Weather Station Build

This Weather Station is just an example of many possible builds using different materials. Depending on what sensors you choose to incorporate, this build is designed to collect up to 4 types of weather data: temperature, humidity, wind speed and (optionally) barometric pressure. Essentially, there are 4 major components that can be adapted to your design preference: Container (+fastener), sensors, processor and transmitter.

Components:

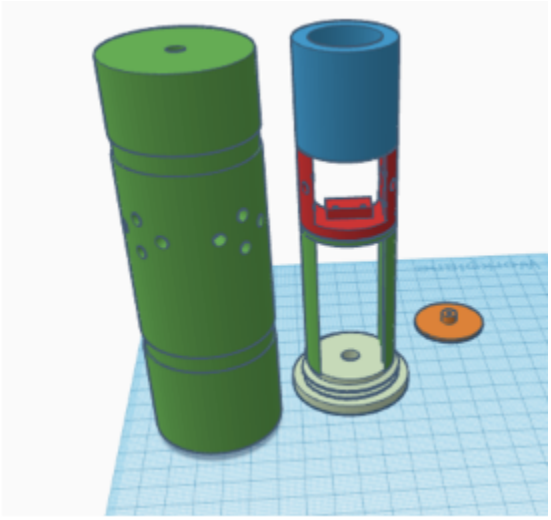
Container:
Purchased or Repurposed

Any container that is big enough to hold the components AND is waterproof (sun and rain). Take home food and soup containers

https://www.amazon.com/Containers-Disposable-Insulated-Microwaveable-Dishwasher/dp/B07BLTDD1V

3D printed

You can design your own (or use our example seen below)



https://github.com/CIJE-STEM/CIJE-WeatherHub/tree/main/3D_files/CIJE_Weather_Station1

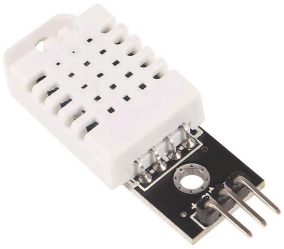
That can be adapted via Tinkercad

https://www.tinkercad.com/things/kYxrZfITcn0-cije-weather-station?sharecode=hQLnVu-pN1FPCwDh6w1JOU05zTvHCK09-pb4Zir9fJ0

Fasteners:

The container needs to be anchored/attached to sometime stable to withstand wind and rain. Zip ties are the easiest

Sensors:

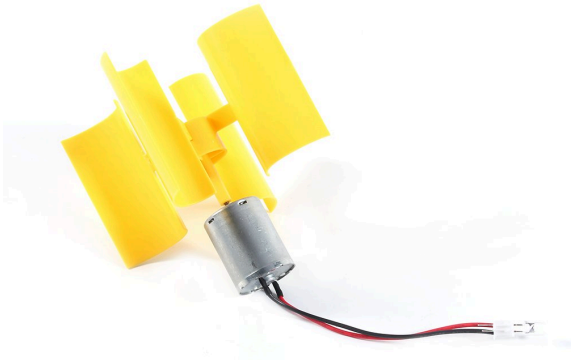


Temperature/Humidity sensor

- **DHT22/DHT11** - I2C device measure temperature and humidity
- **BME280** - I2C device measures temperature, humidity and barometric pressure.

Wind Speed:

Anemometer - We are suggesting an anemometer adapted from a DC motor wind turbine. The voltage output will need to be calibrated to the wind speed.



https://www.amazon.com/Turbine-Vertical-Turbines-Electricity-Generator/dp/B0BWWVQZTG/ref=sr_1_16

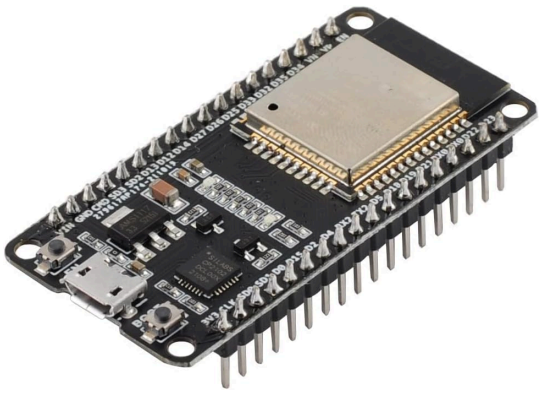
Time: (optional)

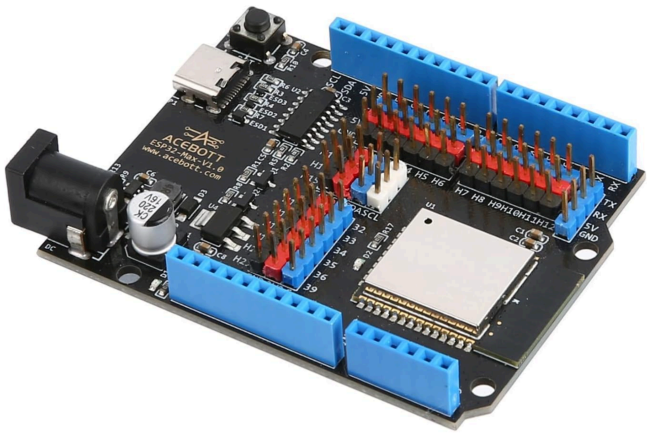
The **DS3231** A real time clock (RTC) is great to have to put the microprocessor to sleep and not use up the power (unless you manage to wire the power) when not in use / when not transmitting the data. There are other versions that are also available. This particular model has a rechargeable battery vs. a coin cell battery that will eventually need to be replaced.

https://www.amazon.com/Dorhea-DS3231-Module-Memory-Raspberry/dp/B08X4H3NBR/ref=sr_1_3?

Processor/Transmitter

ESP32 (Processor AND transmitter}





Build-Up Options:

Your weather station can be built with a wide variety of options which will be explored. These options may be individual or combined depending on your needs.

Here are possible configurations:

Basic:
Internal sensors require minimal fabrication.

- Temperature, Humidity
    - DHT-11, DHT22 Sensors.
    - AHT10 Sensor.
- Temperature, Humidity, Pressure.
    - BME280.
- Temperature, Humidity, Pressure, VOC's (Volatile Organic Compounds - Air Quality)
    - BME680.

Intermediate:
Sensors requiring some physical material fabrication, holes, mounting materials.

- (Basic Components - as listed above).
- Anemometer - DC Generator, resistors and a mounting bracket, hole for wiring.
- UV Sensor
    - Adafruit LTR-390 - requires hole and mount with weather protect window for sensor.

Advanced:
Sensors that require moderate to extensive mechanical fabrication and adaptation, external holes.

- Wind Direction Sensor - rotary magnetic, optical or mechanical switch sensing with an air vane to detect wind air flow direction.
    - AS5600 - rotary magnetic position sensor
    - A3144 Hall Effect sensor or A3144 Hall Effect sensor module board, magnet(s)
    - TCRT5000 Reflectance sensor(s), position disk
- Participation Sensor (Rain Gauge)
    - A3144 Hall Sensor, 3D printed tipping bucket assembly
    - LM393 Rain Drop Sensor Monitor
    - Capacitive Analog Soil Sensor Module
    - Floating Rain gauge sensor with ultrasonic or TOF (Time of Flight) distance sensor

The above options may be selected or combined to create your weather station. We will explore some of these options with design examples to help you get started.

First Steps

- What is the intended application of your weather station?
    - Classroom project, school project?
    - Do you have an intended time frame for installation and use?
    - Can you get permission and a designated place to install the weather station?
- Consider the location of your weather station.
    - Is it to be inside or outside - Are measuring indoor air quality, lighting, etc.
    - Does the location have readily available power?
    - Does the location have readily available WiFi?
    - Is the location easily accessible?
    - Is the location secure?
- What kind of data do you wish to collect?
    - Some sensors are easy to use, and require very little maintenance others may require occasional repair or adjustment.
    - Are you considering microclimate measurement with multiple stations or having a relocatable station?
    - How will you share the data with your students, school, community?
    - Do you expect that your station and data will be used to help plan or make decisions based on the data you collect? For example a community garden, air quality monitoring?

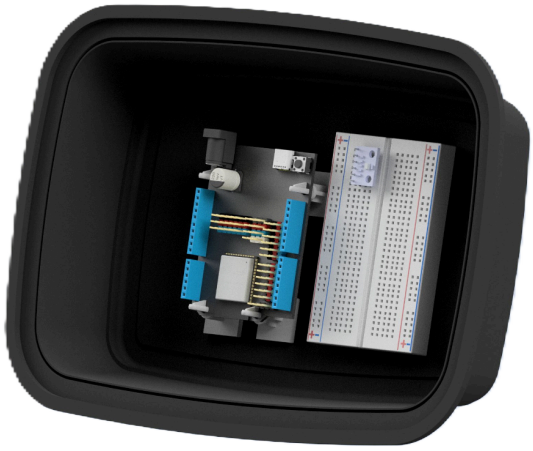
  Based on the questions above hopefully you can formulate the intent and purpose of the weather station as well as what kind of data you wish to acquire/use.

