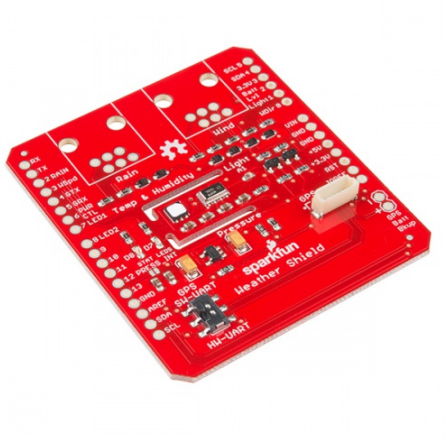
Arduino Weather Shield Hookup Guide V12

Weather Shield Overview

The Arduino Weather Shield from SparkFun is an easy-to-use Arduino shield that grants you access to barometric pressure, relative humidity, luminosity, and temperature. There are also connections to optional sensors such as wind speed/direction, rain gauge, and GPS for location and super accurate timing.



SparkFun Weather Shield

DEV-13956

Things you should know about this shield:

- Uses the Si7021 sensor, MPL3115A2 barometric pressure sensor, and ALS-PT19 light sensor.
- Has connector for the GP-735 compact GPS module
- Has optional RJ11 connector footprints to connect the SparkFun weather meters
- Weather shield can operate from **3V to 10V** and has built in voltage regulators and signal translators
- Typical humidity accuracy of ±2%
- Typical pressure accuracy of ±50Pa
- Typical temperature accuracy of ±0.3C

Suggested Reading

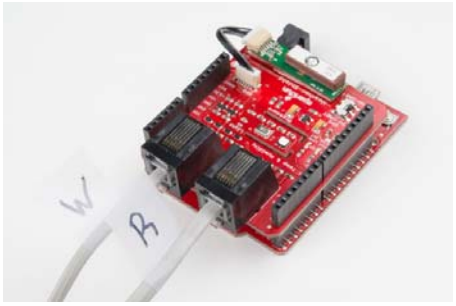
- I^2C Protocol

- Installing an Arduino library
- What is an Arduino Shield
- What are pull-up resistors?
- Si7021 Temperature and Humidity Sensor Hookup Guide
- MPL3115A2 Barometric Sensor Hookup Guide

Hooking It Up

To get up and running with the Weather Shield you'll need the following parts:

- Arduino, RedBoard, or other compatible board
- Arduino Stackable Headers
- Optional: GP-735 GPS Module and 1.75" mating cable
- Optional: Two RJ11 6-pin Connectors
- Optional: Weather Meters



Shield on a RedBoard with optional weather meter ('W'ind and 'R'ain cables) and GPS attached

Assembly

Solder the stackable headers onto the shield, and insert the shield into your Arduino. You are welcome to solder in the RJ11 connectors to the top of the board as well. If you have the GP-735 GPS module, don't worry about attaching it at this time, we'll get to GPS later.

Example Firmware - Basic

Before uploading code to your Arduino with the Weather Shield attached, make sure the GPS UART switch is in the SW-UART position. Having the switch in the opposite position connects the GPS lines to the USB lines and may cause errors while uploading.

Using the Weather Shield example in the Arduino IDE relies on the Si7021 and MPL3115A2 libraries. As of Arduino v1.6.x you can download the libraries through the Arduino Library Manager. Search for and install "SparkFun MPL3115" and "SparkFun Si7021".

For more information, see our tutorial on using the Arduino library manager. For all the latest Arduino Weather Shield code, check out the Github Repository:

WEATHER SHIELD GITHUB REPO https://github.com/sparkfun/Weather_Shield

Open the **Weather_Shield_Basic.ino** sketch.

