Wireless E-Ink Display

Small Project Funding

Done By: Kwa Jian Quan (202525R)

Project motivation

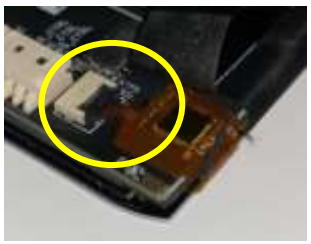
During school, I found it difficult to keep track of the different zoom lessons on my timetable. While I can check it through the calendar app on my phone, there are times where it is inconvenient for me to do so. Hence, with this E-Ink display, I would like to create a calendar events viewer. This will require the use of Google API and HTTPS request protocol. Through this project, I gained a better insight, and have had hands on experience on how these protocol work. Furthermore, I decided to further enhance on my weather station project (right picture), by integrating it into this E-Ink display.

Project summary

Part 1: Testing of e-ink display module

The E-Ink display was set-up by following the GitHub Instructions.[i] This allowed example codes to be uploaded to the ESP32 microcontroller. Following the installation of the drivers, example codes were uploaded to ensure that the E-Ink display was working.

Part 2: Installation of touch screen module



The touch screen module can now be connected to the board. It consists of an adhesive backing and a data cable. After installing of the touch screen module, the data cable was connected to the board (as shown in the picture). An example program which tests the touch screen was ran to ensure that it is working.

Part 3: Testing weather station program (online)

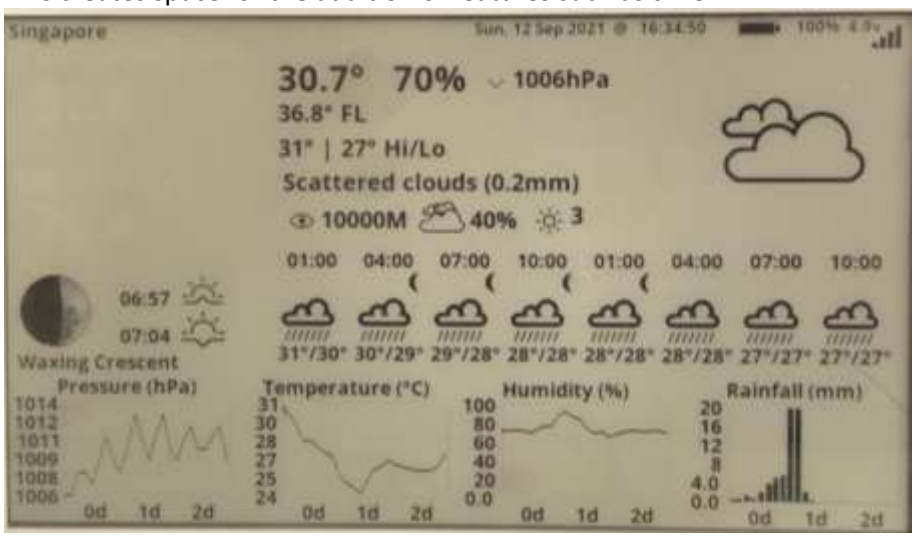
A weather station program[ii] for this E-Ink Display was found and uploaded to the board. An API for Open Weather Map[iii] was created to obtain the current weather data from the internet. The location was set to Singapore and the Wi-Fi credential was entered. The program was working well, but it included additional information, such as moon phases and wind direction, which are not needed for this project.

Part 4: Modifying the program

Currently, this program only obtains information from the internet. As this project will have to display the indoors weather data and calendar events, some of the information must be removed to make space for this information.

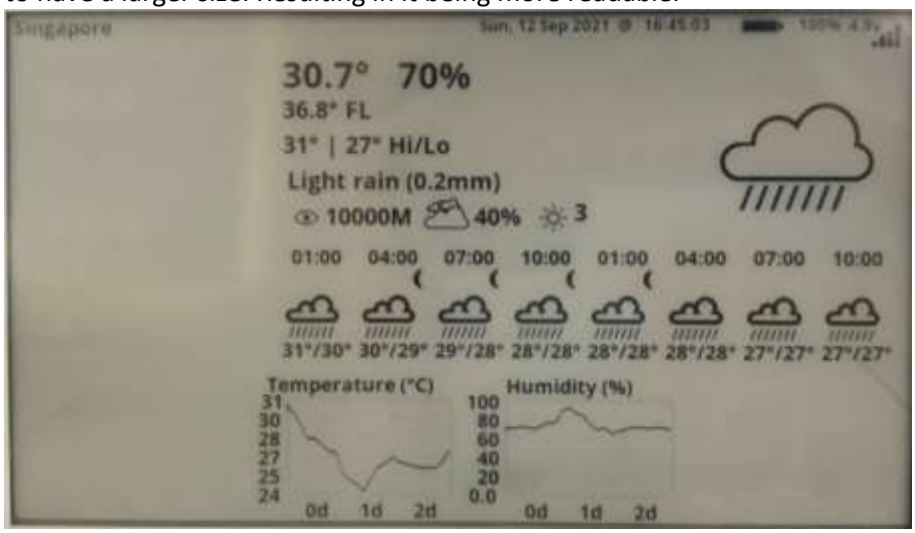
Removing wind direction

This creates space for the addition of features such as time.



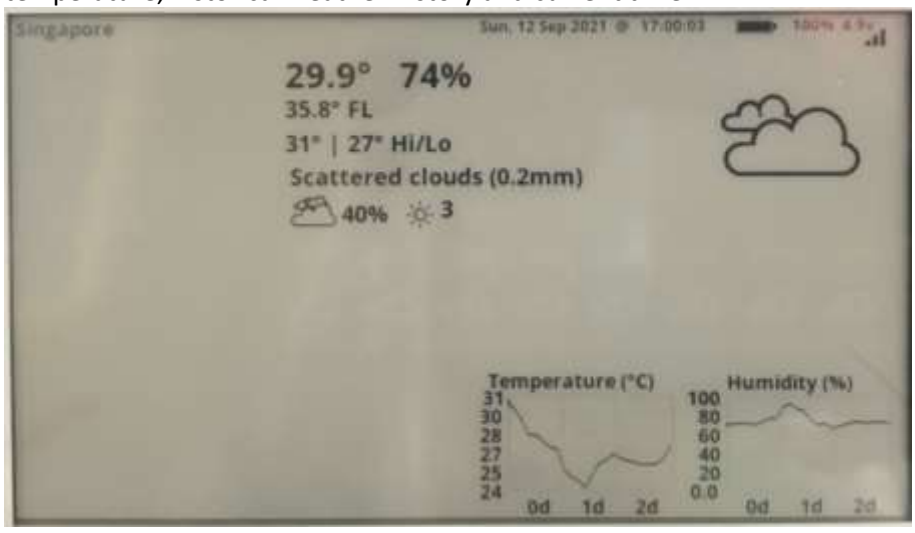
Removing moon phases, sunrise and sunset time, and forecast graphs

This step reduces the number of graphs on the display, which allow the necessary information, temperature and humidity, to have a larger size. Resulting in it being more readable.



