

# 1.0 Executive Summary

**OpenWeather++**

**OpenSource Forcasting**

**Tyler Hoeficker**

**Taylor Grimes**

**Ismael Wayne**

**Travis Critchfield**

The OpenWeather++ project is a collection of technologies based around individual and crowd sourced atmospheric data collection. These technologies include Dallas Semiconductor’s One Wire protocol and it’s integrated circuits, Arduino compatible AT Mega based microcontrollers, and Argent Data Systems weather monitoring circuits. While the use of propriety technologies is currently necessary the project aims to be as open source as possible. All source code used in this project is published under the GNU Public License and is freely available to the public via GitHub.com.

The project aims for three fundamental goals. The first is to provide anyone with basic understanding of Arduino based microcontrollers a guide to construct a standardized weather station. The second fundamental goal of the OpenWeather++ project is to centralize the libraries used in this project. Before this project the required libraries where fragmented across the web with no clear explanation to their use or implementation of version control. The project will host an open repository on GitHub for distribution and version control, as well as a build guide with examples. The third and final fundamental goal is to provide a web based API and framework for accessing the organizing the data into information

Two future goals for the project include expansion into other platforms, such as the ARM based Raspberry Pi, and community based source maintenance.

A future fourth goal for the project is to develop an open source community to maintain the source code and provide further development.

# 2.0 Problem Definition

The OpenWeather++ project arose out of library issues when constructing a weather station kit purchased from SparkFun.com. Most of the libraries where incomplete or outdates with poorly written usage examples. The increased number of weather monitoring modules only furthered this problem, which eventually led to a need for a central repository for source maintenance.

After assembling and updating the necessary libraries came the issue of correctly using methods defined in the libraries. Most code assembled was uncommented and without documentation.

# 3.0 Scope

## 3.1 Target Audience

OpenWeather++ is targeted towards individuals with a willingness to learn, or currently holding the following skill sets: Soldering, C++ based programming, and microcontroller driven development. OpenWeather++ provides these individuals with a base source code, examples and documentation to construct a fully operational weather station.

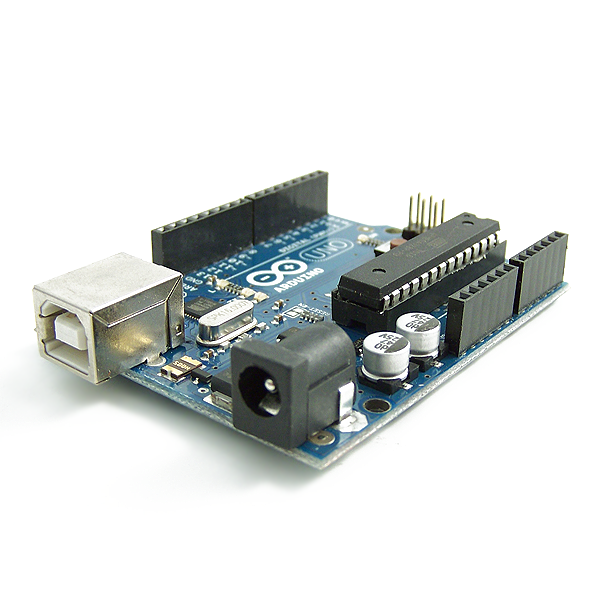
## 3.2 Development Scope

At the time of the first version of this document Aug 14, 2012, OpenWeather++ is in a prototype alpha stage.

