

WeatherDuino Logger for WeatherDuino Pro2 Plus

by engolling

17.02.2019

Get all the data captured by the WeatherDuino Pro2 Plus system and its modules to an additional serial interface with full precision for logging or further processing.

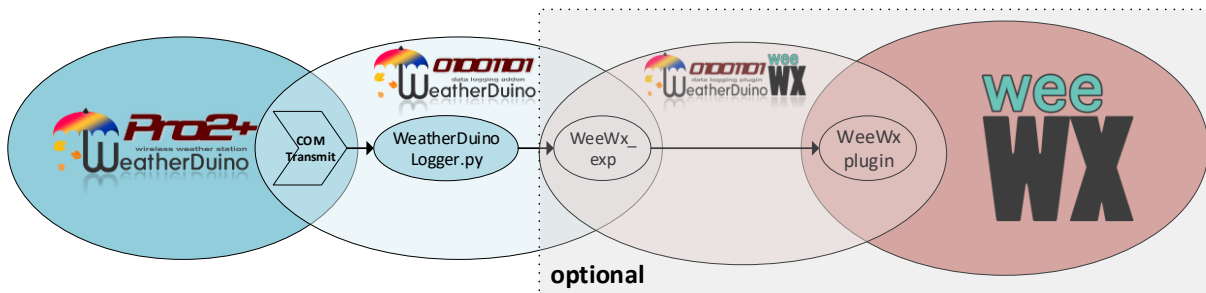
- The sampling rate is one sample per minute.
- Reports via E-Mail if data transmission stops.
- Data is exported in metric units.

This modification does not work on the compact receiver since it has no additional free UART interfaces.

Since the data is written to the serial 3 interface of the meduino there is a corresponding logging script written in python since it is platform independent.

This script logs all selected data to CSV. The script can also export selected signals to a text file. From there you can import additional weather data like the snow height into WeeWx weather software.

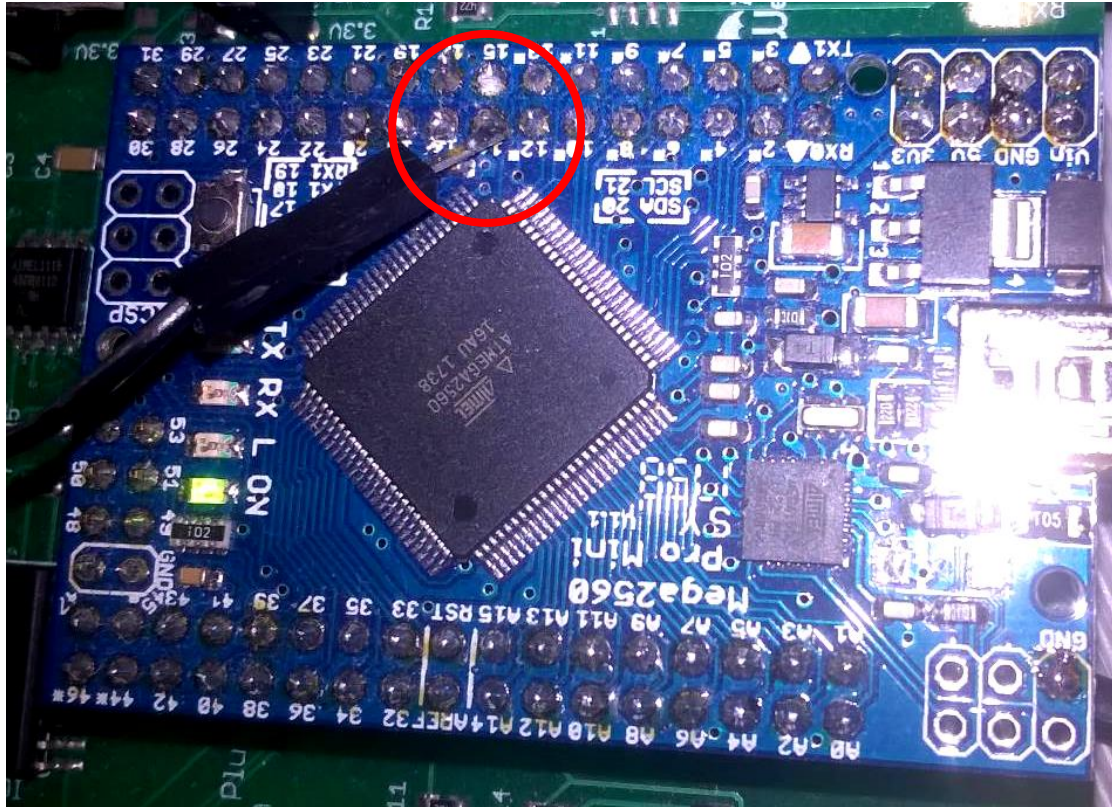
See: <https://www.meteocercal.info/forum/Thread-WeeWx-extra-data-plugin>