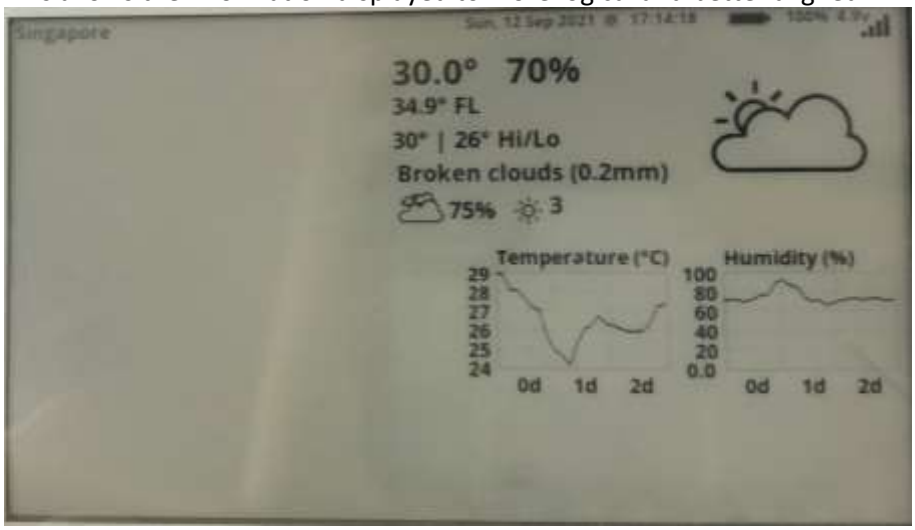
Removing forecast data

The forecast data is not of high priority to this project. Hence, it provides more space for information such as local temperature, historical weather history and current time.



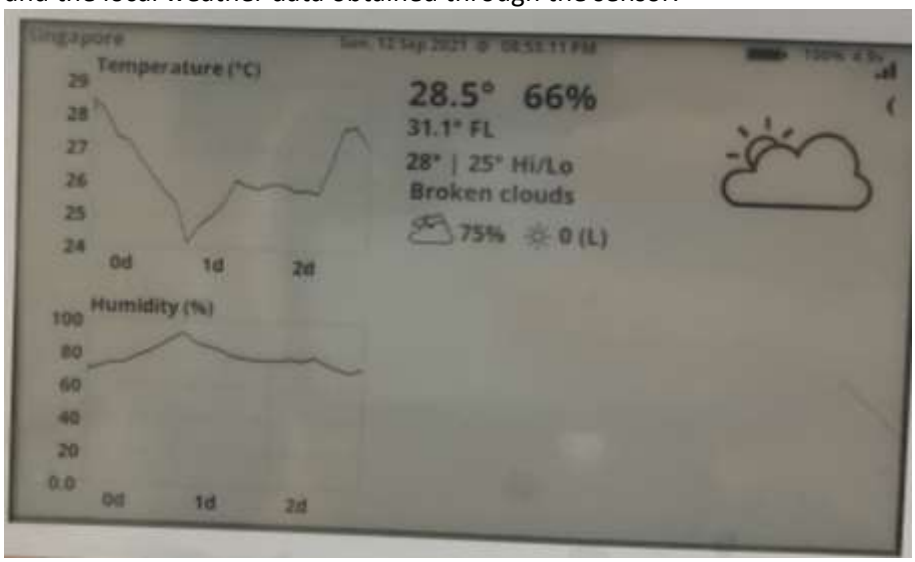
Positioning of information

This allows the information displayed to more logical and better aligned.



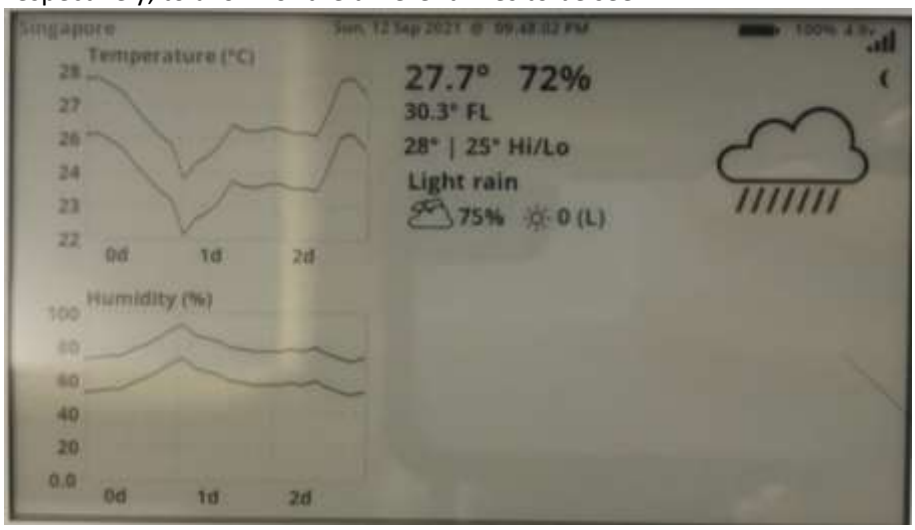
Increasing the size of the graph

This is important as through further development, the graph will display both the weather data obtained from the internet and the local weather data obtained through the sensor.



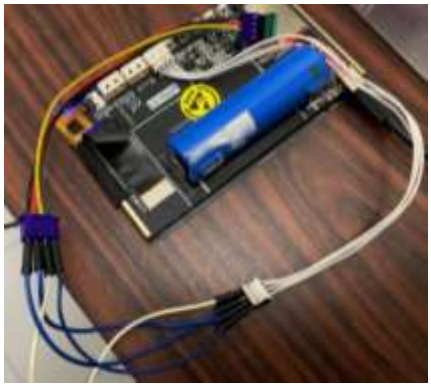
Adding placeholder line for local temperature readings

Currently, the temperature sensor was not installed. Hence, the values were simulated through the weather forecast data obtained through the internet. The data was lowered by 2 degrees and 20% for the temperature and humidity readings respectively, to allow for the different lines to be seen.

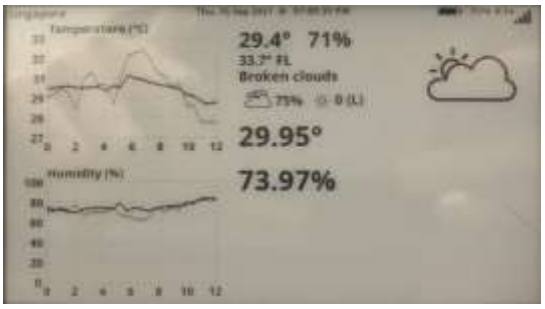


Part 5: Including the temperature sensor

The SHT-35 temperature and humidity sensor, purchased from my previous project (Arduino Weather Station) was transferred over to this project, to display the local temperature and humidity values. It is connected to pins 14 and 15 and retrieved using the i2c communications protocol. As the pins are not in the order of the i2c display, it must be soldered according to the sensor pinout.

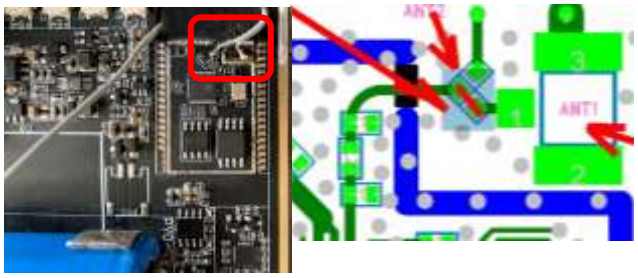


The graph now shows the past weather data instead of the weather forecast data. This was done using an array variable in Arduino. The scale of the x-axis was also changed, to display historical data for a 12-hour period only. The darker line represents the local weather data, obtained by the SHT-35 sensor, and the lighter line represents the weather data obtained from online sources.



