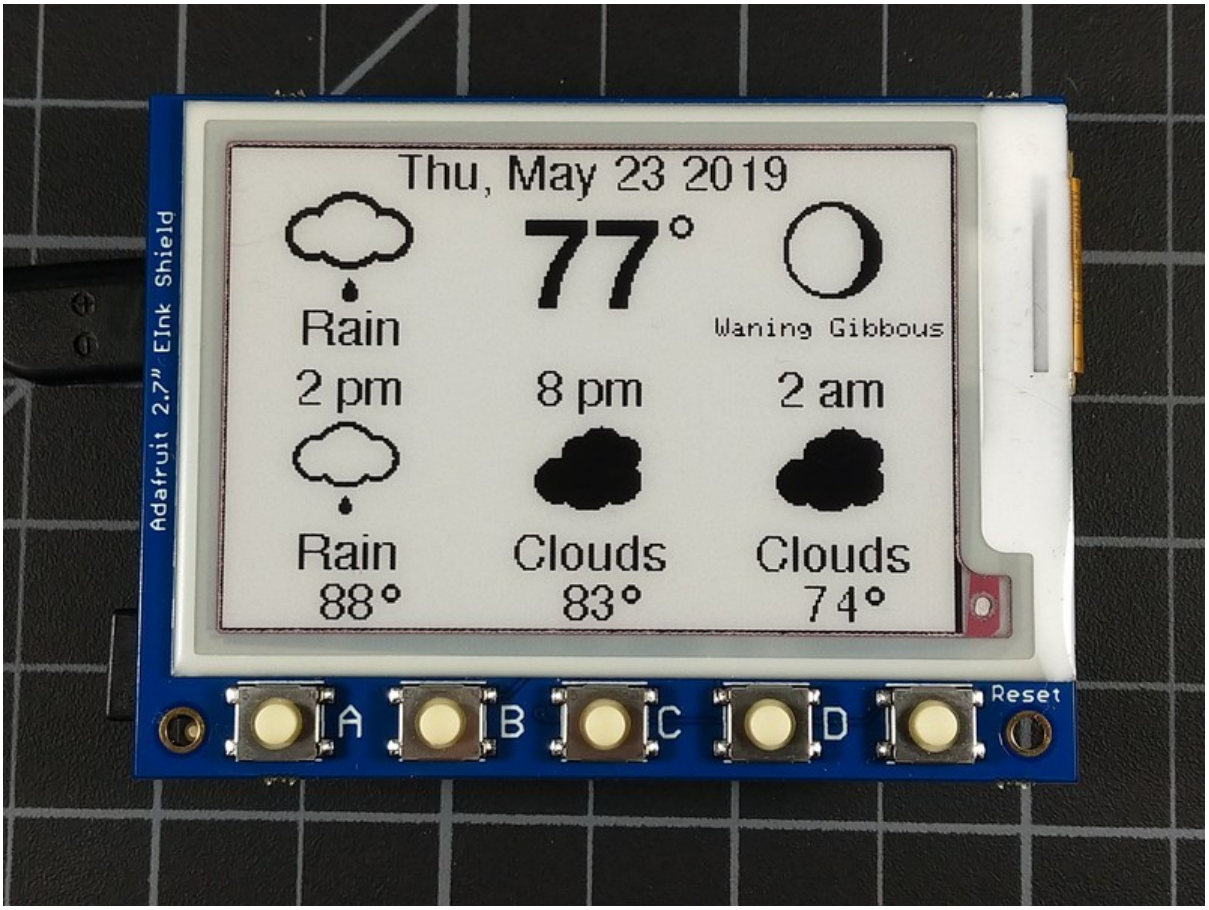
eInk / ePaper Weather Station

Created by Dan Cogliano



https://learn.adafruit.com/epaper-weather-station

Last updated on 2024-06-03 02:47:04 PM EDT

Table of Contents

Overview 3

- Features
- Parts

How it Works 5

Arduino Setup 7

- Installing The Libraries
- Signup For An OpenWeatherMap API key
- Download the Code and Modify The secrets.h File

Use 24

- Status LED

Overview

> So happy together
> How is the weather?
>
> The Turtles, "Happy Together"

In this project, the Adafruit Metro M4 Express Airlift and the Tri-Color ePaper Shield work happily together to create a weather station with the current weather conditions and forecast for your area.

A wonderful thing happened when the Metro M4 Express AirLift was released. It opened Arduino sketches to the Internet, freeing them from the confines of their closed environment. It is now officially a "thing" in a world of the Internet of Things (IoT). In this project, the Metro M4 Express AirLift will grab weather data for your local area from the internet and display it on an easy to read ePaper display

The Adafruit Tri-Color ePaper Shield used with this project is a 2.7" ePaper shield that displays black, white and red pixels. It easily connects to the Metro with no soldering needed since the headers are already assembled on both the Metro and the ePaper shield.

Features

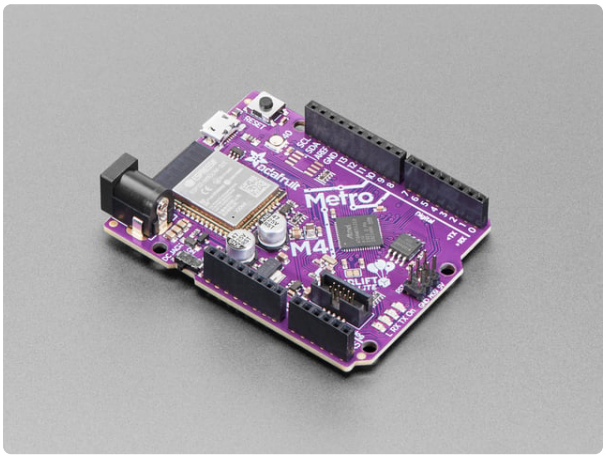
This DIY weather station displays weather information for your local area in either Celsius or Fahrenheit units on an ePaper display. The displayed information includes:

- Current date
- Current temperature
- Current weather conditions
- Forecast data for the next 12 hours in 6 hour increments
- Weather icons for weather conditions and forecasts
- City name
- Display current and forecasted hot temperatures in red
- Sunrise and sunset times
- Description and icon of current moon phase

Parts

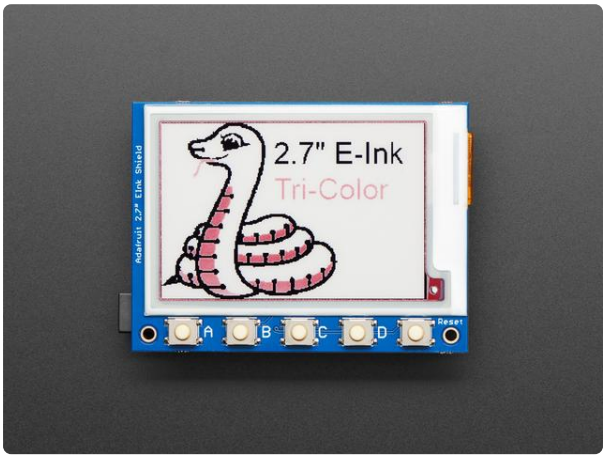
Building this project requires no soldering and uses just two parts: the Adafruit Metro M4 Express AirLift Lite and the Adafruit 2.7" Tri-Color eInk / ePaper Shield with SRAM. To make this project portable, you could add a USB battery pack (http://adafru.it/1959) or the Adafruit PowerBoost 500 Shield (http://adafru.it/2078) and a Li-Po battery and insert it between the Metro and the ePaper shield.