Hardware setup

First, the communication wire has to be soldered to pin 14 of the Meduino, carrying the UART3. We only need pin 14 (TX) since this will be a one-way communication.

Take care to get a proper ground connection; you can get this from one of the I2C connectors on the RX board.



Be aware that the output logic level is at 5V and hence not suitable for direct communication with a PC or a Raspberry Pi. You may use some level shifters to obtain a proper function.

The baud rate of the receiving device has to be set to 115200. Data bits 8, stop bit 1 and parity none.

Additional setup for use with a Raspberry Pi Zero W

I suggest using a Raspberry Pi Zero W for data logging and running the provided logging script. In this case you can use the Raspberry to run a weather software like WeeWx or Cumulus and additionally log all of your captured data in full precision or import some extra data into WeeWx.

Attention: Since the 5V linear regulator on the Meduino gets warm when the Raspberry Zero W is connected, I do not recommend supplying the Raspberry any longer via the WeatherDuino.

Be aware that the ground potentials of the Raspberry and the WeatherDuino have to be connected for a working communication.

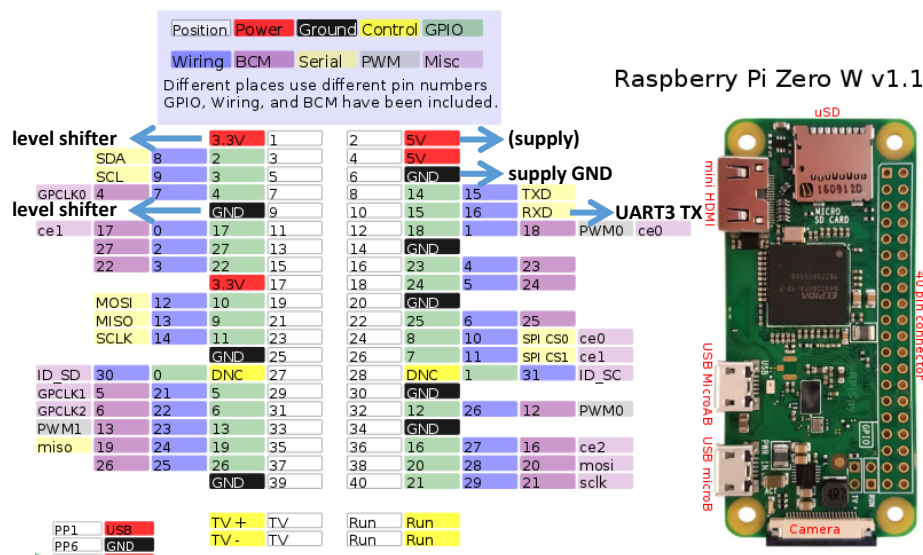
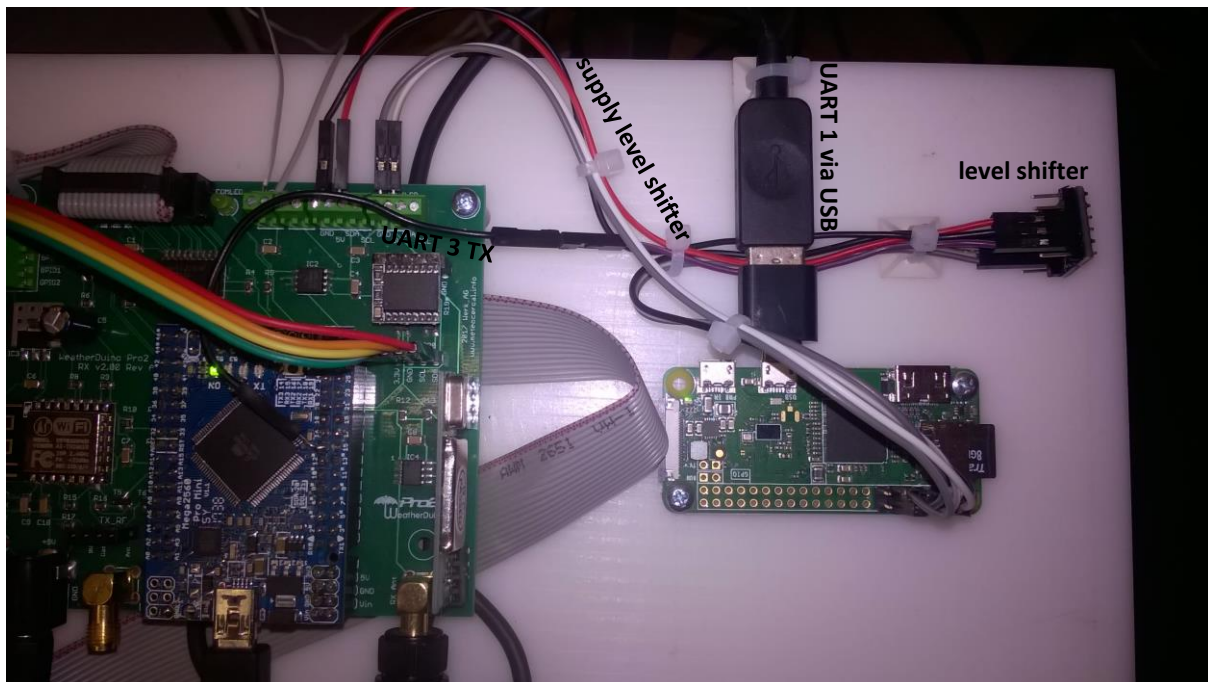


