Building an Arduino based weather station and connecting it as a slave to a control system

Frida Bäfverfeldt
Uppsala University, Uppsala, Sweden
Contents

1 Introduction 3

2 Hardware 4
  2.1 Arduino UNO .................................................. 4
  2.2 LM35 Temperature Sensor ...................................... 4
  2.3 DHT22 Humidity Sensor ....................................... 5
  2.4 BMP280 Barometric Pressure Sensor .......................... 6
  2.5 HC06 Bluetooth Module ...................................... 7
  2.6 The weather station .......................................... 7

3 Software 8
  3.1 Arduino IDE .................................................. 8
  3.2 PuTTY ...................................................... 11
  3.3 MATLAB .................................................... 12
  3.4 EPICS ...................................................... 13

4 Results 15

5 Conclusion 16

6 Acknowledgements 17

7 References 17

8 Appendix 19
  8.1 Circuit diagram .............................................. 19
  8.2 Serial protocol command list ................................ 19
  8.3 Arduino code ................................................ 20
  8.4 MATLAB code ............................................... 22
  8.5 EPICS code ................................................ 24
Building an Arduino based weather station and connecting it as a slave to a control system

Frida Bäverfeldt
Supervisor: Volker Ziemann
Department of Physics and Astronomy
Uppsala University, Uppsala, Sweden

September 27, 2016

Abstract

In this report the process of building an Arduino based weather station is described. The station is then connected to a control system, first one made in MATLAB for testing and later to the EPICS control system in the FREIA laboratory at Uppsala university. The measurements made by the weather station are compared to reference data and found to be consistent.

1 Introduction

When doing a physical experiment, it is important to keep track of the factors that may influence ones measurements. The weather is one such possible factor. In this project we build an Arduino based device for measurements of ambient air temperature, pressure and humidity.

The weather station is to be used at the FREIA laboratory at Uppsala University. The FREIA laboratory’s main research consists of testing equipment for particle accelerators. This is mostly done in high-vacuum pressure chambers which are sensitive to ambient air pressure, why this is one important feature to measure.

Building this device included designing and planning what components to be used as well as wiring and soldering everything together into a working device. Then the microcontroller was programmed to be able to talk to our chosen sensors and to communicate outwards through a serial line.

In this report we are going to take a look at which sensors were used and how they were connected to our microcontroller. The program uploaded to the microcontroller will also be explained as well as how to get MATLAB and EPICS to communicate with the weather station.
2 Hardware

Here we briefly describe the components.

2.1 Arduino UNO

![Arduino UNO board](https://www.arduino.cc/en/Guide/Windows)

Figure 1: The Arduino UNO r3 board
image source: https://www.arduino.cc
/en/Guide/Windows

As a base for our device we need a microcontroller to be able to collect data from our sensors and we chose to work with Arduino since it is user friendly and has an active community with loads of free codes and help to find when needed[10]. In this project we have worked with the Arduino UNO r3 board which is based on an ATmega328p 8-bit microcontroller. The board has an internal clock frequency of 16 MHz, a 32 KB flash memory and number of both analog and digital pins. It operates at 5V and is easiest powered via USB which also provides a serial line for communication and programming from a computer[2].

In the finished product we chose to use just the ATmega328p processor flashed with an Arduino boot loader and internal 8MHz oscillator instead of the whole board, this made the device smaller and easier to handle. For most of the programming and testing we used the UNO r3 board.

2.2 LM35 Temperature Sensor

The LM35 is a small and inexpensive analog temperature sensor. It has three pins, one for data output and the other two for ground and supply voltage respectively. The middle pin, for analog data, returns a voltage between 0 and 1 volt which then can be converted to a temperature by the microcontroller [3].

The measurements from this sensor are quite unstable and the voltage output shifts
around the real value quite a lot and very fast. To get a more accurate value we decided to make 10 readings with short intervals and calculate the mean value of them. This stabilised the readings.

The voltage output from the LM35 is linear to the temperature of the surroundings. The microcontroller reads the voltage output from the sensor and converts this to the related temperature. The Arduino looks for voltages between 0 and 5 volts and maps the input voltages to integer values 0 to 1023 which sets a limit on the resolution of the sensor. However, the sensor only return values up to 1 volt we can get our resolution to be 5 times higher just by telling our microcontroller to look for voltages in the correct range. This is easily done using the function analogReference() and setting it to INTERNAL, but we discuss the software more in the next chapter.