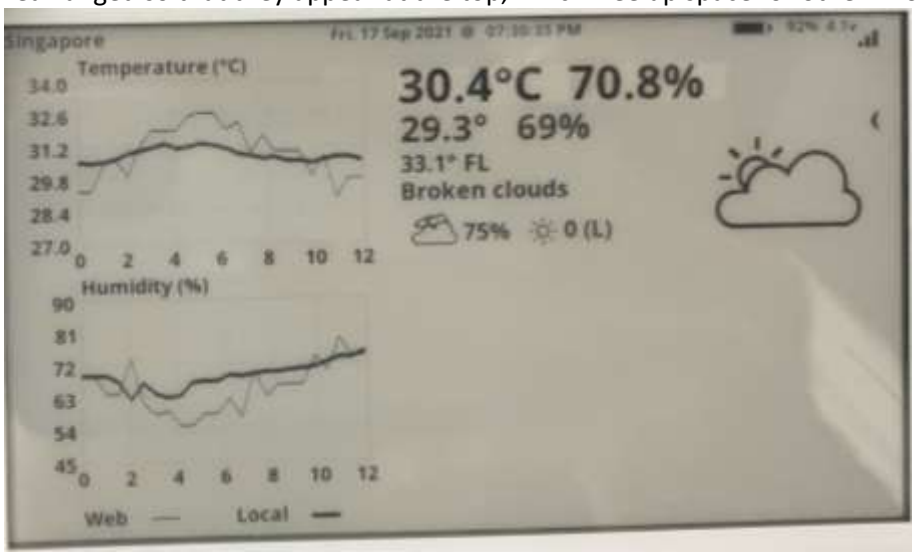
Part 6: Including an external antenna

On occasion, the Wi-Fi signal was weak, and the E-Ink display will fail to obtain the calendar and weather information from the internet. An external antenna was added to rectify this issue. Unfortunately, the U.FL antenna connector, used to connect an external antenna to the device, does not exist. Hence, the antenna was manually soldered to the PCB. This requires a millimeter level precision. This improved the Wi-Fi signal by 600%.



Improving the layout

The graph was reduced in size to make space for the label, located at the bottom of the humidity graph. This allows users to recognize which line belongs to the local, and web weather data. The temperature and humidity readings were also rearranged so that they appear at the top, which free up space for other information to be displayed.



Integrating google Calendar

Google calendar was connected to the Arduino through Google Scripts. It allows the Arduino to request for my calendar data through the google script portal. Google Script will then fetch my personal calendar. This requires the use of HTTPS authentication method and the “GET HTTP” command. The google script code will return the 5 most recent calendar events and send it to the E-Ink display, which will display the information in a card format.

If the calendar portion of the display is clicked, the program will display the 5 most recent calendar events. Clicking the time will display the weather screen.

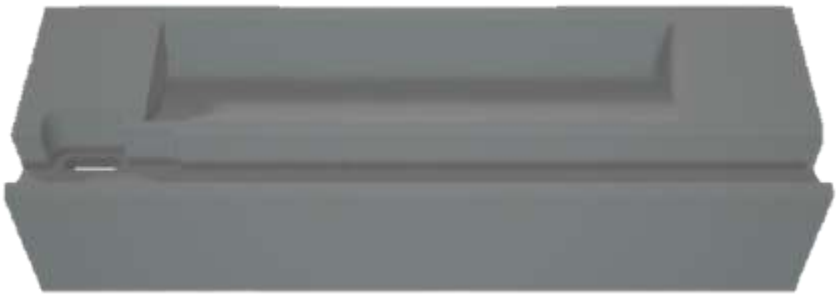
Part 7: Inclusion of the RTC (Real Time Clock)

Battery life is of the upmost important for an e-ink display. The ULP (Ultra Low Power) Coprocessor, in conjunction with the real time clock, will provide the E-Ink display with a longer battery life. The E-Ink display will only fetch the online weather and calendar data every 30 minutes. During this time, the display will also synchronize its internal timing with the web.

The local weather data, and time will be updated every minute, since there is no need to connect to the Wi-Fi network. During the time where the E-Ink display is not updating, it will be set to a deep-sleep mode. This allows the battery to last for a longer time, as the power consumption of the E-Ink display in deep-sleep mode is about 500 times less than when updating the E-Ink display. The image below shows the layout of the E-Ink display, which includes the indoor and outdoor temperature readings, local time, historical reading graphs and a calendar.

Part 8: 3D printing of the base

The base, obtained from the LilyGo GitHub page was 3D printed in makerspace NYP. It provided the user with the convenience of power cable and doubles as a stand.



[i] https://github.com/Xinyuan-LilyGO/LilyGo-EPD47
[ii] https://github.com/markbirss/LilyGo-EPD-4-7-OWM-Weather-Display
[iii] https://openweathermap.org/api

Conclusion

This project has been a good learning experience for me as I have learnt how to obtain data through the internet, using API (Application Programming Interface) and third-party services. While Google Script service was chosen to obtain the calendar data, I have tried many other services such as IFITT, Zapier and Adafruit IO as well. Nevertheless, Google Script was chosen due to it being free to use, even though it was missing several features. Furthermore, I had learnt how to code more efficiently using C, in order to reduce the memory footprint of the program. This includes utilizing Char Arrays and sleep modes.

E-Ink displays proved to be a capable, easy to read and low power device. This project had allowed me to better understand how its underlying technology work, its advantages, and disadvantages.

Google Script to read calendar event

```
function doGet() {
  return ContentService.createTextOutput(GetEvents());
}

function ConvertTime(start, end) {
  var delta = Math.abs(end - start) / 1000;

  // calculate (and subtract) whole days
  var days = Math.floor(delta / 86400);
  delta -= days * 86400;

  // calculate (and subtract) whole hours
  var hours = Math.floor(delta / 3600) % 24;
  delta -= hours * 3600;

  // calculate (and subtract) whole minutes
  var minutes = Math.floor(delta / 60) % 60;

  delta -= minutes * 60;
  return { days, hours, minutes }
}

function GetEvents(mode) {
  var _calendarName = 'name';
  var Cal = CalendarApp.getCalendarsByName(_calendarName)[0];
  var Now = new Date();
  var Later = new Date();
  Later.setDate(Now.getDate() + 14);
  var events = Cal.getEvents(Now, Later);

  var myEvents = {};
  if (events.length > 10) {
    events.length = 10;
  }
  myEvents["Number of events"] = events.length;

  for (var i = 0; i < events.length; i++) {
    str = "";
    time = ConvertTime(Now, events[i].getStartTime())
    if (time.days > 1)
      str += time.days + " Days";
    else if (time.days == 1)
      str += time.days + " Day";
    else {
      if (time.hours > 1)
        str += time.hours + " Hours ";
      else if (time.hours == 1)
        str += time.hours + " Hour ";

      if (time.minutes < 10 && time.days == 0 && time.hours == 0)
        str += "Now";
      else if (time.minutes == 1)
        str += time.minutes + " Minute";
      else
        str += time.minutes + " Minutes";
    }
```

```
    var myEvent = {};

    var title = events[i].getTitle();
    if (title.length >= 20)
    {
      title = title.substring(0,20);
      title += "...";
    }
    myEvent["title"] = title;
    myEvent["time"] = str;
    Logger.log(myEvent);
    myEvents[i] = myEvent;
  }

  output = JSON.stringify(myEvents);
  Logger.log(output);
  return output;
}
```