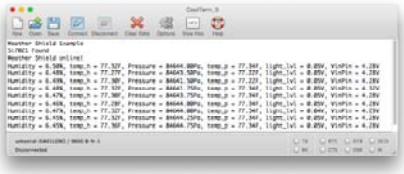
Or copy and paste the code below into the Arduino IDE:

```
/*
 Weather Shield Example
 By: Nathan Seidle
 SparkFun Electronics
 Date: June 10th, 2016
 License: This code is public domain but you buy me a beer if
you use this and we meet someday (Beerware license).

 This example prints the current humidity, air pressure, tempe
rature and light levels.

 The weather shield is capable of a lot. Be sure to checkout t
he other more advanced examples for creating
 your own weather station.

 Updated by Joel Bartlett
 03/02/2017
 Removed HTU21D code and replaced with Si7021
 */

#include <Wire.h> //I2C needed for sensors
#include "SparkFunMPL3115A2.h" //Pressure sensor - Search "Spa
rkFun MPL3115" and install from Library Manager
#include "SparkFun_Si7021_Breakout_Library.h" //Humidity senso
r - Search "SparkFun Si7021" and install from Library Manager

MPL3115A2 myPressure; //Create an instance of the pressure sen
sor
Weather myHumidity;//Create an instance of the humidity sensor

//Hardware pin definitions
//-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=
const byte STAT_BLUE = 7;
const byte STAT_GREEN = 8;

const byte REFERENCE_3V3 = A3;
const byte LIGHT = A1;
const byte BATT = A2;

//Global Variables
//-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=
long lastSecond; //The millis counter to see when a second rol
ls by

void setup()
{
  Serial.begin(9600);
  Serial.println("Weather Shield Example");

  pinMode(STAT_BLUE, OUTPUT); //Status LED Blue
  pinMode(STAT_GREEN, OUTPUT); //Status LED Green

  pinMode(REFERENCE_3V3, INPUT);
  pinMode(LIGHT, INPUT);

  //Configure the pressure sensor
  myPressure.begin(); // Get sensor online
  myPressure.setModeBarometer(); // Measure pressure in Pascal
s from 20 to 110 kPa
  myPressure.setOversampleRate(7); // Set Oversample to the re
commended 128
  myPressure.enableEventFlags(); // Enable all three pressure
and temp event flags
```

```
  //Configure the humidity sensor
  myHumidity.begin();

  lastSecond = millis();

  Serial.println("Weather Shield online!");
}

void loop()
{
  //Print readings every second
  if (millis() - lastSecond >= 1000)
  {
    digitalWrite(STAT_BLUE, HIGH); //Blink stat LED

    lastSecond += 1000;

    //Check Humidity Sensor
    float humidity = myHumidity.getRH();

    if (humidity == 998) //Humidty sensor failed to respond
    {
      Serial.println("I2C communication to sensors is not work
ing. Check solder connections.");

      //Try re-initializing the I2C comm and the sensors
      myPressure.begin();
      myPressure.setModeBarometer();
      myPressure.setOversampleRate(7);
      myPressure.enableEventFlags();
      myHumidity.begin();
    }
    else
    {
      Serial.print("Humidity = ");
      Serial.print(humidity);
      Serial.print("%,");
      float temp_h = myHumidity.getTempF();
      Serial.print(" temp_h = ");
      Serial.print(temp_h, 2);
      Serial.print("F,");

      //Check Pressure Sensor
      float pressure = myPressure.readPressure();
      Serial.print(" Pressure = ");
      Serial.print(pressure);
      Serial.print("Pa,");

      //Check tempf from pressure sensor
      float tempf = myPressure.readTempF();
      Serial.print(" temp_p = ");
      Serial.print(tempf, 2);
      Serial.print("F,");

      //Check light sensor
      float light_lvl = get_light_level();
      Serial.print(" light_lvl = ");
      Serial.print(light_lvl);
      Serial.print("V,");

      //Check batt level
      float batt_lvl = get_battery_level();
      Serial.print(" VinPin = ");
      Serial.print(batt_lvl);
      Serial.print("V");
```

```
      Serial.println();
    }

    digitalWrite(STAT_BLUE, LOW); //Turn off stat LED
  }

  delay(100);
}

//Returns the voltage of the light sensor based on the 3.3V ra
il
//This allows us to ignore what VCC might be (an Arduino plugg
ed into USB has VCC of 4.5 to 5.2V)
float get_light_level()
{
  float operatingVoltage = analogRead(REFERENCE_3V3);

  float lightSensor = analogRead(LIGHT);

  operatingVoltage = 3.3 / operatingVoltage; //The reference v
oltage is 3.3V

  lightSensor = operatingVoltage * lightSensor;

  return (lightSensor);
}

//Returns the voltage of the raw pin based on the 3.3V rail
//This allows us to ignore what VCC might be (an Arduino plugg
ed into USB has VCC of 4.5 to 5.2V)
//Battery level is connected to the RAW pin on Arduino and is
fed through two 5% resistors:
//3.9K on the high side (R1), and 1K on the low side (R2)
float get_battery_level()
{
  float operatingVoltage = analogRead(REFERENCE_3V3);

  float rawVoltage = analogRead(BATT);

  operatingVoltage = 3.30 / operatingVoltage; //The reference
voltage is 3.3V

  rawVoltage = operatingVoltage * rawVoltage; //Convert the 0
to 1023 int to actual voltage on BATT pin

  rawVoltage *= 4.90; //(3.9k+1k)/1k - multiple BATT voltage b
y the voltage divider to get actual system voltage

  return (rawVoltage);
}
```

Open the Serial Monitor. You should see the following output:



Put your hand over the small clear device labeled ‘Light’, and watch the light level change to 0. Blow lightly on the humidity sensor, and watch the humidity change.

Troubleshooting

If there is an error you will see:

```
I2C communication to sensors is not working. Check solder conn
ections.
```

This message appears when the board is unable to get a response from the I2C sensors. This could be because of a faulty solder connection, or if there are other devices on the A5/A4 lines (which are also called SDA/SCL).