### 2.3 DHT22 Humidity Sensor

The DHT22 sensor measures both relative humidity and temperature. The sensor is
protected by a plastic case which makes it a bit bigger (26.7×58.8mm) than the other sensors, but like the LM35 this module has one data output line as well as for voltage and ground. However this sensor is digital and not analog as the LM35 and communicates with the microcontroller using a special protocol used for the DHT sensors which is described in the datasheet. It operates on 3.3 to 5.5V and can measure temperatures between -40°C - 80°C with a ±0.5°C accuracy and humidity 0-100%RH (Relative Humidity) with a ± 2% accuracy [4].

We started off with the DHT11 sensor but decided to instead go with the newer DHT22 during the project since the later one has higher resolution.

2.4 BMP280 Barometric Pressure Sensor


The BMP280 is a combined temperature and pressure sensor made by Bosch. It can communicate in both I2C and SPI, we went with I2C since it is simpler and adequate for our purpose. I2C is a standard communication protocol that allows one or more slave chips to communicate with one or more master chips over short distances, e.g. inside a single device.

The breakout board has 7 pins, for I2C wiring we only need four; voltage, ground and the two logical pins SCK and SDI. The SCK pin is the clock pin which is connected to our ATmega238p’s clock line and the SDI is the data pin, for I2C wiring this line handles both data input and output.

The sensor operates at 3.3V but the breakout board includes a 3.3V regulator so there is no problem connecting it to our microcontroller’s 5V standard.

It measures pressure with ±1 hPa absolute accuracy, and temperature with ±1.0°C accuracy[5].
2.5 HC06 Bluetooth Module

![Figure 5: The HC06 module.](image)


For our device to communicate with a computer it uses a serial communication line, which is standard on an Arduino board. Since we wanted our device to be easy to move to different experiment locations within FREIA, we equipped it with a Bluetooth module that enables wireless serial communication. However, this module transfers data too slowly for it to be convenient to upload new programs via it, and more importantly misses the needed reset line making it impossible to begin with. For uploading we instead used a USB serial adapter (FTDI UB232R) and the Bluetooth module only for transferring data when the program is already up and running.

2.6 The weather station

![Figure 6: The complete device](image)

Once we decided which sensors, microcontroller, etc to use it was time to make that pile of components into an actual device. It took some planning to figure out what was best to put where to minimize the soldering work and to get an as compact device as
possible. After not too much work we managed to fit everything quite nicely together. The circuit diagram for our device can be found in the appendix.

The LM35 is as earlier stated an analog sensor and needed to be connected to one of the ATmega’s analog pins, in our case A0 which is pin no 23 at the ATmega 328p. DHT22 is a digital sensor and was connected to digital pin 4, no 6 at the ATmega. The BMP280 was connected to the clock(SCL) and data(SDA) pins, located at pin 28 and 27 at the ATmega. Last but not least the HC06, connected to the RX and TX pins, pin 2 and 3. Notice that RX on the ATmega needs to be connected to TX on the HC06 and vice versa. All of the sensors were then connected to ground and 5V supply voltage. There is also a capacitor connected in parallel to each of the sensors to smooth out current fluctuations.

Optional add-ons are a reset circuitry and an equivalent to the Arduino board LED. The fist consisting of a 10 kΩ pull up resistor and 100 nF capacitor connected to the reset pin, pin no1. This is needed for uploading new programs to the microprocessor. The LED is a handy tool for diagnostics and for this circuit you need a 220 Ω resistor and a LED connected in series from ATmega pin 19 to the power rails. Neither of these are needed when the device is up and running but are practical until you get there, an option is to have an external circuitry on a breadboard just for uploading programs and to move the microcontroller between these two when needed [7].

Notice that the device needs an external power source to operate unless powered via USB.

3 Software

3.1 Arduino IDE

To program our Arduino microcontroller we used the Arduino coding language. The Arduino language is based on C/C++ and the most basic executable program only needs two functions, a setup() and a loop(), to run. In the setup() function variables, pin modes, serial communication, etc are initialized. This function only runs once. The loop() function is where you write your actual code. As the name suggests, the loop() loops and it loops continuously until your device is powered off [12].