Codes for Arduino E-Ink Display (sensitive data such as passwords are not included)

```
#ifndef BOARD_HAS_PSRAM
#error "Please enable PSRAM !!!"
#endif

#include <esp_task_wdt.h>       // In-built
#include "freertos/FreeRTOS.h"  // In-built
#include "freertos/task.h"      // In-built
#include "epd_driver.h"         // https://github.com/Xinyuan-LilyGO/LilyGo-EPD47
#include "esp_adc_cal.h"        // In-built
#include "Adafruit_SHT31.h"
#include <ArduinoJson.h>  // https://github.com/bblanchon/ArduinoJson
#include <HTTPClient.h>   // In-built
#include <WiFi.h>         // In-built
#include <WiFiClientSecure.h>
#include <time.h>  // In-built
#include <Wire.h>

const char *rootCACertificate =
  "-----BEGIN CERTIFICATE-----\n"
  "MIIDdTCCAl2gAwIBAgILBAAAAAABFUtaw5QwDQYJKoZIhvcNAQEFBQAwVzELMAkG\n"
  "A1UEBhMCQkUxGTAXBgNVBAoTEEdsb2JhbFNpZ24gbnYtc2ExEDAOBgNVBAsTB1Jv\n"
  "b3QgQ0ExGzAZBgNVBAMTEkdsb2JhbFNpZ24gUm9vdCBDQTAeFw05ODA5MDExMjAw\n"
  "MDBaFw0yODAxMjgxMjAwMDBaMFcxCzAJBgNVBAYTAkJFMRkwFwYDVQQKExBHbG9i\n"
  "YWxTaWduIG52LXNhMRAwDgYDVQQLEwdSb290IENBMRswGQYDVQQDExJHbG9iYWxT\n"
  "aWduIFJvb3QgQ0EwggEiMA0GCSqGSIb3DQEBAQUAA4IBDwAwggEKAoIBAQDaDuaZ\n"
  "jc6j40+Kfvvxi4Mla+pIH/EqsLmVEQS98GPR4mdmzxzdzxtIK+6NiY6arymAZavp\n"
  "xy0Sy6scTHAHoT0KMM0VjU/43dSMUBUc71DuxC73/OlS8pF94G3VNTCOXkNz8kHp\n"
  "1Wrjsok6Vjk4bwY8iGlbKk3Fp1S4bInMm/k8yuX9ifUSPJJ4ltbcdG6TRGHRjcdG\n"
  "snUOhugZitVtbNV4FpWi6cgKOOvyJBNPc1STE4U6G7weNLWLBYy5d4ux2x8gkasJ\n"
  "U26Qzns3dLlwR5EiUWMWea6xrkEmCMgZK9FGqkjWZCrXgzT/LCrBbBlDSgeF59N8\n"
  "9iFo7+ryUp9/k5DPAgMBAAGjQjBAMA4GA1UdDwEB/wQEAwIBBjAPBgNVHRMBAf8E\n"
  "BTADAQH/MB0GA1UdDgQWBBRge2YaRQ2XyolQL30EzTSo//z9SzANBgkqhkiG9w0B\n"
  "AQUFAAOCAQEA1nPnfE920I2/7LqivjTFKDK1fPxsnCwrvQmeU79rXqoRSLblCKOz\n"
  "yj1hTdNGCbM+w6DjY1Ub8rrvrTnhQ7k4o+YviiY776BQVvnGCv04zcQLcFGUl5gE\n"
```

```
  "38NflNUVyRRBnMRddWQVDf9VMOyGj/8N7yy5Y0b2qvzfvGn9LhJIZJrglfCm7ymP\n"
  "AbEVtQwdpf5pLGkkeB6zpxxxYu7KyJesF12KwvhHhm4qxFYxldBniYUr+WymXUad\n"
  "DKqC5JlR3XC321Y9YeRq4VzW9v493kHMB65jUr9TU/Qr6cf9tveCX4XSQRjbgbME\n"
  "HMUfpIBvFSDJ3gyICh3WZlXi/EjJKSZp4A==\n"
  "-----END CERTIFICATE-----\n";

#include "owm_credentials.h"
#include "forecast_record.h"
#include "lang.h"

#define SCREEN_WIDTH EPD_WIDTH
#define SCREEN_HEIGHT EPD_HEIGHT

enum alignment { LEFT,
                 RIGHT,
                 CENTER
               };
#define White 0xFF
#define LightGrey 0xBB
#define Grey 0x88
#define DarkGrey 0x44
#define Black 0x00

#define tempMode true
#define humiMode false
#define barchart_on true
#define barchart_off false

boolean LargeIcon = true;
boolean SmallIcon = false;
#define Large 20  // For icon drawing
#define Small 10  // For icon drawing
String Time_str = "--:--";
String Date_str = "-- --- ----";
int wifi_signal, CurrentHour = 0, CurrentMin = 0, CurrentSec = 0, EventCnt = 0, vref = 1100;
//################ PROGRAM VARIABLES and OBJECTS ################################################
#define max_readings 24

Forecast_record_type WxConditions[1];
Forecast_record_type WxForecast[max_readings];

StaticJsonDocument<8 * 1024> calEvent;

float localTemp;
float localHumi;

struct tm timeinfo;

RTC_DATA_ATTR float temperature_web_readings[max_readings] = { 0 };
RTC_DATA_ATTR float temperature_local_readings[max_readings] = { 0 };
RTC_DATA_ATTR float humidity_web_readings[max_readings] = { 0 };
RTC_DATA_ATTR float humidity_local_readings[max_readings] = { 0 };

Adafruit_SHT31 sht31 = Adafruit_SHT31();
```

```
long SleepDuration = 30;  // Sleep time in minutes, aligned to the nearest minute boundary, so if 30 will always update at 00 or 30 past the hour
int WakeupHour = 8;       // Wakeup after 07:00 to save battery power
int SleepHour = 24;       // Sleep  after 23:00 to save battery power
long StartTime = 0;
long SleepTimer = 0;
long Delta = 30;  // ESP32 rtc speed compensation, prevents display at xx:59:yy and then xx:00:yy (one minute later) to save power

//fonts
#include "opensans8b.h"
#include "opensans10b.h"
#include "opensans12b.h"
#include "opensans18b.h"
#include "opensans26b.h"
#include "uvi.h"

GFXfont currentFont;
uint8_t *framebuffer;

HTTPClient http;

void BeginSleep() {
  epd_poweroff_all();
  UpdateLocalTime();
  SleepTimer = 60 - CurrentSec;
  esp_sleep_enable_timer_wakeup(SleepTimer * 1000000LL);  // in Secs, 1000000LL converts to Secs as unit = 1uSec
  Serial.println("Awake for : " + String((millis() - StartTime) / 1000.0, 3) + "-secs");
  Serial.println("Entering " + String(SleepTimer) + " (secs) of sleep time");
  Serial.println("Starting deep-sleep period...");
  esp_deep_sleep_start();  // Sleep for e.g. 30 minutes
}

boolean SyncTime() {
  configTime(gmtOffset_sec, daylightOffset_sec, ntpServer, "time.nist.gov");  //(gmtOffset_sec, daylightOffset_sec, ntpServer)
  // Set the TZ environment variable
  setenv("TZ", Timezone, 1);
  tzset();
  delay(100);
  return UpdateLocalTime();
}

uint8_t StartWiFi() {
  Serial.println("\r\nConnecting to: " + String(ssid));
  IPAddress dns(8, 8, 8, 8);  // Use Google DNS
  WiFi.disconnect();
  WiFi.mode(WIFI_STA);  // switch off AP
  WiFi.setAutoConnect(true);
  WiFi.setAutoReconnect(true);
  WiFi.begin(ssid, password);
  if (WiFi.waitForConnectResult() != WL_CONNECTED) {
    WiFi.disconnect(false);
    delay(200);
```

```
    WiFi.begin(ssid, password);
  }
  if (WiFi.status() == WL_CONNECTED) {
    wifi_signal = WiFi.RSSI();
  } else
    Serial.println("WiFi connection *** FAILED ***");
  return WiFi.status();
}

void StopWiFi() {
  WiFi.disconnect();
  WiFi.mode(WIFI_OFF);
}

void InitialiseSystem() {
  StartTime = millis();
  //  pinMode(TOUCH_INT, INPUT);
  esp_sleep_enable_ext0_wakeup(GPIO_NUM_34, 0);

  Serial.begin(9600);
  Serial.println("Starting...");
  Wire.begin(15, 14);
  epd_init();
  framebuffer = (uint8_t *)ps_calloc(sizeof(uint8_t), EPD_WIDTH * EPD_HEIGHT / 2);
  if (!framebuffer) Serial.println("Memory alloc failed!");
  memset(framebuffer, 0xFF, EPD_WIDTH * EPD_HEIGHT / 2);
  setenv("TZ", Timezone, 1);
  tzset();
}

void loop() {}

void printLocalTime() {
  struct tm timeinfo;
  if (!getLocalTime(&timeinfo)) {
    Serial.println("Failed to obtain time");
    return;
  }
  Serial.println(&timeinfo, "%A, %B %d %Y %H:%M:%S");
}

RTC_DATA_ATTR bool mode = false;
RTC_DATA_ATTR bool UPDATE = false;

