Raspberry Pi with sensors for temperature and humidity

Version 1.0 *(this help file) for* ***Weather Station 2023*** *(desktop app v1.0)*

Image 1: Final status of this project: 1: Raspberry Pi 2B, 2: GPIOs, 3: circuit board connected to five DHT22 sensors, 4: external SSD with the root operating system, 5: USB hub with own power supply, 6: USB drive 8GB for the swap partition of the operating system, 7: DHT22 sensor inside the arena of a Messor barbarus *formicarium (sensor 2 which connects to GPIO (2))*

Content

1 Introduction

In this project a Raspberry Pi will be used to collect temperature and humidity sensor data for several formicariums, which are basically terrariums for ants, but monitoring of any other species is supported. The basic idea is to monitor several of these sensors, even if they are in the same room. But of course, instead of a “species” “room names” could be used to monitor a house or a building.

The project is accompanied by a video playlist on YouTube (in German).

Playlist

It is also available as Open Source Project on Github .

2 Project

This project is divided in several steps which could be called milestones. The section of this document contains these milestones which need to completed in the following order:

1. Soldering the circuit board
2. Connect the circuit board to the RasPi
3. Setup the objects to be monitored (Setup of a formicarium / terrarium)
4. Setup of the Raspberry Pi to collect sensor data[1]
5. Setup the Desktop app to display data visually (Weather Station 2023)

3 Soldering the circuit board

This part of the project is highly depending on what someone wants to achieve. It depends on the RasPi that will be used as well as the number of sensors that should be used. During this project a circuit board for **five** sensors will be used, but the maximum number is limited by the connection pins (GPIOs[2]) of the RasPi model used.

3.1 Item list

The following was needed for five sensors:

- 5× DHT22 sensors
- 5× 10kΩ resistors
- 1× circuit board (28×6 holes)
- Silver wire
- 7× female plugs (out to RasPi)
- 5× 3pin-male (input for the sensors)

1 The “Raspberry Pi” will in the following be abbreviated as “RasPi”
2 GPIO: General Purpose Input Output

- Heat shrink tube
- Wire: 3 different colors (e.g. red, orange, black), 10m each

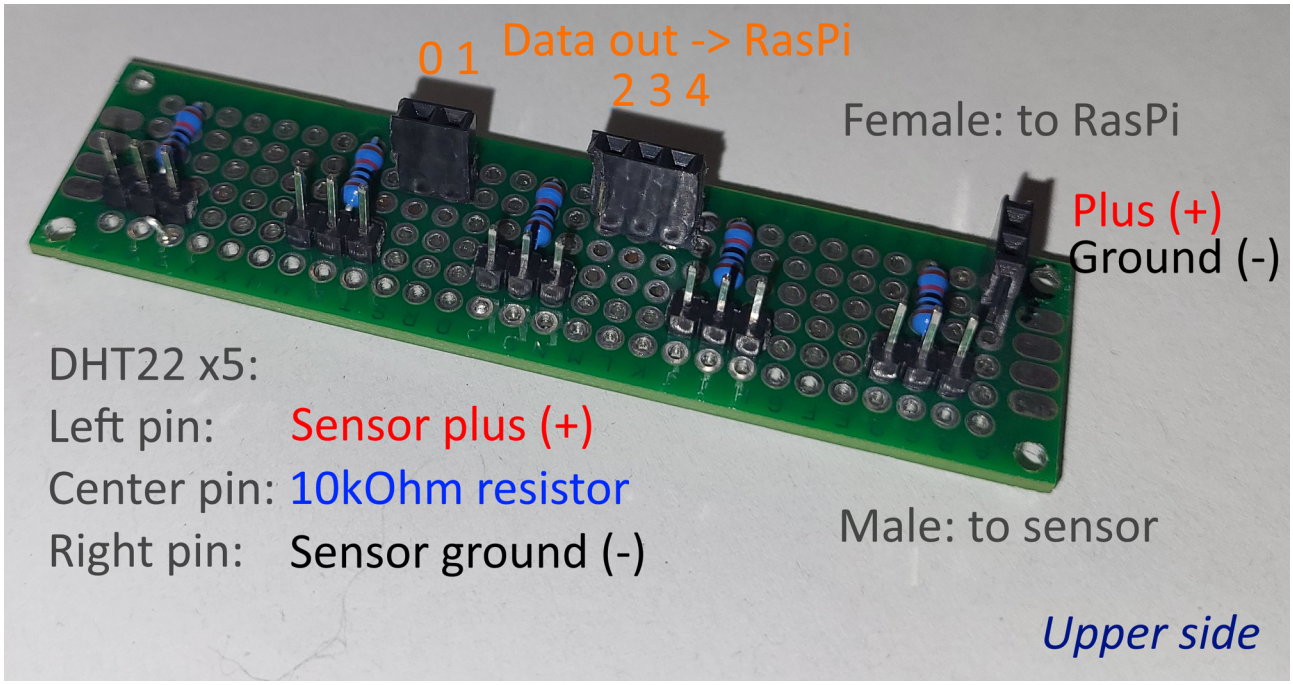


Image 3.1: The upper side of the soldered circuit board.