# 4.0 Requirements Analysis

## 4.1 Hardware

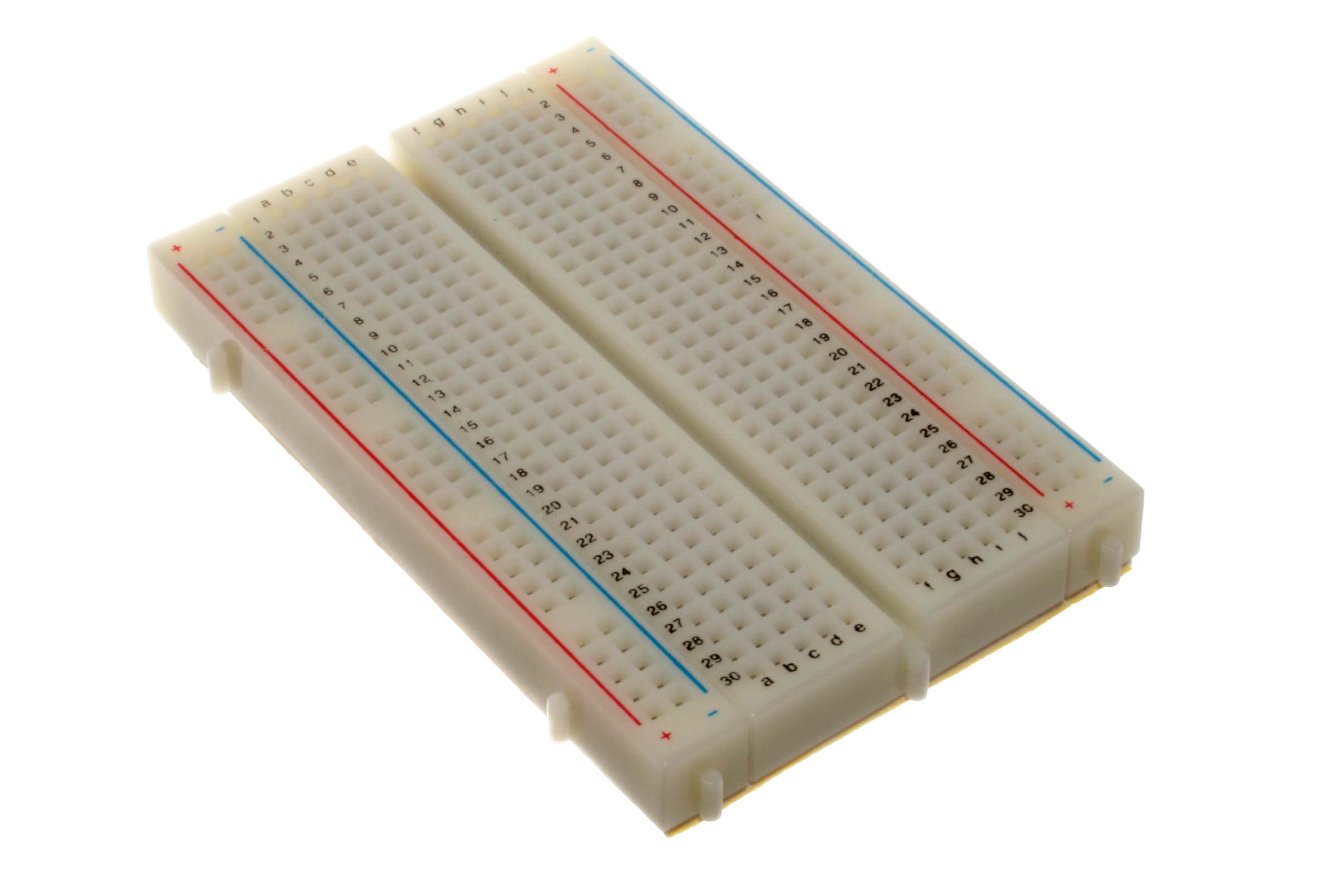
### 4.1.1 Arduino Compatible Prototyping Board (Arduino UNO)



Description: Atmel AVR Atmega 328 based development board. The Arduino UNO, while not required for a final prototype is necessary for testing and development. The Arduino UNO contains 6 analog input/output pins, 13 digital input/output pins, pulse width modulation powered by a Atmel Atmega328 processor. The Atmega328 processor is capable of storing 32 kilobytes of programming instructions that easily fits the 10 kilobytes of programming instructions needed for OpenWeather++.

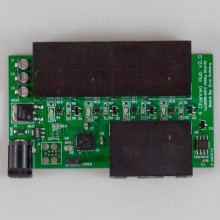
<http://www.arduino.cc/>

### 4.1.2 Solder less Breadboard and Pin Jumpers



Description: Solder less breadboards are used for prototyping electronic designs before creating more permanent PCB layouts. The pin jumpers provide a means of connecting electrical components to each on breadboard. Breadboards can be purchased from local electronics shops and a variety of online retailers including amazon.com, mouser.com, adafruit.com and Sparkfun.com

