Weather Station

(ESP32 to Bluetooth Low Energy App)

Objective:

Using an ESP32 construct a weather station that can communicate via bluetooth to either an Android or Apple App and display the following data:

- **Temperature**
- **Humidity**
- **Wind Speed**

Data types will be characters (char) formatted in the following way:
Temperature - T(Value) , *T20, *T100
Humidity - H(Value) , *H30
Wind Speed - W(Value), *W5

Each value will be terminated by a carriage return (CR), line feed character (LF) , example

```
Serial.print("*T");
Serial.println(temp);
```

Data may be in imperial or metric form but must be "displayed" with proper units, i.e. m/s, mph, ○C, etc.

Note:

Example data is formatted to allow for text parsing in the Android Bluetooth Electronics app. It may however be changed at your discretion to other identifiers depending on your needs.

Recommended Supplies:

	Device	Qty	Notes
1	ESP32	1	Set Power Jumper to 3.3V
2	DHT-11	1	https://github.com/offcircuit/SDHT
3	Anemometer	1	Wind Turbine Kit
4	47Ω		Anemometer Load
5	1KΩ	1	Voltage Divider for Anemometer
6	2KΩ	1	Voltage Divider for Anemometer
7	3V Battery Box	1	
8	Micro to USB Cable	1	

Preparation Instructions:

Shown below are the steps that we will be taking to set up the weather station for operation. We will perform some code configuration in the ESP32 for the BLE configuration. This will allow us to customize the device so that it will be easy to identify in either our BLE app or with another BLE enabled device (ex. ESP32).

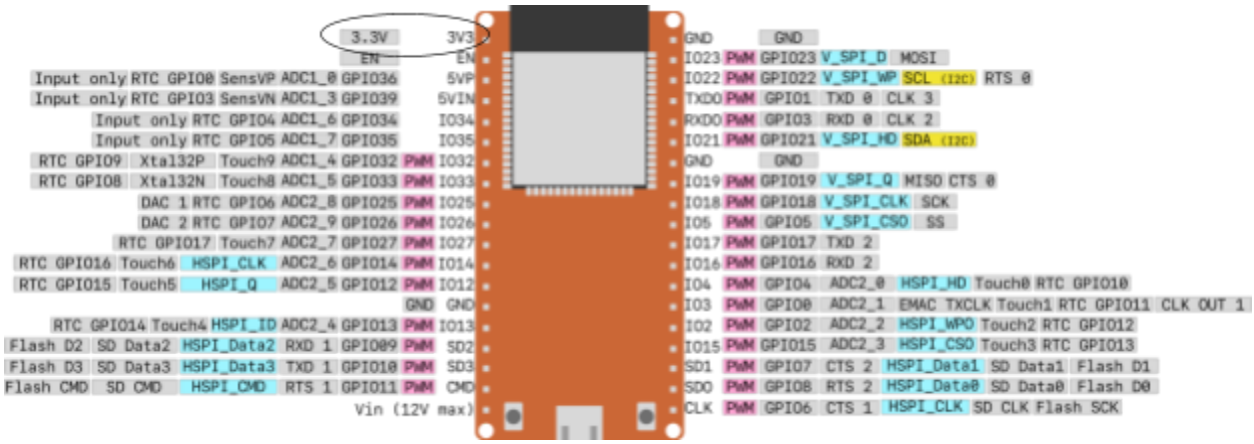
1. **Verify you have the parts list in the above table, each kit has been packed with various devices, please let us know if anything is missing.**
2. **Use battery box for power for, place jumpers to Vin for 3.3V operation.**
3. **Using the available Dupont Jumper Sets wire the devices to the ESP32. Double check you work and if you can have another person check your wiring that is advised.**
4. **Power up your ESP32 if working properly the power led should be "on" the ESP32.**
5. **You will need to load a Weather Station Transmitter into the Arduino IDE and follow the code configuration instructions that follow.**
6. **After verifying the configuration of your ESP32 (you will need an Android or Apple device) you will load a Weather Station Transmitter program onto the ESP32.**

7. **Verify that you can through the Serial monitor see, Temperature, Humidity, and Wind Speed. Confirm that these values change, for instance blowing on the DHT11 you should see a change in temperature and humidity. The anemometer may take some trial and error to get values.**

Hardware Preparation:

Step 2:

The ESP32 operates on 3.3V and all I/O pins require 3.3V signals. The Vin pin can operate up to 12V and will provide power to the 5V and 3.3V pins.

