

APÉNDICE K

```
#define __SIM900_H__
#define __GPRS_SHIELD_ARDUINO_H__
#include <Wire.h> //I2C needed for sensors
#include "SparkFunMPL3115A2.h" //Pressure sensor - Search "SparkFun MPL3115" and
install from Library Manager
#include "SparkFunHTU21D.h" //Humidity sensor - Search "SparkFun HTU21D" and
install from Library Manager
```

```
MPL3115A2 myPressure; //Create an instance of the pressure sensor
HTU21D 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 rolls by
```

```
byte seconds; //When it hits 60, increase the current minute
```

```
byte seconds_2m; //Keeps track of the "wind speed/dir avg" over 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/dir 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 average

#define WIND_DIR_AVG_SIZE 120
int winddiravg[WIND_DIR_AVG_SIZE]; //120 ints to keep track of 2 minute average
float windgust_10m[10]; //10 floats to keep track of 10 minute max
int windgustdirection_10m[10]; //10 ints to keep track of 10 minute max
volatile float rainHour[60]; //60 floating numbers to keep track 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 software specific time period]
int windgustdir = 0; // [0-360 using software specific time period]
float windspdmpm_avg2m = 0; // [mph 2 minute average wind speed mph]
int winddir_avg2m = 0; // [0-360 2 minute average wind direction]
float windgustmph_10m = 0; // [mph past 10 minutes wind gust mph ]
int windgustdir_10m = 0; // [0-360 past 10 minutes wind gust direction]
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 in local time]
//float baromin = 30.03; // [barom in] - It's hard to calculate baromin locally, do this in the
agent
float pressure = 0;
//float dewptf; // [dewpoint F] - It's hard to calculate dewpoint 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;

//-----

#include <SoftwareSerial.h>
SoftwareSerial gprsSerial(10,11);
int sensorMetano;
int sensorHidrogeno;
int sensorMonoxido;
float temperatura;
float htierra;

void setup()
{

```

```

gprsSerial.begin(19200);
Serial.begin(19200);

Serial.println("Config SIM900...");
delay(15000);
Serial.println("Done!...");
gprsSerial.flush();
Serial.flush();

//Verificaci3n funcionamiento SIM 900
gprsSerial.println("AT");
delay(100);
toSerial();

//Devuelve el estado de calidad de la se3al de cobertura
gprsSerial.println("AT+CSQ");
delay(100);
toSerial();

// attach or detach from GPRS service
gprsSerial.println("AT+CGATT?");
delay(100);
toSerial();

// bearer settings
gprsSerial.println("AT+SAPBR=3,1,\"CTYPE\",\"GPRS\"");
delay(2000);
toSerial();

// bearer settings
//Conexi3n al servidor y enviar los dos datos requeridos por el m3todo GET a la
base de datos.
gprsSerial.println("AT+SAPBR=3,1,\"APN\",\"internet.movistar.com.co\"");
delay(2000);
toSerial();

// bearer settings
gprsSerial.println("AT+SAPBR=1,1");
delay(2000);
toSerial();

gprsSerial.println("AT+SAPBR=2,1");
delay(2000);
toSerial();

//*****Placa Sensores

```

```

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

pinMode(WSPPEED, INPUT_PULLUP); // input from wind meters windspeed 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 Pascals from 20 to 110 kPa
myPressure.setOversampleRate(7); // Set Oversample to the recommended 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();

//*****Fin Placa
}

void loop()
{
//***** Placa sensores
//Keep track of which minute it is
if(millis() - lastSecond >= 1000)
{
digitalWrite(STAT1, LOW); //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 120 second to get 2 minute average
float currentSpeed = get_wind_speed();
windspeedmph = currentSpeed;
//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 rainfall 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
}

//***** Fin Placa sensores

// initialize http service
gprsSerial.println("AT+HTTPINIT");
delay(2000);
toSerial();

gprsSerial.println("AT+HTTPPARA=\"CID\",1");