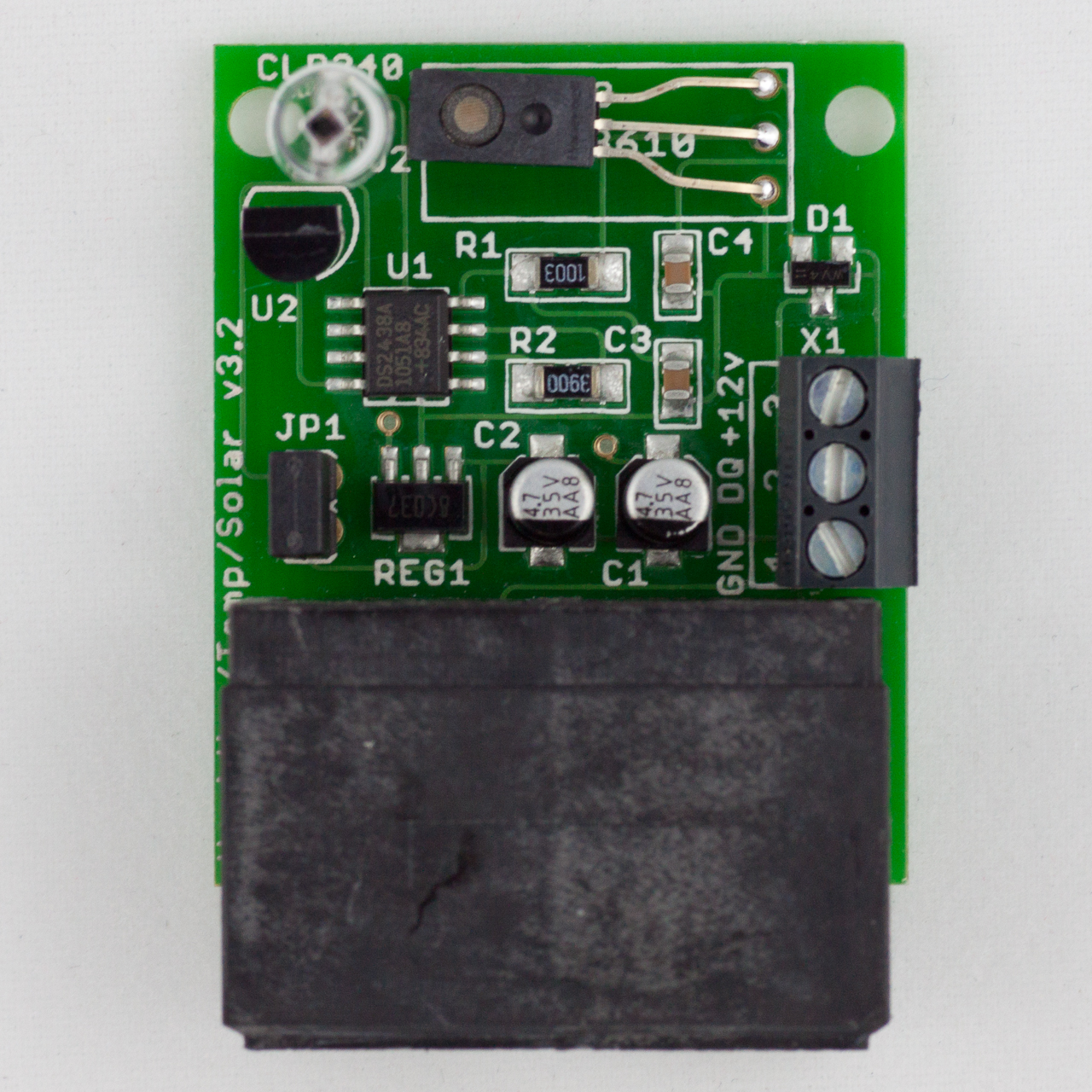
### 4.1.3 Hobby Boards 6 or 4 Channel Power Hub



Description: A hub to provide the required power of five volts across the One-Wire bus for operating multiple modules. Connections provided via RJ-45, with a barrel jack to accommodate DC in from the accompanying AC to DC power adapter.