If you are interested in ePaper displays for other projects, check out the entire line of Adafruit's ePaper displays (https://adafru.it/ExU).



Adafruit Metro M4 Express AirLift (WiFi) - Lite

Give your next project a lift with AirLift - our witty name for the ESP32 co-processor that graces this Metro M4. You already know about the Adafruit Metro... https://www.adafruit.com/product/4000

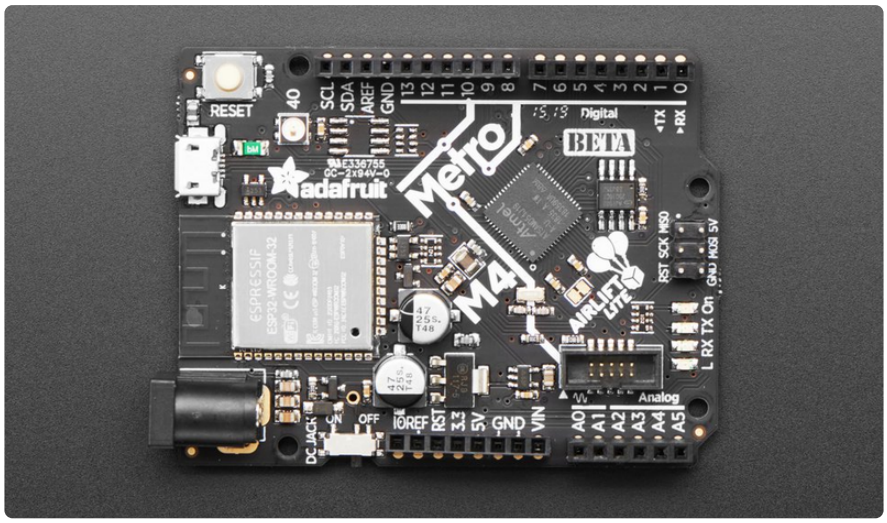


Adafruit 2.7" Tri-Color eInk / ePaper Shield with SRAM

Easy e-paper finally comes to microcontrollers, with this breakout that's designed to make it a breeze to add a tri-color eInk display. Chances are you've seen one of those... https://www.adafruit.com/product/4229

USB cable - USB A to Micro-B
This here is your standard A to micro-B USB cable, for USB 1.1 or 2.0. Perfect for connecting a PC to your Metro, Feather, Raspberry Pi or other dev-board or...
https://www.adafruit.com/product/592

How it Works



WIth the AirLift coprocessor board, this Metro M4 Express is going places! Well, at least for this sketch, it is going to get your local weather and display it on the ePaper shield. There are a number of sites with weather APIs that can get your local area's current conditions and upcoming forecast. A smaller number of these sites offer this data for free for non-commercial purposes. We went with the OpenWeatherMap.com API, which offers free weather data for current conditions and forecasts.

For weather and moon icons, we are taking advantage of the Adafruit GFX Library's support for custom fonts. A font is ideal for this type of display, as we need simple, line drawn icons in different point sizes that look nice on a monochrome ePaper display. For weather icons, we chose the Meteocon (https://adafru.it/fNi) font set, a free font containing over 40 weather icons. For the moon phase font, we went with the Moon Phases (https://adafru.it/EZf) font by Curtis Clark, which is free for non-commercial use. The font files need to be converted for use with the GFX Library, but don't worry, we have done that already in various point sizes so you can use these fonts for this and other projects.

Speaking of moon phases, the OpenWeatherMap does not include moon phases in their API. That is not a problem, however, as we used math to calculate the current moon phase. We know there is, on average, a new moon every 29.5305882 days. Grabbing the current date and time from OpenWeatherMap, and if we have a known point in time when a new moon occurred, we can calculate the moon phase for the current date.

The weather data from OpenWeatherMap includes the current time and times of future forecasts, like the weather forecast 3 hours from now. This time is in UTC format. We need to convert it to local time so the times will display accurately. Fortunately, OpenWeatherMap recently added the local timezone offset to the data (actually, as we were writing this guide!) so the local time can now be easily determined by taking the current time in UTC and adding the local timezone offset.

The next step is to put it all together: getting the weather data and formatting the display. We modified a portion of the code from the excellent Weather Station Project (https://adafru.it/EZg) by Daniel Eichhorn so it was not exclusive for any particular microcontroller. This allows other processors, including the Airlift coprocessor, to retrieve the weather data from the Internet.

Arduino Setup

If you don't have it already, you will need the Arduino IDE installed on your computer to upload this sketch to the Metro M4 Express AirLift. You will find information on installing Arduino in this learning guide (https://adafru.it/CfF). You will also need to configure the Metro M4 Express AirLift board to work with the Arduino IDE. There is a great article on setting up this board (https://adafru.it/EZh) with Arduino and other environments in the Adafruit Learning Guides site (https://adafru.it/dlu).

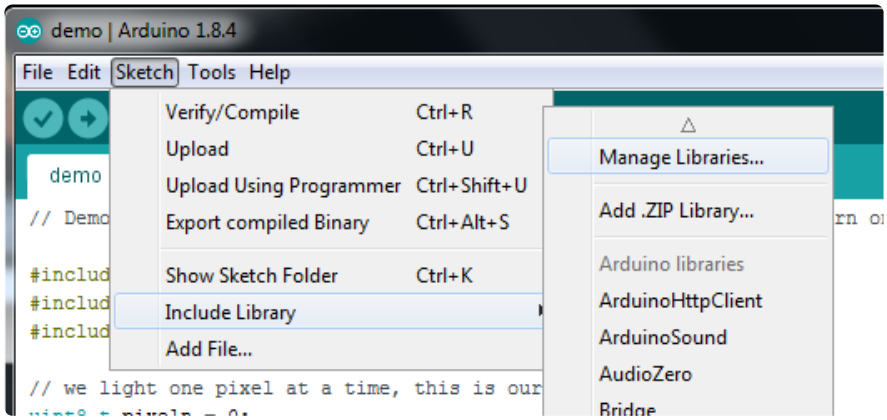
Installing The Libraries

For this sketch, you will need to install these libraries:

- Adafruit EPD (ePaper display) library
- Adafruit GFX library
- Adafruit BusIO
- Adafruit NeoPixel library
- Arduino JSON library
- Adafruit variant of the WiFiNINA library

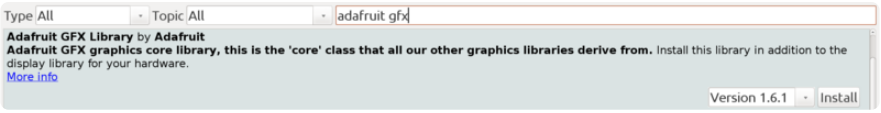
These libraries should be in the Arduino Library Manager in the latest Arduino IDE (1.8.7 and above).

Open up the Arduino library manager:



Search for the **Adafruit EPD** library and install it

Search for the **Adafruit GFX** library and install it



If using an earlier version of the Arduino IDE (prior to 1.8.10), also locate and install **Adafruit_BusIO** (newer versions will install this dependency automatically).

Search for the **Adafruit NeoPixel** library and install it

Search for the **ArduinoJSON** library and install it

You'll have to install WiFiNina manually with the link below

Download Adafruit's version of WiFiNina

https://adafru.it/Evm

Signup For An OpenWeatherMap API key

You will need an API key from OpenWeatherMap in order to download weather data from their site. You can signup for free at OpenWeatherMap.org (https://adafru.it/EZj) . A free account allows access to the current weather and 5 day / 3 hour weather API's.