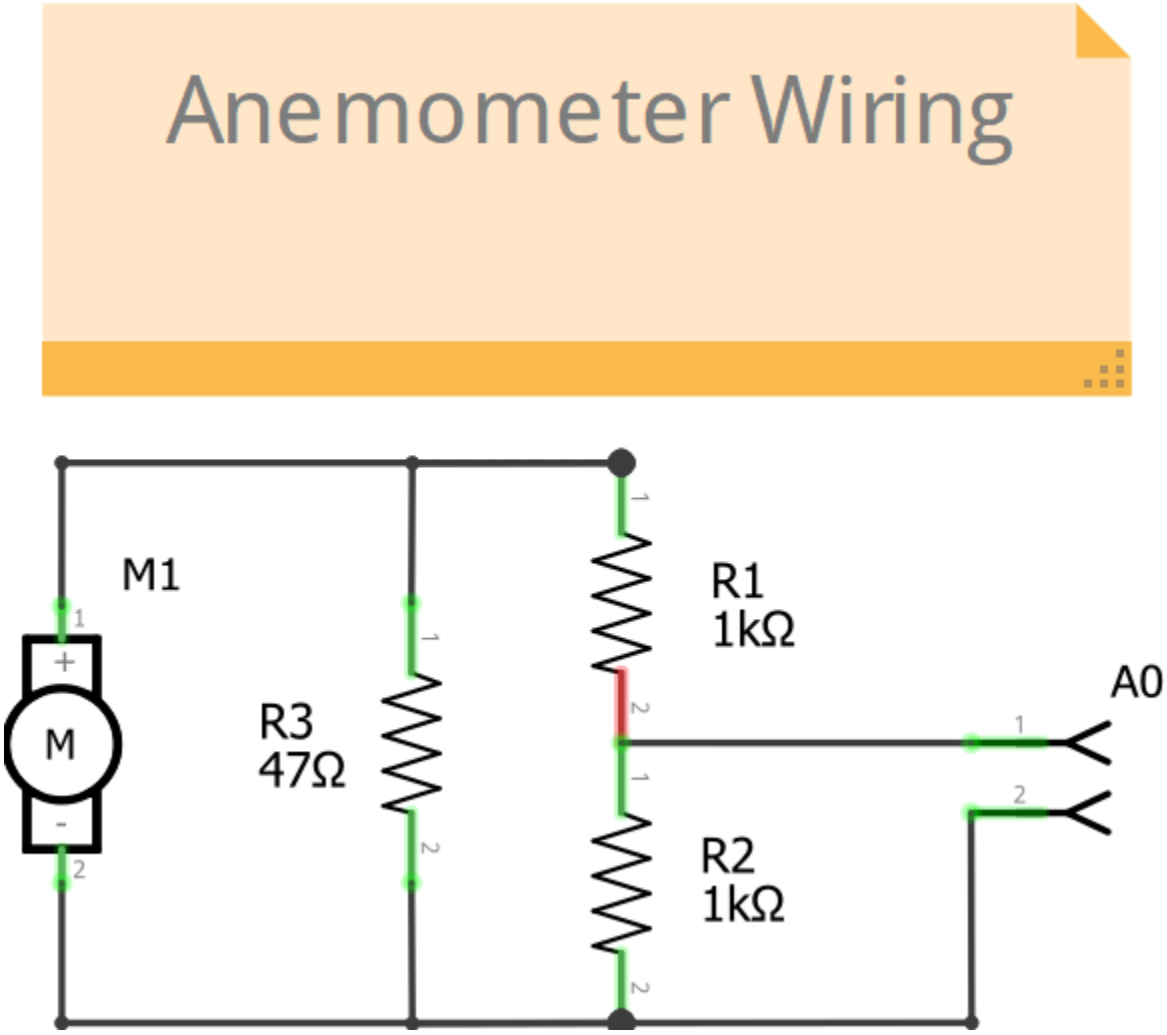


Step 3:

Next we will wire the DHT11, Anemometer and the BLE Module to the ESP32 using the following table:

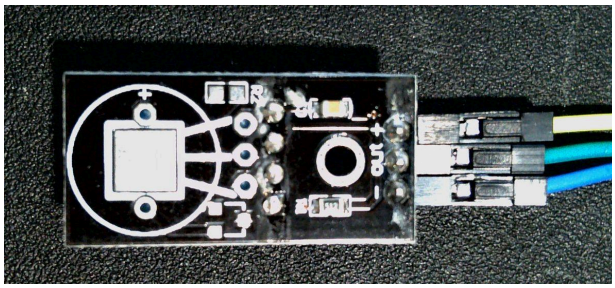
ESP32	Pin		Pin	Device
GND	GND		-	DHT11
3.3V	VCC		+	DHT11
GPIO 17/TXD 2	17		OUT	DHT11
GND	GND		BLK	ANEMOMETER
GPIO 36/ADC1_0	36		OUT	ANEMOMETER

Shown below is the configuration of the Anemometer:



Shown below are the pins of the various devices:

DHT11

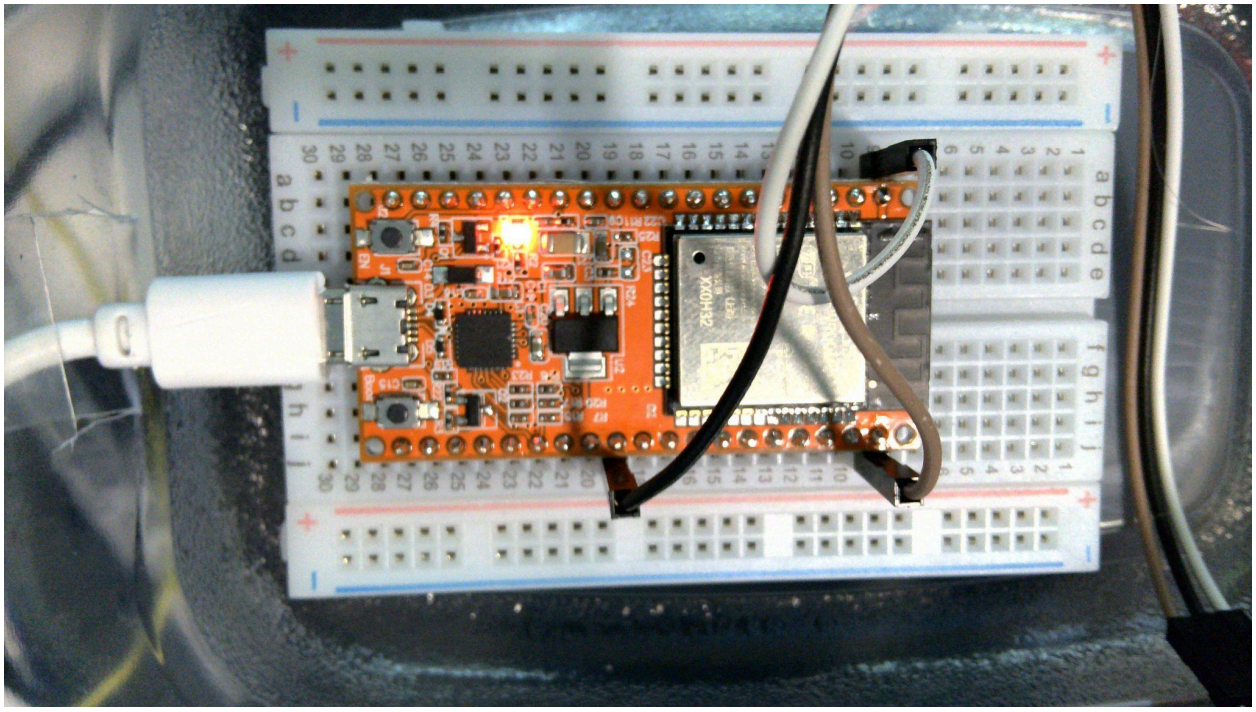


The anemometer will have two wires that pass from the mount through a section of heat shrink tubing and two exit wires.

After setting the jumpers in place, double check all of your connections. The DHT11 can be damaged by reversals of power and ground.