<http://www.hobby-boards.com/store/products/4-Channel-Hub.html>

### 4.1.4 Hobby Boards Humidity/Temperature/Solar Sensor

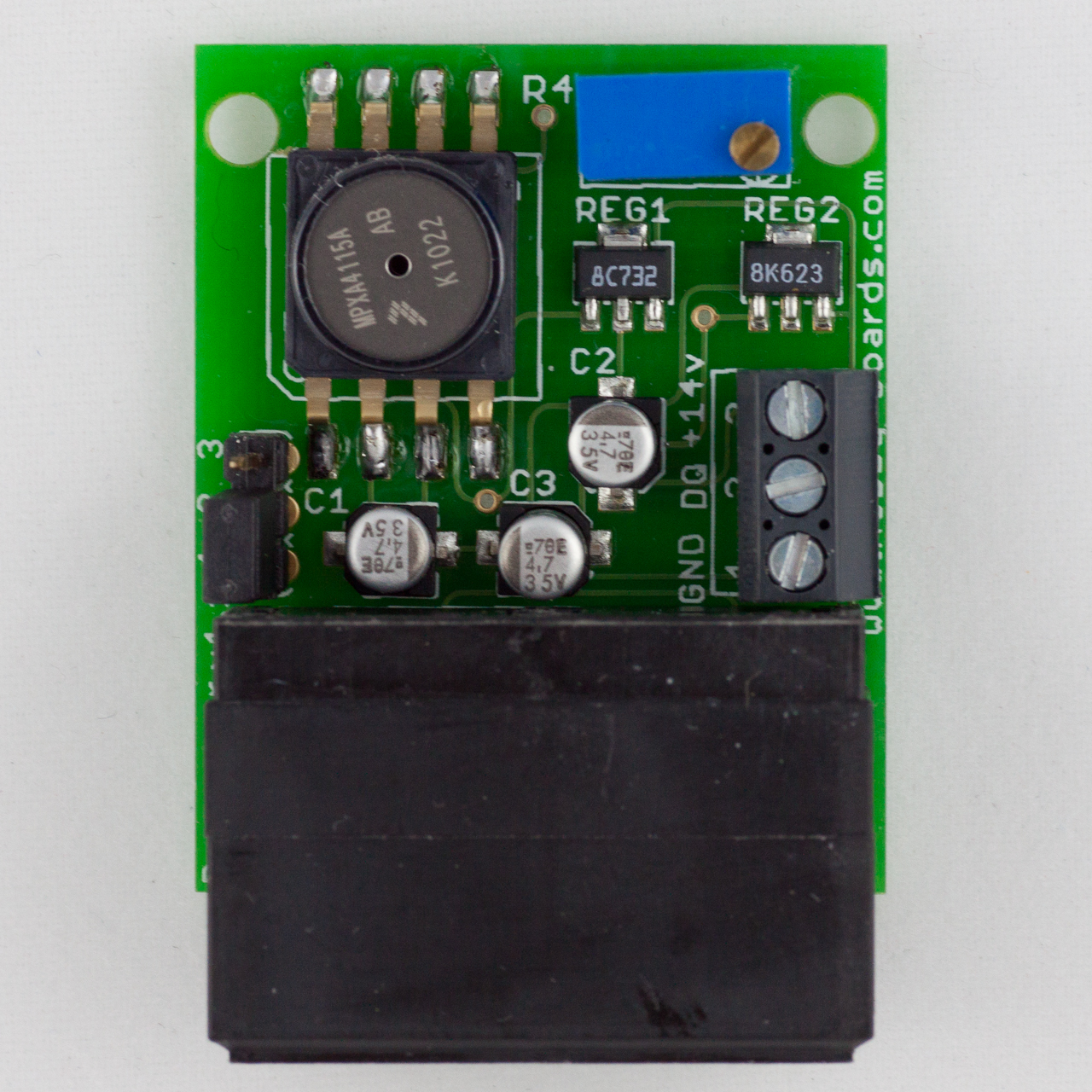


Description: HobbyBoards.com humidity, temperature, and solar sensing module provides a single node for gathering humidity, temperature and visible light levels. The temperature is calculated using a DS1820 digital thermometer communication over the OneWire protocol. Humidity sensing is provided by a HIH-3610, and visible light levels measured by a Clarion CLD240 photodiode. Being that the last to sensors are analog sensors, a DS2438 battery monitor is used to convert their analog data into a digital form. The module required 5 volts to operate.

<http://www.hobby-boards.com/store/products/Humidity%7B47%7DTemp%7B47%7DSolar.html>

*Note: HobbyBoards.com may have changed or will change certain sensors on the module in the future. It is likely that the provided code in OpenWeather++ will work, however it would be still be wise to check your version to see if any code modifications are needed.*

### 4.1.5 Hobby Boards Barometer



Description: The Barometer measures atmospheric pressure, and can be used on its own or as a part of your total weather monitoring system. It reads pressure from 28-32 inHg (948-1083 mb). Its resolution is approximately 0.01 inHg (0.34 mb). Since indoor and outdoor pressure are typically close to the same, we recommend installing this board indoors to avoid possible sensitivity to moisture in an outdoor setting.

This device comes with dual RJ45 connectors, as well as screw terminals, for easy connection to your 1-wire network, and requires power of 15-24 volts DC.

<http://www.hobby-boards.com/store/products/Barometer.html>

### 4.1.6 Hobby Boards Lightning Detector



Description: The Lightning Detector counts lightning strikes, giving you a fascinating new perspective on the electrical storms in your area. The lightning detector needs to be well grounded. We house ours in a length of PVC pipe, which is inexpensive and easy to assemble.

With the 24" antenna this lightning detector will be able to pick up lightning more than 50 miles away. This device comes with dual RJ45 connectors for easy connection to your 1-wire network.

The Lightning Detector requires a 9v battery (not included), this battery should last for at least a couple of years.

<http://www.hobby-boards.com/store/products/Lightning-Detector.html>

### 4.1.7 Development Computer

Description: An x86 based computer to run the Arduino API for development. The computer must be able to run the Arduino API as well as communicate to the Arduino UNO via USB.

### 4.1.8 Web Server

Description: This requirement is optional if the OpenWeather++ build is not using the Wunderground API. The webserver will not only serve as a data collection node, but will also be responsible for serving the weather data to end-users.

## 4.2 Software

### 4.2.1 Arduino Integrated Development Environment (IDE)

Description: Graphical Integrated Development Environment used for development and prototyping with Arduino boards.

<http://arduino.cc/en/Main/Software>

### 4.2.2 Dallas Semiconductor One-Wire Arduino Library

Description: Library for AVR chips that allow the processors of the Arduino to speak and listen to the One Wire protocol.

Included in OpenWeather++ Repository

### 4.2.3 Dallas Semiconductor DS2438 Smart Battery Monitor Library (Patched)

Description: Updated library for communicating and switching modes on the DS2438 Smart Battery Monitor used to meter humidity and solar levels.