Download the Code and Modify The secrets.h File

The download is available on GitHub. It contains code and font files for the weather and moon phase icons.

The **secrets.h** must be modified to include your WiFi settings, your OpenWeatherMap API key, and the city where you want the weather data. You may need to include your 2 letter country code if your city name occurs in more than one country, like Venice, Italy and Venice, Florida. So, for example, Venice, Florida would be entered as "Venice, US".

Other options available in this file include the choice of metric or English units for temperature and the language for the weather descriptions.

```
// SPDX-FileCopyrightText: 2019 Dan Cogliano for Adafruit Industries
//
// SPDX-License-Identifier: MIT

#pragma once

// secrets.h
// Define your WIFI and OpenWeatherMap API key and location in this file

#define WIFI_SSID "{wifi ssid}"
#define WIFI_PASSWORD "{wifi password}"

#define OWM_KEY "{OpenWeatherMap.com key}"
#define OWM_LOCATION "Your City"
//example
//#define OWM_LOCATION "New York,US"

// update the weather at this interval, in minutes
#define UPDATE_INTERVAL 15

// Set to true to show temperatures in Celsius, false for Fahrenheit
#define OWM_METRIC false

// temperature will display in red at or above this temperature
// set to a high number (i.e. >200) to not show temperatures in red
#define METRIC_HOT  32
#define ENGLISH_HOT  90

/*
Arabic - ar, Bulgarian - bg, Catalan - ca, Czech - cz, German - de, Greek - el,
English - en, Persian (Farsi) - fa, Finnish - fi, French - fr, Galician - gl,
Croatian - hr, Hungarian - hu, Italian - it, Japanese - ja, Korean - kr,
Latvian - la, Lithuanian - lt, Macedonian - mk, Dutch - nl, Polish - pl,
Portuguese - pt, Romanian - ro, Russian - ru, Swedish - se, Slovak - sk,
Slovenian - sl, Spanish - es, Turkish - tr, Ukrainian - ua, Vietnamese - vi,
Chinese Simplified - zh_cn, Chinese Traditional - zh_tw.
*/
#define OWM_LANGUAGE "en"
```

```
// SPDX-FileCopyrightText: 2019 Dan Cogliano for Adafruit Industries
//
// SPDX-License-Identifier: MIT

#include <time.h>
#include <Adafruit_GFX.h>    // Core graphics library
#include <Adafruit_ThinkInk.h>
#include <Adafruit_NeoPixel.h>
#include <ArduinoJson.h>        //https://github.com/bblanchon/ArduinoJson
#include <SPI.h>
#include <WiFiNINA.h>

#include "secrets.h"
```

```
#include "OpenWeatherMap.h"

#include "Fonts/meteocons48pt7b.h"
#include "Fonts/meteocons24pt7b.h"
#include "Fonts/meteocons20pt7b.h"
#include "Fonts/meteocons16pt7b.h"

#include "Fonts/moon_phases20pt7b.h"
#include "Fonts/moon_phases36pt7b.h"

#include <Fonts/FreeSans9pt7b.h>
#include <Fonts/FreeSans12pt7b.h>
#include <Fonts/FreeSans18pt7b.h>
#include <Fonts/FreeSansBold12pt7b.h>
#include <Fonts/FreeSansBold24pt7b.h>

#define SRAM_CS     8
#define EPD_CS      10
#define EPD_DC      9
#define EPD_RESET -1
#define EPD_BUSY -1

#define NEOPIXELPIN   40

// This is for the 2.7" tricolor EPD
ThinkInk_270_Tricolor_C44 gfx(EPD_DC, EPD_RESET, EPD_CS, SRAM_CS, EPD_BUSY);

AirliftOpenWeatherMap owclient(&Serial);
OpenWeatherMapCurrentData owcdata;
OpenWeatherMapForecastData owfdata[3];

Adafruit_NeoPixel neopixel = Adafruit_NeoPixel(1, NEOPIXELPIN, NEO_GRB +
NEO_KHZ800);

  const char *moonphasenames[29] = {
    "New Moon",
    "Waxing Crescent",
    "Waxing Crescent",
    "Waxing Crescent",
    "Waxing Crescent",
    "Waxing Crescent",
    "Waxing Crescent",
    "Quarter",
    "Waxing Gibbous",
    "Waxing Gibbous",
    "Waxing Gibbous",
    "Waxing Gibbous",
    "Waxing Gibbous",
    "Waxing Gibbous",
    "Full Moon",
    "Waning Gibbous",
    "Waning Gibbous",
    "Waning Gibbous",
    "Waning Gibbous",
    "Waning Gibbous",
    "Waning Gibbous",
    "Last Quarter",
    "Waning Crescent",
    "Waning Crescent",
    "Waning Crescent",
    "Waning Crescent",
    "Waning Crescent",
    "Waning Crescent",
    "Waning Crescent"
};

int8_t readButtons(void) {
  uint16_t reading = analogRead(A3);
  //Serial.println(reading);
```

```
  if (reading > 600) {
    return 0; // no buttons pressed
  }
  if (reading > 400) {
    return 4; // button D pressed
  }
  if (reading > 250) {
    return 3; // button C pressed
  }
  if (reading > 125) {
    return 2; // button B pressed
  }
  return 1; // Button A pressed
}

bool wifi_connect(){

  Serial.print("Connecting to WiFi... ");

  WiFi.setPins(SPIWIFI_SS, SPIWIFI_ACK, ESP32_RESETN, ESP32_GPIO0, &SPIWIFI);

  // check for the WiFi module:
  if(WiFi.status() == WL_NO_MODULE) {
    Serial.println("Communication with WiFi module failed!");
    displayError("Communication with WiFi module failed!");
    while(true);
  }

  String fv = WiFi.firmwareVersion();
  if (fv < "1.0.0") {
    Serial.println("Please upgrade the firmware");
  }

  neopixel.setPixelColor(0, neopixel.Color(0, 0, 255));
  neopixel.show();
  if(WiFi.begin(WIFI_SSID, WIFI_PASSWORD) == WL_CONNECT_FAILED)
  {
    Serial.println("WiFi connection failed!");
    displayError("WiFi connection failed!");
    return false;
  }

  int wifitimeout = 15;
  int wifistatus;
  while ((wifistatus = WiFi.status()) != WL_CONNECTED && wifitimeout > 0) {
    delay(1000);
    Serial.print(".");
    wifitimeout--;
  }
  if(wifitimeout == 0)
  {
    Serial.println("WiFi connection timeout with error " + String(wifistatus));
    displayError("WiFi connection timeout with error " + String(wifistatus));
    neopixel.setPixelColor(0, neopixel.Color(0, 0, 0));
    neopixel.show();
    return false;
  }
  neopixel.setPixelColor(0, neopixel.Color(0, 0, 0));
  neopixel.show();
  Serial.println("Connected");
  return true;
}

void wget(String &url, int port, char *buff)
{
  int pos1 = url.indexOf("/",0);
  int pos2 = url.indexOf("/",8);
  String host = url.substring(pos1+2,pos2);
```

```
  String path = url.substring(pos2);
  Serial.println("to wget(" + host + "," + path + "," + port + ")");
  wget(host, path, port, buff);
}

void wget(String &host, String &path, int port, char *buff)
{
  //WiFiSSLClient client;
  WiFiClient client;

  neopixel.setPixelColor(0, neopixel.Color(0, 0, 255));
  neopixel.show();
  client.stop();
  if (client.connect(host.c_str(), port)) {
    Serial.println("connected to server");
    // Make a HTTP request:
    client.println(String("GET ") + path + String(" HTTP/1.0"));
    client.println("Host: " + host);
    client.println("Connection: close");
    client.println();