Simple as it may sound, it possible to write rather complex programs only using above described structure. In fact, the program running on our weather station, collecting data from the different sensors, is written in this exact form.

Arduino provides a free and easy to use integrated development environment. This tool was developed to help introduce programming to people not used to software development, which made it perfect for us and our project. The Arduino IDE largely consists of a code editor and a compiler but also has some nifty features which makes it easy
for the user to upload programs to their devices and to read input sent back from the device trough a serial communication port [11].

The sensors we used had prewritten libraries for Arduino communication, these are possible to download from the internet. Some of the sensors have more than one library, the ones that we used for this project are; DHT [13] and Adafruit_BMP280 [14], the BMP library also requires access to Wire (comes preinstalled with Arduino IDE), SPI (comes preinstalled with Arduino IDE) and Adafruit_Sensors [17]. In the Arduino language you only need to specify the header files to get access to the libraries. There are libraries for the LM35 sensor as well but we decided to instead read the output voltage using the function analogRead(), which is built into the Arduino language, and then calculate the associated temperature.

The loop() starts with a few lines of code that are looking for input, if there are no user input the device will not do anything but wait.

```
if (Serial.available() > 0) {
    s = Serial.readStringUntil(\'\n\');
    char charBuf[50];
    s.toCharArray(charBuf, 50);
    :
}
```

The rest of the code interprets the input and figure out what sensor to read a value from. If the input is not supported, nothing will happen. There is a full list of supported commands in the appendix. The lines of code that read measurements are quite similar for all the sensors, however there are some slight differences that are worth noticing.
The code for collecting data from the BMP280 is the one most straight ahead.

```c
if (strstr(charBuf, "TP?")) {
    Serial.print("TP ");
    Serial.println((float)bme.readTemperature());
} else if (strstr(charBuf, "P?")) {
    Serial.print("P ");
    Serial.println((float)bme.readPressure());
}
```

It simply checks what data the user is interested in and answer with an echo of the request (TP or T) and associated measurement. bme.readTemperature() and bme.readPressure() are functions defined in the BMP library.

The code for DHT22 is almost the same, but since this sensor is somewhat unstable there is also a check that makes sure the measurement actually was successful, if not it returns a humidity of -1 or a temperature of 1000. Similar as for the BMP, the functions dht.readHumidity() and dht.readTemperature() are defined in the DHT library.

```c
if (strstr(charBuf, "TH?")) {
    tempDHT = dht.readTemperature();
    Serial.print("TH ");
    //Unstable sensor, check reading
    //return -1 if reading failed
    if (isnan(tempDHT)) {
        Serial.println((float)(1000));
    } else {
        Serial.println(tempDHT);
    }
} else if (strstr(charBuf, "H?")) {
    humidityDHT = dht.readHumidity();
    Serial.print("H ");
    //Unstable sensor, check reading
    //return -1 if reading failed
    if (isnan(humidityDHT)) {
        Serial.println((float)(-1));
    } else {
        Serial.println(humidityDHT);
    }
}
```

Last but not least, the LM35. As mentioned in chapter 2.1 readings from the LM tends
to shift fast around the actual value, to stabilize the readings we do ten measurements within a short time interval and then calculating the average. There is also a function that compensates for change in the internal reference voltage (more about this also in chapter 2.1) and for change in value when going from the output voltage to associated temperature. This simply turns out to be a scaling factor of $\frac{1}{9.31}$, which should be understood as a relation between the output voltage and ambient temperature; a change of 9.31 mV represents a change of 1 °C. This number is calculated by finding what percentage of the range (1024) the reading makes up, times the range itself. analogReference(INTERNAL) sets our range to 0 - 1.1V. Then divide 10mV by this number (According to the datasheet analog output changes 10 mV per degree Celsius) [8]

```c
if (strstr(charBuf, "T")) {
    //Read temp from LM sensor 10 times
    //and calqulate mean to get stabilized reading
    sum = 0;
    for (int i = 0; i < 10 ; i++) {
        reading = analogRead(tempPin);
        delay(20);
        //Function compensating for change in internalReference
        //and for going from a voltage to associated temperature
        rescaledReading = reading / 9.31;
        sum = sum + rescaledReading;
    }
    tempLM = sum / 10;
    Serial.print("T ");
    Serial.println(tempLM);
}
```

The complete code can be found in the appendix.