Included in OpenWeather++ Repository

### 4.2.4 Dallas Semiconductor DS2409 Library One Wire Powered Hub Library

Description: Library used for the proper handling of OneWire data and addresses.

Included in OpenWeather++ Repository

### 4.2.5 Dallas Semiconductor DS2423 Analog to Digital IC Library

Description:

Included in OpenWeather++ Repository

### 4.2.6 An Operating System running Apache Web Server, MySQL, and PHP

Description: This requirement is necessary if the OpenWeather++ implementation is to require personally hosted data. Examples: Windows Vista or greater, Linux, BSD variants, Mac OS X

### 4.2.7 MySQL version 4 or greater

Description: An open source database management system used for storing the weather data sent from an OpenWeather++ station. Advanced users may substitute their own database management system, but documentation will not be provided.

<http://www.mysql.com/>

### 4.2.8 PHP version 5 or greater

Description: A server side language used for the backend distribution and services for web apps and web pages.

http://www.php.net

### 4.2.9 Apache Web Server

Descript: An open source web server for serving the weather data to a web app.

http://httpd.apache.org/

# 5.0 Design Specifications

## 5.1 Anemometer Wiring and Programming

The Argent Data Systems produced anemometer connects to the Arduino development board via an RJ-11 cable which is compatible with both RJ-11 and RJ-45 ports. The center two pins on the RJ-11 are used for communicating with the Arduino Uno development board. On the positive side of the connection a 10k ohm pull up resistor is necessary for reading interrupts. Each time the anemometer completes a full revolution it closes a magnet-based contact that can be measured as a digital interrupt by the Arduino Uno. When wiring this into an RJ-45 jack, pin 4 on the jack will be connected to ground. Pin 8 is connected to the Arduino digital pin 2 with a 10k pull up resistor connected to 5 volt line. Note that one interrupt per second is equivalent to a wind speed of 1.492 MPH or 2.4 km/h.

##### The wind vane is initialized with the following statements

1. #define PIN\_ANEMOMETER 2 // Digital 2
2. #define uint unsigned int
3. #define ulong unsigned long

##### The next section of code sets up a few variables for timing and calculation

1. // How often we want to calculate wind speed or direction
2. #define MSECS\_CALC\_WIND\_SPEED 1000
3. volatile int numRevsAnemometer = 0; // Incremented in the interrupt
4. ulong nextCalcSpeed; // When we next calc the wind speed
5. ulong time; // Millis() at each start of loop().

##### Code to add to the setup() function

1. Serial.begin(9600); //For serial debugging
2. pinMode(PIN\_ANEMOMETER, INPUT);
3. digitalWrite(PIN\_ANEMOMETER, HIGH);
4. attachInterrupt(0, countAnemometer, FALLING);
5. nextCalcSpeed = millis() + MSECS\_CALC\_WIND\_SPEED;
6. unsigned long startTime = millis(); //set start time

##### Code to add to the loop() function

1. time = millis(); //get the current time
2. if (time >= nextCalcSpeed) {
3. calcWindSpeed();
4. nextCalcSpeed = time + MSECS\_CALC\_WIND\_SPEED;
5. }

##### The countAnemometer() prototype

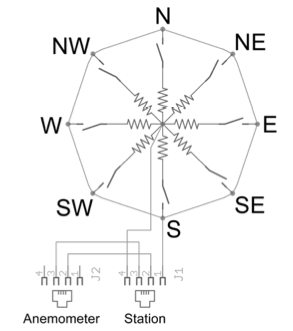
1. void countAnemometer() {
2. numRevsAnemometer++;
3. }

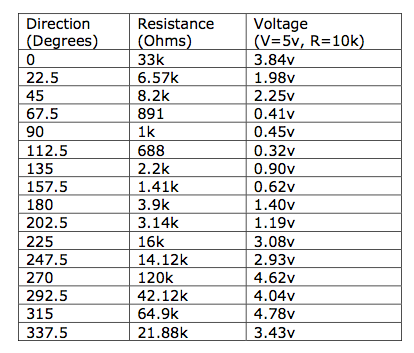
##### The calcWindSpeed() prototype

1. void calcWindSpeed() {
2. int x, iSpeed;
3. // This will produce mph \* 10
4. long speed = 14920;
5. speed \*= numRevsAnemometer;
6. speed /= MSECS\_CALC\_WIND\_SPEED;
7. iSpeed = speed; // Need this for formatting below
8. Serial.print("Wind speed: ");
9. x = iSpeed / 100;
10. Serial.print(x);
11. Serial.print('.');
12. x = iSpeed % 10;
13. Serial.print(x);
14. Serial.println();
15. numRevsAnemometer = 0; // Reset counter
16. }

## 5.2 Wind Direction Wiring and Code

Wind direction is read in from the Argent Data Systems Wind Vane. The wind vane uses eight magnetic switches to measure the position of the wind vane. Each magnetic switch holds a different resistance value and two magnetic switches can be tripped at the same time allowing for a sixteen calculable positions. The wind vane uses the two outer wires on the same RJ-11 socket that the anemometer uses. As with the anemometer, reading the connection requires a 10k ohm pull-up resistor on the positive line. Exact wiring is as follows: Arduino analog pin 5 is connected to RJ-45 port 8 along with a 10k pull up resistor connected to a 5v line. RJ-45 pin 4 is connected to the Arduino ground pin.