Project Planning:

Consider the scope of your project and the resource that you will be able to apply. This project can be incremental and scale based on the resources that you have at hand. Consider a part of this project could be to build a case for expansion and permanence at your school, location. It may be useful to start small, demonstrate utility or interest and then expand.

Design Example - Basic - Web Based Reporting

In this example we want a weather station that can be located outside but next to a classroom window. We may find a better location later but for now we would like to have something that we can experiment with and easily observe. Power will be routed through the window using a 12V DC power adapter. Wi-Fi is available and fairly reliable next to the window and we have permission from our IT department to use the network. Our school is happy and interested in having the data posted on the CIJE Weather Station Hub site, so our IT department will allow outgoing internet traffic from our weather station.



Shown above is an image of a basic weather station using an ESP32 (ACEBOTT), a half bread board and a BME280 sensor. This is a very simple example of a web posting weather station.

We have some wiring to do, coding (most of it is provided) and some testing.

Weather Station Specifications:

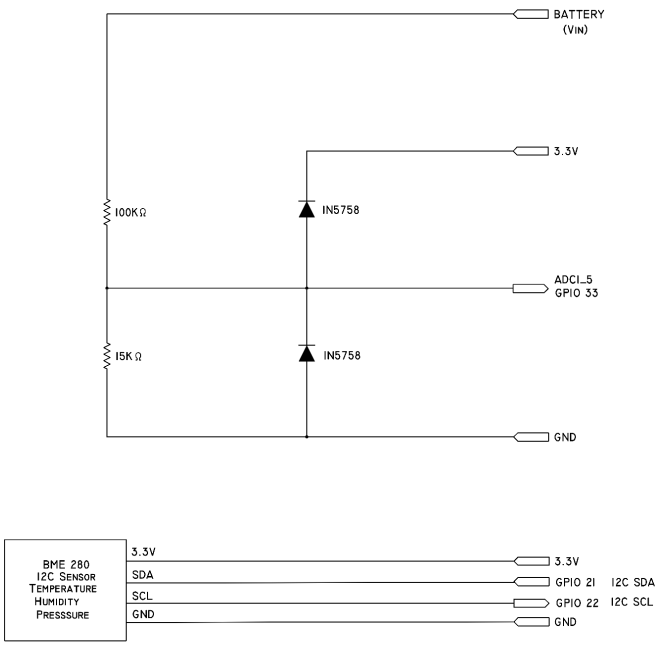
#	Sensing Parameter	Sensor	Notes
1	Temperature	BME280 - I2C - 3.3V-5V	Amazon - BME280 not BMP280
2	Humidity	BME280 - I2C - 3.3V-5V	
3	Pressure	BME280 - I2C - 3.3V-5V	
#	Hardware	Type	
1	ESP32	ESP32 Board	ACEBOTT UNO format ESP32
2	Breadboard	Mini Breadboard	Mount components for easy changes
3	1N5817 Diode (2)	Overvoltage Protection	1N5817 Schottky Diode
4	12VDC Adapter	12V 1A 5.5mm x 2.1mm	Power for Station
5	Plastic Container	48 oz, 7” x 10” x 2”	Station Enclosure - Zip-tied to block

6	Snap-in Support	PCB Mounting Post	Amazon PCB Support
7	Dupont Jumpers	Female to Male (10)	Component interconnects
8	Hot Glue		To secure items if required
	Network		
	SSID	URSchool	
	Password	123456	
	IT Contact	Phone Number	email
	Jim Dandy	555-2134	jd@ITOffice.biz

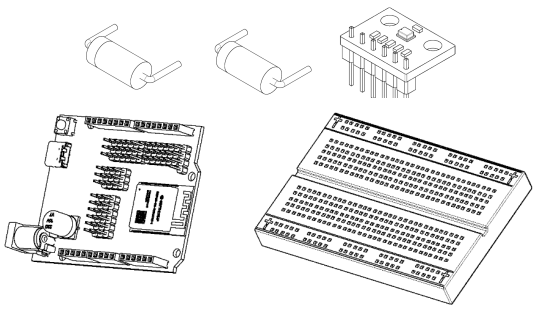
Note:

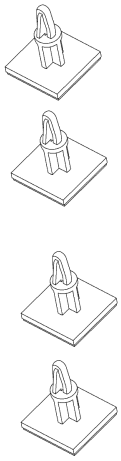
Resources listed are suggested components and vendors, you may use whatever source you see fit. It is recommended that component substitutes be selected based on similar specifications. Please consult your High School Mentor if in doubt prior to purchasing.

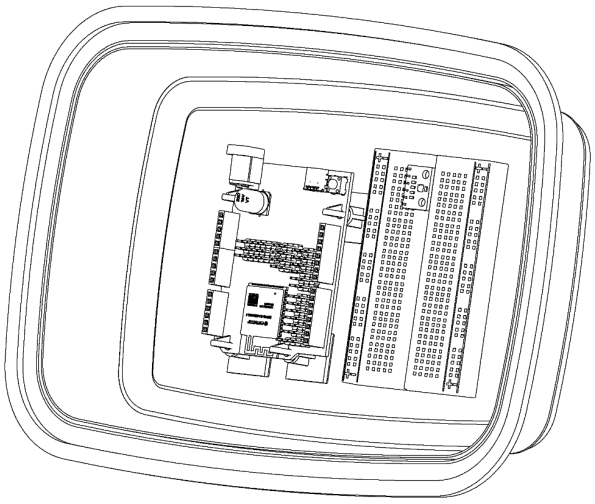
ESP32 with BME280 and Voltage Divider Schematic



Food Tray Container	1
AceBott ESp32	1
Half Breadboard	1
BME280	1
1N5817 Diode	2
Snap-in Support	3-4







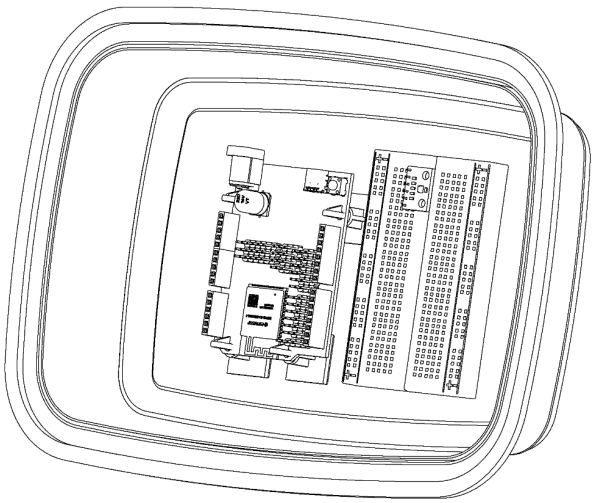
After gathering the above components, place them in the bottom of the Food Tray Container. Do a visual check first by placing them centered both vertically and horizontally next to each other. The easiest way to secure them is to use the adhesive Snap-in Support and the using the adhesive backing on the half breadboard. Mark or try positioning the ESP32 board and the half breadboard first and mark or identify where they go.

Steps:

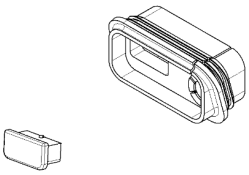
- Insert the supports into the AceBott ESP32 board, note it is suggested that once in place you apply hot glue to the adhesive pads to add another layer of attachment to the food tray.
- Remove the adhesive, using the previous positioning and place it in the bottom.
- Remove the adhesive backing on the half breadboard and firmly place it next to the ESP32.
- You may insert the BME280 into the breadboard where you think it will work best.
    - The BME280 pins VCC, GND, SDA, SCL should be accessible via other pins on the breadboard.
    - You will connect pins via Dupont jumpers to the ESP32.

Now that you have mounted the components, make a couple of notes.

- The location of the programming port.
- The location of the power port.



Using the above information consider how you want to connect those ports to the outside. The programming port will only be used initially to program the weather station and may only be required for changes. Power will be required to operate the weather station and should be both secure (not easily removed) and weatherproof.
We have designed a weather seal that you may 3D print if you like. We have a cutout template as well. A small USB C port plug is also included. You will find the 3D STL file and the cutout template in the weather station resource drive.



After using the cutout and inserting the weather seal you may want to use Silicone RTV to attach the seal. RTV is a good adhesive to use, it is weatherproof, remains pliable and is easy to remove.

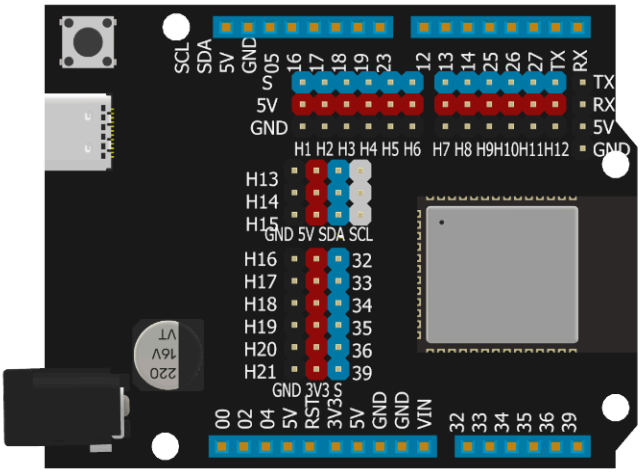
You can use the container lid as the base to mount and seal the project. The weather station in its current configuration can be mounted in any orientation. However if you plan on adding additional sensors it is recommended that the lid be mounted on the bottom and horizontal to the ground. You can secure the lid by any method that you prefer. We mounted the bottom using 4 holes and tie wraps.