### 3.2 PuTTy

PuTTy is a serial console that we used to test our devices wireless communication. The program does in principal the same thing as the built in serial communication window in the Arduino IDE but we wanted to make sure everything worked fine outside Arduinos own enviroment which was what we had been working in up unttill now. Communicating with our device worked just as fine here as in the Arduino IDE, both wireless and through a USB cable.
3.3 MATLAB

We wanted our device to function as a slave to a control system, and also to be able to display the values graphically for prolonged measurements. For these two tasks MATLAB is a great tool.

We wrote a script that once a minute asks the weatherstation for measurements using our serial communication protocol. To get access to the serial line via MATLAB we follow [1]. First three functions needed to be defined:

One that creates a serial object.

```
function serialObj = setupSerial(comPort, baudRate)
    %% Initializes serial port communication between Arduino and MATLAB
    serialObj = serial(comPort);
    set(serialObj, 'DataBits',8);
    set(serialObj, 'StopBits',1);
    set(serialObj, 'BaudRate',baudRate);
    set(serialObj, 'Parity','none');
end
```

One for resetting the serial port and one for closing it.

```
function [] = resetSerial(SerialObj)
    %% Resets the serial input buffer for object obj
    fclose(SerialObj);
    fopen(SerialObj);
end
```

```
function [] = closeSerial()
    %% Close all serial ports
    if ~isempty(instrfind)
        fclose(instrfind);
        delete(instrfind);
    end
end
```

Using the timestamp function in MATLAB, `clock()`, we then made a loop that ask the
weather station for measurements once a minute from defined start time to defined trop time. The values are then saved in separate arrays, three for the different temperature readings, one for humidity and one for pressure.

Example of reading and saving measurements in the loop.

```matlab
fprintf(serialObj, 'H?');
H(i)=str2num(fscanf(serialObj, '%c'));
```

The communication protocol and complete MATLAB code can be found in the appendix.

Then we used the plotting tool to get a graphical overview of how our measurements from the different sensors varied in time.

![Figure 8: Test measurements of the DHT11 and LM35 sensors during 7 hours](image)

Above graph is one of the first test runs we did. This features the LM35 sensor and the DHT11 that we later changed for the newer DHT22 sensor. The big jumps in the DHT11’s measurements are due to the sensor’s low resolution.

### 3.4 EPICS

EPICS (Experimental Physics and Industrial Control System) is an open source software environment for development and management of control systems [9]. For the EPICS system in FREIA to be able to communicate with our weather station it needs access to a protocol file that specifies what commands to be sent to the device and what to expect in return.
# weather2.proto file

Terminator = CR LF;
ReplyTimeout = 2000;

get_pressure {
  out "P?";
  in "P %f"
  ExtraInput=Ignore;
}

Our device returns values a bit slower than standard, to wait longer for returned values
ReplyTimeout is set to 2 sek.

The protocol file gets called from a database file. This is also where the records are
defined, such as what kind of device is used ("stream" in our case) and how often to
fetch new values. Explained in more detail in [1].

# weather2.db file

record(ai, "$(DEVNAM):Pressure") {
  field(DESC,"Pressure in mbar")
  field(SCAN,"10 second")
  field(DTYP,"stream")
  field(INP,"@weather2.proto get_pressure $(PORT)")
}

Database files and proto files for all of the sensors can be found in the appendix.
4 Results

Figure 9 shows the values graphically from two hours of measurements in freia.

Figure 9: Three graphs showing change in temperature, humidity and pressure during two hours. Temperature measured by LM35, DHT22 and BMP280, humidity measured by DHT22 and pressure by BMP280

Figure 10 to 12 shows results from FREIAs EPICS control system. These graphs shows data both from our weather station and reference data from FREIA.

Figure 10: Pressure in FREIA; blue line is data from our BMP280 sensor, red line is reference data
As seen, the DHT22 sensor sometimes returns very high temperatures and low humidity values, this occurs when the microcontroller fails to read from the sensor (further explained in chapter 3.1). We can also see that the temperature readings differ somewhat between the sensors in the weather station and the reference data. This is also the case for our humidity readings. However, the temperature readings differs less than 1 °C and the humidity readings less than 5%, this is within the acceptable margin of error for our purpose. Readings from the pressure sensor are very close to the pressure reference data.

It is possible to calibrate the sensors for more accurate readings if needed.