*­­Note: These resistances are used read via analog in on the Arduino and their values mapped accordingly. When the weather station is emplaced it must be have its read in north aligned with true north.*

##### The wind vane is initialized with the following statements

1. #define PIN\_VANE 5 // Analog 5
2. #define ulong unsigned long //shorthand type
3. #define NUMDIRS 8 //define num directions
4. #define MSECS\_CALC\_WIND\_DIR 1000

##### The next section of code sets up a few variables to deal with the different directions

1. ulong adc[NUMDIRS] = {26, 45, 77, 118, 161, 196, 220, 256};
2. // These directions match 1-for-1 with the values in adc, but
3. // will have to be adjusted as noted above. Modify 'dirOffset'
4. // to which direction is 'away' (it's West here).
5. char \*strVals[NUMDIRS] = {"W","NW","N","SW","NE","S","SE","E"};
6. byte dirOffset=0;
7. ulong nextCalcDir; // When we next calc the direction
8. ulong time; // Millis() at each start of loop().

##### Code to add to the setup() function

1. Serial.begin(9600); //For serial debugging
2. nextCalcDir = millis() + MSECS\_CALC\_WIND\_DIR; //set time to calculate
3. unsigned long startTime = millis(); //set start time

##### Code to add to the loop() function

1. time = millis(); //get the current time
2. if (time >= nextCalcDir) { //if current time is greater or equal to time to calculate
3. calcWindDir(); //call the wind direction calculating function
4. nextCalcDir = time + MSECS\_CALC\_WIND\_DIR; //set the next time to calculate
5. }

##### The calcWindDir() prototype

1. void calcWindDir() {
2. int val;
3. byte x, reading;
4. val = analogRead(PIN\_VANE);
5. val >>=2; // Shift to 255 range
6. reading = val;
7. // Look the reading up in directions table. Find the first value
8. // that's >= to what we got.
9. for (x=0; x<NUMDIRS; x++) {
10. if (adc[x] >= reading)
11. break;
12. }
13. x = (x + dirOffset) % 8; // Adjust for orientation
14. Serial.print("Dir: ");
15. Serial.print(strVals[x]);
16. Serial.println();
17. }

## 5.3 Rain Gauge Wiring and Code

The rain meter module functions on the design of a tip bucket. That is to say that after the bucket inside of the module fills to a certain threshold (.011" of precipitation) it tips to either side causing a contact closure in the circuit. The tip bucket then empties itself and moves to its initial position waiting to fill up again. The rain gauge operates on its own CAT 2 cable. When plugged into an RJ-45 port the rain gauge is grounded on pin 4. Data is transmitted on pin 8. The data line is wired to Arduino pin 4 and must be connected to a 5 volt line via a 10k pull up resistor.

To Recap:

Arduino pin 4 -> 10k pull up resistor to 5v -> RJ-45 pin 8

Arduino ground (or power supply ground) -> RJ-45 pin 4

##### The rain gauge is initialized with the following statements

1. #define PIN\_RAIN 3 // Digital 3
2. #define uint unsigned int
3. #define ulong unsigned long

##### The next section of code sets up a few variables for timing and calculation

1. // How often we want to calculate wind speed or direction
2. #define MSECS\_CALC\_RAIN 60000 //60 second interval
3. volatile int numRainDrops = 0; // Incremented in the interrupt
4. ulong nextCalcRain; // When we next calc the wind speed
5. ulong time; // Millis() at each start of loop().

##### Code to add to the setup() function

1. Serial.begin(9600); //For serial debugging
2. digitalWrite(PIN\_RAIN, HIGH);
3. digitalWrite(PIN\_ANEMOMETER, HIGH);
4. attachInterrupt(1, countRain, FALLING); //Fire countRain when the line goes from high to low
5. nextCalcRain = millis() + MSECS\_CALC\_RAIN;
6. unsigned long startTime = millis(); //set start time

##### Code to add to the loop() function

1. time = millis(); //get the current time
2. if (time >= nextCalcSpeed) {
3. calcWindSpeed();
4. nextCalcSpeed = time + MSECS\_CALC\_WIND\_SPEED;
5. }