Example Firmware - Weather Station

For the more adventurous, we have the **Weather Station** example. This code demonstrates *all* the bells and whistles of the shield. You will need a weather station hooked up to see the wind speed, wind direction and rain values change.

```
/*
 Weather Shield Example
 By: Nathan Seidle
 SparkFun Electronics
 Date: November 16th, 2013
 License: This code is public domain but you buy me a beer if
you use this and we meet someday (Beerware license).

 Much of this is based on Mike Grusin's USB Weather Board cod
e: https://www.sparkfun.com/products/10586

 This is a more advanced example of how to utilize every aspec
t of the weather shield. See the basic
 example if you're just getting started.

 This code reads all the various sensors (wind speed, directio
n, rain gauge, humidity, pressure, light, batt_lvl)
 and reports it over the serial comm port. This can be easily
routed to a datalogger (such as OpenLog) or
 a wireless transmitter (such as Electric Imp).

 Measurements are reported once a second but windspeed and rai
n gauge are tied to interrupts that are
 calculated at each report.

 This example code assumes the GPS module is not used.


  Updated by Joel Bartlett
  03/02/2017
  Removed HTU21D code and replaced with Si7021

 */

#include <Wire.h> //I2C needed for sensors
#include "SparkFunMPL3115A2.h" //Pressure sensor - Search "Spa
rkFun MPL3115" and install from Library Manager
#include "SparkFun_Si7021_Breakout_Library.h" //Humidity senso
r - Search "SparkFun Si7021" and install from Library Manager

MPL3115A2 myPressure; //Create an instance of the pressure sen
sor
Weather myHumidity;//Create an instance of the humidity sensor


//Hardware pin definitions
//-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=
// digital I/O pins
const byte WSPEED = 3;
const byte RAIN = 2;
const byte STAT1 = 7;
const byte STAT2 = 8;

// analog I/O pins
const byte REFERENCE_3V3 = A3;
const byte LIGHT = A1;
const byte BATT = A2;
const byte WDIR = A0;
//-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=

//Global Variables
//-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=
long lastSecond; //The millis counter to see when a second rol
ls by
```

```
byte seconds; //When it hits 60, increase the current minute
byte seconds_2m; //Keeps track of the "wind speed/dir avg" ove
r last 2 minutes array of data
byte minutes; //Keeps track of where we are in various arrays
of data
byte minutes_10m; //Keeps track of where we are in wind gust/d
ir over last 10 minutes array of data

long lastWindCheck = 0;
volatile long lastWindIRQ = 0;
volatile byte windClicks = 0;

//We need to keep track of the following variables:
//Wind speed/dir each update (no storage)
//Wind gust/dir over the day (no storage)
//Wind speed/dir, avg over 2 minutes (store 1 per second)
//Wind gust/dir over last 10 minutes (store 1 per minute)
//Rain over the past hour (store 1 per minute)
//Total rain over date (store one per day)

byte windspdavg[120]; //120 bytes to keep track of 2 minute av
erage

#define WIND_DIR_AVG_SIZE 120
int winddiravg[WIND_DIR_AVG_SIZE]; //120 ints to keep track o
f 2 minute average
float windgust_10m[10]; //10 floats to keep track of 10 minut
e max
int windgustdirection_10m[10]; //10 ints to keep track of 10 m
inute max
volatile float rainHour[60]; //60 floating numbers to keep tra
ck of 60 minutes of rain

//These are all the weather values that wunderground expects:
int winddir = 0; // [0-360 instantaneous wind direction]
float windspeedmph = 0; // [mph instantaneous wind speed]
float windgustmph = 0; // [mph current wind gust, using softwa
re specific time period]
int windgustdir = 0; // [0-360 using software specific time pe
riod]
float windspdmph_avg2m = 0; // [mph 2 minute average wind spee
d mph]
int winddir_avg2m = 0; // [0-360 2 minute average wind directi
on]
float windgustmph_10m = 0; // [mph past 10 minutes wind gust m
ph ]
int windgustdir_10m = 0; // [0-360 past 10 minutes wind gust d
irection]
float humidity = 0; // [%]
float tempf = 0; // [temperature F]
float rainin = 0; // [rain inches over the past hour)] -- the
accumulated rainfall in the past 60 min
volatile float dailyrainin = 0; // [rain inches so far today i
n local time]
//float baromin = 30.03;// [barom in] - It's hard to calculat
e baromin locally, do this in the agent
float pressure = 0;
//float dewptf; // [dewpoint F] - It's hard to calculate dewpo
int locally, do this in the agent

float batt_lvl = 11.8; //[analog value from 0 to 1023]
float light_lvl = 455; //[analog value from 0 to 1023]

// volatiles are subject to modification by IRQs
volatile unsigned long raintime, rainlast, raininterval, rain;
```

```
//-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=

//Interrupt routines (these are called by the hardware interru
pts, not by the main code)
//-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=
void rainIRQ()
// Count rain gauge bucket tips as they occur
// Activated by the magnet and reed switch in the rain gauge,
attached to input D2
{
    raintime = millis(); // grab current time
    raininterval = raintime - rainlast; // calculate interval
between this and last event

    if (raininterval > 10) // ignore switch-bounce glitches le
ss than 10mS after initial edge
    {
        dailyrainin += 0.011; //Each dump is 0.011" of water
        rainHour[minutes] += 0.011; //Increase this minute's a
mount of rain

        rainlast = raintime; // set up for next event
    }
}

void wspeedIRQ()
// Activated by the magnet in the anemometer (2 ticks per rota
tion), attached to input D3
{
    if (millis() - lastWindIRQ > 10) // Ignore switch-bounce g
litches less than 10ms (142MPH max reading) after the reed swi
tch closes
    {
        lastWindIRQ = millis(); //Grab the current time
        windClicks++; //There is 1.492MPH for each click per s
econd.
    }
}


void setup()
{
    Serial.begin(9600);
    Serial.println("Weather Shield Example");

    pinMode(STAT1, OUTPUT); //Status LED Blue
    pinMode(STAT2, OUTPUT); //Status LED Green

    pinMode(WSPEED, INPUT_PULLUP); // input from wind meters w
indspeed sensor
    pinMode(RAIN, INPUT_PULLUP); // input from wind meters rai
n gauge sensor

    pinMode(REFERENCE_3V3, INPUT);
    pinMode(LIGHT, INPUT);

    //Configure the pressure sensor
    myPressure.begin(); // Get sensor online
    myPressure.setModeBarometer(); // Measure pressure in Pasc
als from 20 to 110 kPa
    myPressure.setOversampleRate(7); // Set Oversample to the
recommended 128
    myPressure.enableEventFlags(); // Enable all three pressur
e and temp event flags
```

```
    //Configure the humidity sensor
    myHumidity.begin();

    seconds = 0;
    lastSecond = millis();

    // attach external interrupt pins to IRQ functions
    attachInterrupt(0, rainIRQ, FALLING);
    attachInterrupt(1, wspeedIRQ, FALLING);

    // turn on interrupts
    interrupts();

    Serial.println("Weather Shield online!");

}

void loop()
{
    //Keep track of which minute it is
  if(millis() - lastSecond >= 1000)
    {
        digitalWrite(STAT1, HIGH); //Blink stat LED

    lastSecond += 1000;

        //Take a speed and direction reading every second for
2 minute average
        if(++seconds_2m > 119) seconds_2m = 0;

        //Calc the wind speed and direction every second for 1
20 second to get 2 minute average
        float currentSpeed = get_wind_speed();
        //float currentSpeed = random(5); //For testing
        int currentDirection = get_wind_direction();
        windspdavg[seconds_2m] = (int)currentSpeed;
        winddiravg[seconds_2m] = currentDirection;
        //if(seconds_2m % 10 == 0) displayArrays(); //For test
ing

        //Check to see if this is a gust for the minute
        if(currentSpeed > windgust_10m[minutes_10m])
        {
            windgust_10m[minutes_10m] = currentSpeed;
            windgustdirection_10m[minutes_10m] = currentDirect
ion;
        }

        //Check to see if this is a gust for the day
        if(currentSpeed > windgustmph)
        {
            windgustmph = currentSpeed;
            windgustdir = currentDirection;
        }

        if(++seconds > 59)
        {
            seconds = 0;

            if(++minutes > 59) minutes = 0;
            if(++minutes_10m > 9) minutes_10m = 0;

            rainHour[minutes] = 0; //Zero out this minute's ra
infall amount
```