void drawCalendar() {
  epd_poweron();
  epd_clear();

  bool RxCalendar = false;
  bool Attempts = 1;
  if (StartWiFi() == WL_CONNECTED) {
    SyncTime();
    displayTime(940, 20, false);
    edp_update();
    while (Attempts <= 2) {
```

```
      if (obtainCalendarData()) {
        RxCalendar = true;
        break;
      }
      Attempts++;
    }
  }
  StopWiFi();
  if (RxCalendar) {
    DisplayCalendar(20, 20);

  } else {
    mode = !mode; //go back to weather mode
    UPDATE = true;
  }
}

#define calMode true
#define weatherMode false


void setup() {
  bool WakeUp;

  InitialiseSystem();
  if (!getLocalTime(&timeinfo)) {
    Serial.println("Local time not found...");
    if (StartWiFi() == WL_CONNECTED) {
      SyncTime();

      UPDATE = true;
    }
  }
  if (WakeupHour > SleepHour)
    WakeUp = (timeinfo.tm_hour >= WakeupHour || timeinfo.tm_hour <= SleepHour);
  else
    WakeUp = (timeinfo.tm_hour >= WakeupHour && timeinfo.tm_hour <= SleepHour);


  // mode switcher
  if (esp_sleep_get_wakeup_cause() == 2) {
    mode = !mode;
    UPDATE = true;
  }

  if (mode == calMode) {
    if ((timeinfo.tm_min == 0 || timeinfo.tm_min == 30 || UPDATE) && WakeUp) {
      drawCalendar();
      UPDATE = false;
    } else {
      epd_poweron();
      clearTime(700, 20);
      displayTime(940, 20, true);
    }
  }
```

```
  else {
    getLocalWeather();
    DisplayLocalWeather(418, 50);
    if ((timeinfo.tm_min == 0 || timeinfo.tm_min == 30 || UPDATE) && WakeUp) {
      int Attempts = 1;
      while (Attempts <= 2) {
        if (WiFi.status() == WL_CONNECTED) break;
        StartWiFi();
        SyncTime();
        Attempts += 1;
      }

      WiFiClient client;  // wifi client object
      bool RxWeather = obtainWeatherData(client);

      if (RxWeather) {
        LogWeather();  // Update history for graph
        displayTime(940, 120, false);
        epd_poweron();        // Switch on EPD display
        epd_clear();          // Clear the screen
        DisplayWebWeather();  // Display the weather data
        edp_update();         // Update the display to show the information
        bool RxCalendar = false;
        Attempts = 1;
        while (Attempts <= 2) {
          if (obtainCalendarData()) {
            RxCalendar = true;
            break;
          }
          Attempts++;
        }
        if (RxCalendar) {
          DisplayCalendar(430, 280);
          epd_poweron();  // Switch on EPD display
        }
        UPDATE = false;
      }
    }
    else {
      clearLocalWeather(418, 50);
      clearTime(700, 120);
      displayTime(940, 120, true);
      clearBattery(770, 0);
      DrawBattery(750, 20);
    }
  }
  edp_update();
  BeginSleep();
}

void DisplayCalendar(int x, int y) {
  int menuWidth = 650;
  int menuHeight = 80;
  int N = calEvent["Number of events"];
  if (mode == weatherMode) {
```

```
    menuWidth -= 150;
    menuHeight += 20;
    if (N > 2)
      N = 2;
  }
  for (int i = 0; i < N; i++) {
    const char *title = calEvent[String(i)]["title"];
    const char *time = calEvent[String(i)]["time"];
    drawRect(x, y, menuWidth, menuHeight, Black);
    setFont(OpenSans18B);
    if (mode == calMode)
      drawString(x + 27, y + 20, title, LEFT);
    else
      drawString(x + 20, y + 10, title, LEFT);

    setFont(OpenSans12B);
    if (mode == calMode)
      drawString(x + menuWidth - 20, y + 10, time, RIGHT);
    else
      drawString(x + 20, y + menuHeight - 40, time, LEFT);
    y += menuHeight + 20;
  }
}
void clearTime(int x, int y) {
  Rect_t area = {
    .x = x,
    .y = y,
    .width = 250,
    .height = 65,
  };
  epd_clear_area(area);
}

void clearBattery(int x, int y) {
  Rect_t area = {
    .x = x,
    .y = y,
    .width = 160,
    .height = 24,
  };
  epd_clear_area(area);
}
void displayTime(int x, int y, bool timezone) {
  String hr, min, apm;
  int h = timeinfo.tm_hour;
  int m = timeinfo.tm_min;
  if (timezone) {
    clearTime(x - 250, y);
  } else {
    setFont(OpenSans12B);
    drawString(x, y + 65, Date_str, RIGHT);
  }
  if (h > 24) {
    h -= 24;
  }
```

```
  if (h > 12) {
    hr = String(h - 12);
    apm = "pm";
  } else {
    hr = String(h);
    apm = "am";
  }
  if (m < 10) {
    min = "0" + String(m);
  } else {
    min = String(m);
  }
  setFont(OpenSans26B);
  String timeDisplay = hr + ":" + min + " " + apm;
  drawString(x, y, timeDisplay, RIGHT);
}

void getLocalWeather() {
  byte Attempts = 1;
  sht31.begin(0x44);
  while (Attempts <= 2) {
    localTemp = sht31.readTemperature();
    localHumi = sht31.readHumidity();
    if (!isnan(localTemp) && !isnan(localHumi)) {
      break;
    }
    Attempts++;
  }
}

void clearLocalWeather(int x, int y) {
  Rect_t area = {
    .x = x,
    .y = y,
    .width = 400,
    .height = 48,
  };
  epd_clear_area(area);
}

void DisplayLocalWeather(int x, int y) {
  setFont(OpenSans26B);
  if (!isnan(localTemp) && !isnan(localHumi)) {
    drawString(x, y, String(localTemp, 1) + "°C", LEFT);
    drawString(x + 220, y, String(localHumi, 1) + "%", LEFT);
  } else {
    drawString(x, y, "Not Connected", LEFT);
  }
}
void LogWeather() {
  if (!isnan(localTemp) && !isnan(localHumi)) {
    int r = max_readings - 2;
    while (r >= 0) {  // move all data to the next index
      temperature_local_readings[r + 1] = temperature_local_readings[r];
      temperature_web_readings[r + 1] = temperature_web_readings[r];
```

```
      humidity_local_readings[r + 1] = humidity_local_readings[r];
      humidity_web_readings[r + 1] = humidity_web_readings[r];
      r--;
    }
    r = max_readings - 1;
    temperature_local_readings[0] = localTemp;
    temperature_web_readings[0] = WxConditions[0].Temperature;
    humidity_local_readings[0] = localHumi;
    humidity_web_readings[0] = WxConditions[0].Humidity;
  }
}

bool DecodeWeather(WiFiClient &json) {
  //  Serial.println(F("\nCreating object... "));
  DynamicJsonDocument doc(64 * 1024);                       // allocate the JsonDocument
  DeserializationError error = deserializeJson(doc, json);  // Deserialize the JSON document
  if (error) {                                              // Test if parsing succeeds.
    Serial.print(F("deserializeJson() failed: "));
    Serial.println(error.c_str());