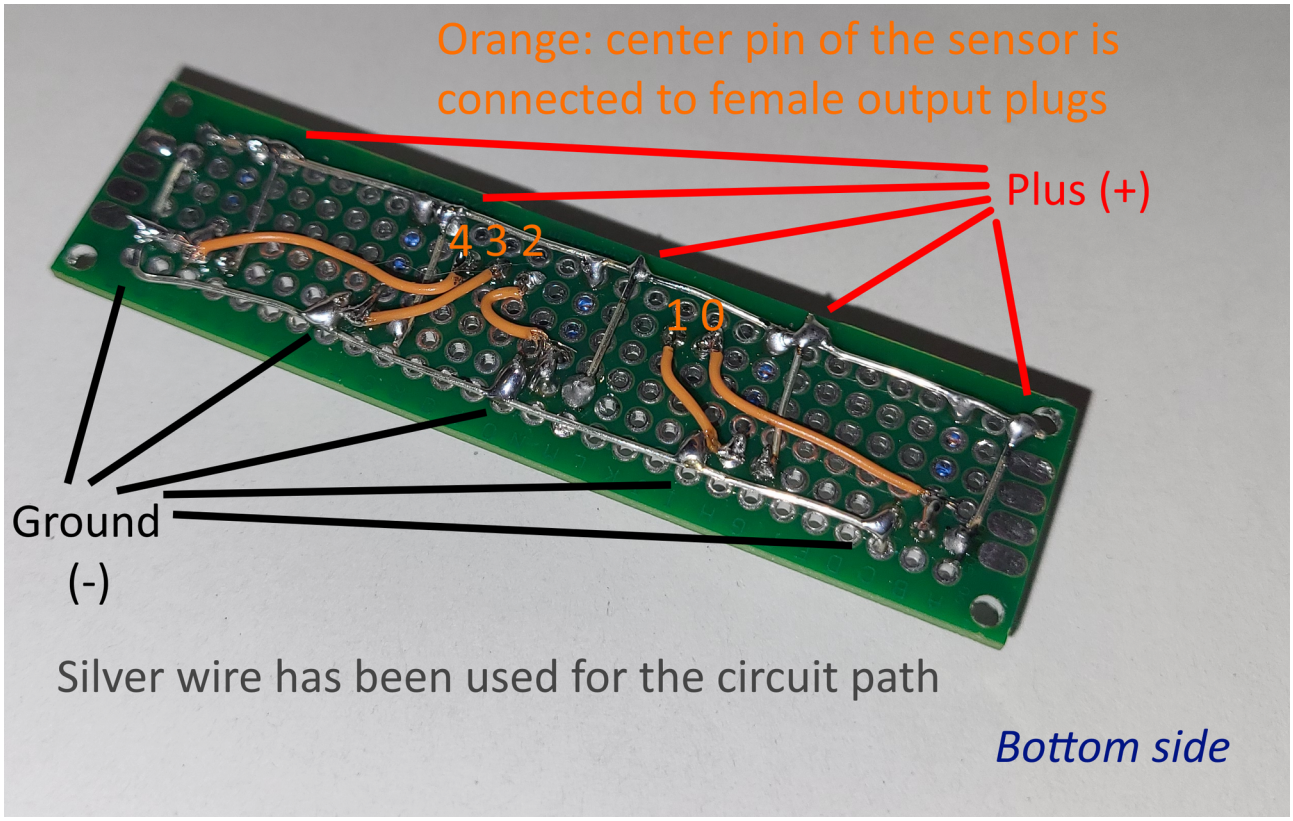


Image 3.2: The bottom side of the soldered circuit board.

3.2 Connect the circuit board to the RasPi

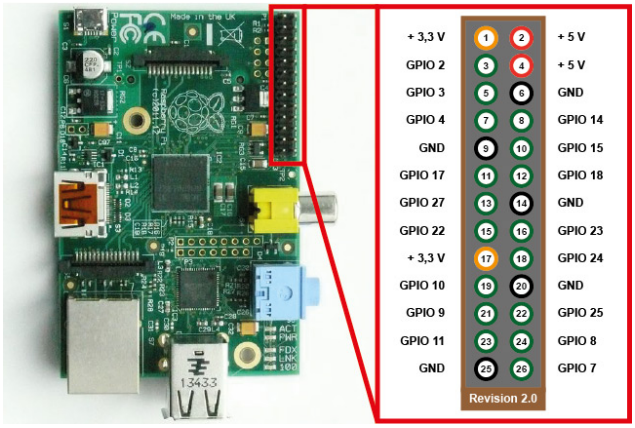


*Image 3.3: Layout of the GPIOs of the RasPi 1B (2011). Pin (1): **+3.3V**, pin (3): data sensor **2**, pin (5): data sensor **3**, pin (6): **GND**, pin (7): data sensor **4**, pin (24): data sensor **8**, pin (26): data sensor **7***

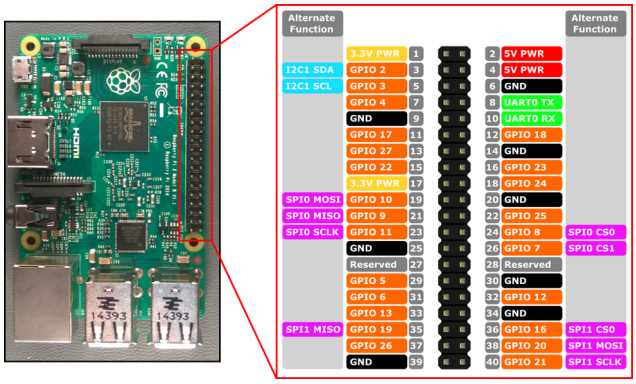


Image 3.4: Layout of the GPIOs of the RasPi 2B (2014). Although the newer version has more GPIOs, the same pins as in the older version can be used.

In this project a very old RasPi[3] was used for development and the GPIO layout might change for different RasPis or their clones. So it is very important to check the pin layout carefully.

As the layout of the circuit board needs only one +3.3V (1) and one GND (6) using a circuit board saves a couple of connections as normally each sensor would need its own +3.3V and GND lines.

E.g. in this help file the **data pins** (orange) used are the following and thus will be shown in screen-shots for the desktop application as examples:

- GPIO 2 → Sensor **2**
- GPIO 3 → Sensor **3**
- GPIO 4 → Sensor **4**
- GPIO 7 → Sensor **7**
- GPIO 8 → Sensor **8**

Please note that the name of the GPIO is different to the pins used on the RasPi!

3 Actually, the project was started with a RasPi 1B (2011) with 512MB RAM and single-core CPU. For the final version of the project a RasPi 2B with 1GB RAM and quad-core CPU is used which seems to be the minimum requirement.

4 Sensors

The requirement for each sensor is that it could be used between 3 and 5V and that it has got three outputs[4]. In this project DHT22-sensors are used, but any sensor that fulfills the requirements should work.

Each sensor owns three output lines which are soldered to the wires. The length of the wires is depending on the location of the RasPi and the location of the object that needs to be monitored.

- **GND -** (black)
- **OUT** (data, yellow/orange)
- **PLUS +** (red)

Image 4.1: Pins of the DHT22

Image 4.2: A small hole in the cap that seals the holes in the upper side of the formicarium allows for enough space that the sensor is actually in the formicarium, but ants or other small animals can not escape. This is a closeup of the setup (sensor 2) used on the title page of this document.

4 There are variants of the DHT22 sensor that has got four pins. But only three lines are used, so make sure to solder the wires to the connected lines **plus**, **out**, **minus**.