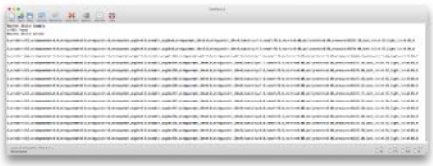
```
            windgust_10m[minutes_10m] = 0; //Zero out this min
ute's gust
        }

        //Report all readings every second
        printWeather();

        digitalWrite(STAT1, LOW); //Turn off stat LED
    }

  delay(100);
}

//Calculates each of the variables that wunderground is expect
ing
void calcWeather()
{
    //Calc winddir
    winddir = get_wind_direction();

    //Calc windspeed
    //windspeedmph = get_wind_speed(); //This is calculated i
n the main loop

    //Calc windgustmph
    //Calc windgustdir
    //These are calculated in the main loop

    //Calc windspdmph_avg2m
    float temp = 0;
    for(int i = 0 ; i < 120 ; i++)
        temp += windspdavg[i];
    temp /= 120.0;
    windspdmph_avg2m = temp;

    //Calc winddir_avg2m, Wind Direction
    //You can't just take the average. Google "mean of circula
r quantities" for more info
    //We will use the Mitsuta method because it doesn't requir
e trig functions
    //And because it sounds cool.
    //Based on: http://abelian.org/vlf/bearings.html
    //Based on: http://stackoverflow.com/questions/1813483/ave
raging-angles-again
    long sum = winddiravg[0];
    int D = winddiravg[0];
    for(int i = 1 ; i < WIND_DIR_AVG_SIZE ; i++)
    {
        int delta = winddiravg[i] - D;

        if(delta < -180)
            D += delta + 360;
        else if(delta > 180)
            D += delta - 360;
        else
            D += delta;

        sum += D;
    }
    winddir_avg2m = sum / WIND_DIR_AVG_SIZE;
    if(winddir_avg2m >= 360) winddir_avg2m -= 360;
    if(winddir_avg2m < 0) winddir_avg2m += 360;

    //Calc windgustmph_10m
    //Calc windgustdir_10m
```

```
    //Find the largest windgust in the last 10 minutes
    windgustmph_10m = 0;
    windgustdir_10m = 0;
    //Step through the 10 minutes
    for(int i = 0; i < 10 ; i++)
    {
        if(windgust_10m[i] > windgustmph_10m)
        {
            windgustmph_10m = windgust_10m[i];
            windgustdir_10m = windgustdirection_10m[i];
        }
    }

    //Calc humidity
    humidity = myHumidity.getRH();
    //float temp_h = myHumidity.readTemperature();
    //Serial.print(" TempH:");
    //Serial.print(temp_h, 2);

    //Calc tempf from pressure sensor
    tempf = myPressure.readTempF();
    //Serial.print(" TempP:");
    //Serial.print(tempf, 2);

    //Total rainfall for the day is calculated within the inte
rrupt
    //Calculate amount of rainfall for the last 60 minutes
    rainin = 0;
    for(int i = 0 ; i < 60 ; i++)
        rainin += rainHour[i];

    //Calc pressure
    pressure = myPressure.readPressure();

    //Calc dewptf

    //Calc light level
    light_lvl = get_light_level();

    //Calc battery level
    batt_lvl = get_battery_level();
}

//Returns the voltage of the light sensor based on the 3.3V ra
il
//This allows us to ignore what VCC might be (an Arduino plugg
ed into USB has VCC of 4.5 to 5.2V)
float get_light_level()
{
    float operatingVoltage = analogRead(REFERENCE_3V3);

    float lightSensor = analogRead(LIGHT);

    operatingVoltage = 3.3 / operatingVoltage; //The referenc
e voltage is 3.3V

    lightSensor = operatingVoltage * lightSensor;

    return(lightSensor);
}

//Returns the voltage of the raw pin based on the 3.3V rail
//This allows us to ignore what VCC might be (an Arduino plugg
ed into USB has VCC of 4.5 to 5.2V)
//Battery level is connected to the RAW pin on Arduino and is
```

```
fed through two 5% resistors:
//3.9K on the high side (R1), and 1K on the low side (R2)
float get_battery_level()
{
    float operatingVoltage = analogRead(REFERENCE_3V3);

    float rawVoltage = analogRead(BATT);

    operatingVoltage = 3.30 / operatingVoltage; //The referenc
e voltage is 3.3V

    rawVoltage = operatingVoltage * rawVoltage; //Convert the
0 to 1023 int to actual voltage on BATT pin

    rawVoltage *= 4.90; //(3.9k+1k)/1k - multiple BATT voltag
e by the voltage divider to get actual system voltage

    return(rawVoltage);
}

//Returns the instataneous wind speed
float get_wind_speed()
{
    float deltaTime = millis() - lastWindCheck; //750ms

    deltaTime /= 1000.0; //Covert to seconds

    float windSpeed = (float)windClicks / deltaTime; //3 / 0.7
50s = 4

    windClicks = 0; //Reset and start watching for new wind
    lastWindCheck = millis();

    windSpeed *= 1.492; //4 * 1.492 = 5.968MPH

    /* Serial.println();
     Serial.print("Windspeed:");
     Serial.println(windSpeed);*/

    return(windSpeed);
}

//Read the wind direction sensor, return heading in degrees
int get_wind_direction()
{
    unsigned int adc;

    adc = analogRead(WDIR); // get the current reading from th
e sensor

    // The following table is ADC readings for the wind direct
ion sensor output, sorted from low to high.
    // Each threshold is the midpoint between adjacent heading
s. The output is degrees for that ADC reading.
    // Note that these are not in compass degree order! See We
ather Meters datasheet for more information.

    if (adc < 380) return (113);
    if (adc < 393) return (68);
    if (adc < 414) return (90);
    if (adc < 456) return (158);
    if (adc < 508) return (135);
    if (adc < 551) return (203);
    if (adc < 615) return (180);
    if (adc < 680) return (23);
```

```
    if (adc < 746) return (45);
    if (adc < 801) return (248);
    if (adc < 833) return (225);
    if (adc < 878) return (338);
    if (adc < 913) return (0);
    if (adc < 940) return (293);
    if (adc < 967) return (315);
    if (adc < 990) return (270);
    return (-1); // error, disconnected?
}


//Prints the various variables directly to the port
//I don't like the way this function is written but Arduino do
esn't support floats under sprintf
void printWeather()
{
    calcWeather(); //Go calc all the various sensors

    Serial.println();
    Serial.print("$,winddir=");
    Serial.print(winddir);
    Serial.print(",windspeedmph=");
    Serial.print(windspeedmph, 1);
    Serial.print(",windgustmph=");
    Serial.print(windgustmph, 1);
    Serial.print(",windgustdir=");
    Serial.print(windgustdir);
    Serial.print(",windspdmph_avg2m=");
    Serial.print(windspdmph_avg2m, 1);
    Serial.print(",winddir_avg2m=");
    Serial.print(winddir_avg2m);
    Serial.print(",windgustmph_10m=");
    Serial.print(windgustmph_10m, 1);
    Serial.print(",windgustdir_10m=");
    Serial.print(windgustdir_10m);
    Serial.print(",humidity=");
    Serial.print(humidity, 1);
    Serial.print(",tempf=");
    Serial.print(tempf, 1);
    Serial.print(",rainin=");
    Serial.print(rainin, 2);
    Serial.print(",dailyrainin=");
    Serial.print(dailyrainin, 2);
    Serial.print(",pressure=");
    Serial.print(pressure, 2);
    Serial.print(",batt_lvl=");
    Serial.print(batt_lvl, 2);
    Serial.print(",light_lvl=");
    Serial.print(light_lvl, 2);
    Serial.print(",");
    Serial.println("#");

}
```