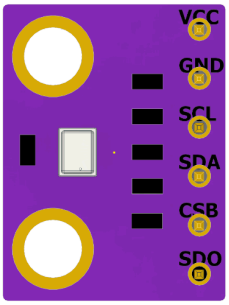
The bottom lid should have one or two small holes to allow for pressure equalization and moisture to exit. 1/16” holes are recommended. Larger holes may encourage insects to build nests in the enclosure and cause problems with the electronics.

Wiring

Wiring the weather station will require minimal preparation

- Acquire 4 Female to Male Dupont Jumpers.
- Using the following chart connect jumpers as shown in the wiring chart.





	Description	ESP32	ESP32 Pin	Sensor	Sensor Pin
1	BME280 Environmental Sensor	5V	H15 5V	5V	VCC
		GND	H15 GND	GND	GND
		SDA	H15 SDA	SDA	SDA
		SCL	H15 SDA	SCL	SCL
2	100KΩ 1/4W Resistor	Receive the Battery voltage from the voltage divider network. Use pin 33 for the analog input.		Form a voltage divider network, with one end of 100KΩ going to the battery supply, and the other to the junction of the 15KΩ. The other end of the 15KΩ should goto GND.	
3	15KΩ 1/4W Resistor				
4	1N5817 Diode (2)	Overvoltage Protection	1N5817 Schottky Diode		1N5817 Diode (2)

Upon completion have someone else double check the connections.

Note:

Resources listed are suggested components and vendors, you may use whatever source you see fit. It is recommended that component substitutes be selected based on similar specifications. Please consult your High School Mentor if in doubt prior to purchasing.

Software:

This may be the most challenging part of this project. If you have never used an ESP32 before there are a few steps that we have to take before we can use the ESP32 with the Arduino IDE.

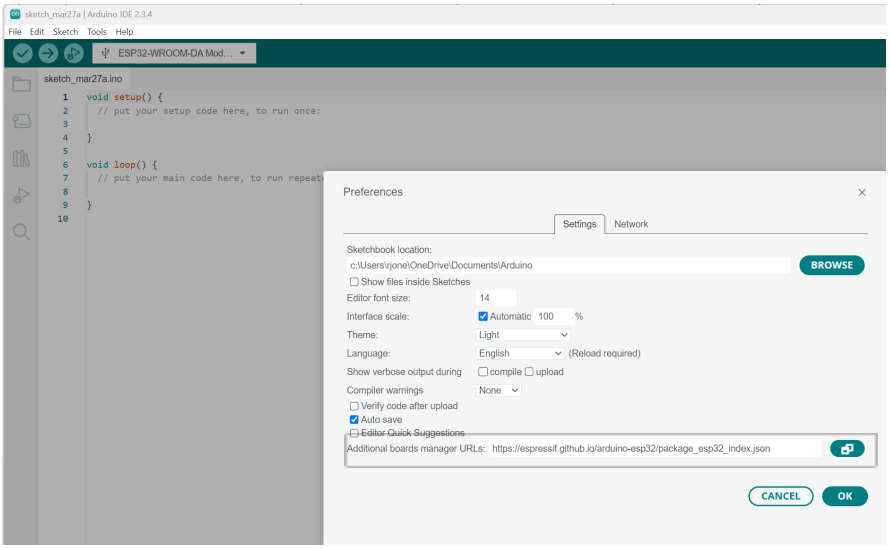
Assuming you have used the Arduino IDE before we will need to open the application and set some parameters:

Open the Arduino IDE and using the File Tab, scroll down to the dialog box called “Preferences”

Notice in the dialog box the section a section that says:
Additional Boards manager URLs.

Click on the green window selector on the right.

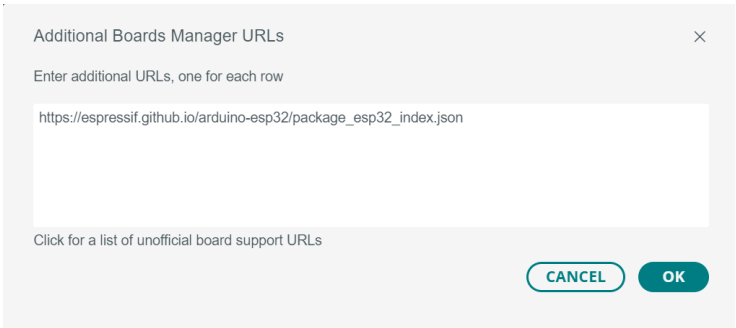
Here we will enter the URL that will point to descriptions of the Esprissif.



https://espressif.github.io/arduino-esp32/package_esp32_index.json

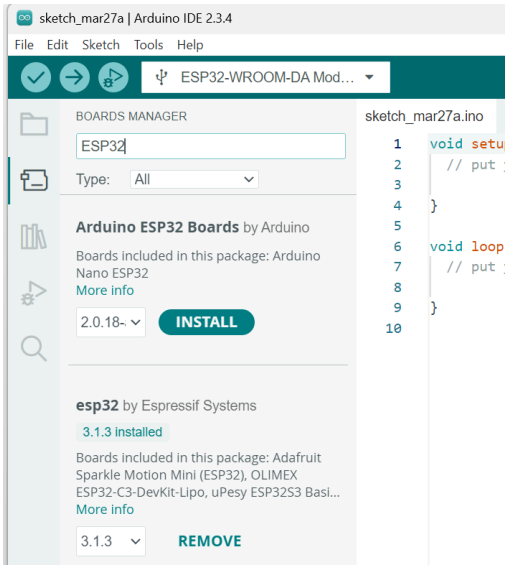
Cut and paste the above URL into the URL dialog box. When complete “Click OK”.

This will begin a process that will request the Arduino IDE to go out on the Internet to download the various board descriptions.



Note:

- You must have internet access to download these files.
- Depending on your Internet access it may take up to a half hour or more to complete.
- You cannot proceed until this is accomplished.

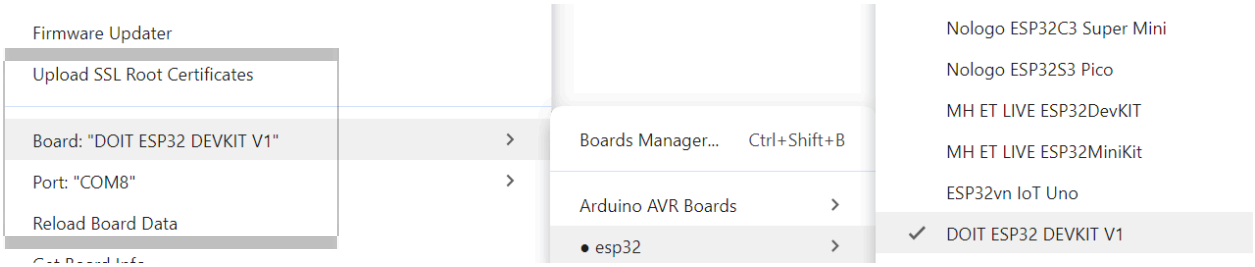


Select the Board Manager to configure the IDE for ESP32 boards. On the Left Dialog tab, find and open the item marked as Board Manager (as shown on the right).

In the Boards Manager Search Box type “ESP32”. Typically two suggestions will pop up.

- Arduino ESP32 Boards by Arduino
- esp32 by Espressif Systems

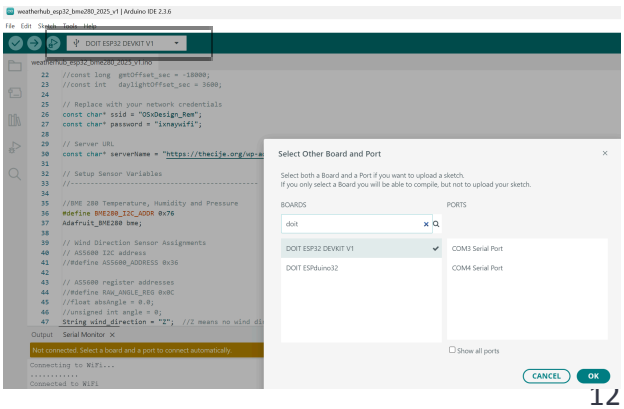
Install esp32 by Espressif Systems.
Installations will take a while so be patient.
Upon completion of the Board Manager move to the “Tools” tab and scroll down to Board and allow the dialog box to open on the right. Select esp32 and allow that dialog box to open. You will see possibly 50 different boards and selecting the “right one” is critical to get your code to compile.



Scroll through the esp32 board tab and look for:

- DOIT ESP32 DEVKIT V1

This module will populate the various parameters of the board into the IDE. If you plug in the ACEBOTT board now you will have to select the com port for programming.



Select from the Tools Com Drop Down Box.