5 Conclusion

We built a weather station that measures temperature, relative humidity and air pressure. The station was programmed to read values from the sensor when a user asks for them. The device was then connected as a slave to MATLAB (as a simulated control system) and to the EPICS control system in the FREIA laboratory. Our measurements from the weather station are consistent with reference measurements from FREIA.
6 Acknowledgements

I would like to express my sincere gratitude to my supervisor Dr. Volker Ziemann for his guidance, patience and support in this project, for sharing his immense knowledge and for inspiring me to take my first steps into the world of electronics.

Furthermore, my dear friend Ebba Fogelström, thank you for your contributions to the project and constant support. 💖

Many thanks also to Måns Holmberg & Adam Hjort for your help in understanding and for fruitful discussions.

7 References

URN: urn:nbn:se:uu:diva-257545


[14] https://github.com/adafruit/Adafruit_BMP280_Library

8 Appendix

8.1 Circuit diagram

![Circuit Diagram](image)

Figure 13

8.2 Serial protocol command list

<table>
<thead>
<tr>
<th>Command</th>
<th>Response</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>T?</td>
<td>T x</td>
<td>Temperature in °C, measured by LM35 sensor</td>
</tr>
<tr>
<td>TH?</td>
<td>TH x</td>
<td>Temperature in °C, measured by DHT22 sensor</td>
</tr>
<tr>
<td>TP?</td>
<td>TP x</td>
<td>Temperature in °C, measured by BMP280 sensor</td>
</tr>
<tr>
<td>H?</td>
<td>H x</td>
<td>Relative humidity in %, measured by DHT22 sensor</td>
</tr>
<tr>
<td>P?</td>
<td>P x</td>
<td>Pressure in Pa, measured by BMP280 sensor</td>
</tr>
</tbody>
</table>

Where x is the numerical value returned by the sensor.
8.3 Arduino code

---

String s;

//LM
int tempPin = A0;
int reading;
float rescaledReading;
float sum;
float tempLM;

//DHT
#include "DHT.h"
#define DHTPIN 4
#define DHTTYPE DHT22
DHT dht(DHTPIN, DHTTYPE);
float humidityDHT;
float tempDHT;

//BMP
#include <Wire.h>
#include <SPI.h>
#include <Adafruit_Sensor.h>
#include <Adafruit_BMP280.h>
#define BMP_SCK 13
#define BMP_MISO 12
#define BMP_MOSI 11
#define BMP_CS 10
Adafruit_BMP280 bme;

void setup() {
  Serial.begin(9600);
  analogReference(INTERNAL);
  dht.begin();
  if (!bme.begin()) {
    Serial.println("Could not find a valid BMP280 sensor, check wiring!");
    while (1);
  }
}
void loop() {

// send data only when you receive data:
if (Serial.available() > 0) {
  s = Serial.readStringUntil(’/n’);
  char charBuf[50];
  s.toCharArray(charBuf, 50);

  //what did user ask for? -> return echo and reading

  //LM
  if (strstr(charBuf, "T?")) {
    //Read temp from LM sensor 10 times
    //and calulate mean to get stabilized reading
    sum = 0;
    for (int i = 0; i < 10 ; i++ ) {
      reading = analogRead(tempPin);
      delay(20);
      //Rescaling due to changed analog reference
      rescaledReading = reading / 9.31;
      sum = sum + rescaledReading;
    }
    tempLM = sum / 10;
    Serial.print("T ");
    Serial.println(tempLM);
  }

  //DHT
  if (strstr(charBuf, "TH?")) {
    tempDHT = dht.readTemperature();
    Serial.print("TH ");
    //Unstable sensor, check reading
    //return -1 if reading failed
    if (isnan(tempDHT)) {
      Serial.println((float)(1000));
    } else {
      Serial.println(tempDHT);
    }
  } else if (strstr(charBuf, "H?")) {
    humidityDHT = dht.readHumidity();
    Serial.print("H ");
    //Unstable sensor, check reading
  }
}
}
//return -1 if reading failed
if (isnan(humidityDHT)) {
    Serial.println((float)(-1));
} else {
    Serial.println(humidityDHT);
}

//BMP
if (strstr(charBuf, "TP?")) {
    Serial.print("TP ");
    Serial.println((float)bme.readTemperature());
} else if (strstr(charBuf, "P?")) {
    Serial.print("P ");
    Serial.println((float)bme.readPressure() / 100);
}
}