    return false;
  }
  // convert it to a JsonObject
  JsonObject root = doc.as<JsonObject>();
  //  Serial.println(" Decoding " + Type + " data");
  WxConditions[0].FTimezone = doc["timezone_offset"];  // "0"
  JsonObject current = doc["current"];
  WxConditions[0].Temperature = current["temp"];
  WxConditions[0].FeelsLike = current["feels_like"];
  WxConditions[0].Humidity = current["humidity"];
  WxConditions[0].UVI = current["uvi"];               //Serial.println("UVin: " + String(WxConditions[0].
UVI));
  WxConditions[0].Cloudcover = current["clouds"];     //Serial.println("CCov: " + String(WxConditions[0].
Cloudcover));
  WxConditions[0].Visibility = current["visibility"];  //Serial.println("Visi: " + String(WxConditions[0].
Visibility));
  JsonObject current_weather = current["weather"][0];
  String Description = current_weather["description"];  // "scattered clouds"
  String Icon = current_weather["icon"];                // "01n"
  WxConditions[0].Forecast0 = Description;              //Serial.println("Fore: " + String(WxConditions[0]
.Forecast0));
  WxConditions[0].Icon = Icon;                          //Serial.println("Icon: " + String(WxConditions[0]
.Icon));
  Serial.println("T: " + String(WxConditions[0].Temperature) + " H: " + String(WxConditions[0].Humidity));
  return true;
}
//#########################################################################################
String ConvertUnixTime(int unix_time) {
  // Returns either '21:12  ' or ' 09:12pm' depending on Units mode
  time_t tm = unix_time;
  struct tm *now_tm = localtime(&tm);
  char output[40];
  strftime(output, sizeof(output), "%I:%M%P %m/%d/%y", now_tm);

  return output;
}
```

```
bool obtainCalendarData() {
  WiFiClientSecure *client = new WiFiClientSecure;
  client->setCACert(rootCACertificate);
  http.setFollowRedirects(HTTPC_STRICT_FOLLOW_REDIRECTS);
  http.begin(*client, "https://script.google.com/macros/s/XXXXXXXXXXXXXXXXXX/exec");
  int httpCode = http.GET();
  if (httpCode == HTTP_CODE_OK) {
    DeserializationError error = deserializeJson(calEvent, http.getString());

    if (error) {
      Serial.print(F("deserializeJson() failed: "));
      Serial.println(error.f_str());
      return false;
    } else {

      http.end();
      delete client;
      return true;
    }
  } else {
    Serial.printf("connection failed, error: %s", http.errorToString(httpCode).c_str());
    http.end();
    delete client;
    return false;
  }
}
//#################################################################################################
bool obtainWeatherData(WiFiClient &client) {
  String uri = "/data/2.5/onecall?lat=" + Latitude + "&lon=" + Longitude + "&units=metric&appid=" + apikey
OWM + "&mode=json&lang=" + Language + "&exclude=minutely,hourly,alerts,daily";
  int Attempts = 1;
  while (Attempts <= 2) {  // Try up-to 2 time for Weather
    client.stop();
    http.begin(client, serverOWM, 80, uri);  //http.begin(uri,test_root_ca); //HTTPS example connection
    if (http.GET() == HTTP_CODE_OK)
      if (DecodeWeather(http.getStream()))
        return true;
  }
  client.stop();
  http.end();
  return false;
}

float hPa_to_inHg(float value_hPa) {
  return 0.02953 * value_hPa;
}

int JulianDate(int d, int m, int y) {
  int mm, yy, k1, k2, k3, j;
  yy = y - (int)((12 - m) / 10);
  mm = m + 9;
  if (mm >= 12) mm = mm - 12;
  k1 = (int)(365.25 * (yy + 4712));
  k2 = (int)(30.6001 * mm + 0.5);
```

```
  k3 = (int)((int)((yy / 100) + 49) * 0.75) - 38;
  // 'j' for dates in Julian calendar:
  j = k1 + k2 + d + 59 + 1;
  if (j > 2299160) j = j - k3;  // 'j' is the Julian date at 12h UT (Universal Time) For Gregorian calenda
r:
  return j;
}

float SumOfPrecip(float DataArray[], int readings) {
  float sum = 0;
  for (int i = 0; i <= readings; i++) sum += DataArray[i];
  return sum;
}

String TitleCase(String text) {
  if (text.length() > 0) {
    String temp_text = text.substring(0, 1);
    temp_text.toUpperCase();
    return temp_text + text.substring(1);  // Title-case the string
  } else
    return text;
}

void DisplayWebWeather() {                    // 4.7" e-paper display is 960x540 resolution
  DisplayStatusSection(600, 20, wifi_signal);  // Wi-Fi signal strength and Battery voltage
  DisplayGeneralInfoSection();                // Top line of the display
  DisplayMainWeatherSection(450, 160);        // Centre section of display for Location, temperature, Wea
ther report, current Wx Symbol
  DisplayGraphSection(320, 60);               // Graphs of temperature and humidity
}

void DisplayGeneralInfoSection() {
  setFont(OpenSans10B);
  drawString(5, 2, City, LEFT);
  setFont(OpenSans8B);
  drawString(SCREEN_WIDTH / 2, 2, Date_str + "  @   " + Time_str, CENTER);
}

void DisplayMainWeatherSection(int x, int y) {
  setFont(OpenSans8B);
  DisplayTempHumiPressSection(x, y - 60);
  DisplayForecastTextSection(x - 60, y + 5);
  DisplayVisiCCoverUVISection(x - 5, y + 55);
}

void DisplayTempHumiPressSection(int x, int y) {
  setFont(OpenSans18B);
  drawString(x - 30, y, String(WxConditions[0].Temperature, 1) + "°C   " + String(WxConditions[0].Humidity
, 0) + "%", LEFT);
  setFont(OpenSans12B);
  int Yoffset = 42;
  drawString(x - 30, y + Yoffset, String(WxConditions[0].FeelsLike, 1) + "° FL", LEFT);  // Show FeelsLike
 temperature if windspeed > 0
}
```

```
void DisplayForecastTextSection(int x, int y) {
#define lineWidth 34
  setFont(OpenSans12B);
  String Wx_Description = WxConditions[0].Forecast0;
  Wx_Description.replace(".", "");  // remove any '.'
  int spaceRemaining = 0, p = 0, charCount = 0, Width = lineWidth;
  while (p < Wx_Description.length()) {
    if (Wx_Description.substring(p, p + 1) == " ") spaceRemaining = p;
    if (charCount > Width - 1) {  // '~' is the end of line marker
      Wx_Description = Wx_Description.substring(0, spaceRemaining) + "~" + Wx_Description.substring(spaceR
emaining + 1);
      charCount = 0;
    }
    p++;
    charCount++;
  }
  if (WxForecast[0].Rainfall > 0) Wx_Description += " (" + String(WxForecast[0].Rainfall, 1) + "mm)";
  String Line1 = Wx_Description.substring(0, Wx_Description.indexOf("~"));
  String Line2 = Wx_Description.substring(Wx_Description.indexOf("~") + 1);
  drawString(x + 30, y + 5, TitleCase(Line1), LEFT);