    uint32_t bytes = 0;
    int capturepos = 0;
    bool capture = false;
    int linelength = 0;
    char lastc = '\0';
    while(true)
    {
      while (client.available()) {
        char c = client.read();
        //Serial.print(c);
        if((c == '\n') && (lastc == '\r'))
        {
          if(linelength == 0)
          {
            capture = true;
          }
          linelength = 0;
        }
        else if(capture)
        {
          buff[capturepos++] = c;
          //Serial.write(c);
        }
        else
        {
          if((c != '\n') && (c != '\r'))
            linelength++;
        }
        lastc = c;
        bytes++;
      }

      // if the server's disconnected, stop the client:
      if (!client.connected()) {
        //Serial.println();
        Serial.println("disconnecting from server.");
        client.stop();
        buff[capturepos] = '\0';
        Serial.println("captured " + String(capturepos) + " bytes");
        break;
      }
    }
  }
  else
  {
    Serial.println("problem connecting to " + host + ":" + String(port));
    buff[0] = '\0';
  }
```

```
  neopixel.setPixelColor(0, neopixel.Color(0, 0, 0));
  neopixel.show();
}

int getStringLength(String s)
{
  int16_t  x = 0, y = 0;
  uint16_t w, h;
  gfx.getTextBounds(s, 0, 0, &x, &y, &w, &h);
  return w + x;
}

/*
return value is percent of moon cycle ( from 0.0 to 0.999999), i.e.:

0.0: New Moon
0.125: Waxing Crescent Moon
0.25: Quarter Moon
0.375: Waxing Gibbous Moon
0.5: Full Moon
0.625: Waning Gibbous Moon
0.75: Last Quarter Moon
0.875: Waning Crescent Moon

*/
float getMoonPhase(time_t tdate)
{

  time_t newmoonref = 1263539460; //known new moon date (2010-01-15 07:11)
  // moon phase is 29.5305882 days, which is 2551442.82048 seconds
  float phase = abs( tdate - newmoonref) / (double)2551442.82048;
  phase -= (int)phase; // leave only the remainder
  if(newmoonref > tdate)
  phase = 1 - phase;
  return phase;
}

void displayError(String str)
{
    // show error on display
    neopixel.setPixelColor(0, neopixel.Color(255, 0, 0));
    neopixel.show();

    Serial.println(str);

    gfx.setTextColor(EPD_BLACK);
    gfx.powerUp();
    gfx.clearBuffer();
    gfx.setTextWrap(true);
    gfx.setCursor(10,60);
    gfx.setFont(&FreeSans12pt7b);
    gfx.print(str);
    gfx.display();
    neopixel.setPixelColor(0, neopixel.Color(0, 0, 0));
    neopixel.show();
}

void displayHeading(OpenWeatherMapCurrentData &owcdata)
{

  time_t local = owcdata.observationTime + owcdata.timezone;
  struct tm *timeinfo = gmtime(&local);
  char datestr[80];
  // date
  //strftime(datestr,80,"%a, %d %b %Y",timeinfo);
  strftime(datestr,80,"%a, %b %d",timeinfo);
  gfx.setFont(&FreeSans18pt7b);
  gfx.setCursor((gfx.width()-getStringLength(datestr))/2,30);
  gfx.print(datestr);
```

```
  // city
  strftime(datestr,80,"%A",timeinfo);
  gfx.setFont(&FreeSansBold12pt7b);
  gfx.setCursor((gfx.width()-getStringLength(owcdata.cityName))/2,60);
  gfx.print(owcdata.cityName);
}

void displayForecastDays(OpenWeatherMapCurrentData &owcdata,
OpenWeatherMapForecastData owfdata[], int count = 3)
{
  for(int i=0; i < count; i++)
  {
    // day

    time_t local = owfdata[i].observationTime + owcdata.timezone;
    struct tm *timeinfo = gmtime(&local);
    char strbuff[80];
    strftime(strbuff,80,"%I",timeinfo);
    String datestr = String(atoi(strbuff));
    strftime(strbuff,80,"%p",timeinfo);
    // convert AM/PM to lowercase
    strbuff[0] = tolower(strbuff[0]);
    strbuff[1] = tolower(strbuff[1]);
    datestr = datestr + " " + String(strbuff);
    gfx.setFont(&FreeSans9pt7b);
    gfx.setCursor(i*gfx.width()/3 + (gfx.width()/3-getStringLength(datestr))/2,94);
    gfx.print(datestr);

    // weather icon
    String wicon = owclient.getMeteoconIcon(owfdata[i].icon);
    gfx.setFont(&meteocons20pt7b);
    gfx.setCursor(i*gfx.width()/3 + (gfx.width()/3-getStringLength(wicon))/2,134);
    gfx.print(wicon);

    // weather main description
    gfx.setFont(&FreeSans9pt7b);
    gfx.setCursor(i*gfx.width()/3 + (gfx.width()/3-
getStringLength(owfdata[i].main))/2,154);
    gfx.print(owfdata[i].main);

    // temperature
    int itemp = (int)(owfdata[i].temp + .5);
    int color = EPD_BLACK;
    if((OWM_METRIC && itemp >= METRIC_HOT)|| (!OWM_METRIC && itemp >= ENGLISH_HOT))
      color = EPD_RED;
    gfx.setTextColor(color);
    gfx.setFont(&FreeSans9pt7b);
    gfx.setCursor(i*gfx.width()/3 + (gfx.width()/3-getStringLength(String(itemp)))/
2,172);
    gfx.print(itemp);
    gfx.drawCircle(i*gfx.width()/3 + (gfx.width()/3-getStringLength(String(itemp)))/
2 + getStringLength(String(itemp)) + 6,163,3,color);
    gfx.drawCircle(i*gfx.width()/3 + (gfx.width()/3-getStringLength(String(itemp)))/
2 + getStringLength(String(itemp)) + 6,163,2,color);
    gfx.setTextColor(EPD_BLACK);
  }
}

void displayForecast(OpenWeatherMapCurrentData &owcdata, OpenWeatherMapForecastData
owfdata[], int count = 3)
{
  gfx.powerUp();
  gfx.clearBuffer();
  neopixel.setPixelColor(0, neopixel.Color(0, 255, 0));
  neopixel.show();

  gfx.setTextColor(EPD_BLACK);
  displayHeading(owcdata);
```

```
  displayForecastDays(owcdata, owfdata, count);
  gfx.display();
  gfx.powerDown();
  neopixel.setPixelColor(0, neopixel.Color(0, 0, 0));
  neopixel.show();
}

void displayAllWeather(OpenWeatherMapCurrentData &owcdata,
OpenWeatherMapForecastData owfdata[], int count = 3)
{
  gfx.powerUp();
  gfx.clearBuffer();
  neopixel.setPixelColor(0, neopixel.Color(0, 255, 0));
  neopixel.show();

  gfx.setTextColor(EPD_BLACK);

  // date string
  time_t local = owcdata.observationTime + owcdata.timezone;
  struct tm *timeinfo = gmtime(&local);
  char datestr[80];
  // date
  //strftime(datestr,80,"%a, %d %b %Y",timeinfo);
  strftime(datestr,80,"%a, %b %d %Y",timeinfo);
  gfx.setFont(&FreeSans9pt7b);
  gfx.setCursor((gfx.width()-getStringLength(datestr))/2,14);
  gfx.print(datestr);

  // weather icon
  String wicon = owclient.getMeteoconIcon(owcdata.icon);
  gfx.setFont(&meteocons24pt7b);
  gfx.setCursor((gfx.width()/3-getStringLength(wicon))/2,56);
  gfx.print(wicon);

  // weather main description
  gfx.setFont(&FreeSans9pt7b);
  gfx.setCursor((gfx.width()/3-getStringLength(owcdata.main))/2,72);
  gfx.print(owcdata.main);

  // temperature
  gfx.setFont(&FreeSansBold24pt7b);
  int itemp = owcdata.temp + .5;
  int color = EPD_BLACK;