Image source: https://cdn.sparkfun.com/assets/learn_tutorials/6/7/6/PiZero_1.pdf by Sparkfun, retrieved on 27.08.2018

I used a bidirectional level shifter based on a BSS138, those can be found in almost any online shop. Since the level conversion has only to be unidirectional, a resistor divider or the usage of a zener diode would also be possible.

Software setup

The original WeatherDuino source code has to be modified as following in order to retrieve all the stored sensor data each minute.

First init serial 3 on the Meduino:

In WeatherDuino_P2AT2560_RX_[Version].ino line 453 find:

```
//-----  
//  Setup Things  
//-----  
void setup()  
{  
  Serial.begin(19200);
```

Change to:

```
//-----  
//  Setup Things  
//-----  
void setup()  
{  
  Serial.begin(19200);  
  Serial3.begin(115200);
```

Next, insert the sending routine in the 1-minute timer:

In Routines.ino line 333 find:

```
float conv_factor = 1.0;
#if (RAIN_DISPLAY_UNIT == 1)
  conv_factor = 0.03937;
#endif
Rain_LastHour = RainTips_LastHour * (COLLECTOR_TYPE[RAIN_SOURCE] / 1000.0) * conv_factor;
//Serial.print("Rain Last Hour :"); Serial.println(Rain_LastHour);
Rain_Last24Hours = RainTips_Last24Hours * (COLLECTOR_TYPE[RAIN_SOURCE] / 1000.0) *
conv_factor;
//Serial.print("Rain last 24 hours :"); Serial.println(Rain_Last24Hours);
```

Change to:

```
float conv_factor = 1.0;
#if (RAIN_DISPLAY_UNIT == 1)
  conv_factor = 0.03937;
#endif
Rain_LastHour = RainTips_LastHour * (COLLECTOR_TYPE[RAIN_SOURCE] / 1000.0) * conv_factor;
//Serial.print("Rain Last Hour :"); Serial.println(Rain_LastHour);
Rain_Last24Hours = RainTips_Last24Hours * (COLLECTOR_TYPE[RAIN_SOURCE] / 1000.0) *
conv_factor;
//Serial.print("Rain last 24 hours :"); Serial.println(Rain_Last24Hours);

// --- Transmit raw data via Serial 3
Com_Transmit();
```