```

```

delay(2000);
toSerial();

// set http param value
String cadena =
"AT+HTTTPARA=\"URL\", \"http://www.telecoteam.com/estacion/index.php?Estacion=";
int Estacion=1;
sensorMetano = analogRead(A15);
sensorHidrogeno = analogRead(A14);
sensorMonoxido = analogRead(A13);
delay(1000);
htierra=analogRead(A12);
temperatura=(5.0*(analogRead(A11))*100)/(1023);

cadena = cadena + Estacion;
cadena = cadena + "&Metano=";
cadena = cadena + sensorMetano;
cadena = cadena + "&Hidrogeno=";
cadena = cadena + sensorHidrogeno;
cadena = cadena + "&Monoxido=";
cadena = cadena + sensorMonoxido;
cadena = cadena + "&DirViento=";
cadena = cadena + winddir;
cadena = cadena + "&VelocidadViento=";
cadena = cadena + windspeedmph;
cadena = cadena + "&VelocidadRafaga=";
cadena = cadena + windgustmph;
cadena = cadena + "&DirRafaga=";
cadena = cadena + windgustdir;
cadena = cadena + "&Humedad=";
cadena = cadena + htierra;
cadena = cadena + "&Temperatura=";
cadena = cadena + temperatura;
cadena = cadena + "&Lluvia=";
cadena = cadena + rainin;
cadena = cadena + "&Presion=";
cadena = cadena + pressure;
cadena = cadena + "&Bateria=";
cadena = cadena + batt_lvl;
cadena = cadena + "&Luz=";
cadena = cadena + light_lvl;
cadena = cadena + "\"";
gprsSerial.println(cadena);
delay(2000);
toSerial();

// set http action type 0 = GET, 1 = POST, 2 = HEAD

```

```

gprsSerial.println("AT+HTTPACTION=0");
delay(6000);
toSerial();

// read server response
gprsSerial.println("AT+HTTPREAD");
delay(1000);
toSerial();

gprsSerial.println("");
gprsSerial.println("AT+HTTPTERM");
toSerial();
delay(300);

gprsSerial.println("");
delay(10000);
}

void toSerial()
{
  while(gprsSerial.available() != 0)
  {
    Serial.write(gprsSerial.read());
  }
}

//Interrupt routines (these are called by the hardware interrupts, 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 less than 10mS after initial edge
  {
    dailyrainin += 0.011; //Each dump is 0.011" of water
    rainHour[minutes] += 0.011; //Increase this minute's amount of rain

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

void wspeedIRQ()
// Activated by the magnet in the anemometer (2 ticks per rotation), attached to input D3

```

```

{
  if (millis() - lastWindIRQ > 10) // Ignore switch-bounce glitches less than 10ms (142MPH
max reading) after the reed switch closes
  {
    lastWindIRQ = millis(); //Grab the current time
    windClicks++; //There is 1.492MPH for each click per second.
  }
}

```

//Calculates each of the variables that wunderground is expecting

void calcWeather()

```

{
  //Calc winddir
  winddir = get_wind_direction();

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

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

  //Calc windspdmp_h_avg2m
  float temp = 0;
  for(int i = 0 ; i < 120 ; i++)
    temp += windspdavg[i];
  temp /= 120.0;
  windspdmp_h_avg2m = temp;

  //Calc winddir_avg2m, Wind Direction
  //You can't just take the average. Google "mean of circular quantities" for more info
  //We will use the Mitsuta method because it doesn't require trig functions
  //And because it sounds cool.
  //Based on: http://abelian.org/vlf/bearings.html
  //Based on: http://stackoverflow.com/questions/1813483/averaging-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.readHumidity();
//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 interrupt
//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 rail
//This allows us to ignore what VCC might be (an Arduino plugged 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 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 plugged 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 by the voltage divider to get
actual system voltage

    return(rawVoltage);
}

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

```

```

deltaTime /= 1000.0; //Covert to seconds

float windSpeed = (float)windClicks / deltaTime; //3 / 0.750s = 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 direction sensor output, sorted from
    low to high.
    // Each threshold is the midpoint between adjacent headings. The output is degrees for that
    ADC reading.
    // Note that these are not in compass degree order! See Weather 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 doesn'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(htierra, 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("#");
}

```