  if (Line1 != Line2) drawString(x + 30, y + 30, Line2, LEFT);
}

void DisplayVisiCCoverUVISection(int x, int y) {
  setFont(OpenSans12B);
  CloudCover(x + 10, y, WxConditions[0].Cloudcover);
  Display_UVIndexLevel(x + 120, y, WxConditions[0].UVI);
}

void Display_UVIndexLevel(int x, int y, float UVI) {
  String Level = "";
  if (UVI <= 2) Level = " (L)";
  if (UVI >= 3 && UVI <= 5) Level = " (M)";
  if (UVI >= 6 && UVI <= 7) Level = " (H)";
  if (UVI >= 8 && UVI <= 10) Level = " (VH)";
  if (UVI >= 11) Level = " (EX)";
  drawString(x + 20, y - 5, String(UVI, (UVI < 0 ? 1 : 0)) + Level, LEFT);
  DrawUVI(x - 10, y - 5);
}


void DisplayGraphSection(int x, int gy) {
  int r = 0;
  int gwidth = 300, gheight = 170;
  int gx = (SCREEN_WIDTH - gwidth * 2) / 5 + 8;
  int gap = gheight + 60;
  // (x,y,width,height,MinValue, MaxValue, Title, Data Array, AutoScale)
  DrawGraph(gx, gy, gwidth, gheight, TXT_TEMPERATURE_C, temperature_web_readings, temperature_local_readin
gs, max_readings, tempMode, 1);
  DrawGraph(gx, gy + gap, gwidth, gheight, TXT_HUMIDITY_PERCENT, humidity_web_readings, humidity_local_rea
dings, max_readings, humiMode, 10);
  drawLegend(gx, gy + 2 * gap - 10);
}

void drawLegend(int x, int y) {
```

```
#define lineLength 30
  setFont(OpenSans10B);
  drawString(x, y, String("Web"), LEFT);

  x += 70;
  y += 10;

  // web dataset
  drawLine(x, y - 1, x + lineLength, y - 1, Black);  // Two lines for hi-res display
  drawLine(x, y, x + lineLength, y, Black);

  x += lineLength + 60;
  y -= 10;

  drawString(x, y, String("Local"), LEFT);

  x += 80;
  y += 10;

  // local dataset
  drawLine(x, y - 2, x + lineLength, y - 2, Black);
  drawLine(x, y - 1, x + lineLength, y - 1, Black);
  drawLine(x, y, x + lineLength, y, Black);
  drawLine(x, y + 1, x + lineLength, y + 1, Black);
  drawLine(x, y + 2, x + lineLength, y + 2, Black);
}


void arrow(int x, int y, int asize, float aangle, int pwidth, int plength) {
  float dx = (asize - 10) * cos((aangle - 90) * PI / 180) + x;  // calculate X position
  float dy = (asize - 10) * sin((aangle - 90) * PI / 180) + y;  // calculate Y position
  float x1 = 0;
  float y1 = plength;
  float x2 = pwidth / 2;
  float y2 = pwidth / 2;
  float x3 = -pwidth / 2;
  float y3 = pwidth / 2;
  float angle = aangle * PI / 180 - 135;
  float xx1 = x1 * cos(angle) - y1 * sin(angle) + dx;
  float yy1 = y1 * cos(angle) + x1 * sin(angle) + dy;
  float xx2 = x2 * cos(angle) - y2 * sin(angle) + dx;
  float yy2 = y2 * cos(angle) + x2 * sin(angle) + dy;
  float xx3 = x3 * cos(angle) - y3 * sin(angle) + dx;
  float yy3 = y3 * cos(angle) + x3 * sin(angle) + dy;
  fillTriangle(xx1, yy1, xx3, yy3, xx2, yy2, Black);
}

void DrawSegment(int x, int y, int o1, int o2, int o3, int o4, int o11, int o12, int o13, int o14) {
  drawLine(x + o1, y + o2, x + o3, y + o4, Black);
  drawLine(x + o11, y + o12, x + o13, y + o14, Black);
}


void DisplayStatusSection(int x, int y, int rssi) {
  setFont(OpenSans8B);
```

```
  DrawRSSI(x + 305, y + 15, rssi);
  DrawBattery(x + 150, y);
}

void DrawRSSI(int x, int y, int rssi) {
  int WIFIsignal = 0;
  int xpos = 1;
  for (int _rssi = -100; _rssi <= rssi; _rssi = _rssi + 20) {
    if (_rssi <= -20) WIFIsignal = 30;  //            <-20dbm displays 5-bars
    if (_rssi <= -40) WIFIsignal = 24;  //  -40dbm to  -21dbm displays 4-bars
    if (_rssi <= -60) WIFIsignal = 18;  //  -60dbm to  -41dbm displays 3-bars
    if (_rssi <= -80) WIFIsignal = 12;  //  -80dbm to  -61dbm displays 2-bars
    if (_rssi <= -100) WIFIsignal = 6;  // -100dbm to  -81dbm displays 1-bar
    fillRect(x + xpos * 8, y - WIFIsignal, 6, WIFIsignal, Black);
    xpos++;
  }
}

boolean UpdateLocalTime() {
  char time_output[30], day_output[30], update_time[30];
  while (!getLocalTime(&timeinfo, 5000)) {  // Wait for 5-sec for time to synchronise
    Serial.println("Failed to obtain time");
    return false;
  }
  CurrentHour = timeinfo.tm_hour;
  CurrentMin = timeinfo.tm_min;
  CurrentSec = timeinfo.tm_sec;
  //See http://www.cplusplus.com/reference/ctime/strftime/
  Serial.println(&timeinfo, "%a %b %d %Y   %H:%M:%S");  // Displays: Saturday, June 24 2017 14:05:49
  sprintf(day_output, "%s, %02u %s %04u", weekday_D[timeinfo.tm_wday], timeinfo.tm_mday, month_M[timeinfo.
tm_mon], (timeinfo.tm_year) + 1900);
  strftime(update_time, sizeof(update_time), "%r", &timeinfo);  // Creates: '@ 02:05:49pm'
  sprintf(time_output, "%s", update_time);
  Date_str = day_output;
  Time_str = time_output;
  return true;
}

void DrawBattery(int x, int y) {
  setFont(OpenSans8B);
  uint8_t percentage = 100;
  esp_adc_cal_characteristics_t adc_chars;
  esp_adc_cal_value_t val_type = esp_adc_cal_characterize(ADC_UNIT_1, ADC_ATTEN_DB_11, ADC_WIDTH_BIT_12, 1
100, &adc_chars);
  if (val_type == ESP_ADC_CAL_VAL_EFUSE_VREF) {
    vref = adc_chars.vref;
  }
  float voltage = analogRead(36) / 4096.0 * 6.566 * (vref / 1000.0);
  if (voltage > 1) {  // Only display if there is a valid reading
    Serial.println("\nVoltage = " + String(voltage));
    percentage = 2836.9625 * pow(voltage, 4) - 43987.4889 * pow(voltage, 3) + 255233.8134 * pow(voltage, 2
) - 656689.7123 * voltage + 632041.7303;
    if (voltage >= 4.20) percentage = 100;
    if (voltage <= 3.20) percentage = 0;  // orig 3.5
    drawRect(x + 25, y - 14, 40, 15, Black);
```

```
    fillRect(x + 65, y - 10, 4, 7, Black);
    fillRect(x + 27, y - 12, 36 * percentage / 100.0, 11, Black);
    drawString(x + 85, y - 14, String(percentage) + "%  " + String(voltage, 1) + "v", LEFT);
  }
}

// Symbols are drawn on a relative 10x10grid and 1 scale unit = 1 drawing unit
void addcloud(int x, int y, int scale, int linesize) {
  fillCircle(x - scale * 3, y, scale, Black);
  // Left most circle
  fillCircle(x + scale * 3, y, scale, Black);
  // Right most circle
  fillCircle(x - scale, y - scale, scale * 1.4, Black);
  // left middle upper circle
  fillCircle(x + scale * 1.5, y - scale * 1.3, scale * 1.75, Black);
  // Right middle upper circle
  fillRect(x - scale * 3 - 1, y - scale, scale * 6, scale * 2 + 1, Black);
  // Upper and lower lines
  fillCircle(x - scale * 3, y, scale - linesize, White);
  // Clear left most circle
  fillCircle(x + scale * 3, y, scale - linesize, White);