Note:
Resources listed are suggested components and vendors, you may use

whatever source you see fit. It is recommended that component substitutes be selected based on similar specifications. Please consult your High School Mentor if in doubt prior to purchasing.

The com port that the board is attached to.

In the example to the right
“COM8 Serial Port (USB)”
yours will probably be different.

Click “OK” when complete.

If you do not see an available COM port please contact your instructor or IT specialist to identify the port location or Serial to USB driver for your PC (this process is different for PC’s, Mac’s and Chromebooks).

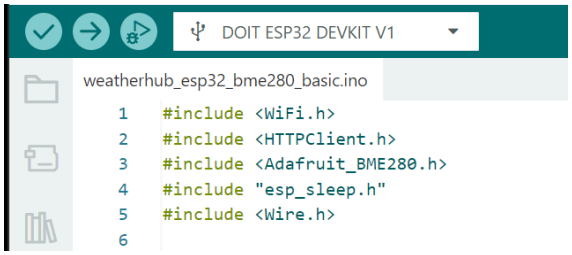
After completing these steps we can begin configuring the IDE for the code, and sensors of your weather station.

For the basic weather station using the BME280 and the ACEBOTT ESP32 goto the following URL and download the following Arduino Program Code:

Code Operation:

Lines 1-5 are code inclusion statements for the various libraries need to:

- Enable WiFi operation
- Enable Web Server communication
- Provide driver interface for BME280 Temperature, Humidity and Pressure sensor
- Enable ESP32 low power sleep modes
- Enable I^2C two wire communication



Lines 15-23 setup variables to use with:

- CIJE Weather Station ID # and Passkey, obtained by registering your station on the CIJE Weather Station Site
- Require Network credentials, SSID and Password
- ULR location of the CIJE Weather station Server for data posting

```
7  //------------------------------------------------------------------------
8  // Setup variables
9  //------------------------------------------------------------------------
10
11 // Setup Network Credentials and Remote Server Address
12 // Replace with your registered station_id and passkey
13 // Register at: https://www.cijeweatherhub.site
14
15 const int station_id = #;        // Use your assigned station ID number
16 const char* passkey = "XXXXXX";  // Replace with your actual passkey
17
18 // Replace with your network credentials
19 const char* ssid = "YOURSSID";
20 const char* password = "PASSWORD";
21
22 // Server URL
23 const char* serverName = "https://thecije.org/wp-admin/admin-ajax.php";
24
```

Lines 28-55 contain variable assignments and default variable values:

- BME-280 sensor instance assignment and I²C address
- Variables for
    - Wind Speed
    - Precipitation
    - Battery Voltage Sensor pin
    - Battery Voltage A/D count
    - Battery Voltage Analog value
    - UV Index value
- ESP32 Specific variables for
    - Deep Sleep Interval
    - Wifi Connection Wait Timeout value

```
28  //BME 280 Temperature, Humidity and Pressure
29  #define BME280_I2C_ADDR 0x76
30  Adafruit_BME280 bme;
31
32  //Wind Direction
33  String wind_direction = "Z";  //Z means no wind direction detected or collected
34
35  //Wind Speed
36  float wind_speed = 0.0;   // Wind Speed in MPH this a scaled value computed later
37
38  // Precipitation
39  float precipitation = 0.0;
40
41  //Battery Voltage
42  int battVoltagePin = 33;  // Battery Voltage Analog Measurement Pin
43  int battDigital = 0;      // A/D Count from Battery Voltage Divider
44
45  float battery = 0.0;  // 0.0 could mean no battery data collected
46
47  // UV Index Sensor
48  int uv_index = 1;  //// 0.0  means no UV data collected
49
50  // Variable for sleep interval (in seconds)
51  const uint64_t sleep_interval_seconds = 3600;  // 3600 = 1hour in seconds
52  const uint64_t sleep_interval_us = sleep_interval_seconds * 1000000ULL;
53
54  // WiFi connection timeout (in milliseconds)
55  const unsigned long wifi_timeout_ms = 30000;  // 30 seconds
```

Lines 61-84 provide initialization instructions in void setup()

- Serial Monitor (115200 Baud)
- I²C Initialization
- Find and verify BME-280 Sensor operation
- Connect to WFi and verify operation
- Post Weather Data to CIJE Weather Station Web Portal
- Go into Deep Sleep mode

```
61  void setup() {
62    Serial.begin(115200);
63    Wire.begin();  // Initialize I2C
64
65    // Initialize sensors
66    if (!bme.begin(BME280_I2C_ADDR)) {
67      Serial.println("Could not find a valid BME280 sensor, check wiring!");
68      while (1) ;
69    }
70
71    delay(1000);
72
73    // Connect to Wi-Fi
74    if (!connectToWiFi()) {
75      Serial.println("Failed to connect to WiFi. Going to sleep.");
76      goToSleep();
77    }
78
79    // Post weather data
80    postWeatherData();
81
82    // Go to deep sleep after posting data
83    goToSleep();
84  }
```

Lines 91-93 indicate null operation of the void loop().

```
91  void loop() {
92    // Nothing to do in the loop; the ESP will sleep after posting data.
93  }
```

- In this program's operational schema the void loop() is not used, since all operations are performed in void setup().
- In terms of sequence due the use of Deep Sleep mode, after booting all operations are performed, then the processor goes into Deep Sleep mode. After a programmed interval the Processor essentially “reboots”. Any operation performed in void loop() would be “lost” and could potentially corrupt the reboot. Therefore it is not recommended that any code operations be performed in void loop(), (potential reentry errors).

Function Descriptions:

Lines 99-118 Provide the necessary code for connecting and authenticating with your WiFi network

- Using your SSID and Password begin the WiFi connection process
- Determine if the the WiFi connection has been established within the assigned time
- Print the status of the attempted WiFi communication
- Upon exit of the function return a “True” or “False” status of the attempt

```
99  bool connectToWiFi() {
100   WiFi.begin(ssid, password);
101   Serial.println("Connecting to WiFi...");
102
103   unsigned long startAttemptTime = millis();
104
105   while (WiFi.status() != WL_CONNECTED) {
106     if (millis() - startAttemptTime >= wifi_timeout_ms) {
107       Serial.println("WiFi connection timeout.");
108       return false;  // Failed to connect within the timeout
109     }
110     delay(500);
111     Serial.print(".");
112   }
113
114   Serial.println("\nConnected to WiFi");
115   Serial.print("IP Address: ");
116   Serial.println(WiFi.localIP());
117   return true;
118 }
```

Lines 122-192 Contain a lengthy set of instructions to perform posting weather data to the CIJE Weather Station Portal.
Lines 122-156 shown on the right and described below.

- Verify WiFi is operational before Posting Data, if not post “WiFi Disconnected”
- Read various Sensors
  - BME280
  - Battery Voltage
- Check Data for proper formatting
- Open the HTTP Web Portal for data posting
- Post Data to Portal

```
122 void postWeatherData() {
123
124   if (WiFi.status() == WL_CONNECTED) {
125     float temperature, humidity, pressure;
126
127     // Get Temp, Humidity and Pressure from BME280 Sensor
128     temperature = bme.readTemperature();
129     humidity = bme.readHumidity();
130     pressure = bme.readPressure() / 100.0F;  // Convert Pa to hPa
131
132     // Read Battery Voltage
133     battDigital = analogRead(battVoltagePin);
134
135     //Prepost data format check
136     if (isnan(temperature) || isnan(humidity) || isnan(pressure) || isnan(wind_speed) || isnan(precipitation)) {
137       Serial.println("Failed to read sensor data!");
138       return;
139     }
140
141     // Login in to CIJE Database and post results
142     HTTPClient http;
143     http.begin(serverName);
144     http.addHeader("Content-Type", "application/x-www-form-urlencoded");
145
146     String httpRequestData = "action=post_weather_data";
147     httpRequestData += "&station_id=" + String(station_id);
148     httpRequestData += "&passkey=" + String(passkey);
149     httpRequestData += "&temperature=" + String(temperature);
150     httpRequestData += "&humidity=" + String(humidity);
151     httpRequestData += "&pressure=" + String(pressure);
152     httpRequestData += "&wind_speed=" + String(wind_speed);
153     httpRequestData += "&precipitation=" + String(precipitation);
154     httpRequestData += "&battery=" + String(battery);
155     httpRequestData += "&wind_direction=" + String(wind_direction);
156     httpRequestData += "&uv_index=" + String(uv_index);
```

Lines 158-192 Contain Post data verification checks.

- Check with the Web Portal and gather it’s post response
- From post response, print response code
- Using response code print response message
- End the HTTP Web portal connection
- Print Wifi Disconnected Message

```
158     Serial.print("HTTP Request Data: ");
159     Serial.println(httpRequestData);
160
161     // Determine error code is any on post
162     int httpResponseCode = http.POST(httpRequestData);
163
164     if (httpResponseCode > 0) {
165       String response = http.getString();
166       Serial.print("Response Code: ");
167       Serial.println(httpResponseCode);
168       Serial.print("Response: ");
169       Serial.println(response);
170
171       if (response.indexOf("Invalid station ID or passkey") != -1) {
172         Serial.println("Error: Invalid station ID or passkey");
173       } else if (response.indexOf("Data out of range") != -1) {
174         Serial.println("Error: Data out of range");
175       } else if (response.indexOf("Post too soon") != -1) {
176         Serial.println("Error: Post too soon. Please wait an hour.");
177       } else if (response.indexOf("Failed to insert data") != -1) {
178         Serial.println("Error: Failed to insert data");
179       } else {
180         Serial.println("Data posted successfully");
181       }
182     } else {
183       Serial.print("Error on sending POST: ");
184       Serial.println(httpResponseCode);
185     }
186
187     http.end();
188   }
189   else {
190     Serial.println("WiFi Disconnected");
191   }
192 }
```

Lines 198-203 ESP32 Specific code instructions to enable Deep Sleep Power down mode.