  if((OWM_METRIC && (int)itemp >= METRIC_HOT)|| (!OWM_METRIC && (int)itemp >=
ENGLISH_HOT))
    color = EPD_RED;
  gfx.setTextColor(color);
  gfx.setCursor(gfx.width()/3 + (gfx.width()/3-getStringLength(String(itemp)))/2,
58);
  gfx.print(itemp);
  gfx.setTextColor(EPD_BLACK);

  // draw temperature degree as a circle (not available as font character
  gfx.drawCircle(gfx.width()/3 + (gfx.width()/3 + getStringLength(String(itemp)))/2
+ 8, 58-30,4,color);
  gfx.drawCircle(gfx.width()/3 + (gfx.width()/3 + getStringLength(String(itemp)))/2
+ 8, 58-30,3,color);

  // draw moon
  // draw Moon Phase
  float moonphase = getMoonPhase(owcdata.observationTime);
  int moonage = 29.5305882 * moonphase;
  //Serial.println("moon age: " + String(moonage));
  // convert to appropriate icon
  String moonstr = String((char)((int)'A' + (int)(moonage*25./30)));
  gfx.setFont(&moon_phases20pt7b);
  // font lines look a little thin at this size, drawing it a few times to thicken
the lines
```

```
  gfx.setCursor(2*gfx.width()/3 + (gfx.width()/3-getStringLength(moonstr))/2,56);
  gfx.print(moonstr);
  gfx.setCursor(2*gfx.width()/3 + (gfx.width()/3-getStringLength(moonstr))/2+1,56);
  gfx.print(moonstr);
  gfx.setCursor(2*gfx.width()/3 + (gfx.width()/3-getStringLength(moonstr))/2,56-1);
  gfx.print(moonstr);

  // draw moon phase name
  int currentphase = moonphase * 28. + .5;
  gfx.setFont(); // system font (smallest available)
  gfx.setCursor(2*gfx.width()/3 + max(0,(gfx.width()/3 -
getStringLength(moonphasenames[currentphase]))/2),62);
  gfx.print(moonphasenames[currentphase]);


  displayForecastDays(owcdata, owfdata, count);
  gfx.display();
  gfx.powerDown();
  neopixel.setPixelColor(0, neopixel.Color(0, 0, 0));
  neopixel.show();

}

void displayCurrentConditions(OpenWeatherMapCurrentData &owcdata)
{
  gfx.powerUp();
  gfx.clearBuffer();
  neopixel.setPixelColor(0, neopixel.Color(0, 255, 0));
  neopixel.show();

  gfx.setTextColor(EPD_BLACK);
  displayHeading(owcdata);

  // weather icon
  String wicon = owclient.getMeteoconIcon(owcdata.icon);
  gfx.setFont(&meteocons48pt7b);
  gfx.setCursor((gfx.width()/2-getStringLength(wicon))/2,156);
  gfx.print(wicon);

  // weather main description
  gfx.setFont(&FreeSans9pt7b);
  gfx.setCursor(gfx.width()/2 + (gfx.width()/2-getStringLength(owcdata.main))/2,
160);
  gfx.print(owcdata.main);

  // temperature
  gfx.setFont(&FreeSansBold24pt7b);
  int itemp = owcdata.temp + .5;
  int color = EPD_BLACK;
  if((OWM_METRIC && (int)itemp >= METRIC_HOT)|| (!OWM_METRIC && (int)itemp >=
ENGLISH_HOT))
    color = EPD_RED;
  gfx.setTextColor(color);
  gfx.setCursor(gfx.width()/2 + (gfx.width()/2-getStringLength(String(itemp)))/2,
130);
  gfx.print(itemp);
  gfx.setTextColor(EPD_BLACK);

  // draw temperature degree as a circle (not available as font character
  gfx.drawCircle(gfx.width()/2 + (gfx.width()/2 + getStringLength(String(itemp)))/2
+ 10, 130-26,4,color);
  gfx.drawCircle(gfx.width()/2 + (gfx.width()/2 + getStringLength(String(itemp)))/2
+ 10, 130-26,3,color);

  gfx.display();
  gfx.powerDown();
  neopixel.setPixelColor(0, neopixel.Color(0, 0, 0));
  neopixel.show();
}
```

```
void displaySunMoon(OpenWeatherMapCurrentData &owcdata)
{

  gfx.powerUp();
  gfx.clearBuffer();
  neopixel.setPixelColor(0, neopixel.Color(0, 255, 0));
  neopixel.show();

  gfx.setTextColor(EPD_BLACK);
  displayHeading(owcdata);

  // draw Moon Phase
  float moonphase = getMoonPhase(owcdata.observationTime);
  int moonage = 29.5305882 * moonphase;
  // convert to appropriate icon
  String moonstr = String((char)((int)'A' + (int)(moonage*25./30)));
  gfx.setFont(&moon_phases36pt7b);
  gfx.setCursor((gfx.width()/3-getStringLength(moonstr))/2,140);
  gfx.print(moonstr);

  // draw moon phase name
  int currentphase = moonphase * 28. + .5;
  gfx.setFont(&FreeSans9pt7b);
  gfx.setCursor(gfx.width()/3 + max(0,(gfx.width()*2/3 -
getStringLength(moonphasenames[currentphase]))/2),110);
  gfx.print(moonphasenames[currentphase]);

  // draw sunrise/sunset

  // sunrise/sunset times
  // sunrise

  time_t local = owcdata.sunrise + owcdata.timezone + 30;  // round to nearest
minute
  struct tm *timeinfo = gmtime(&local);
  char strbuff[80];
  strftime(strbuff,80,"%I",timeinfo);
  String datestr = String(atoi(strbuff));
  strftime(strbuff,80,":%M %p",timeinfo);
  datestr = datestr + String(strbuff) + " - ";
  // sunset
  local = owcdata.sunset + owcdata.timezone + 30; // round to nearest minute
  timeinfo = gmtime(&local);
  strftime(strbuff,80,"%I",timeinfo);
  datestr = datestr + String(atoi(strbuff));
  strftime(strbuff,80,":%M %p",timeinfo);
  datestr = datestr + String(strbuff);

  gfx.setFont(&FreeSans9pt7b);
  int datestrlen = getStringLength(datestr);
  int xpos = (gfx.width() - datestrlen)/2;
  gfx.setCursor(xpos,166);
  gfx.print(datestr);

  // draw sunrise icon
  // sun icon is "B"
  String wicon = "B";
  gfx.setFont(&meteocons16pt7b);
  gfx.setCursor(xpos - getStringLength(wicon) - 12,174);
  gfx.print(wicon);

  // draw sunset icon
  // sunset icon is "A"
  wicon = "A";
  gfx.setCursor(xpos + datestrlen + 12,174);
  gfx.print(wicon);

  gfx.display();
```

```
  gfx.powerDown();
  neopixel.setPixelColor(0, neopixel.Color(0, 0, 0));
  neopixel.show();
}

void setup() {
  neopixel.begin();
  neopixel.show();

  gfx.begin();
  Serial.println("ePaper display initialized");
  gfx.setRotation(2);
  gfx.setTextWrap(false);

}

void loop() {
  char data[4000];
  static uint32_t timer = millis();
  static uint8_t lastbutton = 1;
  static bool firsttime = true;

  int button = readButtons();

  // update weather data at specified interval or when button 4 is pressed
  if((millis() >= (timer + 1000*60*UPDATE_INTERVAL)) || (button == 4) || firsttime)
  {
    Serial.println("getting weather data");
    firsttime = false;
    timer = millis();
    int retry = 6;
    while(!wifi_connect())
    {
      delay(5000);
      retry--;
      if(retry < 0)
      {
        displayError("Can not connect to WiFi, press reset to restart");
        while(1);
      }
    }
    String urlc = owclient.buildUrlCurrent(OWM_KEY,OWM_LOCATION);
    Serial.println(urlc);
    retry = 6;
    do
    {
      retry--;
      wget(urlc,80,data);
      if(strlen(data) == 0 && retry < 0)
      {
        displayError("Can not get weather data, press reset to restart");
        while(1);
      }
    }
    while(strlen(data) == 0);