Step 4:

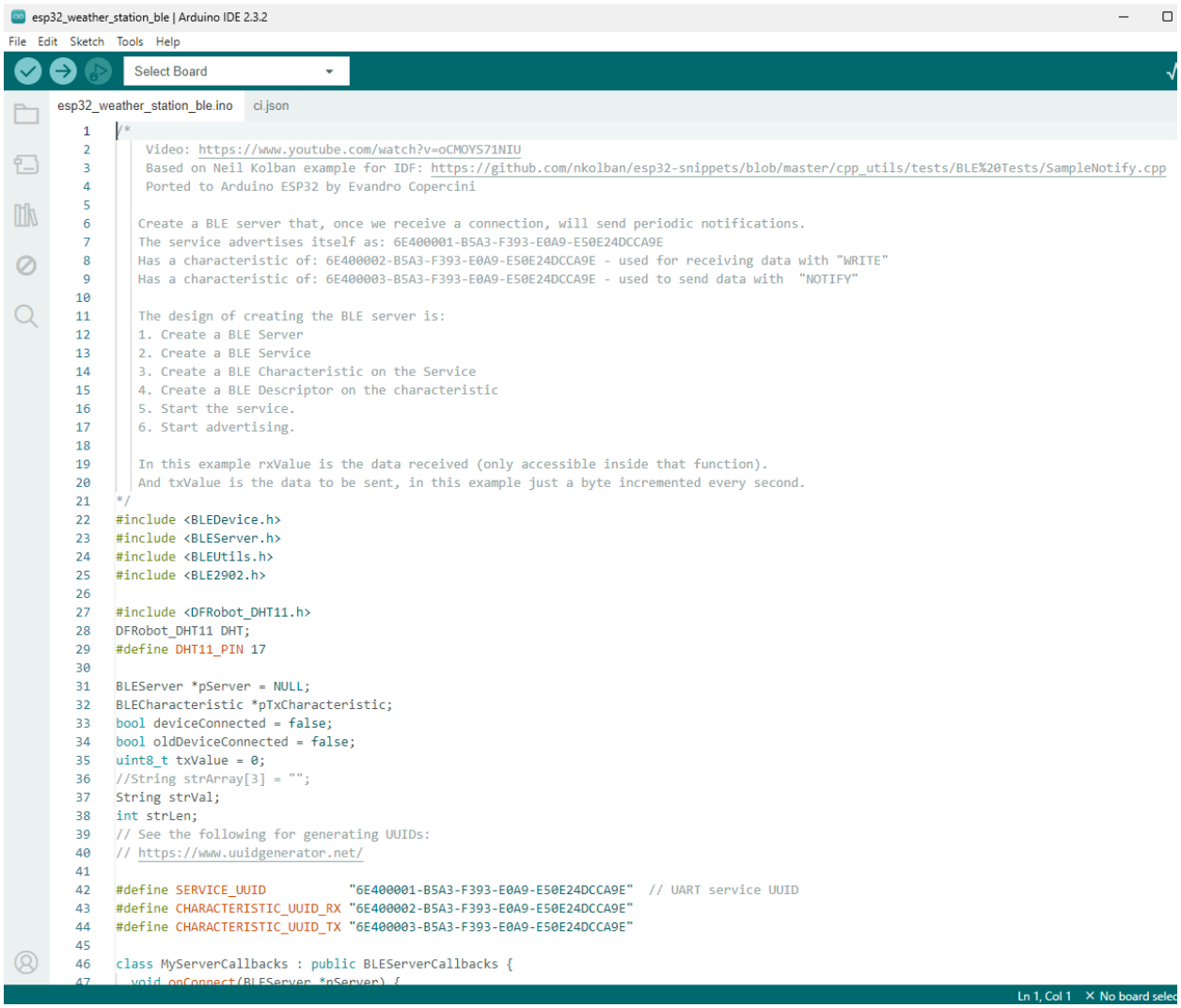
Secure all jumpers and double check the connections. When plugged in properly the ESP32 will have the power led "on".



Step 5:

Load up the the ESP32 Weather Hub to App INO into your laptop computer and the Arduino IDE. You will need a USB to Micro USB programming cable. IF you are using the ESP32 for the first time you will need to configure both the Board Manager Preferences , configure the board manager for the ESP32. Included in this presentation is a link to the setup.

Show below is the Weather Station Program:



In simple terms the BLE module takes character data (numbers, text, etc) from your programs and converts them into a digital wireless signal that another BLE device compatible module then can convert back to the original character data.

In the case of the ESP32 we have built-in hardware that handles all of the house keeping for BLE communications. We do however have to load the proper library for us to use it and have to configure the code for our purposes.

The example program has added both the DHT11 Temperature and Humidity sensor. On line 25 you can see the entries for the DHT11 library and the instance to use the sensor.

```
27  #include <DFRobot_DHT11.h>
28  DFRobot_DHT11 DHT;
29  #define DHT11_PIN 17
```

Lines 74-78 are used to "setup" some items that will inform our A/D converter (in part because it is programmable) and to set the BLE identity that we want to advertise to the outside world.

```
73  void setup() {
74    Serial.begin(115200);
75    analogSetAttenuation(ADC_11db);  //150mV - 2600mV
76    analogReadResolution(12);
77    // Create the BLE Device
78    BLEDevice::init("cije32WSTX");
```

Line 37-38 has been added so we can translate our sensor values into the proper format for transmitting.

```
36  //String strArray[3] = "";
37  String strVal;
38  int strLen;
```

Lines 120-158 show us how to send data using the ESP32 UART program to another BLE module or App. The UART program sends HEX data and it is not readily translated on the other end. So we have to "break down" the information that we send into the proper Hex codes so that it is received and reconstructed for easy reading.

Line 120 reads the Temperature and Humidity values from the DHT11 and places them in instance variables DHT.temperature, and DHT.humidity.

Since the data transmitted is in HEX we need to convert the characters for transmitting. From out original specifications we suggested the following:

> **Data types will be characters (char) formatted in the following way:**
> **Temperature - T(Value) , *T20, *T100**
> **Humidity - H(Value) , *H30**
> **Wind Speed - W(Value), *W5**

Each value will be terminated by a carriage return (CR), line feed character (LF)

The combination of code to do this looks like this:

```
120     DHT.read(DHT11_PIN);
121     //_____________________________________________
122     //Temperature convert - send
123     txValue = '*';  //"*"
124     pTxCharacteristic->setValue(&txValue, 1);
125     pTxCharacteristic->notify();
126     Serial.print(txValue);
127
128     txValue = 'T'; //"T"
129     pTxCharacteristic->setValue(&txValue, 1);
130     pTxCharacteristic->notify();
131     Serial.print(txValue);
132
133     delay(20);
134     strVal = String(DHT.temperature);
135     strLen = strVal.length();
136     Serial.println(strLen);
137
138     for(int ptr = 0; ptr < strLen; ptr++)
139     {
140       txValue = strVal.charAt(ptr);;
141       pTxCharacteristic->setValue(&txValue, 1);
142       pTxCharacteristic->notify();
143       Serial.print(txValue);
144     }
145
146     //Send a carriage return and line feed
147     delay(20);
148     txValue = 10;
149     pTxCharacteristic->setValue(&txValue, 1);
150     pTxCharacteristic->notify();
151     Serial.print(txValue);
152
153     delay(20);
154     txValue = 13;
155     pTxCharacteristic->setValue(&txValue, 1);
156     pTxCharacteristic->notify();
157     Serial.print(txValue);
158
```

The code breaks down the various character and send them individually to be "reconstructed" at the other end.