5 Setup of a formicarium / terrarium

A “formicarium” is a special type of terrarium. They are used in this example for the sensors, but any type of terrarium is supported. Therefore a species name can be defined for each monitoring object. The wires of the sensor should be long enough to connect both, the formicarium and the circuit board/RasPi.

There are too many options depending on the individual setup, therefore this can not be discussed in this document. See the photos in this document for an example setup.

Feel free to use the comments below the videos of the YouTube Playlist to ask directly for help in case of uncertainties.

6 Setup of the Raspberry Pi[5]

6.1 Headless installation via WiFi

The RasPi can be installed without monitor via SSH-client. PuTTY can be used to access the RasPi which is obviously a Linux system[6]. There are a lot of installation guides on the internet which might be different according to the version used.

In this project a working RasPi that can be accessed via (W)LAN is a prerequisite and will not be discussed further.

For useful links, please check chapter 10.1 of this document.

6.2 User configuration

The user “**root**” is only to be used to setup the RasPi, but to access the database via a desktop application the user “**ant_admin**” is created. Ideally, this user should only have read rights while only the DHT2SQL.py can write to the database. Python scripts run as user “**root**”, but are restricted to the host “**localhost**” (127.0.0.1), thus “**root**” can not connect via (W)LAN for security reasons.

Follow this guide to setup MariaDB for LAN access and user creation:

https://mariadb.com/kb/en/configuring-mariadb-for-remote-client-access/

6.3 Database configuration

After MariaDB has been installed, the table which stores sensor data needs to be created.

5 This can be totally different for other versions of the RasPi or its clones. Use this is an exemplary guide which steps are needed.

6 I’m writing this as a Windows user. Linux users might laugh about me and could do it better, but everything in this project was done on a Windows 11 laptop, while the RasPi is controlled via Putty.

This step is *critical* and the database columns are used for Weather Station 2023 to display sensor data (Database setup). Whenever a column name is different to these defaults, it should be considered throughout this help file.

Also, DHT2SQL.py refers to these database columns.

```
MariaDB [(none)]> create table antstation.raw_data (
    -> id int auto_increment primary key,
    -> temp float not null,
    -> hygro float not null,
    -> sensor_id int not null,
    -> created_at timestamp);
Query OK, 0 rows affected (0.065 sec)
```

Image 6.1: Creation of the MariaDB table to store the sensor data.

6.4 DHT2SQL.py

This script is used to write data from the sensors via Cron job (6.5) to a MariaDB, thus it needs to have database access as well. The script can be found in *Sources*, but the database connection should be changed. Otherwise the script will not be working.

DHT2SQL.py

In this script the SQL connection needs to be changed to the user "**root**" of the RasPi.

```
233  #ANTSTATION SQL
234  try:
235      conn = mariadb.connect(
236          user="root",           # We're using root as this script runs only on localhost
237          password="password",   # I'm using a different pwd, of course ;)
238          host="127.0.0.1",      # localhost, no access from outside
239          port=3306,             # same port as from outside
240          database="antstation"  # Ants.Cologne: in this db we're storing our sensor data!
241      )
242  except mariadb.Error as e:
243      print(f"Error connecting to MariaDB Platform: {e}")
244      sys.exit(1)
245
246  # Get Cursor
247  cur = conn.cursor()
248  #print("SQL works")
```

Image 6.2: SQL section of DHT2SQL.py which needs to be changed to own settings.

6.5 Cron job

Cron is a job scheduler for Unix-like operating systems, so it's already available and can be used when the script used has been tested successfully. pigpio / pigpio Daemon must also be started via Cron.

```
> sudo crontab -e
```

The following lines are added:

```
*/10[7] * * * * python3 /home/pi/DHT2SQL.py 2 3 4 7 8[8]
@reboot sudo pigpiod
```