Load it onto your Arduino, and open the serial monitor at 9600. You should see output similar to the following:



Example with GPS

Shield on a RedBoard with optional weather meter connectors and GPS attached

Attach the GP-735 GPS module using the short cable. To secure the module, there is space on the shield to attach the module using double-stick tape.

Serial pins are connected to digital pins 4 and 5 when Serial is set to soft and are attached to the internal UART when set to hard.

There is a switch labeled **Serial** on the shield. This is to select which pins on the Arduino to connect the GPS to. In almost all cases the switch should be set to 'Soft'. This will attach the GPS serial pins to digital pins 5 (TX from the GPS) and 4 (RX into the GPS).

Grab the GPS example sketch from the GitHub repo that demonstrates using the GP-735 with all the other sensors. Load it onto your Arduino, and open the serial monitor at 9600.

You can also copy the code below:

```
/*
 Weather Shield Example
 By: Nathan Seidle
 SparkFun Electronics
 Date: November 16th, 2013
 License: This code is public domain but you buy me a beer if
you use this and we meet someday (Beerware license).

 Much of this is based on Mike Grusin's USB Weather Board cod
e: https://www.sparkfun.com/products/10586

 This code reads all the various sensors (wind speed, directio
n, rain gauge, humidty, pressure, light, batt_lvl)
 and reports it over the serial comm port. This can be easily
routed to an datalogger (such as OpenLog) or
 a wireless transmitter (such as Electric Imp).

 Measurements are reported once a second but windspeed and rai
n gauge are tied to interrupts that are
 calcualted at each report.

 This example code assumes the GP-735 GPS module is attached.

 Updated by Joel Bartlett
 03/02/2017
 Removed HTU21D code and replaced with Si7021
 */

#include <Wire.h> //I2C needed for sensors
#include "SparkFunMPL3115A2.h" //Pressure sensor - Search "Spa
rkFun MPL3115" and install from Library Manager
#include "SparkFun_Si7021_Breakout_Library.h" //Humidity senso
r - Search "SparkFun Si7021" and install from Library Manager
#include <SoftwareSerial.h> //Needed for GPS
#include <TinyGPS++.h> //GPS parsing - Available through the L
ibrary Manager.

TinyGPSPlus gps;

static const int RXPin = 5, TXPin = 4; //GPS is attached to pi
n 4(TX from GPS) and pin 5(RX into GPS)
SoftwareSerial ss(RXPin, TXPin);

MPL3115A2 myPressure; //Create an instance of the pressure sen
sor
Weather myHumidity;//Create an instance of the humidity sensor

//Hardware pin definitions
//-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=
// digital I/O pins
const byte WSPEED = 3;
const byte RAIN = 2;
const byte STAT1 = 7;
const byte STAT2 = 8;
const byte GPS_PWRCTL = 6; //Pulling this pin low puts GPS to
sleep but maintains RTC and RAM

// analog I/O pins
const byte REFERENCE_3V3 = A3;
const byte LIGHT = A1;
const byte BATT = A2;
const byte WDIR = A0;
//-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=
```

```
//Global Variables
//-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=
long lastSecond; //The millis counter to see when a second rol
ls by
byte seconds; //When it hits 60, increase the current minute
byte seconds_2m; //Keeps track of the "wind speed/dir avg" ove
r last 2 minutes array of data
byte minutes; //Keeps track of where we are in various arrays
of data
byte minutes_10m; //Keeps track of where we are in wind gust/d
ir over last 10 minutes array of data

long lastWindCheck = 0;
volatile long lastWindIRQ = 0;
volatile byte windClicks = 0;

//We need to keep track of the following variables:
//Wind speed/dir each update (no storage)
//Wind gust/dir over the day (no storage)
//Wind speed/dir, avg over 2 minutes (store 1 per second)
//Wind gust/dir over last 10 minutes (store 1 per minute)
//Rain over the past hour (store 1 per minute)
//Total rain over date (store one per day)

byte windspdavg[120]; //120 bytes to keep track of 2 minute av
erage
int winddiravg[120]; //120 ints to keep track of 2 minute aver
age
float windgust_10m[10]; //10 floats to keep track of 10 minut
e max
int windgustdirection_10m[10]; //10 ints to keep track of 10 m
inute max
volatile float rainHour[60]; //60 floating numbers to keep tra
ck of 60 minutes of rain

//These are all the weather values that wunderground expects:
int winddir = 0; // [0-360 instantaneous wind direction]
float windspeedmph = 0; // [mph instantaneous wind speed]
float windgustmph = 0; // [mph current wind gust, using softwa
re specific time period]
int windgustdir = 0; // [0-360 using software specific time pe
riod]
float windspdmph_avg2m = 0; // [mph 2 minute average wind spee
d mph]
int winddir_avg2m = 0; // [0-360 2 minute average wind directi
on]
float windgustmph_10m = 0; // [mph past 10 minutes wind gust m
ph ]
int windgustdir_10m = 0; // [0-360 past 10 minutes wind gust d
irection]
float humidity = 0; // [%]
float tempf = 0; // [temperature F]
float rainin = 0; // [rain inches over the past hour)] -- the
accumulated rainfall in the past 60 min
volatile float dailyrainin = 0; // [rain inches so far today i
n local time]
//float baromin = 30.03;// [barom in] - It's hard to calculat
e baromin locally, do this in the agent
float pressure = 0;
//float dewptf; // [dewpoint F] - It's hard to calculate dewpo
int locally, do this in the agent

float batt_lvl = 11.8; //[analog value from 0 to 1023]