Modify the receiving routine that wind values of each TX-Module are buffered:

In RX_TX.ino find line 137 – 157:

```
if (TX_UnitID == WIND_SOURCE)
{
  uint32_t gust;
  gust = RX_Data[2];
  if (gust <= 67000)                                // - First step Spike Removal - Process if Gust <= 241 Km/h
  {
    #if (WIND_SPIKECONTROL == 1)
      static uint32_t last_gust;
      uint16_t spike_limit;
      if (last_gust < 11111 ) spike_limit = 8333; // < 40 Km/h -> Spike Limit 30 Km/h
      else if (last_gust < 18055 ) spike_limit = 15277; // < 65 Km/h -> Spike Limit 55 Km/h
      else spike_limit = 50000;                // > 65 Km/h -> Spike Limit 180 Km/h
      if (gust > last_gust + spike_limit) gust = last_gust;
      last_gust = gust;
    #endif

    RX_WindPackets = RX_WindPackets + 1;

    TX_Unit[TX_UnitID].Wind_average = RX_Data[1];
    TX_Unit[TX_UnitID].Wind_gust = gust;
    TX_Unit[TX_UnitID].Wind_dir = RX_Data[3];
  }
}
```

Change to:

```
uint32_t gust;
gust = RX_Data[2];
if (gust <= 67000)                                // - First step Spike Removal - Process if Gust <= 241 Km/h
{
  #if (WIND_SPIKECONTROL == 1)
    static uint32_t last_gust;
    uint16_t spike_limit;
    if (last_gust < 11111 ) spike_limit = 8333; // < 40 Km/h -> Spike Limit 30 Km/h
    else if (last_gust < 18055 ) spike_limit = 15277; // < 65 Km/h -> Spike Limit 55 Km/h
    else spike_limit = 50000;                // > 65 Km/h -> Spike Limit 180 Km/h
    if (gust > last_gust + spike_limit) gust = last_gust;
    last_gust = gust;
  #endif

  TX_Unit[TX_UnitID].Wind_average = RX_Data[1];
  TX_Unit[TX_UnitID].Wind_gust = gust;
  TX_Unit[TX_UnitID].Wind_dir = RX_Data[3];

  if (TX_UnitID == WIND_SOURCE)
  {
    RX_WindPackets = RX_WindPackets + 1;
  }
}
```

Modify the receiving routine that rain values of each TX-Module are buffered:

In **RX_TX.ino** find line **228 – 237**:

```
#if (RAIN_SOURCE != 3)
  if (TX_UnitID == RAIN_SOURCE)
  {
    COLLECTOR_TYPE[TX_UnitID] = RX_Data[1];      // Collector tip value * 1000
    TX_Unit[TX_UnitID].Rainftipshour = RX_Data[2];
    TX_Unit[TX_UnitID].TotalRain_tips = RX_Data[3];
    //Serial.print("TotalRain_tips :"); Serial.println(TX_Unit[TX_UnitID].TotalRain_tips);

    MainSensors_lastTimeRX[2] = now_millis;
    bitWrite(MainSensorsRX, 2, 1);
```

Change to:

```
#if (RAIN_SOURCE != 3)
  COLLECTOR_TYPE[TX_UnitID] = RX_Data[1];      // Collector tip value * 1000
  TX_Unit[TX_UnitID].Rainftipshour = RX_Data[2];
  TX_Unit[TX_UnitID].TotalRain_tips = RX_Data[3];
  //Serial.print("TotalRain_tips :"); Serial.println(TX_Unit[TX_UnitID].TotalRain_tips);

  if (TX_UnitID == RAIN_SOURCE)
  {
    MainSensors_lastTimeRX[2] = now_millis;
    bitWrite(MainSensorsRX, 2, 1);
```

Modify the receiving routine that the temperature and humidity values of the AQM are buffered:

In **RX_TX.ino** line **417 - 420** find:

```
case 12:
{
  AQI_Monitor.GAS_1 = RX_Data[3]; // GAS_1 from AQI Monitor
  AQI_Monitor.GAS_2 = RX_Data[4]; // GAS_1 from AQI Monitor
```

Change to:

```
case 12:
{
  AQI_Monitor.GAS_1 = RX_Data[3]; // GAS_1 from AQI Monitor
  AQI_Monitor.GAS_2 = RX_Data[4]; // GAS_1 from AQI Monitor
  AQI_Monitor.AQI_Temp = RX_Data[1]/10; // Temp from AQI Monitor
  AQI_Monitor.AQI_Hum = RX_Data[2]/10; // Humidity from AQI Monitor
```

to provide temperature and humidity data in your extra logging.