-------------------------------

8.4 MATLAB code

-------------------------------

function [] = closeSerial()
% Close all serial ports

if ~isempty(instrfind)
    fclose(instrfind);
    delete(instrfind);
end
end

-------------------------------

function [] = resetSerial(SerialObj)
% Reset the serial input buffer for object obj

fclose(SerialObj);
fopen(SerialObj);
end
function serialObj = setupSerial(comPort, baudRate)
    %% Initializes serial port communication between Arduino and MATLAB
    serialObj = serial(comPort);
    set(serialObj, 'DataBits', 8);
    set(serialObj, 'StopBits', 1);
    set(serialObj, 'BaudRate', baudRate);
    set(serialObj, 'Parity', 'none');
end

clear all;
close all;
start = [2016 9 19 10 0 0]; % define date and time for start measuring
stop = [2016 9 19 12 0 0]; % define data and time for stop measuring

closeSerial;
% create a serial obj at chosen sreial port and baud rate
serialObj = setupSerial('COM7', 9600);
resetSerial(serialObj);

minutes = etime(stop, start)/60; % calculate nr of minuts the script will be running
TH = zeros(1, length(minutes));
T = zeros(1, length(minutes));
H = zeros(1, length(minutes));
TP = zeros(1, length(minutes));
P = zeros(1, length(minutes));
i = 0;

% run loop from defined start time to defined stop time
while ((etime(fix(clock), start)>-5) && (etime(fix(clock), stop)<0))
    c = fix(clock) % creates new array with current date and time
    if c(6) == 0 % when second==0 in date time array, start messuring
        i = i + 1;
        fprintf(serialObj, 'TP?');
        inputTP = strsplit(fscanf(serialObj, '%c'));
        TP(i) = str2num(inputTP{2});
    end
end

23
fprintf(serialObj, 'H?');
inputH=strsplit(fscanf(serialObj, '%c'));
    H(i)=str2num(inputH(2));

fprintf(serialObj, 'TH?');
inputTH = strsplit(fscanf(serialObj, '%c'));
    TH(i)=str2num(inputTH(2));

fprintf(serialObj, 'T?');
inputT = strsplit(fscanf(serialObj, '%c'));
    T(i)=str2num(inputT(2));

fprintf(serialObj, 'P?');
inputP = strsplit(fscanf(serialObj, '%c'));
    P(i)=str2num(inputP(2));

pause(5);
end
end

8.5 EPICS code

# weather2.proto file

Terminator = CR LF;

generate_pressure {  
    out "P?";  
    in "P %f";  
    ExtraInput=Ignore;  
}
generate_humidity {  
    out "H?";  
    in "H %f";  
    ExtraInput=Ignore;  
}
generate_temp {  
    out "T?";  
    in "T %f";  
    ExtraInput=Ignore;  
}
get_tempHumidity {
    out "TH?";
    in "TH %f";
    ExtraInput=Ignore;
}

get_tempPressure {
    out "TP?";
    in "TP %f";
    ExtraInput=Ignore;
}

# weather2.db file

record(ai, "$(DEVNAM):Pressure") {
    field(DESC,"Pressure in mbar")
    field(SCAN,"10 second")
    field(DTYP,"stream")
    field(INP,"@weather2.proto get_pressure $(PORT)")
}

record(ai, "$(DEVNAM):Humidity") {
    field(DESC,"Relative humidity in %")
    field(SCAN,"10 second")
    field(DTYP,"stream")
    field(INP,"@weather2.proto get_humidity $(PORT)")
}

record(ai, "$(DEVNAM):Temp") {
    field(DESC,"Temperature in degrees celcius")
    field(SCAN,"10 second")
    field(DTYP,"stream")
    field(INP,"@weather2.proto get_temp $(PORT)")
}

record(ai, "$(DEVNAM):TempHumidity") {
    field(DESC,"Temperature from humidity sensor, in degrees celcius")
    field(SCAN,"10 second")
    field(DTYP,"stream")
}
record(ai, "$(DEVNAM):TempPressure") {
    field(Desc, "Temperature from pressure sensor, in degrees celcius")
    field(SCAN, "10 second")
    field(DTYP, "stream")
    field(INP, "@weather2.proto get_tempPressure $(PORT)")
}