- Print “Sleep” message
- Set Deep Sleep interval
- Enable Deep Sleep

```
198  void goToSleep() {
199
200    Serial.println("Going to sleep now...");
201    esp_sleep_enable_timer_wakeup(sleep_interval_us);
202    esp_deep_sleep_start();
203  }
```

The previous code description outlines a basic Weather Station. Additional sensors and operations can be added to advance the capability of the Weather Station for Intermediate, and Advanced Operation.

Design Example - Basic - Local Reporting

In this example we want a weather station that can be located outside but next to a classroom window. We may find a better location later but for now we would like to have something that we can experiment with and easily observe. Power will be routed through the window using a 12V DC power adapter. We will use NRF2401 RF modules to wirelessly connect two (2) CIJE Ardunio Unos together. Our goal is to display basic temperature and humidity data to an LCD Display in our classroom. We will add additional sensors later but will likely keep or upgrade the LCD display.

Shown above is an image of a basic weather station using an Arduino UNO (CIJE UNO), a half breadboard and a DHT-22 sensor. Local posting of data is done through another Arduino UNO (CIJE UNO) via an NRF2401 RF module. In this example numeric data is displayed on a 1602 I^2C LCD with temperature and humidity also displayed through mini servos.

We have some wiring to do, coding (most of it is provided) and some testing.

UNO based Weather Station Transmitter

Weather Station Specifications:

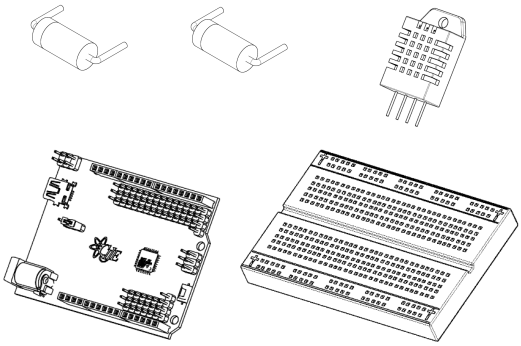
#	Sensing Parameter	Sensor	Notes
1	Temperature	DHT-22 3.3V-5V	Amazon - DHT-22
2	Humidity	DHT-22 3.3V-5V	
#	Hardware	Type	
1	Arduino UNO	UNO	KeyStudio or CIJE
2	Breadboard	Mini Breadboard	Mount components for easy changes
4	1N5817 Diode (2)	Overvoltage Protection	1N5817 Schottky Diode
4	12VDC Adapter		Power for Station
5	Plastic Container	48 oz, 7” x 10” x 2”	Station Enclosure - taped to block
6	Snap-in Support	PCB Mounting Post	Amazon PCB Support
7	Dupont Jumpers	Female to Male (10)	Component interconnects
8	Hot Glue		
	Local Display		
1	Arduino UNO	UNO	KeyStudio or CIJE
2	LCD 1602	16 Character x 2 Line I^2C	Amazon or CIJE
3	SG90 Micro Servo	180° Micro Servo (2)	Amazon or CIJE
4	12V DC Adapter	12V 1A 5.5mm x 2.1mm	Power for Station

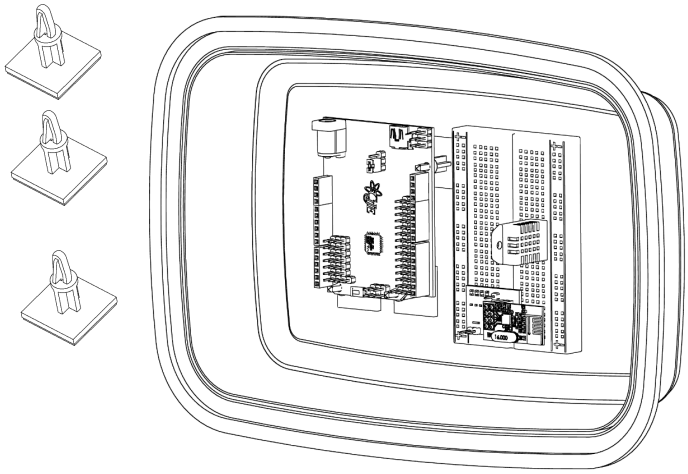
5	Dupont Jumpers	Female to Male (10)	Component interconnects
6	Plastic Container	48 oz, 7” x 10” x 2”	Station Enclosure - Zip-tied to block

Note:

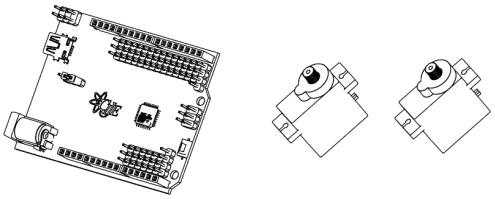
Resources listed are suggested components and vendors, you may use whatever source you see fit. It is recommended that component substitutes be selected based on similar specifications. Please consult your High School Mentor if in doubt prior to purchasing.

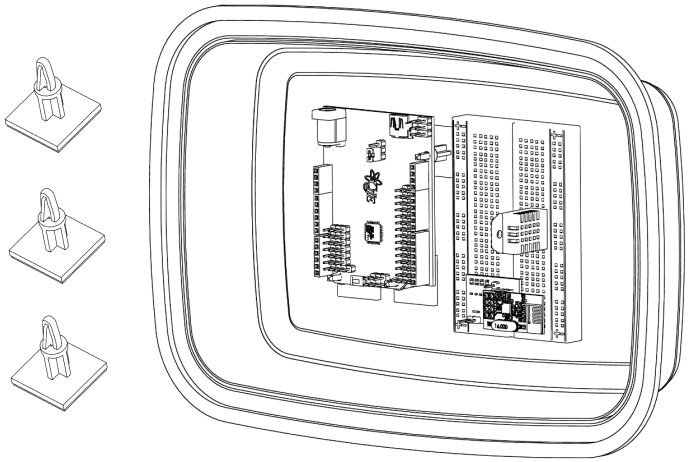
Food Tray Container	1
CIJE Arduino UNO	1
Half Breadboard	1
DHT-22	1
1N5817 Diode	2
Snap-in Support	3





Food Tray Container	1
CIJE Arduino UNO	1
1602 I^{2}C LCD	1
SG90 Micro Servo	2
Snap-in Support	3





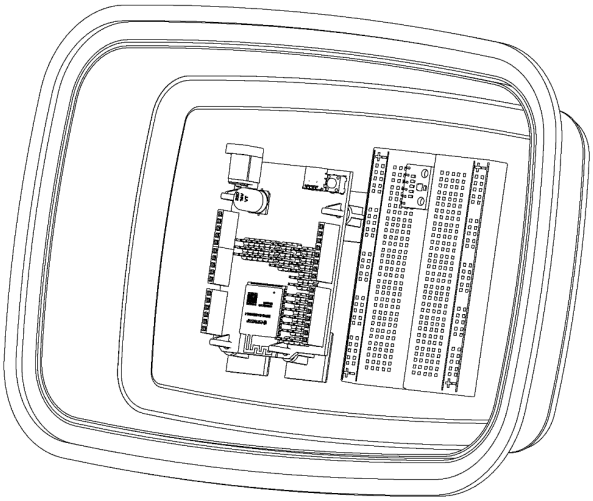
Remote Transmitter Module

After gathering the above components, place them in the bottom of the Food Tray Container. Do a visual check first by placing them centered both vertically and horizontally next to each other. The easiest way to secure them is to use the adhesive Snap-in Support and the using the adhesive backing on the half breadboard. Mark or try positioning the Arduino UNO board and the half breadboard first and mark or identify where they go.

You may now secure the DHT22 sensor, NRF2401 RF module on the half-breadboard (use double sided tape for the RF module). On the half-breadboard place a 1KΩ and a 3.3KΩ resistor in series to form a voltage divider network. The top of the 3.3KΩ will go to your battery or power source. Where the 3.3KΩ and 1KΩ meet run a wire to the A0 analog input. Make sure the other end of the 1KΩ is grounded. This will form your voltage divider network to measure the power source (Vin).