https://en.wikipedia.org/wiki/Cron

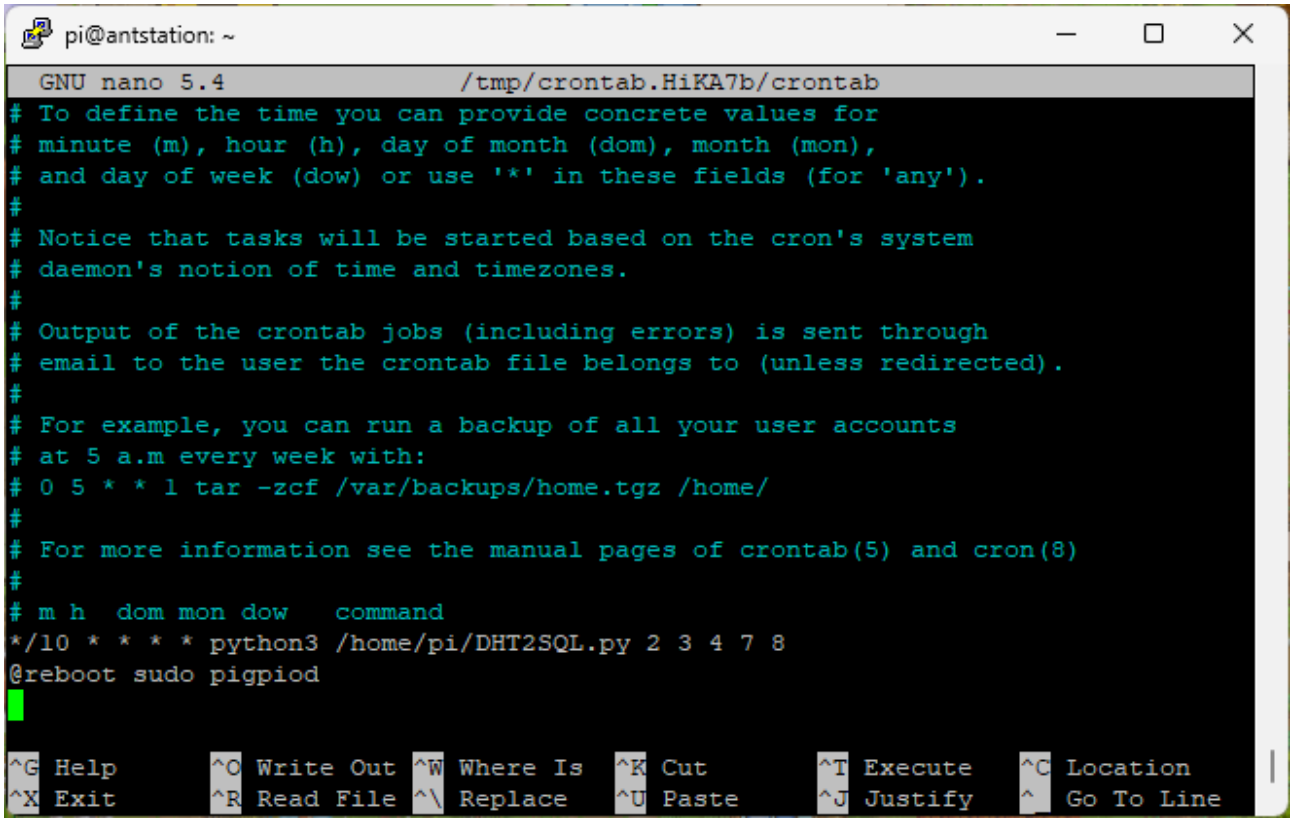


Image 6.3: crontab *should look like this when everything is set up.*

6.6 Moving the database to an external Solid State Drive[9]

As the RasPi is used 24/7 as a database server, changing from the default SD-card to an SSD is highly recommended. The SD-card will then only be used for booting while all the load of the database is handled by the SSD which should also result in less latency of common operations (read, write, access via (W)LAN).

But the main reason is, that the first version of this project (Ant Station 1.0, 2021) only worked for 15 months: then the SD-card has crashed, all data was lost and research on the internet showed that SD-cards are not for operating system tasks, but designed and optimized for multimedia usage.

For this project the external SSD was connected to an external USB hub with its own power supply. This is critical, as the used RasPi can not supply a drive with its own power which could result in making the RasPi inaccessible in the network.

7 Every 10 minutes: change for different intervals.
8 Five sensors are used on GPIOs 2, 3, 4, 7, and 8. Must be changed for own projects.
9 Solid State Drives will be abbreviated as SSD in this document.

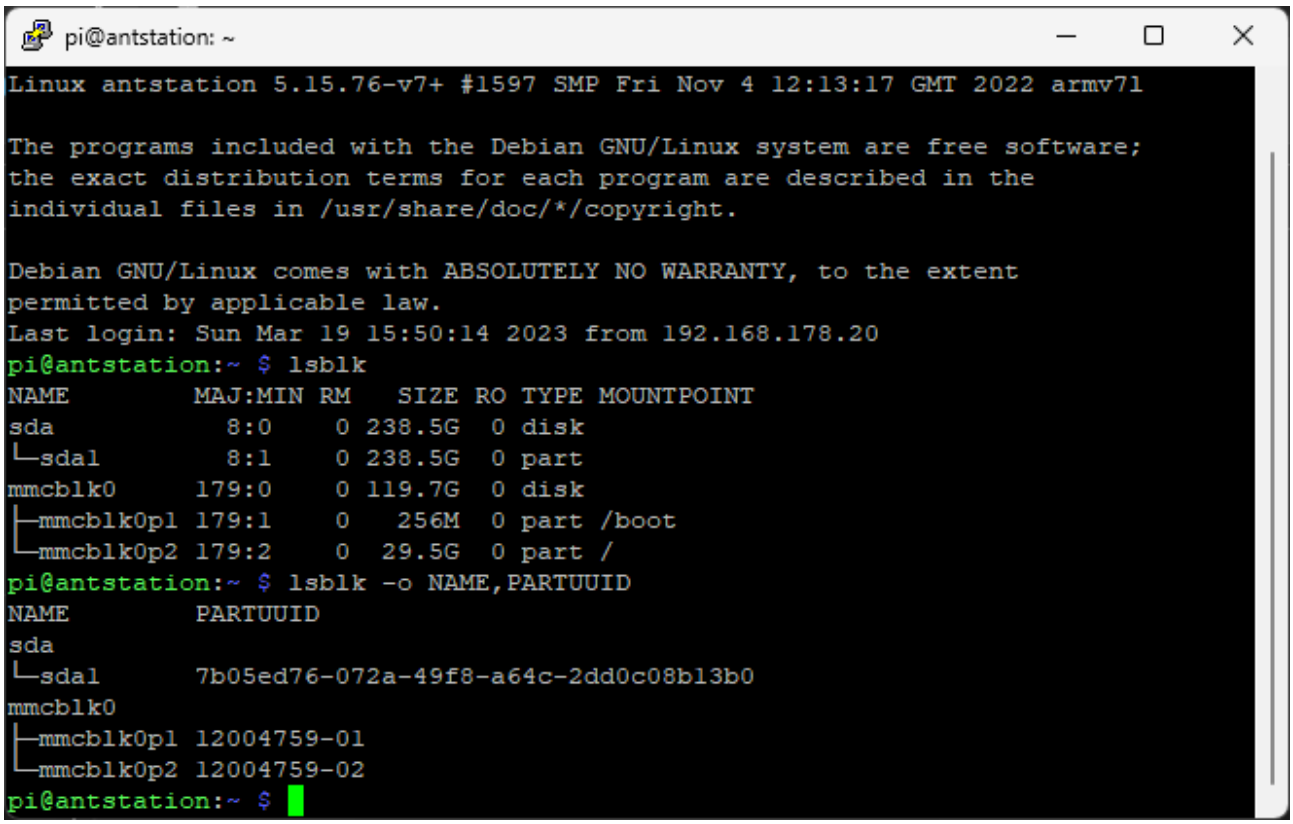


Image 6.4: **lsblk** *is used to show all connected drives, but we can also find the associated PARTUUID.*

```
pi@antstation:~ $ lsblk -o NAME,PARTUUID
NAME        PARTUUID
sda
└─sda1      7b05ed76-072a-49f8-a64c-2dd0c08b13b0
sdb
└─sdb1      3a1c433e-01
mmcblk0
├─mmcblk0p1 12004759-01
└─mmcblk0p2 12004759-02
pi@antstation:~ $ lsblk
NAME        MAJ:MIN RM   SIZE RO TYPE MOUNTPOINT
sda           8:0    0 238.5G  0 disk
└─sda1        8:1    0 238.5G  0 part /
sdb           8:16   1   7.2G  0 disk
└─sdb1        8:17   1   7.2G  0 part [SWAP]
mmcblk0     179:0    0 119.7G  0 disk
├─mmcblk0p1 179:1    0   256M  0 part /boot
└─mmcblk0p2 179:2    0  29.5G  0 part
pi@antstation:~ $ df -h
Filesystem      Size  Used Avail Use% Mounted on
/dev/root       234G  2.2G  220G   1% /
devtmpfs        333M     0  333M   0% /dev
tmpfs           462M     0  462M   0% /dev/shm
tmpfs           185M 1004K  184M   1% /run
tmpfs           5.0M  4.0K  5.0M   1% /run/lock
/dev/mmcblk0p1  255M   50M  206M  20% /boot
tmpfs            93M     0   93M   0% /run/user/1000
```

Image 6.5: This is how it should look like: sda1 is mounted as "/" and an additional swap partition is available on sdb1.