A more detailed description follows:

Line 123-126 sets the character * and sends it.

```
123     txValue = '*';  //"*"
124     pTxCharacteristic->setValue(&txValue, 1);
125     pTxCharacteristic->notify();
126     Serial.print(txValue);
```

Then lines 128-131 sets the character T and sends it.

```
128     txValue = 'T'; //"T"
129     pTxCharacteristic->setValue(&txValue, 1);
130     pTxCharacteristic->notify();
131     Serial.print(txValue);
132
```

Next we read the temperature which could be up to a three digit value to a special kind of array called a String. The String holds the character that makes up our temperature data. So for instance it could be "25" or it could be "115" these are 3 separate HEX values that then must be sent in sequence.

```
133     delay(20);
134     strVal = String(DHT.temperature);
135     strLen = strVal.length();
136     Serial.println(strLen);
137
138     for(int ptr = 0; ptr < strLen; ptr++)
139     {
140       txValue = strVal.charAt(ptr);;
141       pTxCharacteristic->setValue(&txValue, 1);
142       pTxCharacteristic->notify();
143       Serial.print(txValue);
144     }
145
```

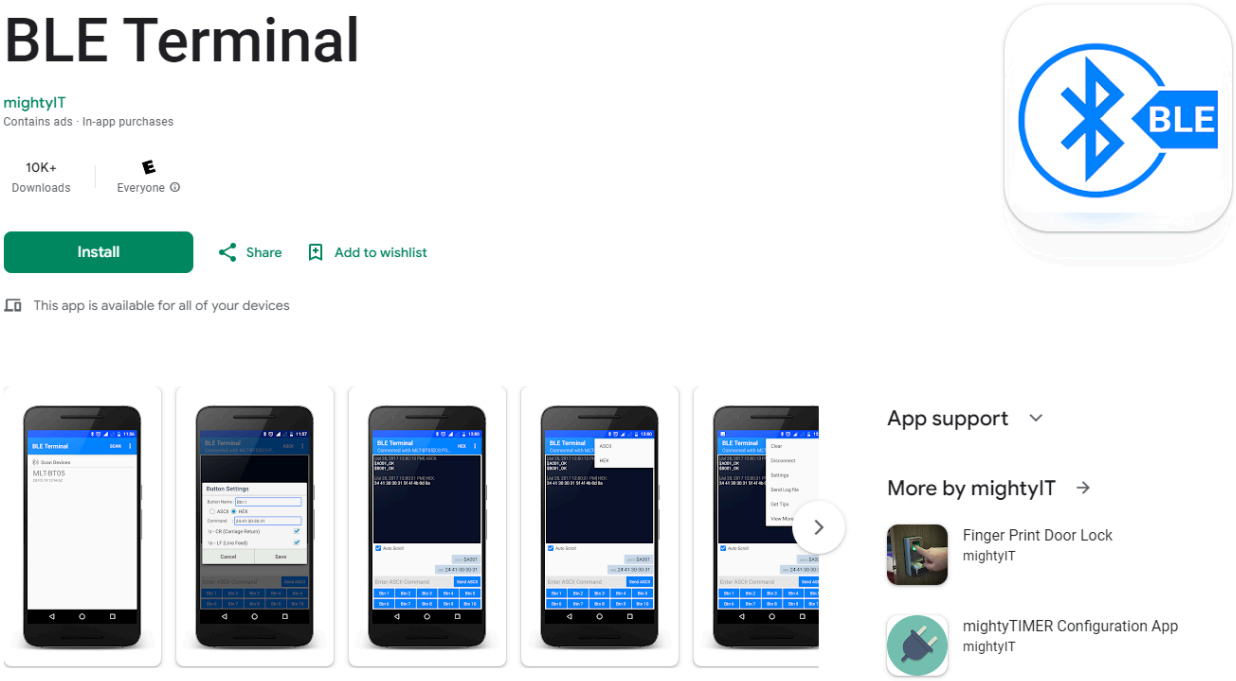
After this we must send a Carriage Return (CR) and a Line Feed (LF) character so that the data is sent on separate lines.

```
146     //Send a carriage return and line feed
147     delay(20);
148     txValue = 10;
149     pTxCharacteristic->setValue(&txValue, 1);
150     pTxCharacteristic->notify();
151     Serial.print(txValue);
152
153     delay(20);
154     txValue = 13;
155     pTxCharacteristic->setValue(&txValue, 1);
156     pTxCharacteristic->notify();
157     Serial.print(txValue);
158
```

In this way we can send the HEX data so it can be "seen". This process is repeated for the Humidity and the Wind Speed.

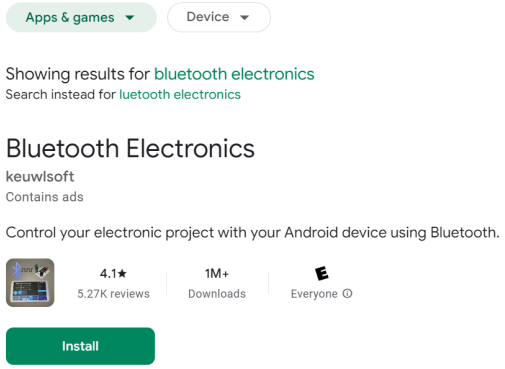
If you think about the code that we have used, certain sequences could be condensed using functions. The Wind Speed section has been left unfinished so you can try your hand at formatting the data.

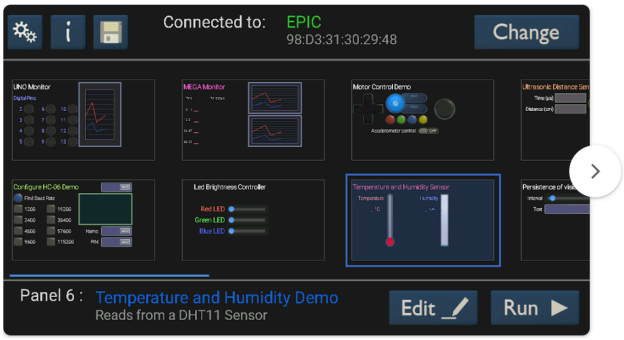
Section 6:

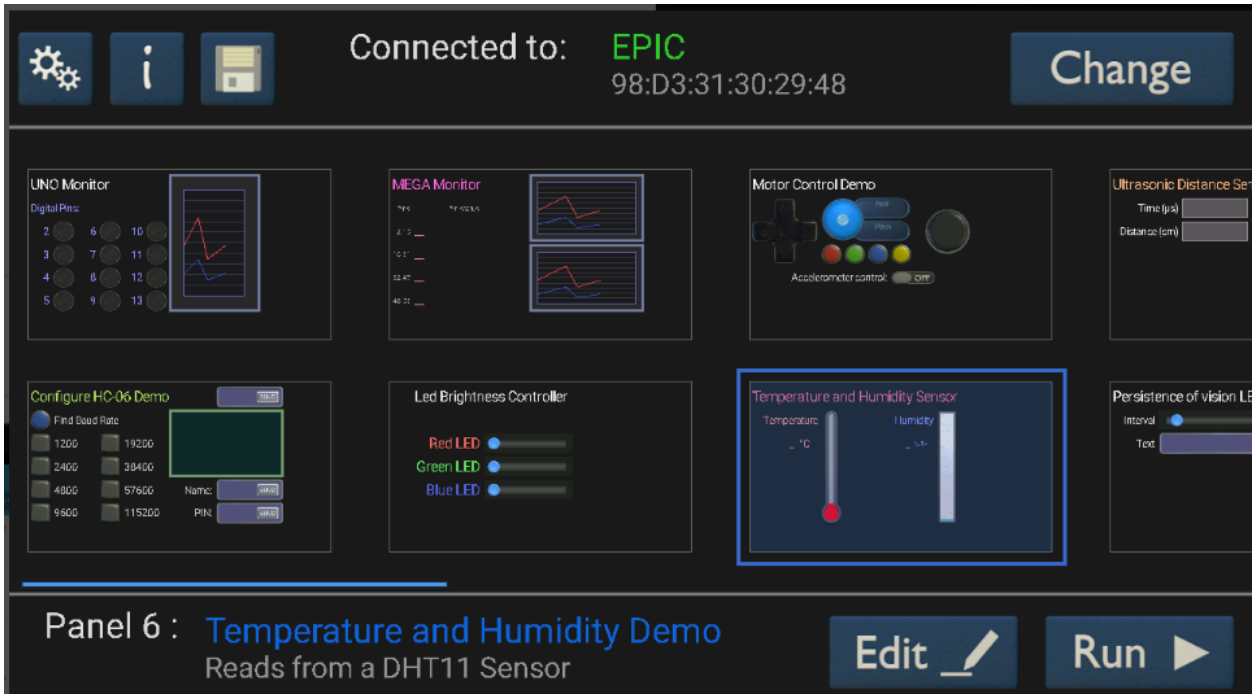


The next application is called Bluetooth Electronics and it allows us to create various “no code” instruments. We will use this later to create a no code display of our Temperature, Humidity and Wind Speed.

Goto the Google Play store and search for Bluetooth Electronics. Download and install the app to your phone or tablet. We will walk through a simple interface later.



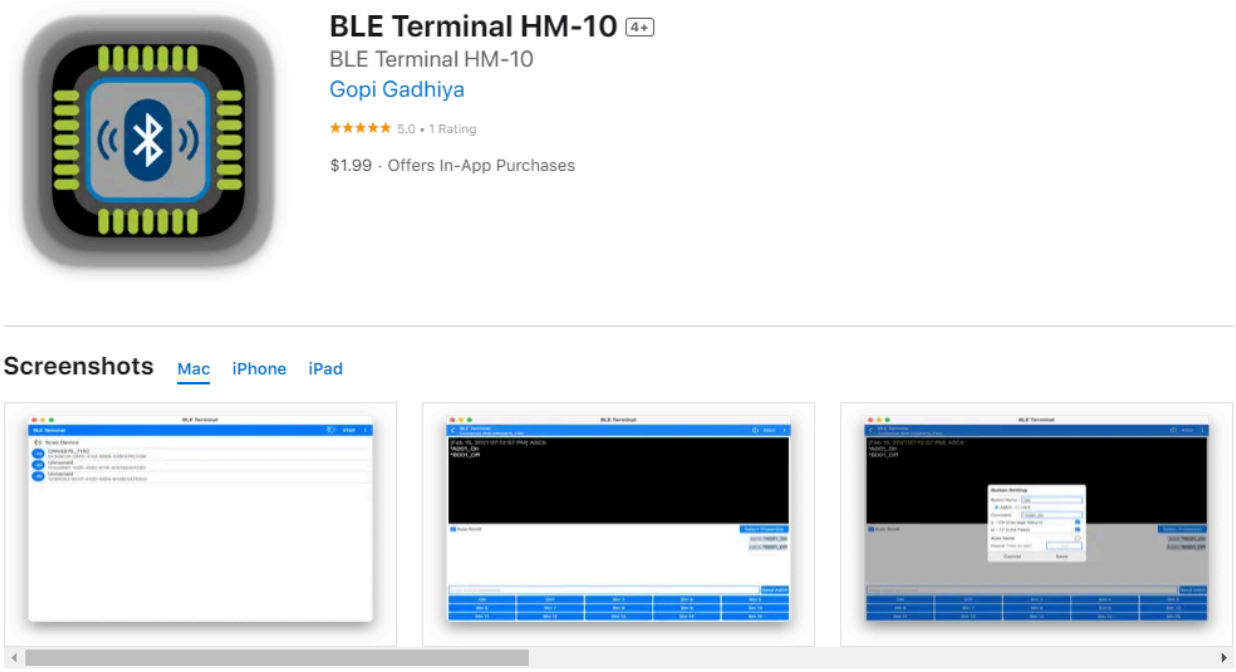




iPhone

BLE Terminal HM-10 and it allows us to scan and interact with BLE devices. It will also allow us to display Temperature, Humidity and Wind Speed data.

Goto the Apple App Store and search for BLE Terminal. Download and install the app to your phone or tablet. We will walk through a simple interface later.