In **data_structs.h** line **49** find:

```
uint16_t AQI_Temp;
```

Change to:

```
int16_t AQI_Temp;
```

Finally copy the new provided file “COM_Transmit_v.ino” into the project folder. This file contains the sending function that starts a data stream according to the enclosed pattern. The data layout can be found in the file “Transmission_Layout_[Version]”.

Logging script setup

The python script “Weatherduino_Logger_.py” can display and log all of the WeatherDuino Pro2 Plus data to a *.csv file. This data can be easily displayed with any csv viewer. Moreover, it can generate a dataset for import into WeeWx.

The logging scripts runs on Python 2.7 with the [pytz](#) library installed.

In the upper part of the logging script you can make some settings for debugging, logging and mail notifications.

Be sure the path to the “Transmission_Layout_xx” file is correct. I had some trouble with relative paths, so I suggest to use an absolute path pointing to that file.

The main settings concerning signal preparation and logging are made in the “Transmission_Layout_” csv file. The columns separator must be ‘;’.

Line 1	Groups the origin where the data was collected at the WeatherDuino
Line 2	Names the signals referring to the names in the WeatherDuino software
Line 3	Space to give an alias name to be written in the log file. This overwrites the Names given in line 2
Line 4	By writing a ‘1’ in the field, logging is enabled for the corresponding signal
Line 5	Names the signal for WeeWx export. This name overwrites the signal names given in line 2 and maybe overwritten in line 3
Line 6	By writing a ‘1’ in the field, WeeWx export is enabled for the corresponding signal
Line 7	Names the unit group for WeeWx
Line 8	Defines the variable type, as it is defined in the WeatherDuino software. This is needed for encoding the serial data string.
Line 9	The factor to divide the integer value coming from the WeatherDuino to get a decimal
Line 10	The unit of the signal to be plotted in the header of the csv log file

You should only change line 3 - 6 unless you know what you are doing.

Since WeeWx does not like signal names with a space ‘ ’ in it I suggest not to use spaces in any of the names in the Transmission_Layout file. Better use underscores ‘_’ instead.

To make the script working properly it should run all the time since it captures the serial data sent by the WeatherDuino, decodes it and logs it or stores it in the export file for WeeWx.

Integration of additional data

The layout file can be extended after the termination signal of the WeatherDuino with additional input signal (see Figure 1). These variable types has to be ‘import’ and the data has to be written in a vector called extra data in the corresponding section of the logging script. This can be found around line 466 in the python code.

In case of the shown example the list extraData must contain three items with the corresponding signal values for Input1, Input2 and Input3 at the end.

	ByteSum	CRC	END			
WiFi_H3	ByteSum	CRC_8	Termination	Input1	Input2	Input3
0	0	0	0	0	1	1
group_percent						
int16_t	uint16_t	uint8_t	uint16_t	import	import	import
0	10	1	1	1	1	1
% r.H.						

Figure 1: Extension of layout file

Integration of calculation data

Like the 'import' signals you can add some extra signals of type 'calc' to the layout file. You can do the necessary calculations in the script around line 523. A automatic formula parser might be added in future.

Automatic start of logging script

In case of using a linux machine I suggest using a cron job to start the script.

In a terminal window type `crontab -e`

Add the following command at the bottom of the file and exit saving it:

`@reboot /usr/bin/python /home/pi/WeatherDuino/WeatherDuino_Logger.py &`

You have to adjust the paths that it fits your requirements.

I suggest using absolute paths, since my cron job did not work correctly when using relative path.

After a reboot, the script should run properly in the background. You can check the log file and the error log if there is actual data.