Steps:

- Insert the supports into the UNO board, note it is suggested that once in place you apply hot glue to the adhesive pads to add another layer of attachment to the food tray
- Remove the adhesive, using the previous positioning and place it in the bottom.
- Remove the adhesive backing on the half breadboard and firmly place it next to the UNO board.
- You may insert the DHT-22 into the breadboard where you think it will work best.
    - The DHT-22 pins VCC, GND, Signal (Data) should be accessible via other pins on the breadboard.
    - You will connect pins via Dupont jumpers to the UNO.
- Place the two voltage divider resistors (3.3KΩ, 1KΩ) on the breadboard.
- Wire them accordingly to form a resistor divider with the one end of the 3.3KΩ to be attached to the battery positive terminal.
- At the junction of the 3.3KΩ and the 1KΩ run a Dupount jumper to the A0 analog input on the UNO.
- Make sure the other end of the 1KΩ goes to GND.



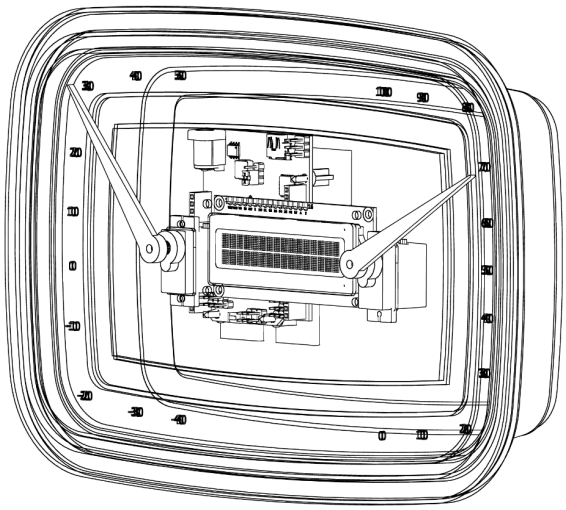
Now that you have mounted the components, make a couple of notes.

- The location of the programming port.
- The location of the power port.

Remote Receiver (Display Module)

After gathering the above components, place them in the bottom of the Food Tray Container. Do a visual check first by placing them centered both vertically and horizontally next

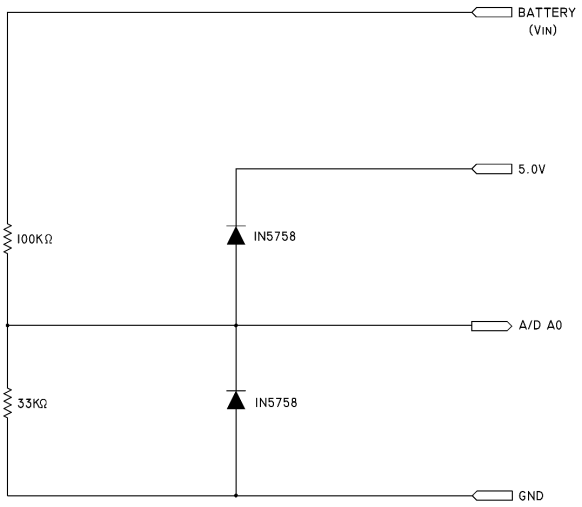
to each other. The easiest way to secure them is to use the adhesive Snap-in Support and the using the adhesive backing on the half breadboard. Mark or try positioning the Arduino UNO board and NRF2401 RF Module first and mark or identify where they go.

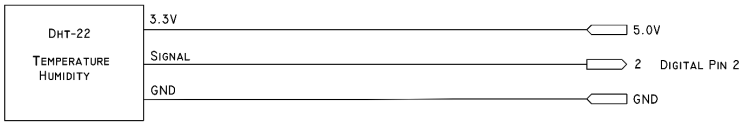


Using the clear cover center and mark the location of the cuts for the LCD display and the two servos. We have included a paper cutout template for you to use as well. Carefully cut out the sections so that the LCD and the servos can be mounted to the clear cover. We have included a 3D printed bracket, and 3D dial pointers if you want to use them. The CNC cutout has the Temperature and Humidity Legend printed on it to help with assembly.

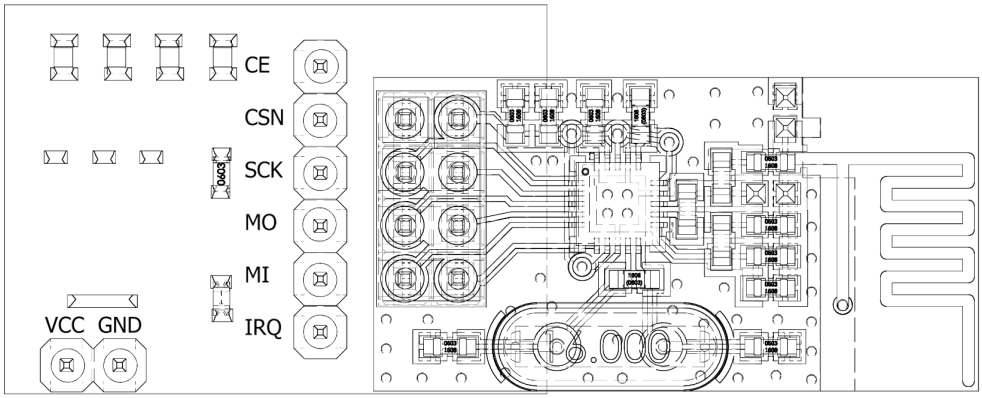
Shown below is the wiring table for both the weather transmitter and the receiver.

CIJE Weather Station DHT-22 Transmitter



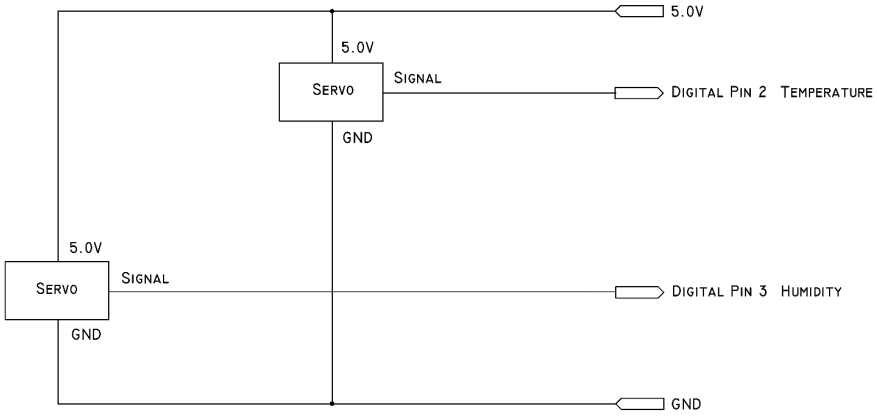


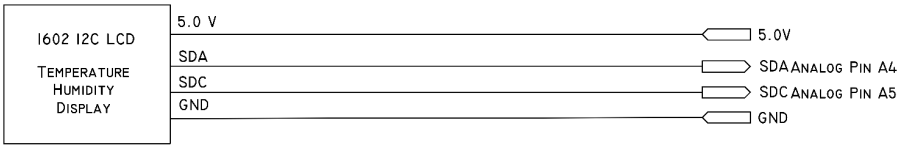
Weather Station Transmitter Wiring Table

	Description	UNO	UNO Pin	Sensor	Sensor Pin
1	DHT-22 Temperature and Humidity Sensor	5V	D2 5V	VCC	+
		GND	D2 GND	GND	-
		S	D2 S	Sensor Data	S
					
2	NRF2401 RF Module Note: The NRF2401 requires a power adapter to operate with 5V systems. Top view of the power adapter and NRF2401 is shown above.	5V	D13 VCC	VCC	5V
		GND	D13 GND	GND	GND
		D9	D9 S	CE	CE
		D10	D10 S	CSN	CSN
		D11	D11 S	MOSI	MO
		D12	D12 S	MISO	MI
		D13	D13 S	SCK	SCK

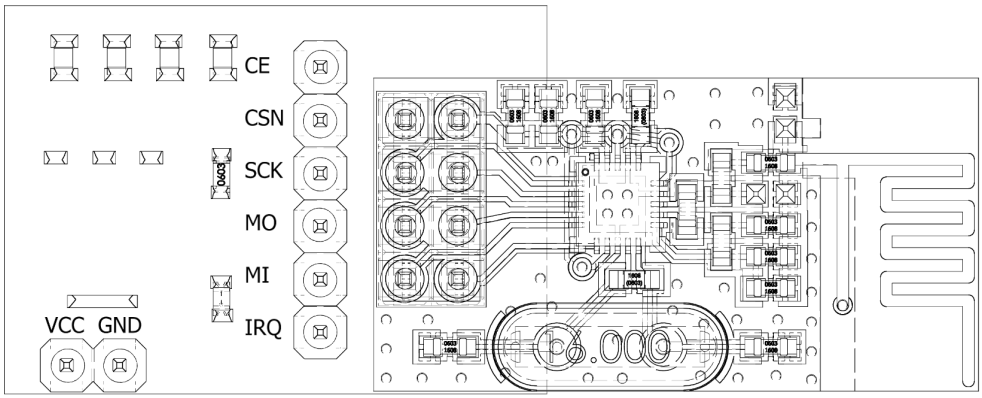
Upon completion have someone else double check the connections.