    Serial.println("data retrieved:");
    Serial.println(data);
    retry = 6;
    while(!owclient.updateCurrent(owcdata,data))
    {
      retry--;
      if(retry < 0)
      {
        displayError(owclient.getError());
        while(1);
      }
      delay(5000);
    }
```

```
    String urlf = owclient.buildUrlForecast(OWM_KEY,OWM_LOCATION);
    Serial.println(urlf);
    wget(urlf,80,data);
    Serial.println("data retrieved:");
    Serial.println(data);
    if(!owclient.updateForecast(owfdata[0],data,0))
    {
      displayError(owclient.getError());
      while(1);
    }
    if(!owclient.updateForecast(owfdata[1],data,2))
    {
      displayError(owclient.getError());
      while(1);
    }
    if(!owclient.updateForecast(owfdata[2],data,4))
    {
      displayError(owclient.getError());
      while(1);
    }

    switch(lastbutton)
    {
      case 1:
        displayAllWeather(owcdata,owfdata,3);
        break;
      case 2:
        displayCurrentConditions(owcdata);
        break;
      case 3:
        displaySunMoon(owcdata);
        break;
    }
  }

  if (button == 0) {
    return;
  }

  Serial.print("Button "); Serial.print(button); Serial.println(" pressed");

  if (button == 1) {
    displayAllWeather(owcdata,owfdata,3);
    lastbutton = button;
  }
  if (button == 2) {
    //displayForecast(owcdata,owfdata,3);
    displayCurrentConditions(owcdata);
    lastbutton = button;
  }
  if (button == 3) {
    displaySunMoon(owcdata);
    lastbutton = button;
  }

  // wait until button is released
  while (readButtons()) {
    delay(10);
  }

}
```

```
// SPDX-FileCopyrightText: 2019 Dan Cogliano for Adafruit Industries
//
// SPDX-License-Identifier: MIT

#pragma once
```

```
#include "secrets.h"
#include <ArduinoJson.h>          //https://github.com/bblanchon/ArduinoJson

typedef struct OpenWeatherMapCurrentData {
  // "lon": 8.54,
  float lon;
  // "lat": 47.37
  float lat;
  // "id": 521,
  uint16_t weatherId;
  // "main": "Rain",
  String main;
  // "description": "shower rain",
  String description;
  // "icon": "09d"
  String icon;
  String iconMeteoCon;
  // "temp": 290.56,
  float temp;
  // "pressure": 1013,
  uint16_t pressure;
  // "humidity": 87,
  uint8_t humidity;
  // "temp_min": 289.15,
  float tempMin;
  // "temp_max": 292.15
  float tempMax;
  // visibility: 10000,
  uint16_t visibility;
  // "wind": {"speed": 1.5},
  float windSpeed;
  // "wind": {deg: 226.505},
  float windDeg;
  // "clouds": {"all": 90},
  uint8_t clouds;
  // "dt": 1527015000,
  time_t observationTime;
  // "country": "CH",
  String country;
  // "sunrise": 1526960448,
  time_t sunrise;
  // "sunset": 1527015901
  time_t sunset;
  // "name": "Zurich",
  String cityName;
  time_t timezone;
} OpenWeatherMapCurrentData;

typedef struct OpenWeatherMapForecastData {
  // {"dt":1527066000,
  time_t observationTime;
  // "main":{
  //   "temp":17.35,
  float temp;
  //   "temp_min":16.89,
  float tempMin;
  //   "temp_max":17.35,
  float tempMax;
  //   "pressure":970.8,
  float pressure;
  //   "sea_level":1030.62,
  float pressureSeaLevel;
  //   "grnd_level":970.8,
  float pressureGroundLevel;
  //   "humidity":97,
  uint8_t humidity;
  //   "temp_kf":0.46
  // },"weather":[{
  //   "id":802,
```

```
  uint16_t weatherId;
  //   "main":"Clouds",
  String main;
  //   "description":"scattered clouds",
  String description;
  //   "icon":"03d"
  String icon;
  String iconMeteoCon;
  // }],"clouds":{"all":44},
  uint8_t clouds;
  // "wind":{
  //   "speed":1.77,
  float windSpeed;
  //   "deg":207.501
  float windDeg;
  // rain: {3h: 0.055},
  float rain;
  // },"sys":{"pod":"d"}
  // dt_txt: "2018-05-23 09:00:00"
  String observationTimeText;

} OpenWeatherMapForecastData;

class AirliftOpenWeatherMap{
  private:
    Stream *Serial;
    String currentKey;
    String currentParent;
    //OpenWeatherMapCurrentData *data;
    uint8_t weatherItemCounter = 0;
    bool metric = true;
    String language;
    String _error;

  public:
    AirliftOpenWeatherMap(Stream *serial){Serial = serial;};
    String buildUrlCurrent(String appId, String locationParameter);
    String buildUrlForecast(String appId, String locationParameter);
    bool updateCurrent(OpenWeatherMapCurrentData &data,String json);
    bool updateForecast(OpenWeatherMapForecastData &data,String json, int day = 0);

    void setMetric(bool metric) {this->metric = metric;}
    bool isMetric() { return metric; }
    void setLanguage(String language) { this->language = language; }
    String getLanguage() { return language; }
    void setError(String error){_error = error;}
    String getError(){return _error;}

    String getMeteoconIcon(String icon);

};
```

```
// SPDX-FileCopyrightText: 2019 Dan Cogliano for Adafruit Industries
//
// SPDX-License-Identifier: MIT

#include "OpenWeatherMap.h"

String AirliftOpenWeatherMap::buildUrlCurrent(String appId, String location) {
  String units = OWM_METRIC ? "metric" : "imperial";
  return "http://api.openweathermap.org/data/2.5/weather?q=" + location + "&appid="
+ appId + "&units=" + units + "&lang=" + String(OWM_LANGUAGE);
}

String AirliftOpenWeatherMap::buildUrlForecast(String appId, String location) {
  String units = OWM_METRIC ? "metric" : "imperial";
  return "http://api.openweathermap.org/data/2.5/forecast?q=" + location +
```

```
"&cnt=6&appid=" + appId + "&units=" + units + "&lang=" + String(OWM_LANGUAGE);
}

String AirliftOpenWeatherMap::getMeteoconIcon(String icon) {
  // clear sky
  // 01d
  if (icon == "01d")    {
    return "B";
  }
  // 01n
  if (icon == "01n")    {
    return "C";
  }
  // few clouds
  // 02d
  if (icon == "02d")    {
    return "H";
  }
  // 02n
  if (icon == "02n")    {
    return "4";
  }
  // scattered clouds
  // 03d
  if (icon == "03d")    {
    return "N";
  }
  // 03n
  if (icon == "03n")    {
    return "5";
  }
  // broken clouds
  // 04d
  if (icon == "04d")    {
    return "Y";
  }
  // 04n
  if (icon == "04n")    {
    return "%";
  }
  // shower rain
  // 09d
  if (icon == "09d")    {
    return "R";
  }
  // 09n
  if (icon == "09n")    {
    return "8";
  }
  // rain
  // 10d
  if (icon == "10d")    {
    return "Q";
  }
  // 10n
  if (icon == "10n")    {
    return "7";
  }
  // thunderstorm
  // 11d
  if (icon == "11d")    {
    return "P";
  }
  // 11n
  if (icon == "11n")    {
    return "6";
  }
  // snow
  // 13d
```

