._temperature = "%d °F" % ((temperature * 9 / 5) + 32)

        description = weather["weather"][0]["description"]
        description = description[0].upper() + description[1:]
        print(description)
        self._description = description
        # "thunderstorm with heavy drizzle"

        self.update_time()

    def update_time(self):
        now = datetime.now()
        self._time_text = now.strftime("%I:%M %p").lstrip("0").replace(" 0", " ")
        self.update_display()

    def update_display(self):
        self.display.fill(Adafruit_EPD.WHITE)
        image = Image.new("RGB", (self.display.width, self.display.height),
color=WHITE)
        draw = ImageDraw.Draw(image)

        # Draw the Icon
        (font_width, font_height) = icon_font.getsize(self._weather_icon)
        draw.text(
            (
                self.display.width // 2 - font_width // 2,
                self.display.height // 2 - font_height // 2 - 5,
            ),
            self._weather_icon,
            font=icon_font,
            fill=BLACK,
        )

        # Draw the city
        draw.text(
            (5, 5), self._city_name, font=self.medium_font, fill=BLACK,
        )

        # Draw the time
        (font_width, font_height) = medium_font.getsize(self._time_text)
        draw.text(
            (5, font_height * 2 - 5),
```

```python
        self._time_text,
        font=self.medium_font,
        fill=BLACK,
    )

    # Draw the main text
    (font_width, font_height) = large_font.getsize(self._main_text)
    draw.text(
        (5, self.display.height - font_height * 2),
        self._main_text,
        font=self.large_font,
        fill=BLACK,
    )

    # Draw the description
    (font_width, font_height) = small_font.getsize(self._description)
    draw.text(
        (5, self.display.height - font_height - 5),
        self._description,
        font=self.small_font,
        fill=BLACK,
    )

    # Draw the temperature
    (font_width, font_height) = large_font.getsize(self._temperature)
    draw.text(
        (
            self.display.width - font_width - 5,
            self.display.height - font_height * 2,
        ),
        self._temperature,
        font=self.large_font,
        fill=BLACK,
    )

    self.display.image(image)
    self.display.display()
```