6.7 Security

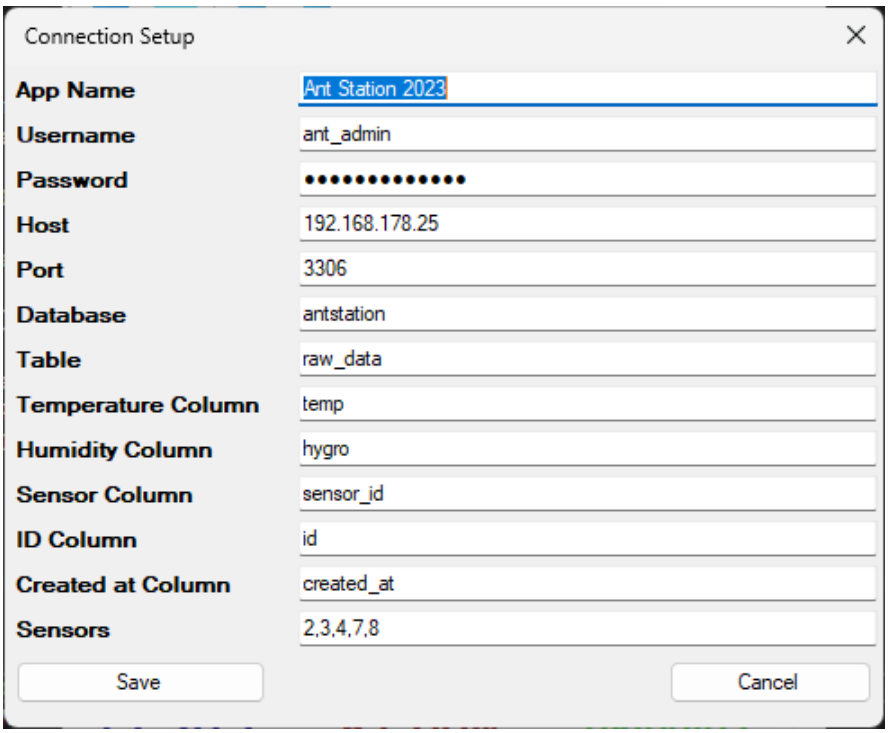
In the context of this project the RasPi is only available in LAN, via SSH and a different user than root for the database connection. Therefore it can only be recommended to follow more in-depth security advises if this solution should be used in other environments.

It is not recommended to use this “home grown solution” in any production environment without further security audits.

7 Weather Station 2023

The desktop application can be used after all sensors have been connected to the RasPi, which also writes data via Cron job every *x* minutes to the database.

7.1 Database setup



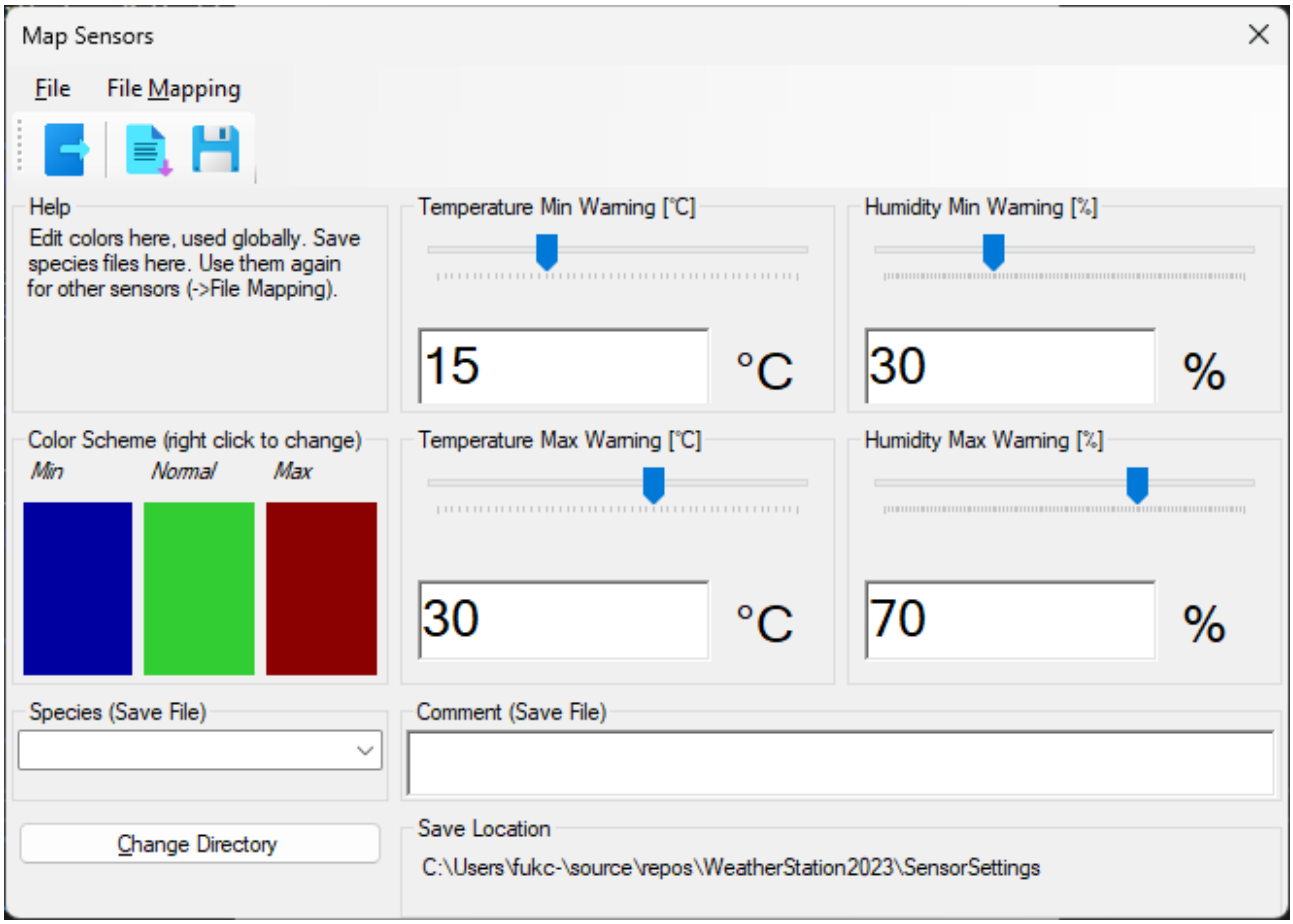
Without filling in the correct values to this form the application will not run. Make sure that the database and user is set up correctly and that the database is accessibly from the client computer.