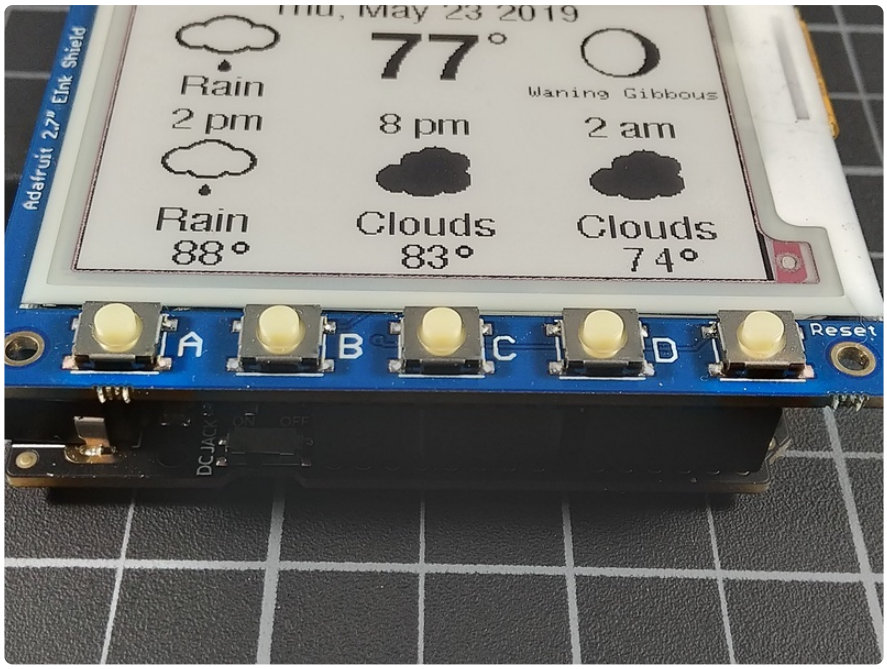
```
  if (icon == "13d")    {
    return "W";
  }
  // 13n
  if (icon == "13n")    {
    return "#";
  }
  // mist
  // 50d
  if (icon == "50d")    {
    return "M";
  }
  // 50n
  if (icon == "50n")    {
    return "M";
  }
  // Nothing matched: N/A
  return ")";

}

bool AirliftOpenWeatherMap::updateCurrent(OpenWeatherMapCurrentData &data, String
json)
{
  Serial->println("updateCurrent()");
  DynamicJsonDocument doc(2000);
  //StaticJsonDocument<2000> doc;

  DeserializationError error = deserializeJson(doc, json);
  if (error) {
    Serial->println(String("deserializeJson() failed: ") + (const char
*)error.c_str());
    Serial->println(json);
    setError(String("deserializeJson() failed: ") + error.c_str());
    return false;
  }

  int code = (int) doc["cod"];
  if(code != 200)
  {
    Serial->println(String("OpenWeatherMap error: ") + (const char
*)doc["message"]);
    setError(String("OpenWeatherMap error: ") + (const char *)doc["message"]);
    return false;
  }

  data.lat = (float) doc["coord"]["lat"];
  data.lon = (float) doc["coord"]["lon"];

  data.main = (const char*) doc["weather"][0]["main"];
  data.description = (const char*) doc["weather"][0]["description"];
  data.icon = (const char*) doc["weather"][0]["icon"];

  data.cityName = (const char*) doc["name"];
  data.visibility = (uint16_t) doc["visibility"];
  data.timezone = (time_t) doc["timezone"];

  data.country = (const char*) doc["sys"]["country"];
  data.observationTime = (time_t) doc["dt"];
  data.sunrise = (time_t) doc["sys"]["sunrise"];
  data.sunset = (time_t) doc["sys"]["sunset"];

  data.temp = (float) doc["main"]["temp"];
  data.pressure = (uint16_t) doc["main"]["pressure"];
  data.humidity = (uint8_t) doc["main"]["humidity"];
  data.tempMin = (float) doc["main"]["temp_min"];
  data.tempMax = (float) doc["main"]["temp_max"];

  data.windSpeed = (float) doc["wind"]["speed"];
```

```
  data.windDeg = (float) doc["wind"]["deg"];
  return true;
}

bool AirliftOpenWeatherMap::updateForecast(OpenWeatherMapForecastData &data, String
json, int day)
{
  Serial->println("updateForecast()");
  DynamicJsonDocument doc(5000);
  //StaticJsonDocument<5000> doc;

  DeserializationError error = deserializeJson(doc, json);
  if (error) {
    Serial->println(String("deserializeJson() failed: ") + (const char
*)error.c_str());
    Serial->println(json);
    setError(String("deserializeJson() failed: ") + error.c_str());
    return false;
  }

  int code = (int) doc["cod"];
  if(code != 200)
  {
    Serial->println(String("OpenWeatherMap error: ") + (const char
*)doc["message"]);
    setError(String("OpenWeatherMap error: ") + (const char *)doc["message"]);
    return false;
  }

  data.observationTime = (time_t) doc["list"][day]["dt"];

  data.temp = (float) doc["list"][day]["main"]["temp"];
  data.pressure = (uint16_t) doc["list"][day]["main"]["pressure"];
  data.humidity = (uint8_t) doc["list"][day]["main"]["humidity"];
  data.tempMin = (float) doc["list"][day]["main"]["temp_min"];
  data.tempMax = (float) doc["list"][day]["main"]["temp_max"];

  data.main = (const char*) doc["list"][day]["weather"][0]["main"];
  data.description = (const char*) doc["list"][day]["weather"][0]["description"];
  data.icon = (const char*) doc["list"][day]["weather"][0]["icon"];
  return true;
}
```