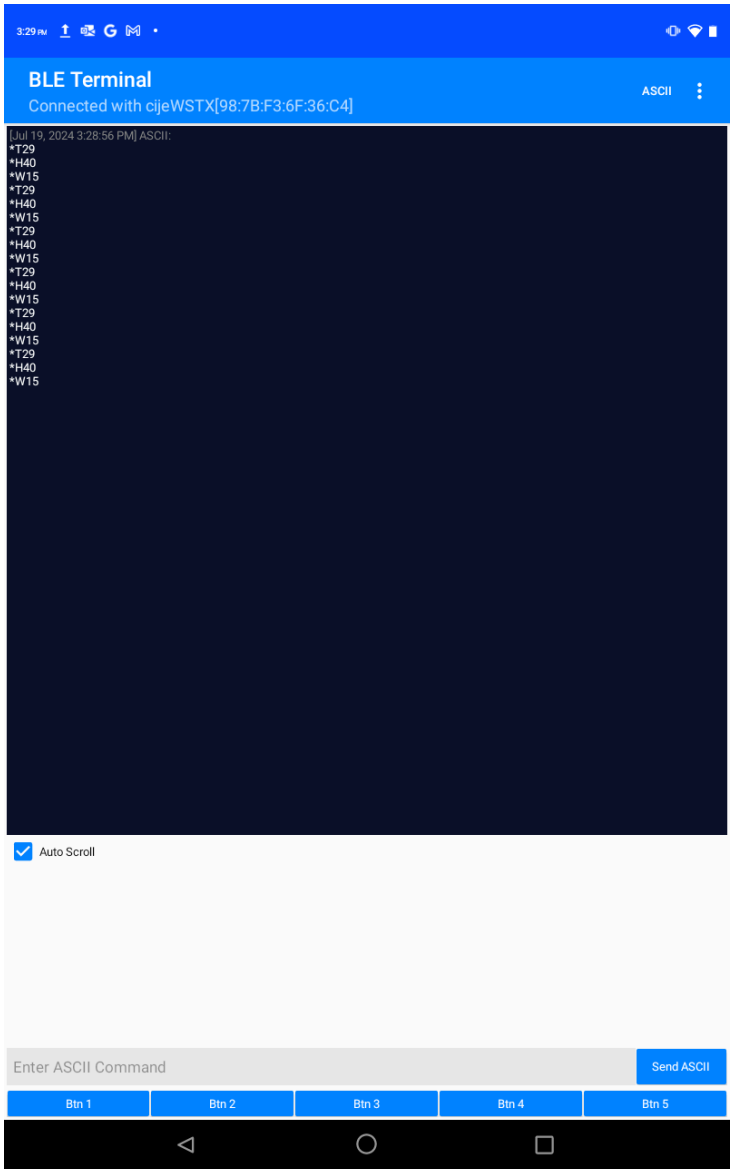
Now that the apps have been installed depending on your platform you will open either the Android app - BLE Terminal or the Apple app BLE Termin HM-10

When you open the app you see a screen similar to that on the right. The app will begin to scan for BLE devices. You may have to scroll through the many devices to find the BLE name that you labeled your device. In this example the CIJE Weather Station was named "cijeWSTX". You can see on the right the number below it is call a UUID, it is a Universally Unique Identifier newer devices have this added along with other advanced features,

Bluetooth Electronics

In this example using an Android phone or tablet you can create a number of different "displays" using the built in editor. We will create three graphs that will display temperature, humidity and wind speed.

If you click on the line showing cijeWSTX another screen will open. This screen is the terminal screen. Given that you have "connected" any data that is exchanged between devices will be shown. The data you see here is data from the Weather Station after the Weather station software has been installed.



iPhone

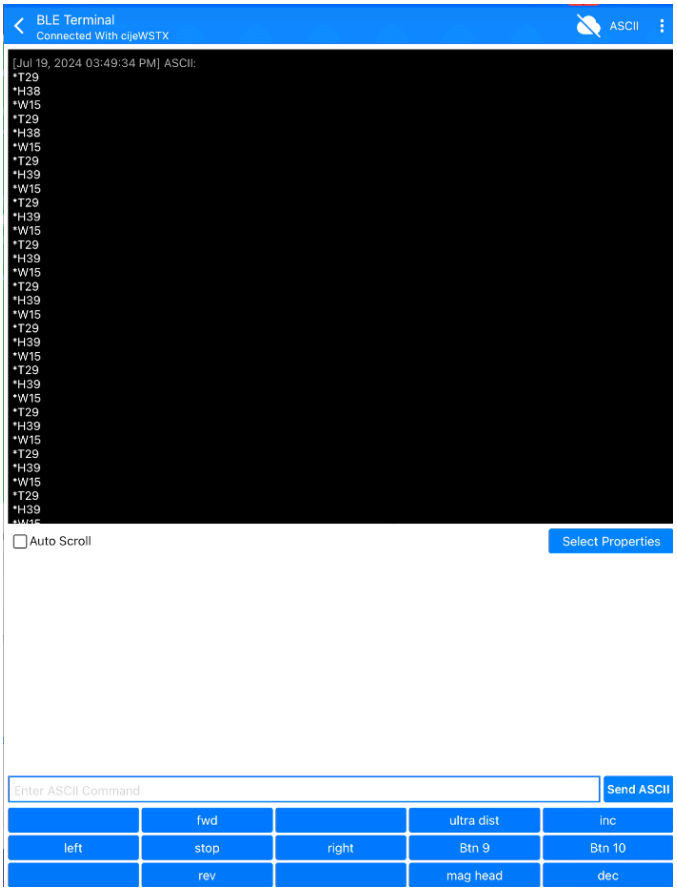
Open the app BLE Terminal HM-10.

The screen begins scanning for BLE devices and in a similar fashion search for your BLE device. It may require an extensive search since

there may be many BLE devices in your location. Notice that our "cijeWSTX" weather station is in the scan list. Notice the UUID shown as well.

Press on the line containing cijeWSTX and a new screen should appear. This is the terminal screen and will display any data that is exchanged with your BLE device. The screen here is showing data being sent from the Weather Station.

You may load the Weather Station Program onto your ESP32 and see if you can see any data following through. You may have to disconnect, reconnect after loading the software.



Step 7:

If after completing the Wind Speed section, try it out. If you are seeing the data that you intended you are all set.

Verify that all components work and are displayed on your BLE Terminal app. The code is posted below:

Android Users - Instrumentation App Bluetooth Electronics Instructions:

We have provided a resource file to get you started:

Bluetooth Weather:

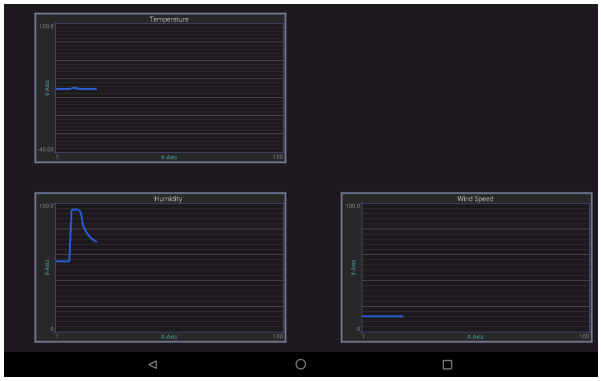
https://drive.google.com/file/d/1R5izjsBgzMfYSGape5w5ORg9xXeMz53P/view?usp=drive_link

We have provided a resource file to get you started:

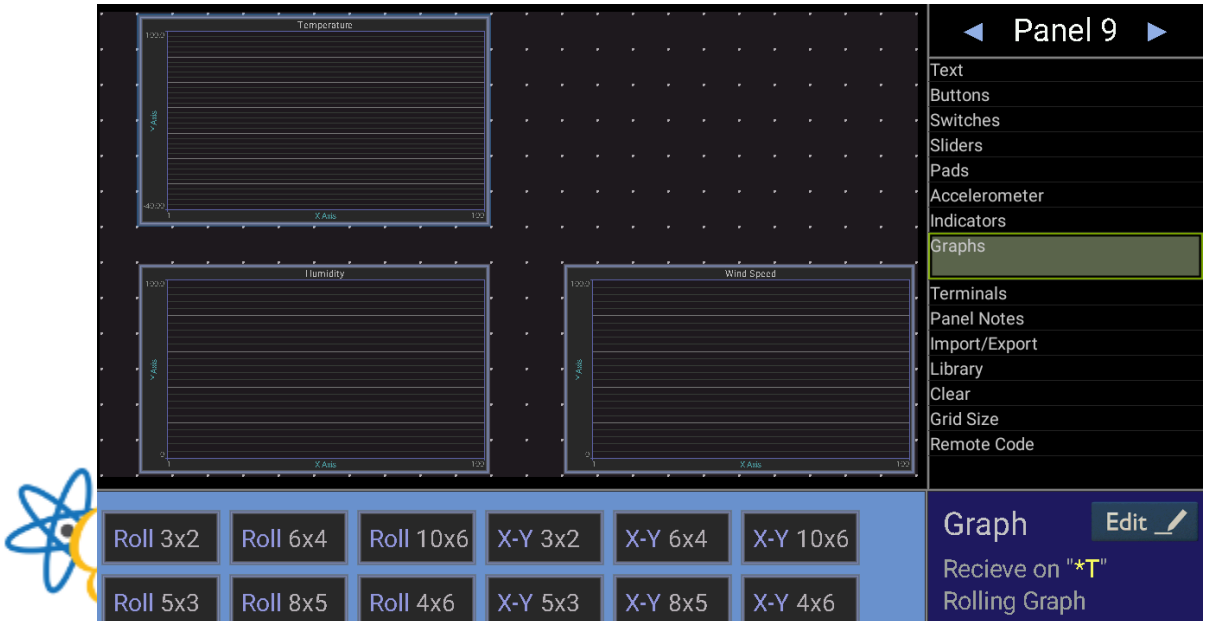
Bluetooth Weather:

https://drive.google.com/file/d/1R5izjsBgzMfYSGape5w5ORg9xXeMz53P/view?usp=drive_link

The graphs shown on the right show realtime data from the Weather station. They collect Temp, Humidity and Wind Speed and display it in three separate graphs. This a simple but powerful example of what this no code app can do.



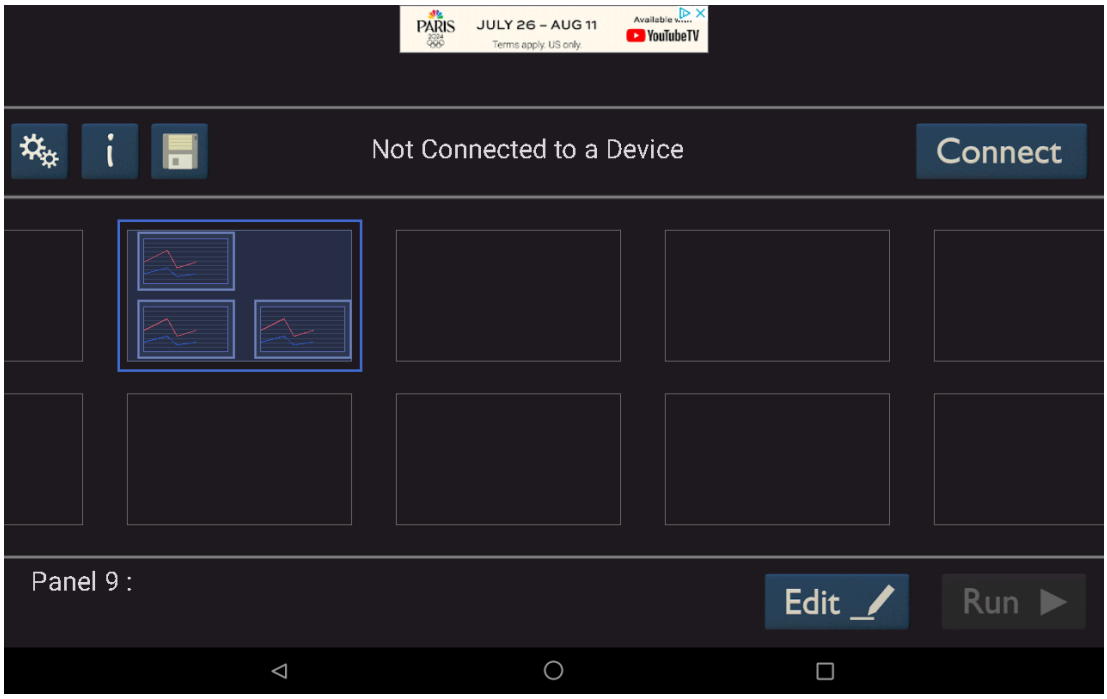
To create your own graphs select an empty panel on the opening screen or edit a previous example screen to modify.



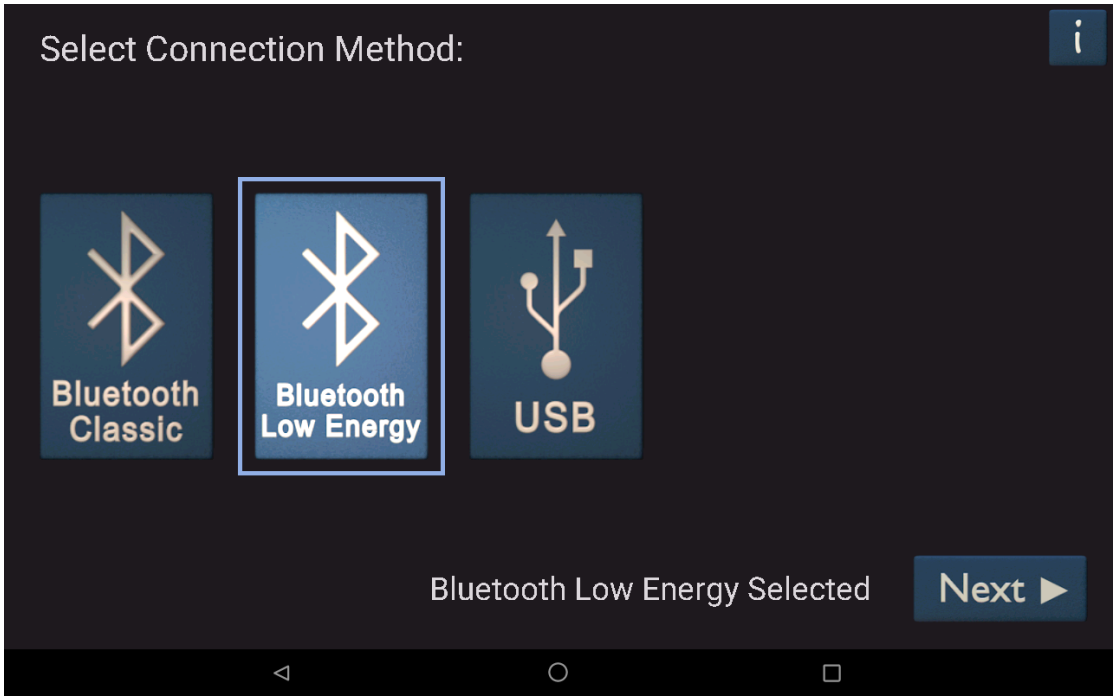
The graphs are created by selecting a new instrument.