App Name	The title of the main window, as shown e.g. in the taskbar, can be changed, e.g. if someone wants to monitor frogs instead of ants. **Default**: *Weather Station 2023*
Username	Name of the MariaDB user used to write to the database.
Password	Password of this user.
Host	IP-address of the RasPi – should be LAN only, thus IPv4 is used in this project. IPv6 could work, but was not tested.
Port	MariaDB port **Default**: *3306*
Database	Name of the database as defined during setup.
Table	Table of sensor data used by DHT2SQL.py.
Temperature Column	Name of the temperature column.

Humidity Column	Name of the humidity column.
Sensor Column	Name of the sensor column.
ID Column	Name of the database ID column.
Created at Column	Name of the created_at column (timestamp).
Sensors	List of Sensor-IDs (int), can be separated by " " (space), ";" or ",". A sensor is defined by the pin (sensor id) to which the data cable of the DHT22 sensor is connected. It is the same ID that is used by the Python script DHT2SQL.py for writing to the database.

7.2 Species setup

Basically, and what makes the difference to other weather monitoring applications, this application is designed to monitor certain animal species in their terrariums (closed spaces).

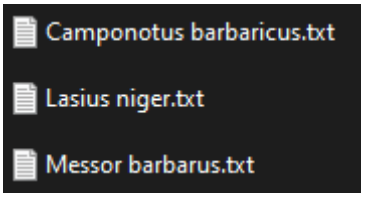


This form is used to create *species files* ("*.txt")[10]. As a *species file* could be used by more than one sensor no specific details are used here. It is only used to color the temperature and humidity values,

10 Although *species files* are basically text files and are easily readable by humans, they should only created with the application itself as there are certain requirements for a valid file. Bad files loaded during startup of the application might lead to a crash.

but the color scheme changes also some other coloring. The colors can be changed so that people with e.g. a red-green blindness could set better colors for them which are used globally.

If you want to use the possibility to monitor several different species, you should start creating as many different *species files* as needed, even if they only contain the default values first. Values can be changed later, as long as a *species file* exists.



7.3 Valid species file

```
1  [antconfig 1.0]
2  # Lasius niger - Ants Cologne
3  Lasius niger
4  12
5  25
6  30
7  80
8
```

Line 1 is essential and must be: `[antconfig 1.0]`

Line 2 is a comment (only one line allowed) and can be used for authorship, versions, testing etc.

Line 3 is the name of the species. In other projects it could be used for labels like “living room”, “kitchen”, “bed room” etc.

Line 4: Temperature minimum.

Line 5: Temperature maximum.

Line 6: Humidity minimum.

Line 7: Humidity maximum.

Line 8: Always empty (EOF)

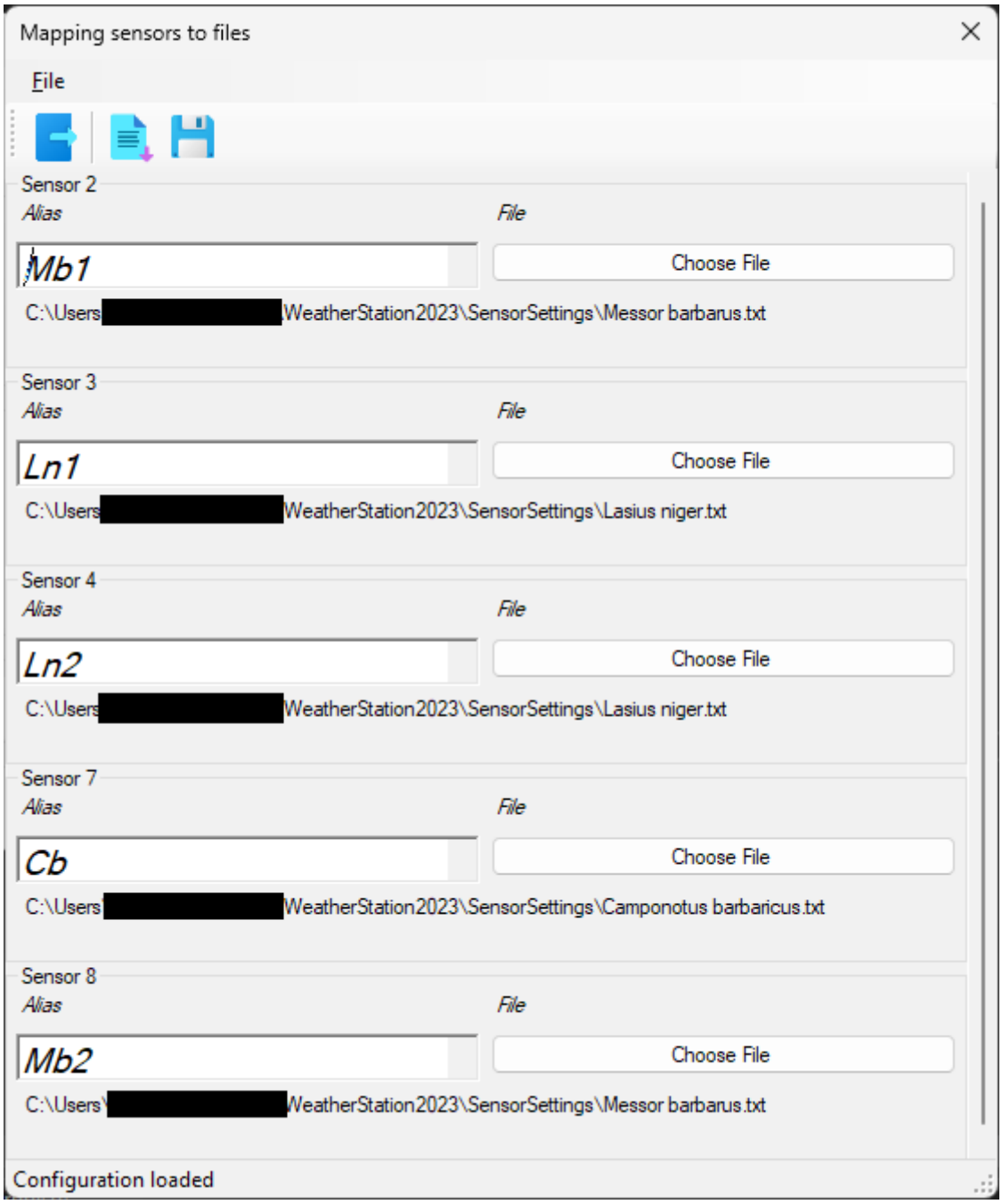
A valid file has got the “.txt” extension, fulfills the requirements of lines 1, 2 and 8.