float light_lvl = 455; //[analog value from 0 to 1023]
```

```
//Variables used for GPS
//float flat, flon; // 39.015024 -102.283608686
//unsigned long age;
//int year;
//byte month, day, hour, minute, second, hundredths;

// volatiles are subject to modification by IRQs
volatile unsigned long raintime, rainlast, raininterval, rain;

//-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=

//Interrupt routines (these are called by the hardware interru
pts, not by the main code)
//-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=
void rainIRQ()
// Count rain gauge bucket tips as they occur
// Activated by the magnet and reed switch in the rain gauge,
attached to input D2
{
  raintime = millis(); // grab current time
  raininterval = raintime - rainlast; // calculate interval be
tween this and last event

    if (raininterval > 10) // ignore switch-bounce glitches le
ss than 10mS after initial edge
  {
    dailyrainin += 0.011; //Each dump is 0.011" of water
    rainHour[minutes] += 0.011; //Increase this minute's amoun
t of rain

    rainlast = raintime; // set up for next event
  }
}

void wspeedIRQ()
// Activated by the magnet in the anemometer (2 ticks per rota
tion), attached to input D3
{
  if (millis() - lastWindIRQ > 10) // Ignore switch-bounce gli
tches less than 10ms (142MPH max reading) after the reed switc
h closes
  {
    lastWindIRQ = millis(); //Grab the current time
    windClicks++; //There is 1.492MPH for each click per secon
d.
  }
}


void setup()
{
  Serial.begin(9600);
  Serial.println("Weather Shield Example");

  ss.begin(9600); //Begin listening to GPS over software seria
l at 9600. This should be the default baud of the module.

  pinMode(STAT1, OUTPUT); //Status LED Blue
  pinMode(STAT2, OUTPUT); //Status LED Green

  pinMode(GPS_PWRCTL, OUTPUT);
  digitalWrite(GPS_PWRCTL, HIGH); //Pulling this pin low puts
GPS to sleep but maintains RTC and RAM

  pinMode(WSPEED, INPUT_PULLUP); // input from wind meters win
```

```
dspeed sensor
  pinMode(RAIN, INPUT_PULLUP); // input from wind meters rain
gauge sensor

  pinMode(REFERENCE_3V3, INPUT);
  pinMode(LIGHT, INPUT);

  //Configure the pressure sensor
  myPressure.begin(); // Get sensor online
  myPressure.setModeBarometer(); // Measure pressure in Pascal
s from 20 to 110 kPa
  myPressure.setOversampleRate(7); // Set Oversample to the re
commended 128
  myPressure.enableEventFlags(); // Enable all three pressure
and temp event flags

  //Configure the humidity sensor
  myHumidity.begin();

  seconds = 0;
  lastSecond = millis();

  // attach external interrupt pins to IRQ functions
  attachInterrupt(0, rainIRQ, FALLING);
  attachInterrupt(1, wspeedIRQ, FALLING);

  // turn on interrupts
  interrupts();

  Serial.println("Weather Shield online!");

}

void loop()
{
  //Keep track of which minute it is
  if(millis() - lastSecond >= 1000)
  {
    digitalWrite(STAT1, HIGH); //Blink stat LED

    lastSecond += 1000;

    //Take a speed and direction reading every second for 2 mi
nute average
    if(++seconds_2m > 119) seconds_2m = 0;

    //Calc the wind speed and direction every second for 120 s
econd to get 2 minute average
    float currentSpeed = get_wind_speed();
    //float currentSpeed = random(5); //For testing
    int currentDirection = get_wind_direction();
    windspdavg[seconds_2m] = (int)currentSpeed;
    winddiravg[seconds_2m] = currentDirection;
    //if(seconds_2m % 10 == 0) displayArrays(); //For testing

    //Check to see if this is a gust for the minute
    if(currentSpeed > windgust_10m[minutes_10m])
    {
      windgust_10m[minutes_10m] = currentSpeed;
      windgustdirection_10m[minutes_10m] = currentDirection;
    }

    //Check to see if this is a gust for the day
    if(currentSpeed > windgustmph)
    {
```

```
      windgustmph = currentSpeed;
      windgustdir = currentDirection;
    }

    if(++seconds > 59)
    {
      seconds = 0;

      if(++minutes > 59) minutes = 0;
      if(++minutes_10m > 9) minutes_10m = 0;

      rainHour[minutes] = 0; //Zero out this minute's rainfal
l amount
      windgust_10m[minutes_10m] = 0; //Zero out this minute's
gust
    }

    //Report all readings every second
    printWeather();

    digitalWrite(STAT1, LOW); //Turn off stat LED
  }

  //smartdelay(800); //Wait 1 second, and gather GPS data
}

//While we delay for a given amount of time, gather GPS data
static void smartdelay(unsigned long ms)
{
  unsigned long start = millis();
  do
  {
    while (ss.available())
      gps.encode(ss.read());
  } while (millis() - start < ms);
}


//Calculates each of the variables that wunderground is expect
ing
void calcWeather()
{
  //Calc winddir
  winddir = get_wind_direction();

  //Calc windspeed
  //windspeedmph = get_wind_speed(); //This is calculated in t
he main loop

  //Calc windgustmph
  //Calc windgustdir
  //Report the largest windgust today
  //windgustmph = 0;
  //windgustdir = 0;

  //Calc windspdmph_avg2m
  float temp = 0;
  for(int i = 0 ; i < 120 ; i++)
    temp += windspdavg[i];
  temp /= 120.0;
  windspdmph_avg2m = temp;

  //Calc winddir_avg2m
  temp = 0; //Can't use winddir_avg2m because it's an int
  for(int i = 0 ; i < 120 ; i++)
```

```
    temp += winddiravg[i];
  temp /= 120;
  winddir_avg2m = temp;

  //Calc windgustmph_10m
  //Calc windgustdir_10m
  //Find the largest windgust in the last 10 minutes
  windgustmph_10m = 0;
  windgustdir_10m = 0;