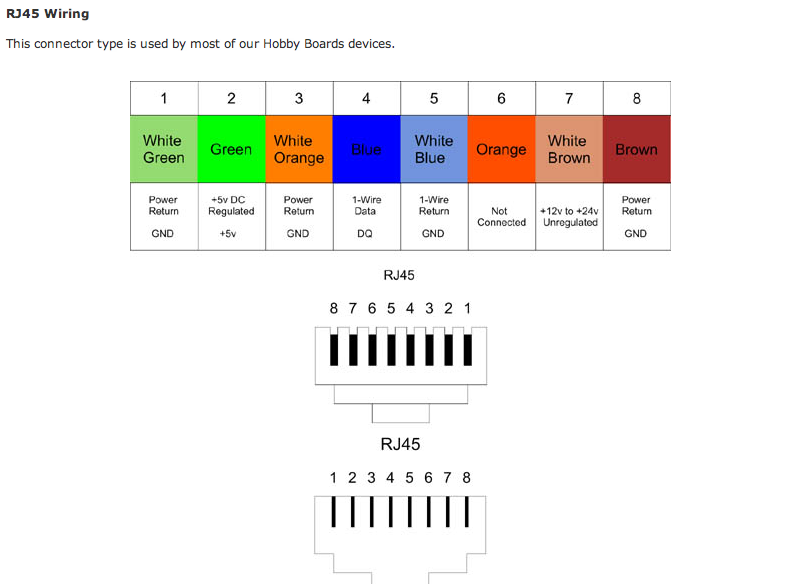
##### The countRain() prototype

1. void countRain() {
2. numRainDrops++;
3. }

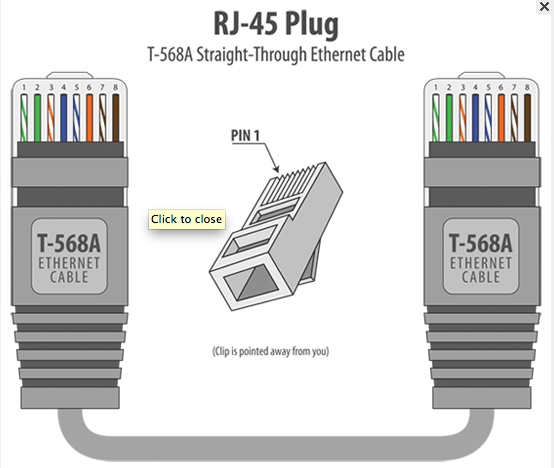
##### The calcRainFall() prototype

1. void calcRainFall() {
2. Serial.print("Rain Fall: "); //debugging mode
3. Serial.print(numRainDrops \* .011);
4. }

## 5.4 Wiring of One Wire Based Devices

One Wire modules rely on the use of RJ45 ports along with CAT 5 or CAT 6 cables for communication and power. A “Strait-Through” Ethernet wiring scheme is to be used with the Ethernet cables.

Power is supplied from the One Wire Hub while data is transferred over Ethernet pins four and five (these are colors blue and blue/white respectively). Pin four provides data and is connected to digital pin four on the Arduino while pin five is connected to ground.



## 5.5 Temperature

Temperature is for the OpenWeather++ system is calculated within the HobbyBoards Temperature, Humidity, and Solar module. This process is accomplished with the Dallas Semiconductor DS18S20. The DS18S20 is a One Wire IC belonging to a family of DS18xx digital thermometers. These digital thermometers require little circuitry to operate on the One Wire bus and can be be wired in as an individual module if your build is requires a reduced cost. The devices at the time of writing this build guide can be purchased from 2 to 5 US dollars. The DS18S20 provides an 8-bit temperature resolution. If greater accuracy is required, the DS18B20 should be used instead. Weather used inside the HobbyBoards module or as a stand-alone meter, the cost, simplicity and accuracy of the DS18xx line of thermometers makes it a must in any form of OpenWeather++ build.

##### Required Libraries

1. #include <OneWire.h>

##### Initialization

1. OneWire oneWire(4); //OneWire instantiation on Arduino Digital Pin 4

##### Code to add to the setup() function

1. Serial.begin(9600); //For serial debugging

##### Code to add to the loop() function

1. getTemp();//Call our function to return a temp;

##### getTemp function prototype

1. //function to retrieve temp in cel from oneWire18s20 or oneWire1820
2. float getTemp(){
3. byte i;
4. byte data[12];
5. int Temp;
6. oneWire.reset();
7. oneWire.select(oneWire18x20\_address);
8. oneWire.write(0x44,1); // start conversion, with parasite power on at the end
9. // Resolution 9 bit 10 bit 11 bit 12 bit
10. // Conversion Time (ms) 93.75 187.5 375 750
11. // LSB (Â°C) 0.5 0.25 0.125 0.0625
12. delay(750);
13. oneWire.reset();
14. oneWire.select(oneWire18x20\_address);
15. oneWire.write(0xBE); // Read Scratchpad
16. for ( i = 0; i < 9; i++) { // we need 9 bytes (9 byte resolution on oneWire1820 & oneWire18s20)
17. data[i] = oneWire.read();
18. }
19. Temp=(data[1]<<8)+data[0];//take the two bytes from the response relating to temperature
20. Temp=Temp>>1;//divide by 16 to get pure Celsius readout
21. Temp=Temp\*1.8+32; // comment this line out to get fahrenheit
22. float t = Temp;
23. return t;
24. }