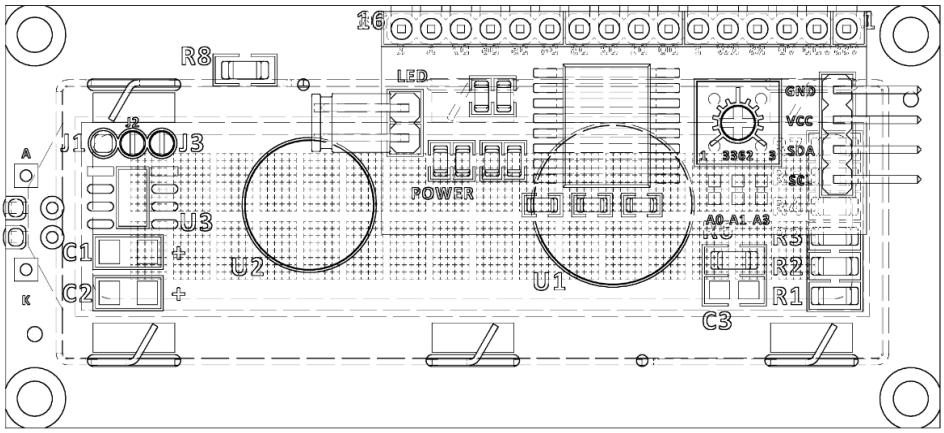
CIJE Weather Station DHT-22
Receiver





Weather Station Receiver Wiring Table

	Description	UNO	UNO Pin	Sensor	Sensor Pin
					

1	NRF2401 RF Module Note: The NRF2401 requires a power adapter to operate with 5V systems. Top view of the power adapter and NRF2401 is shown above.	5V	D13 VCC	VCC	5V
		GND	D13 GND	GND	GND
		D9	D9 S	CE	CE
		D10	D10 S	CSN	CSN
		D11	D11 S	MOSI	MO
		D12	D12 S	MISO	MI
		D13	D13 S	SCK	SCK
					
2	1602 I²C LCD	GND	I²C GND	GND	GND
		VCC	I²C VCC	VCC	VCC
		SDA	I²C SDA	SDA	SDA
		SCK	I²C SCK	SCK	SCK
3	Temperature Servo	D2	D2 GND	GND	Brown
		D2	D2 VCC	VCC	Red
		D2	D2 S	Signal	Orange
4	Humidity Servo	D3	D3 GND	GND	Brown
		D3	D3 VCC	VCC	Red
		D3	D3 S	Signal	Orange

Upon completion have someone else double check the connections.

Operational Details

The Arduino UNO Transmitter is configured to sample data from the DH-22 to obtain temperature and humidity data. This is done periodically (about 1 second), during this sampling period the battery voltage is sampled through the A0 analog port through a resistor divider network.

The data is then placed in a C++ “Structure” in the code so that the NRF2401 transmit library can form packets for transmission. The NRF2401 library provides data transparency and packet recovery to ensure that the data set is delivered securely.

The Arduino UNO receiver is configured to use the NRF2401 receiver library to accept incoming data from the weather station transmitter. Data packets are received and placed in the C++ “Structure” for decoding. The data is decoded to provide inputs for the 2 line x 16 character LCD display and for two servos. Data decoded for the LCD display includes Temperature (°C), Humidity (%) and the Battery voltage (V) at the transmitter. The two servos are used to provide an analog indication of Temperature and Humidity. The servo library provides the control signals to the servo while the C++ “map” function is used to convert Temperature and Humidity data into meaningful position commands.

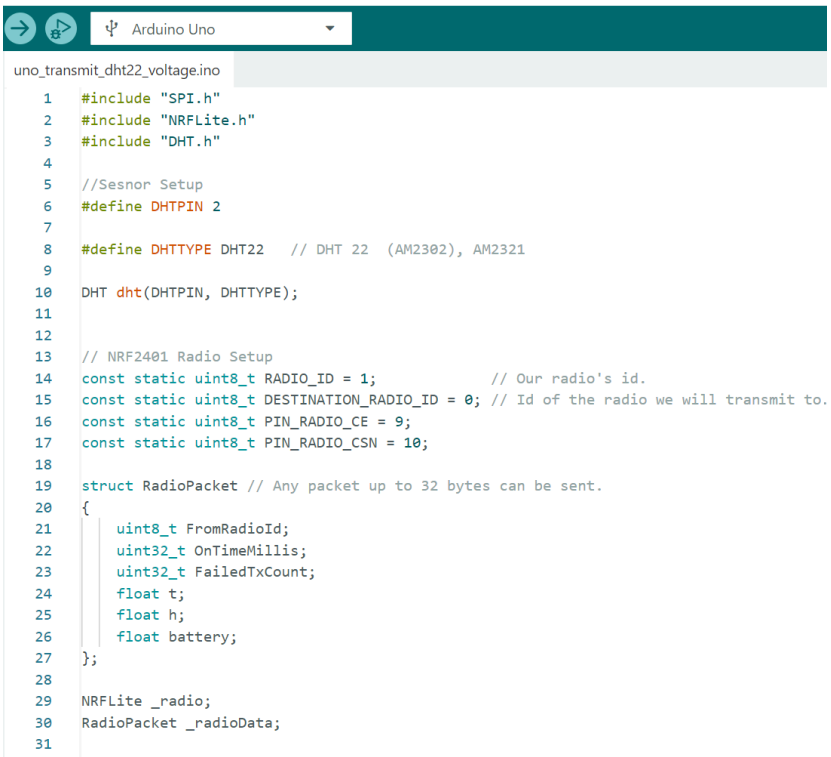
The location of the remote unit may require some trial and error. The NRF2401 operates on the same frequency band as WiFi and depending on your location may experience interference. If you find that the remote or local unit has intermittent connectivity try relocating the remote or local unit closer or in a different location.

Transmitter Code Description:

Lines 1-38 Contain various elements required to include and assign variables for the the Transmitter Module to operate
Lines 1-3

- SPI (Serial Peripheral Interface) Library
- NRF2401 RF module minimal code library (Lite)
- DHT-22 Temperature and Humidity Sensor Library

Lines 6-38




```
1  #include "SPI.h"
2  #include "NRFLite.h"
3  #include "DHT.h"
4
5  //Sesnor Setup
6  #define DHTPIN 2
7
8  #define DHTTYPE DHT22   // DHT 22  (AM2302), AM2321
9
10 DHT dht(DHTPIN, DHTTYPE);
11
12
13 // NRF2401 Radio Setup
14 const static uint8_t RADIO_ID = 1;             // Our radio's id.
15 const static uint8_t DESTINATION_RADIO_ID = 0; // Id of the radio we will transmit to.
16 const static uint8_t PIN_RADIO_CE = 9;
17 const static uint8_t PIN_RADIO_CSN = 10;
18
19 struct RadioPacket // Any packet up to 32 bytes can be sent.
20 {
21     uint8_t FromRadioId;
22     uint32_t OnTimeMillis;
23     uint32_t FailedTxCount;
24     float t;
25     float h;
26     float battery;
27 };
28
29 NRFLite _radio;
30 RadioPacket _radioData;
31
```

- Create sensor instance for the DHT-22 Sensor, and assign the Sensor Pin Number
- Create “static variables” for the operation of the NRF2401 RF module
    - Radio ID for TX, this may need to be reassigned in a multiple Transmitter, receive environment
    - In a similar fashion the Destination Radio (receiver) may have to be reassigned to avoid interference with other radios
    - Assign the hardware pin numbers for the CE (Chip Enable) and CSN (Chip Select Not) is used to enable various transmit and receive mode operations
- Create and assign a data “**Struct**ure” to hold and transfer data between the Transmitter the Receiver modules
- Create instances for the various NRF2401 Radio functions
- Declare the “Type” and default values for the battery voltage measurement

Lines 40-53 Using the void setup() function initialize the Serial Monitor, the DHT-22 Sensor and verify that the NRF2401 radio module is operational

- Being the Serial Monitor
- Begin DHT-22 Operation
- Check for the NRF2401 Operation
- Assign the Local Radio ID value

```
40  void setup()
41  {
42    Serial.begin(115200);
43    // Start DHT22 Sensor
44    dht.begin();
45
46    if (!_radio.init(RADIO_ID, PIN_RADIO_CE, PIN_RADIO_CSN))
47    {
48      Serial.println("Cannot communicate with radio");
49      while (1); // Wait here forever.
50    }
51
52    _radioData.FromRadioId = RADIO_ID;
53  }
```

Line 55-121 The void loop() functions performs the various functions of reading sensors, formatting and sending data through the NRF2401 RF Module

Lines 59-75 Gather Sensor Data

- Read the battery voltage from the A/D I/O port
- Convert and calculate the equivalent Analog Battery Voltage, an offset of 0.45 Volts is added to the measurement due to a diode that is present in the Vin line on the UNO
- Read and Assign the various temperature, humidity values from the DH-22 sensor. Not all are used or posted (these may be added at your discretion to your local weather station if desired)