  //Step through the 10 minutes
  for(int i = 0; i < 10 ; i++)
  {
    if(windgust_10m[i] > windgustmph_10m)
    {
      windgustmph_10m = windgust_10m[i];
      windgustdir_10m = windgustdirection_10m[i];
    }
  }

  //Calc humidity
  humidity = myHumidity.getRH();
  //float temp_h = myHumidity.readTemperature();
  //Serial.print(" TempH:");
  //Serial.print(temp_h, 2);

  //Calc tempf from pressure sensor
  tempf = myPressure.readTempF();
  //Serial.print(" TempP:");
  //Serial.print(tempf, 2);

  //Total rainfall for the day is calculated within the interr
upt
  //Calculate amount of rainfall for the last 60 minutes
  rainin = 0;
  for(int i = 0 ; i < 60 ; i++)
    rainin += rainHour[i];

  //Calc pressure
  pressure = myPressure.readPressure();

  //Calc dewptf

  //Calc light level
  light_lvl = get_light_level();

  //Calc battery level
  batt_lvl = get_battery_level();

}

//Returns the voltage of the light sensor based on the 3.3V ra
il
//This allows us to ignore what VCC might be (an Arduino plugg
ed into USB has VCC of 4.5 to 5.2V)
float get_light_level()
{
  float operatingVoltage = analogRead(REFERENCE_3V3);

  float lightSensor = analogRead(LIGHT);

  operatingVoltage = 3.3 / operatingVoltage; //The reference v
oltage is 3.3V

  lightSensor = operatingVoltage * lightSensor;
```

```
  return(lightSensor);
}

//Returns the voltage of the raw pin based on the 3.3V rail
//This allows us to ignore what VCC might be (an Arduino plugg
ed into USB has VCC of 4.5 to 5.2V)
//Battery level is connected to the RAW pin on Arduino and is
fed through two 5% resistors:
//3.9K on the high side (R1), and 1K on the low side (R2)
float get_battery_level()
{
  float operatingVoltage = analogRead(REFERENCE_3V3);

  float rawVoltage = analogRead(BATT);

  operatingVoltage = 3.30 / operatingVoltage; //The reference
voltage is 3.3V

  rawVoltage = operatingVoltage * rawVoltage; //Convert the 0
to 1023 int to actual voltage on BATT pin

  rawVoltage *= 4.90; //(3.9k+1k)/1k - multiple BATT voltage b
y the voltage divider to get actual system voltage

  return(rawVoltage);
}

//Returns the instataneous wind speed
float get_wind_speed()
{
  float deltaTime = millis() - lastWindCheck; //750ms

  deltaTime /= 1000.0; //Covert to seconds

  float windSpeed = (float)windClicks / deltaTime; //3 / 0.750
s = 4

  windClicks = 0; //Reset and start watching for new wind
  lastWindCheck = millis();

  windSpeed *= 1.492; //4 * 1.492 = 5.968MPH

  /* Serial.println();
   Serial.print("Windspeed:");
   Serial.println(windSpeed);*/

  return(windSpeed);
}

//Read the wind direction sensor, return heading in degrees
int get_wind_direction()
{
  unsigned int adc;

  adc = analogRead(WDIR); // get the current reading from the
sensor

  // The following table is ADC readings for the wind directio
n sensor output, sorted from low to high.
  // Each threshold is the midpoint between adjacent heading
s. The output is degrees for that ADC reading.
  // Note that these are not in compass degree order! See Weat
her Meters datasheet for more information.

  if (adc < 380) return (113);
```

```
  if (adc < 393) return (68);
  if (adc < 414) return (90);
  if (adc < 456) return (158);
  if (adc < 508) return (135);
  if (adc < 551) return (203);
  if (adc < 615) return (180);
  if (adc < 680) return (23);
  if (adc < 746) return (45);
  if (adc < 801) return (248);
  if (adc < 833) return (225);
  if (adc < 878) return (338);
  if (adc < 913) return (0);
  if (adc < 940) return (293);
  if (adc < 967) return (315);
  if (adc < 990) return (270);
  return (-1); // error, disconnected?
}


//Prints the various variables directly to the port
//I don't like the way this function is written but Arduino do
esn't support floats under sprintf
void printWeather()
{
  calcWeather(); //Go calc all the various sensors

  Serial.println();
  Serial.print("$,winddir=");
  Serial.print(winddir);
  Serial.print(",windspeedmph=");
  Serial.print(windspeedmph, 1);
  /*Serial.print(",windgustmph=");
  Serial.print(windgustmph, 1);
  Serial.print(",windgustdir=");
  Serial.print(windgustdir);
  Serial.print(",windspdmph_avg2m=");
  Serial.print(windspdmph_avg2m, 1);
  Serial.print(",winddir_avg2m=");
  Serial.print(winddir_avg2m);
  Serial.print(",windgustmph_10m=");
  Serial.print(windgustmph_10m, 1);
  Serial.print(",windgustdir_10m=");
  Serial.print(windgustdir_10m);*/
  Serial.print(",humidity=");
  Serial.print(humidity, 1);
  Serial.print(",tempf=");
  Serial.print(tempf, 1);
  Serial.print(",rainin=");
  Serial.print(rainin, 2);
  Serial.print(",dailyrainin=");
  Serial.print(dailyrainin, 2);
  Serial.print(",pressure=");
  Serial.print(pressure, 2);
  Serial.print(",batt_lvl=");
  Serial.print(batt_lvl, 2);
  Serial.print(",light_lvl=");
  Serial.print(light_lvl, 2);

  Serial.print(",lat=");
  Serial.print(gps.location.lat(), 6);
  Serial.print(",lat=");
  Serial.print(gps.location.lng(), 6);
  Serial.print(",altitude=");
  Serial.print(gps.altitude.meters());
  Serial.print(",sats=");
```

```
  Serial.print(gps.satellites.value());

  char sz[32];
  Serial.print(",date=");
  sprintf(sz, "%02d/%02d/%02d", gps.date.month(), gps.date.day
(), gps.date.year());
  Serial.print(sz);

  Serial.print(",time=");
  sprintf(sz, "%02d:%02d:%02d", gps.time.hour(), gps.time.minu
te(), gps.time.second());
  Serial.print(sz);

  Serial.print(",");
  Serial.println("#");

}
```