  // Clear right most circle
  fillCircle(x - scale, y - scale, scale * 1.4 - linesize, White);
  // left middle upper circle
  fillCircle(x + scale * 1.5, y - scale * 1.3, scale * 1.75 - linesize, White);
  // Right middle upper circle
  fillRect(x - scale * 3 + 2, y - scale + linesize - 1, scale * 5.9, scale * 2 - linesize * 2 + 2, White);
  // Upper and lower lines
}

void CloudCover(int x, int y, int CloudCover) {
  addcloud(x - 9, y, Small * 0.3, 2);      // Cloud top left
  addcloud(x + 3, y - 2, Small * 0.3, 2);  // Cloud top right
  addcloud(x, y + 15, Small * 0.6, 2);     // Main cloud
  drawString(x + 30, y, String(CloudCover) + "%", LEFT);
}


void DrawUVI(int x, int y) {
  Rect_t area = {
    .x = x, .y = y, .width = uvi_width, .height = uvi_height
  };
  epd_draw_grayscale_image(area, (uint8_t *)uvi_data);
}

void DrawGraph(int x_pos, int y_pos, int gwidth, int gheight, String title, float DataArrayWeb[], float Da
taArrayLocal[], int readings, boolean temperature, int margin) {
#define temperature_margin 0  // Sets the autoscale increment, so axis steps up after a change of e.g. 3
#define y_minor_axis 5        // 5 y-axis division markers
  setFont(OpenSans10B);
  // set the max and min as first value
  int maxYscale = DataArrayWeb[0];
  int minYscale = DataArrayWeb[0];
  int last_x, last_web_y, last_local_y;
  float x2, y2_web, y2_local;
```

```
  for (int i = 0; i < readings; i++) {
    if (DataArrayWeb[i] != 0 && DataArrayLocal[i] != 0) {
      if (DataArrayWeb[i] >= maxYscale) maxYscale = DataArrayWeb[i];
      if (DataArrayWeb[i] <= minYscale) minYscale = DataArrayWeb[i];
      if (DataArrayLocal[i] >= maxYscale) maxYscale = DataArrayLocal[i];
      if (DataArrayLocal[i] <= minYscale) minYscale = DataArrayLocal[i];
    }
  }
  int Y1Max = round(maxYscale + 0.5) + margin;
  int Y1Min = round(minYscale) - margin;
  // Draw the graph line
  last_x = x_pos + 1;
  last_web_y = y_pos + (Y1Max - constrain(DataArrayWeb[1], Y1Min, Y1Max)) / (Y1Max - Y1Min) * gheight;
  last_local_y = y_pos + (Y1Max - constrain(DataArrayLocal[1], Y1Min, Y1Max)) / (Y1Max - Y1Min) * gheight;
  drawRect(x_pos, y_pos, gwidth + 3, gheight + 2, Grey);
  drawString(x_pos, y_pos - 28, title, LEFT);
  for (int gx = 0; gx < readings; gx++) {
    if (DataArrayWeb[gx] != 0 && DataArrayLocal[gx] != 0) {
      x2 = x_pos + gx * gwidth / (readings - 1) - 1;  // max_readings is the global variable that sets the
 maximum data that can be plotted
      y2_web = y_pos + (Y1Max - constrain(DataArrayWeb[gx], Y1Min, Y1Max)) / (Y1Max - Y1Min) * gheight + 1
;
      y2_local = y_pos + (Y1Max - constrain(DataArrayLocal[gx], Y1Min, Y1Max)) / (Y1Max - Y1Min) * gheight
 + 1;

      // web dataset
      drawLine(last_x, last_web_y - 1, x2, y2_web - 1, Black);  // Two lines for hi-res display
      drawLine(last_x, last_web_y, x2, y2_web, Black);

      // local dataset
      drawLine(last_x, last_local_y - 2, x2, y2_local - 2, Black);
      drawLine(last_x, last_local_y - 1, x2, y2_local - 1, Black);
      drawLine(last_x, last_local_y, x2, y2_local, Black);
      drawLine(last_x, last_local_y + 1, x2, y2_local + 1, Black);
      drawLine(last_x, last_local_y + 2, x2, y2_local + 2, Black);

      last_x = x2;
      last_web_y = y2_web;
      last_local_y = y2_local;
    }
    last_x = x_pos + gx * gwidth / (readings - 1) - 1;
  }
  //Draw the Y-axis scale
#define number_of_dashes 20
  float y_lbl;
  for (int spacing = 0; spacing <= y_minor_axis; spacing++) {
    for (int j = 0; j < number_of_dashes; j++) {  // Draw dashed graph grid lines
      if (spacing < y_minor_axis) drawFastHLine((x_pos + 3 + j * gwidth / number_of_dashes), y_pos + (ghei
ght * spacing / y_minor_axis), gwidth / (2 * number_of_dashes), Grey);
    }
    y_lbl = Y1Max - (float)(Y1Max - Y1Min) / y_minor_axis * spacing;
    if (temperature) {
      drawString(x_pos - 10, y_pos + gheight * spacing / y_minor_axis - 5, String(y_lbl, 1), RIGHT);
    } else {
      drawString(x_pos - 7, y_pos + gheight * spacing / y_minor_axis - 5, String(y_lbl, 0), RIGHT);
```

```
    }
  }

  int graphHours = max_readings / (60 / SleepDuration);
  // Todo: add var for time div
  for (int i = 0; i < graphHours + 1; i += 2) {
    drawString(x_pos + gwidth / graphHours * i, y_pos + gheight + 10, String(i), CENTER);
    if (i < graphHours - 2) drawFastVLine(x_pos + gwidth / graphHours * (i + 1) + gwidth / graphHours, y_p
os, gheight, LightGrey);
  }
}

void drawString(int x, int y, String text, alignment align) {
  char *data = const_cast<char *>(text.c_str());
  int x1, y1;  //the bounds of x,y and w and h of the variable 'text' in pixels.
  int w, h;
  int xx = x, yy = y;
  get_text_bounds(&currentFont, data, &xx, &yy, &x1, &y1, &w, &h, NULL);
  if (align == RIGHT) x = x - w;
  if (align == CENTER) x = x - w / 2;
  int cursor_y = y + h;
  write_string(&currentFont, data, &x, &cursor_y, framebuffer);
}

void fillCircle(int x, int y, int r, uint8_t color) {
  epd_fill_circle(x, y, r, color, framebuffer);
}

void drawFastHLine(int16_t x0, int16_t y0, int length, uint16_t color) {
  epd_draw_hline(x0, y0, length, color, framebuffer);
}

void drawFastVLine(int16_t x0, int16_t y0, int length, uint16_t color) {
  epd_draw_vline(x0, y0, length, color, framebuffer);
}

void drawLine(int16_t x0, int16_t y0, int16_t x1, int16_t y1, uint16_t color) {
  epd_write_line(x0, y0, x1, y1, color, framebuffer);
}

void drawCircle(int x0, int y0, int r, uint8_t color) {
  epd_draw_circle(x0, y0, r, color, framebuffer);
}

void drawRect(int16_t x, int16_t y, int16_t w, int16_t h, uint16_t color) {
  epd_draw_rect(x, y, w, h, color, framebuffer);
}

void fillRect(int16_t x, int16_t y, int16_t w, int16_t h, uint16_t color) {
  epd_fill_rect(x, y, w, h, color, framebuffer);
}

void fillTriangle(int16_t x0, int16_t y0, int16_t x1, int16_t y1,
                  int16_t x2, int16_t y2, uint16_t color) {
  epd_fill_triangle(x0, y0, x1, y1, x2, y2, color, framebuffer);
```

```
}

void drawPixel(int x, int y, uint8_t color) {
  epd_draw_pixel(x, y, color, framebuffer);
}

void setFont(GFXfont const &font) {
  currentFont = font;
}
void edp_update() {
  epd_draw_grayscale_image(epd_full_screen(), framebuffer);  // Update the screen
}
```