We selected the roll 10 x 6 graph (Roll 10x6) and dragged it into the instrument space. Selecting Edit in the lower right corner allows you to edit the transmit character and graph particulars, range, headers, line types, etc.

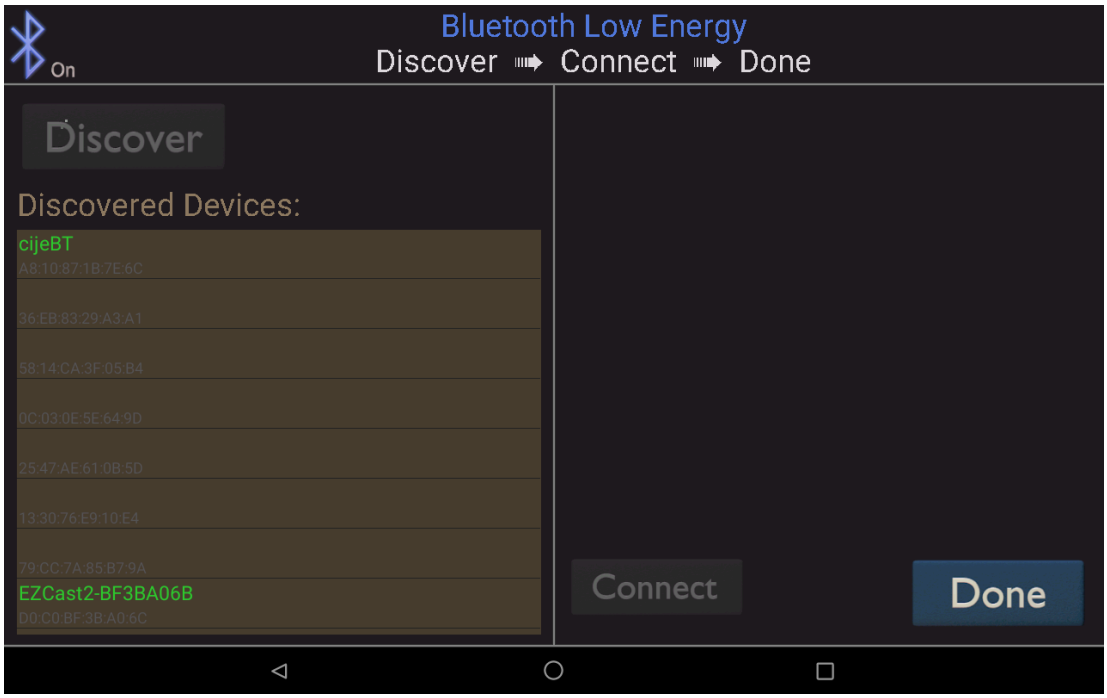
After you are satisfied with the instrument, back out to the panel screen and make sure that you have selected the instrument you want to connect to the remote bluetooth device.



Select connect to goto the connection screen, use Bluetooth Low Energy.

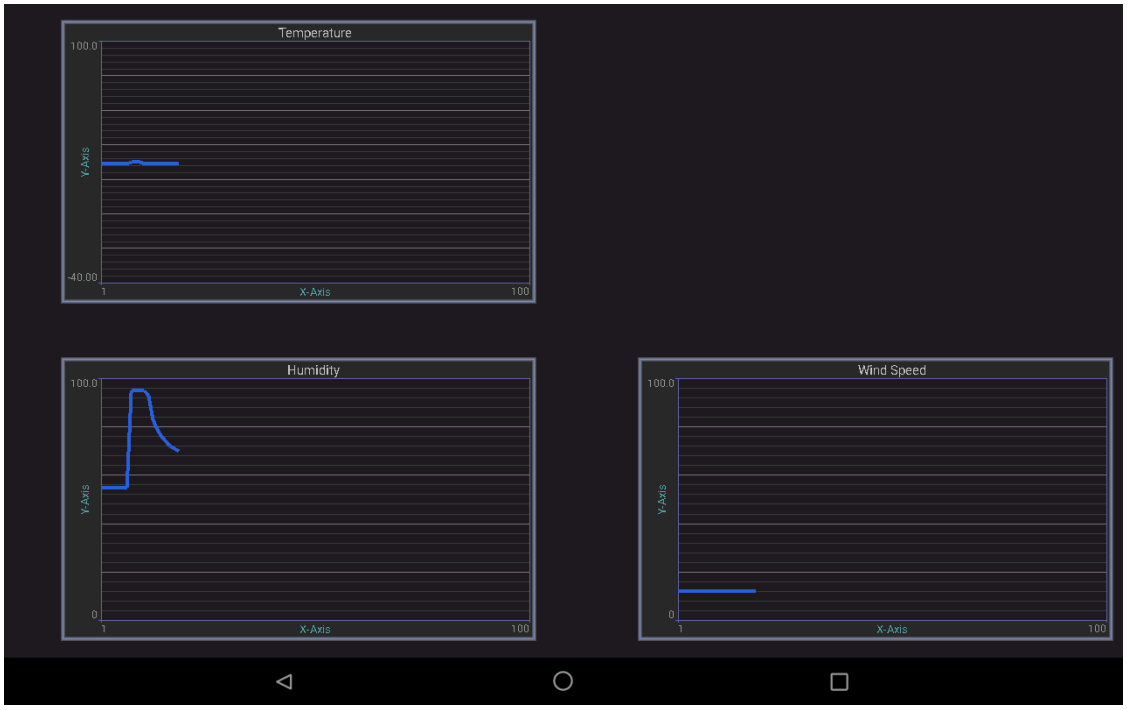


Scan for the bluetooth device that you have previously configured (ex. cijeBT)



Once you have selected and connected with the device press done.

In order to see activity you will need to "reset" or reload your ESP32 bluetooth device to get it to start sending data.


Temperature
100.0
Y-Axis
-40.00
1
X-Axis
100
Humidity
100.0
Y-Axis
0
1
X-Axis
100
Wind Speed
100.0
Y-Axis
0
1
X-Axis
100