Lines 4 – 7 must contain integer numbers.

Remark:

Currently species values from https://www.antstore.net/ are used as reference. Please note that there are differences between nest and arena values.

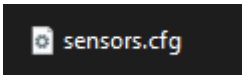
7.4 Sensor configuration



In this form everything is put together. The "**Choose File**" button loads a *species file* for each sensor and an **alias** can be given.

This is completely optional and can be skipped if no coloring of the sensor data is needed.

If used, everything will be saved in "`sensors.cfg`" in the working directory, which is the only "*.cfg"-file in the directory. As it is the only *configuration file* allowed, there is no dialog to select it. Only the last status can be loaded or a changed status be saved.



7.5 Valid sensor.cfg

The sensor configuration file contains one line per sensor. It should never be created manually. As there can be multiple installations of WeatherStation2023 on the same system, each one can contain its own configuration file (if different user directories are used), but it is also possible to use the same directory including species files for multiple installations[11].

```
1  [antconfig 1.0]
2  Sensor 2    Mb1 C:\Users\[redacted]\WeatherStation2023\SensorSettings\Messor barbarus.txt
3  Sensor 3    Ln1 C:\Users\[redacted]\WeatherStation2023\SensorSettings\Lasius niger.txt
4  Sensor 4    Ln2 C:\Users\[redacted]\WeatherStation2023\SensorSettings\Lasius niger.txt
5  Sensor 7    Cb  C:\Users\[redacted]\WeatherStation2023\SensorSettings\Camponotus barbaricus.txt
6  Sensor 8    Mb2 C:\Users\[redacted]\WeatherStation2023\SensorSettings\Messor barbarus.txt
7
```

Line 1 is essential and must be: `[antconfig 1.0]`[12]

Line 2 – 6: five sensors, tab-separated: *Sensor Name/ID, Alias, Species file*

Last line (7): Always empty (EOF)

7.6 Error codes

Even though the focus during the development of this application was to make it thread-safe by using background workers and exception handling, still errors can occur, e.g. if the RasPi is not reachable. Errors related to *Background Workers* (Bgw) might show that there are problems with the database, e.g. because the RasPi is busy and does not respond. If restarting the application does not help, it might be a good start to **reboot** the RasPi[13].

Error code	Component	Message	Solution
0001_BgwMf-<n>	MainForm Background Worker <n>	*	**
0002_BgwSen	Sensor Background Worker	*	**
0003_BgwSsf_r	SensorStatisticsForm Background Worker Read	*	**
0003_BgwSsf_c	SensorStatisticsForm Background Worker Complete	*	**
0005_BgwSf	Statistics Form Background Worker	*	**

11 Using several installations of WeatherStation2023 is possible, but will not be discussed in detail here. Normally, also multiple databases on multiple RasPis would also be necessary and require a lot more prerequisites. The application title could be used for host identification.

12 *Species files* and *configuration files* are distinguished mainly by their extension. If the file format changes (other than "antconfig 1.0"), a new version of WeatherStation2023 would be required.

13 Try if a SSH connection to the RasPi is possible. Enter "`sudo reboot`" in the prompt. If the RasPi does not respond, disconnect power to restart.

0006_Mfconf	Error loading *sensors.cfg* to MainForm	`Not a valid sensor configuration file!`	Delete *sensors.cfg* and create a new one
0007_Mfconf	Corrupt *sensor.cfg*		Delete *sensors.cfg* and create a new one
0008_Mfspec	Error loading species to MainForm	`Not a valid species file: + path`	Delete <path> and create a new one
0009_Mfspec	Corrupt species file	`Corrupt species file: + path`	Delete <path> and create a new one
0010_Mfcol	Colors from species file could not be loaded	`Colors could not be loaded. Not a valid species file: " + path`	Delete <path> and create a new one
0011_MapFcfg	Error loading *sensors.cfg* to Mapping-Form	`Not a valid sensor configuration file!`	Delete *sensors.cfg* and create a new one
0012_MapFsav	Error saving *sensors.cfg* in Mapping-Form	*	
0013_BgStatFrun	Statistics Background Worker	*	**
0014_CFldtxt	Error loading TXT file to Color Values Form	*	Check format of TXT files
0015_CFsvtxt	Error saving species file	*	
0016_CFval	Wrong value in number field	Value should be between x and y!	Only integer numbers are allowed
0017_CFldtxts	Error loading TXT file to Color Values Form	*	Check format of TXT files

* **Message** is generated by the system. Maybe there will be a collection of common errors in the future, but this is currently not planned. Community feedback would be highly appreciated.
** **Background Workers** are handling multiple database access tasks and can sometimes throw errors, because the RasPi is not available for handling requests. They often work the next time and there is nothing a user can actually do about those errors. Therefore these errors can be switched-off in the application menu.