Lines 77-100 Format data and report to the serial monitor the data values.

```
55  void loop() {
56    // Wait a few seconds between measurements.
57
58    // Read Battery Voltage
59    battDigital = analogRead(battVoltagePin);
60    // Calculate battery voltage
61    // Voltage calculated on A/D 1/2023 of 5V X the divider ratio 4.03 + a
62    // reverse polarity protection diode voltage of 0.45v
63    battery = (4.03 * ((5.0 / 1023.0) * (battDigital))) + 0.45;  //
64
65    // Reading temperature or humidity takes about 250 milliseconds!
66    // Sensor readings may also be up to 2 seconds 'old' (its a very slow sensor)
67    float h = dht.readHumidity();
68    // Read temperature as Celsius (the default)
69    float t = dht.readTemperature();
70    // Read temperature as Fahrenheit (isFahrenheit = true)
71    float f = dht.readTemperature(true);
72      // Compute heat index in Fahrenheit (the default)
73    float hif = dht.computeHeatIndex(f, h);
74    // Compute heat index in Celsius (isFahreheit = false)
75    float hic = dht.computeHeatIndex(t, h, false);
76
77    Serial.print(F("Humidity: "));
78    Serial.print(h);
79    Serial.print(F("%  Temperature: "));
80    Serial.print(t);
81    Serial.print(F("°C "));
82    Serial.print("Battery: ");
83    Serial.print(battery);
84    Serial.print(" V");
85
86    Serial.println();
87
88    _radioData.OnTimeMillis = millis();
89    _radioData.t = t;
90    _radioData.h = h;
91    _radioData.battery = battery;
92
93      Serial.print("Sending ");
94      Serial.print(_radioData.OnTimeMillis);
95      Serial.print(" ms");
96
97
98      Serial.print(_radioData.t);
99      Serial.print(_radioData.h);
100     Serial.print(_radioData.battery);
```

- Print to the Serial monitor the various sensor and battery voltage readings
- Assign sensor data values to the necessary structure variables for transmission
- Report on data transmission
- Print values to be sent via the NRF2401 Radio

Line 102-121 Perform data transmission and transmission status

- Check on transmission status and report on success of transmission
- Set the “rest interval” between sensor data gathering and data transmission

```
102     // By default, 'send' transmits data and waits for an acknowledgement.  If no acknowledgement is
103     // it will try again up to 16 times.  This retry logic is built into the radio hardware itself, s
104     // You can also perform a NO_ACK send that does not request an acknowledgement.  In this situatio
105     // will only be transmitted a single time and there is no verification of delivery.  So NO_ACK se
106     // situations where performance is more important than reliability.
107     //   _radio.send(DESTINATION_RADIO_ID, &_radioData, sizeof(_radioData), NRFLite::REQUIRE_ACK) //
108     //   _radio.send(DESTINATION_RADIO_ID, &_radioData, sizeof(_radioData), NRFLite::NO_ACK)
109
110     if (_radio.send(DESTINATION_RADIO_ID, &_radioData, sizeof(_radioData))) // Note how '&' must be p
111     {
112         Serial.println("...Success");
113     }
114     else
115     {
116         Serial.println("...Failed");
117         _radioData.FailedTxCount++;
118     }
119
120 delay(2000);  // Replace with deep sleep interval (approximately 1 Minute)
121 }
```

Receiver Code Description:

Lines 1-50 Provide NRF2401 hookup instructions, interfacing with LCD and data structures for the received remote data.
Lines 6-14 Provide NRF2401 Radio Pin number assignments
Lines 17-21 Provide Library assignments for the various used in the Weather Station Receiver

- I^2C Library for the LCD
- The required library to communicate with the 1602/2004 I^2C LCD
- The required library for the SPI (Serial Peripheral Interface) of the NRF2401 RF module
- The Transmission/Reception Library of the NRF2401
- The Library for using Servos with the UNO

```
1   /*
2   Demonstrates simple RX and TX operation.
3   Any of the Basic_TX examples can be used as a transmitter.
4   Please read through 'NRFLite.h' for a description of all the methods available in the library.
5
6   Radio    Arduino
7   CE    -> 9
8   CSN   -> 10 (Hardware SPI SS)
9   MOSI  -> 11 (Hardware SPI MOSI)
10  MISO  -> 12 (Hardware SPI MISO)
11  SCK   -> 13 (Hardware SPI SCK)
12  IRQ   -> No connection
13  VCC   -> No more than 3.6 volts
14  GND   -> GND
15
16  */
17  #include <Wire.h>
18  #include <LiquidCrystal_I2C.h>
19  #include "SPI.h"
20  #include "NRFLite.h"
21  #include <Servo.h>
22
23  LiquidCrystal_I2C lcd(0x27,16,2);  // set the LCD address to 0x27 for a 16 chars and 2 line display
24
25
26  // Assign local variables
27  Servo temp;
28  Servo humd;
29  float temperature = 0;
30  float humidity = 0;
31  int tempPos = 0;
32  int humdPos = 0;
33
34  const static uint8_t RADIO_ID = 0;       // Our radio's id.  The transmitter will send to this id.
35  const static uint8_t PIN_RADIO_CE = 9;
36  const static uint8_t PIN_RADIO_CSN = 10;
37
38  // Setlocal sturct to match remote structure for data transfer
39  struct RadioPacket // Any packet up to 32 bytes can be sent.
40  {
41      uint8_t FromRadioId;
42      uint32_t OnTimeMillis;
43      uint32_t FailedTxCount;
44      float t;
45      float h;
46      float battery;
47  };
48
49  NRFLite _radio;
50  RadioPacket _radioData;
```

Lines 27-50 Provide the various variables use to store and present data received front the remote weather station transmitter

- Create an instance for the LCD, noting its address (0x27) and alternate address is 0x3F, and display format (16 character, 2 Lines)
- Various variable assignments for temperature, humidity and servo position values

- Variables required for use with the NRF2401 Radio module, note values for the Radio ID must match that of the Transmitter
- Assignment the necessary data structure for use with the NRF2401
- Create instances for the various NRF2401 Radio functions

Lines 52-75 Using void setup() initialize various hardware elements used in the weather station receiver

- Initialize the Serial Monitor
- Connect the Temperature and Humidity display servos
- Initialize the LCD and turn on the backlight
- Verify that the NRF2401 radio is operational and report status

```
52  void setup()
53  {
54      Serial.begin(115200);
55      temp.attach(2);
56      humd.attach(3);
57
58      lcd.init();
59    // Print a message to the LCD.
60      lcd.backlight();
61
62      // By default, 'init' configures the radio to use a 2MBPS bitrate on cha
63      // Both the RX and TX radios must have the same bitrate and channel to c
64      // You can run the 'ChannelScanner' example to help select the best chan
65      // You can assign a different bitrate and channel as shown below.
66      //   _radio.init(RADIO_ID, PIN_RADIO_CE, PIN_RADIO_CSN, NRFLite::BITRATE
67      //   _radio.init(RADIO_ID, PIN_RADIO_CE, PIN_RADIO_CSN, NRFLite::BITRATE
68      //   _radio.init(RADIO_ID, PIN_RADIO_CE, PIN_RADIO_CSN, NRFLite::BITRATE
69
70      if (!_radio.init(RADIO_ID, PIN_RADIO_CE, PIN_RADIO_CSN))
71      {
72          Serial.println("Cannot communicate with radio");
73          while (1); // Wait here forever.
74      }
75  }
```

Lines 81-129 The void loop() is used to access the NRF2401 Radio Receiver Module and to display the remote data Line 81-104 Create an active receive loop for incoming data from the NRF2401

- Receive data from the remote and place data into the radio data structure
- Using received data create a message to display via the serial monitor
- Using the LCD display gather the received data format it for displaying on the LCD
- Using the received temperature and humidity data “map” the data values to the range of allowable servo values (0-180 degrees) Also note that displays are on the left and right. The left servo display will require inverting the data values for proper orientation
- Send the formatted data to the servos

```
81      while (_radio.hasData())
82      {
83          _radio.readData(&_radioData); // Note how '&' must be placed in fro
84
85
86          // Assign temperature, humidity to local variables
87          temperature = _radioData.t;
88          humidity = _radioData.h;
89
90          String msg = "Radio ";
91          msg += _radioData.FromRadioId;
92          msg += ", ";
93          msg += _radioData.OnTimeMillis;
94          msg += " ms, ";
95          msg += _radioData.FailedTxCount;
96          msg += " Failed TX ";
97          msg += _radioData.t;
98          msg += " C, ";
99          msg += _radioData.h;
100         msg += " %, ";
101         msg += _radioData.battery;
102         msg += " V";
103
104         Serial.println(msg);
105
106         // Display to LCD Temperature, Humdidty and Battery Voltage
107         lcd.clear();
108         lcd.setCursor(0,0);
109         lcd.print(_radioData.t);
110         lcd.print("C");
111         lcd.setCursor(10,0);
112         lcd.print(_radioData.h);
113         lcd.print("%");
114         lcd.setCursor(5,1);
115         lcd.print(_radioData.battery);
116         lcd.print("V");
117
118         // Scale and set Servo positions for Temperature and Humdidty
119         tempPos = map(temperature,-40,50,180,0);
120         humdPos = map(humidity,0,100,0,180);
121         //Serial.print(tempPos);
122         //Serial.print("  ");
123         //Serial.print(humdPos);
124         Serial.println();
125
126         temp.write(tempPos);
127         humd.write(humdPos);
128     }
129 }
```