The font files can be found in the GitHub repository here (https://adafru.it/F1o).

Use

The shield includes 4 programmable buttons, labeled "A" to "D", and a reset button. These buttons have been programmed show different weather displays. When first booting up the device, the display shows the current and forecast weather. You can also view this display by pressing the "A" or reset button. Button "B" shows the current weather conditions, and button "C" shows the current lunar phase and sunrise and sunset times for the day. The "D" button will retrieve the latest weather data from the Internet and display it using the last display mode. This is particularly useful if you have a long update interval and you want to update the display with the latest weather data.



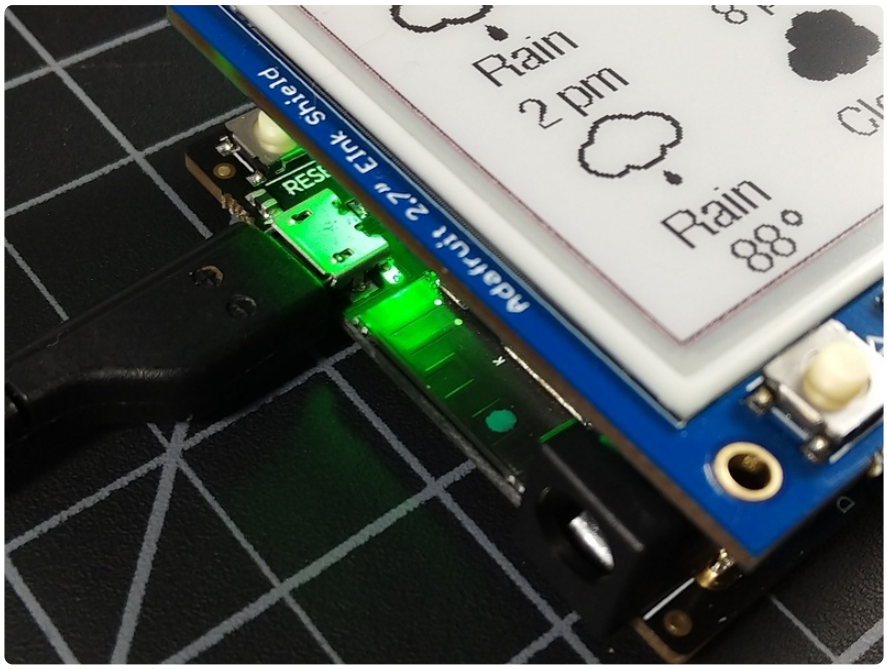
Status LED

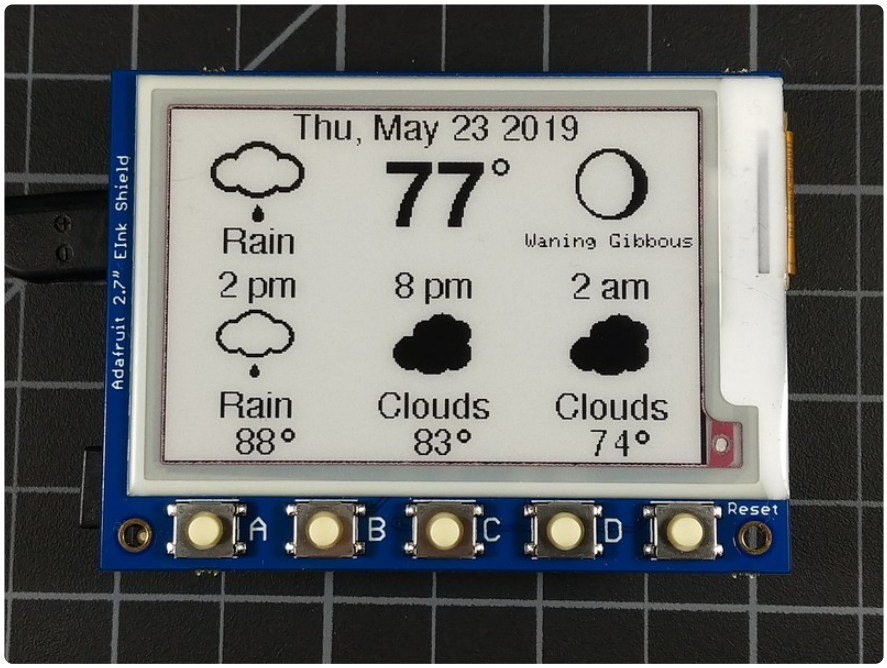
The Metro M4 Express Airlift also comes with a single NeoPixel. This project uses the NeoPixel as a project status indicator.

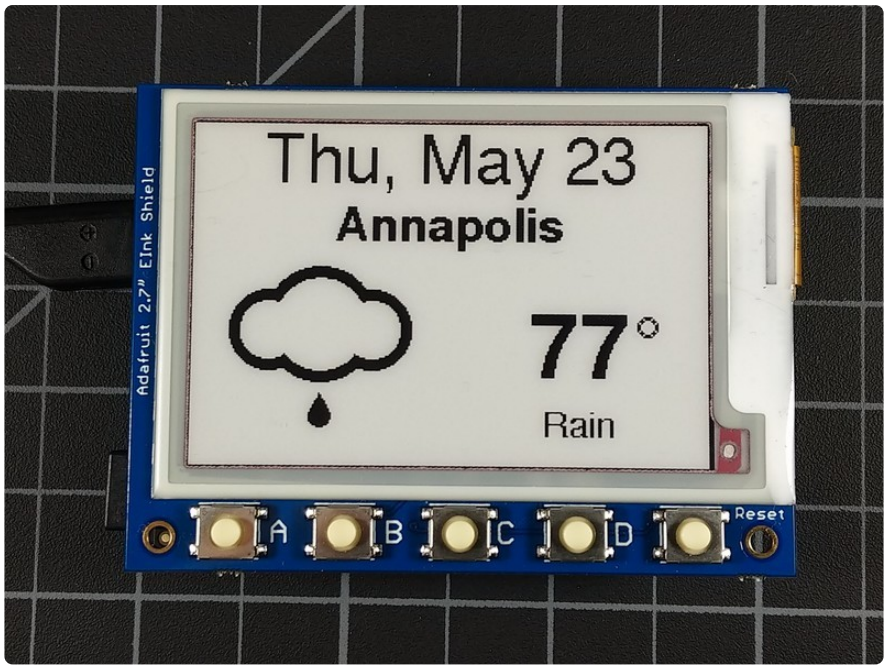
A **blue** NeoPixel status means the sketch is currently accessing the Internet, either connecting to the WiFi hotspot or grabbing the date and time.

A **green** NeoPixel status means the sketch is currently updating the display and will turn off when it is completed. This can take a few seconds since ePaper displays are not very speedy at refreshing their screens.

A **red** NeoPixel status means there is a network communication issue. Pressing the buttons will have no effect if the NeoPixel is showing one of these status colors. Wait for the NeoPixel to turn off before pressing one of the buttons.


Rain
2 pm
Rain
88°
Adafruit 2.7" EInk Shield


Thu, May 23 2019
77°
Rain
Waning Gibbous
2 pm
8 pm
2 am
Rain
88°
Clouds
83°
Clouds
74°
Adafruit 2.7" EInk Shield
A
B
C
D
Reset


Thu, May 23
Annapolis
77°
Rain
Adafruit 2.7" EInk Shield
A
B
C
D
Reset

Thu, May 23
Annapolis
Waning Gibbous
5:47 AM - 8:18 PM
Adafruit 2.7" EInk Shield
A
B
C
D
Reset