# 6.0 External Documents

Argent Data Systems Weather Sensor Assembly p/n 80422 Data Sheet

HobbyBoards One Wire Hub Data Sheet

HobbyBoards Lightning Detector Schematic

HobbyBoards Barometer Schematic

MPXA4115A Schematic

http://www.arduino.cc/playground/Learning/OneWire

# 7.0 Development Methodology

The OpenWeather++ project uses a Spiral Development Methodology. The reason the team chose this methodology is that spiral development offers advantages that tailor to the OpenWeather++ project. The main advantage is that it allows us to build a very basic prototype to confirm working units, and then progress to more advanced models offering different levels of features along the way.

# 8.0 Testing

## 8.1 Rain Gauge Testing

Testing of the rain gauge is done with a cup of water, and exactly 5 cubic inches of water, a container to catch the water exiting the gauge, and a computer for data logging.

1. Begin data logging with the computer and slowly pour the water into the rain gauge.
2. After all the water has been poured into the rain gauge check the total reported precipitation
3. If the rain gauge is reports a sum with a tolerance greater than .1 inches check the code to ensure each interrupt is measured as .011".
4. If the code is correct and the gauge is has a tolerance of error greater than .5" the unit is defective and needs to be replaced.

## 8.2 Anemometer Testing

## Unit testing for the anemometer is accomplished with an automobile with an accurate speed gauge or a wind chamber. Testing with a motor vehicle should be done in a controlled environment and not on an open road. For testing with an automobile complete the following steps.

## With the anemometer either helped by an assistant or secured to the interior passenger side of the vehicle along with a computer for reading the data, position the anemometer slightly outside of the passenger window.

## Safely begin operating the vehicle at a steady 10 miles per hour. Once the vehicle has reached 10 MPH, begin data logging. Keep the speed for a long enough period of time to get enough data for a good average speed reading from the anemometer.

## After stopping the vehicle, calculate the average speed readings. If the average is greater than or less than 10 MPH by 1 MPH, annotate as a note and proceed with the next step.

## Repeat steps one through three increasing the speed of the vehicle by 10 MPH with each increment all the way up to 30 MPH.

## If the error of the anemometer is linear, it may be corrected in the code with an adjustment. If the error is not linear and is sporadic there is an error in the module and it needs to be replaced.

## 8.3 Wind Vane Testing

Unit testing is performed with the wind vane, an Arduino linked to a PC with serial out debugging enabled, or another means of reading the reported wind vane direction, and a compass that has an error tolerance no greater than 2°. Perform the following steps to ensure the wind vane is within an acceptable means of error tolerance, which for this module is 15°.

1. Emplace the weather unit in a fixed stable position for testing
2. Find magnetic north with the reference compass
3. Rotate the wind vane until its reading show north. There are small cardinal direction guide markings on the top of the non-moving element of the wind vane
4. Keeping the wind vane module static & ensuring that it stays reading north, rotate the weather station assembly until the wind vane is aligned with the compass's magnetic north azimuth
5. Rotate the wind vane 45° clockwise
6. Using the compass, shoot an azimuth aligned with the wind vane. The wind vane should read NE while the compass 45°.
7. Rinse and repeat for all cardinal and ordinal directions
8. If your wind vane module is off at any directions this probably means it has an error tolerance greater than 15° and needs to be replaced as calibration is not within the scope of this tutorial

## 8.4 Thermometer Testing

The DS18S20 comes pre-calibrated and tested from the factory, with an error tolerance of .5 °Celsius between -10 °C and 85 °C. However to ensure proper reading of the thermometer, a unit test should be done after interfacing with the DS18S20. To test a second thermometer known to be accurate will be needed as a control. The test will also require a computer and microcontroller or other form of communication to log readings from the DS18S20.

1. Let both thermometers sit in the current environment for one minute to ensure stable readings.
2. After one minute, begin data polling both thermometers every 30 seconds for two minutes.
3. Once the two minutes of testing is up, compute the average for each thermometer and compare results.
4. If the thermometers differ more than two degrees check the source code of the DS18S20 for accuracy.

Note: Although the thermometers may have subtle differences in readings, this does not indicate that the DS18S20 is faulty.

# 9.0 Maintenance and Life Cycle

# 10.0 Definitions of terminology used in this document

AVR: Advanced Virtual RISC. Architecture of microcontrollers produced by the Atmel Corporation.

ARM: Advanced RISC Machines. A company that designs microprocessor architectures.

PHP: Pre-Hypertext Processor. A backend web server language.

MySQL: A relational database management system used for storing data.

Ethernet: A Layer 2 network standard.

CAT2/5: A Layer 1 network standard.