You should see output similar to the following:

Note: The `batt_lvl` is indicating 4.08V. This is correct and is the actual voltage read from the Arduino powered over USB. The GPS module will add 50-80mA to the overall power consumption. If you are using a long or thin USB cable you may see significant voltage drop similar to this example. There is absolutely no harm in this! The Weather Shield runs at 3.3V and the Arduino will continue to run just fine down to about 3V. The reading is very helpful for monitoring your power source (USB, battery, solar, etc).

This example demonstrates how you can get location, altitude, and time from the GPS module. This would be helpful with weather stations that are moving such as balloon satellites, AVL, package tracking, and even static stations where you need to know precise altitude or timestamps.

Resources and Going Further

The Weather Shield example firmware outputs regular barometric pressure. This is very different from the pressure that weather stations report. For more information, see the definition of “altimeter setting pressure”. For an example of how to calculate altimeter setting type barometric pressure see the MPL3115A2 hook-up guide. Also check out the MPL3115A2 library, specifically the `BarometricHgInch` example.

Datasheets

There’s a lot of technology on this shield. Here’s the datasheets in case you need to reference them:

- Si7021 Temperature and Humidity
- MPL3115A2 Pressure
- ALS-PT19 Light
- GP-735 GPS
- Weather Meters

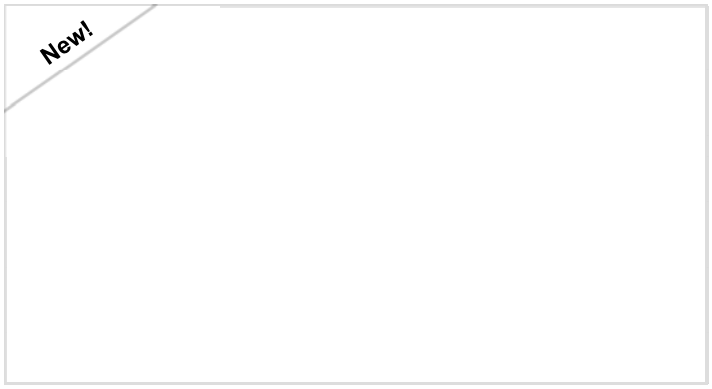
Additional resources and projects to check out:

- Si7021 Repo and Library

- MPL3115A2 Pressure Repo and Library
- If you're interested in using GPS with Arduino definitely check out Mikal Hart's TinyGPS++ library
- Consider adding an OpenLog for datalogging the weather readings over time.
- The Photon from Particle is a good way to add WiFi to get a truly wireless weather station. We also sell an Arduino-shaped version of the Photon, the SparkFun Photon RedBoard. This Weather Shield is compatible with that board as well.

The Arduino Weather Shield attached to a Photon RedBoard.

For more Internet-connected weather fun, check out our Photon Weather Shield Hookup Guide.



Photon Weather Shield Hookup Guide V11

MARCH 2, 2017

Create Internet-connected weather projects with the SparkFun Weather Shield for the Photon.