7.7 Github

The C#-code for the desktop application can be found here:

https://github.com/Ants-Cologne/WeatherStation2023

7.7.1 WeatherStation2023.pdf

This project is still in development and changes to the help file are expected.

If there was an update to the help file, downloading it from Github is sufficient and there is no need to re-install the application. Just copy **WeatherStation2023.pdf** to the installation directory.

https://github.com/Ants-Cologne/WeatherStation2023/blob/master/WeatherStation2023.pdf

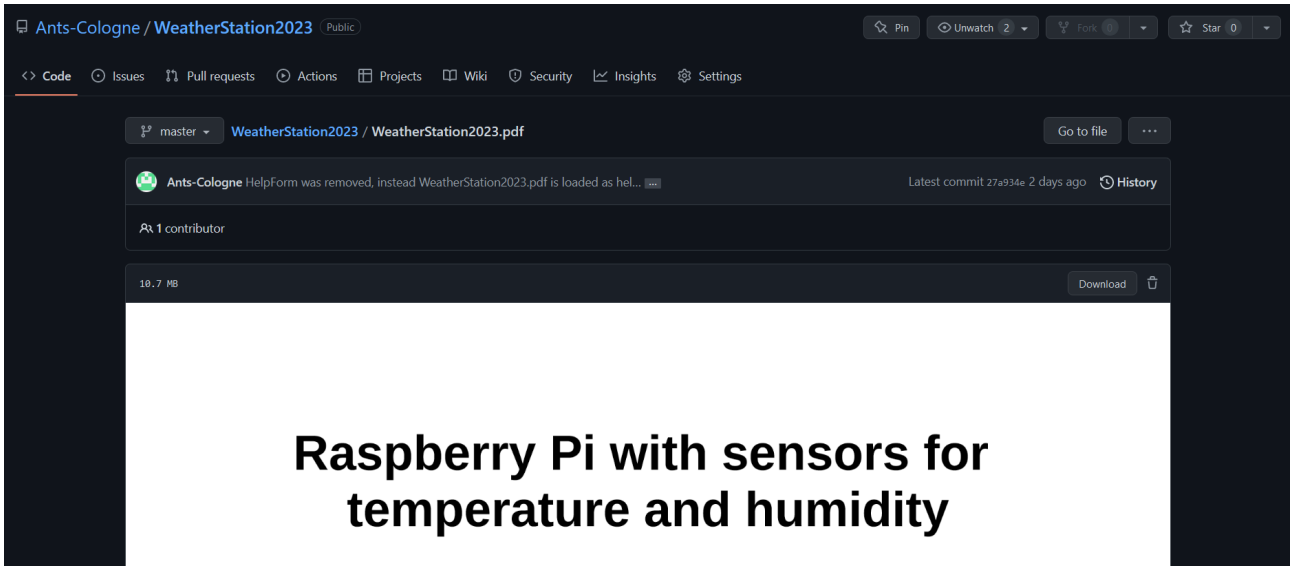


Image 8.1: Logo

8 YouTube videos (German)

8.1 Playlist

This project was accompanied by a German YouTube tutorial playlist which can be found here:

Ants Cologne RasPi DHT22-Sensor-Tutorial

To support this project please consider a channel subscription of **Ants Cologne** on YouTube.

8.2 Videos

If you need more details, maybe one of these videos can give advise, especially about how to solder the circuit board. English translation on YouTube should work.

RasPi DHT22-Sensor-Tutorial (1) - Hardware (1)	Overview circuit board, DHT22 sensor, RasPi.
RasPi DHT22-Sensor-Tutorial (2) - Hardware (2) - Die Platine	Soldering the circuit board, missing hardware needed (silver wire), preparation of the sensors: wire configuration with different length.
RasPi DHT22-Sensor-Tutorial (3) - Hardware (3) - Finales Platinenlayout	Finalization of the circuit board layout for five DHT22 sensors.
RasPi DHT22-Sensor-Tutorial (4): Das Skript	The circuit board has been tested, sensors are connected and now the RasPi is installed. The DHT2SQL.py script is introduced to read data from sensors and write it to the database.
RasPi DHT22-Sensor-Tutorial (5): Drei Sensoren an der Platine	Finalization of the sensor wiring. Setup of the first formicariums.
Messor barbarus Update 18.12.2022 - Winterruhe?!	All sensors connected, some problems still exist. At this point data is continuously written to the database. *This video is not part of the RasPi playlist.*
RasPi DHT22-Sensor-Tutorial (6) - Die Desktop-Anwendung mit C# (1)	Starting the programming of *Weather Station 2.0* to solve some problems experienced with *AntStation 1.0* (2021).
RasPi-DHT22-Sensor-Tutorial (7) - Die Desktop-Anwendung (2): Backgroundworker und Sensor-Klasse	The basic problems with *AntStation 1.0* have been resolved by using background workers which will now be used throughout the project.
RasPi-DHT22-Sensor-Tutorial (8) - WeatherStation2023	Weather Station 2023 was a complete re-design of the Windows application and was the first release that was feature-complete and released on Github .
RasPi-DHT22-Tutorial (9): Die finale Anwendung	Demonstration of all functions of the first release of Weather Station 2023 after upgrading to a RasPi 2B.

9 Stackoverflow Credits

Some problems were not searchable, so the author asked some questions on stackoverflow.com and wants to thank these persons as they helped him when a continuation was limited:

IVSoftware for answering https://stackoverflow.com/questions/75277474/groupbox-created-in-code-doesnt-dock-correctly-to-parent-tablelayoutpanel

IVSoftware even provided a Github example of their solution, which was a great help as they gave an example how to use custom user controls, such as a GroupBox, something that the author never used in own projects before:

https://github.com/IVSoftware/group-boxes-fill-parent-width

Compman2408 for answering https://stackoverflow.com/questions/75033883/c-sharp-sqlconnection-child-form-closing-but-data-is-not-updated-in-mainform/75034158#75034158

Stackoverflow is a great platform and helped a lot with answers when searching for C# problems.

Therefore: a lot of thanks and kudos to **Stackoverflow**!

10 List of software used for this project

This list includes ALL software used for this project. Only the RasPi part is essential, the Windows software is just an example what was be needed to run everything on a Windows 11 client. For this operating system Weather Station 2023 is optimized, but it should run on older Windows versions as well.

10.1 Raspberry Pi

10.1.1 Raspberry Pi OS Lite (32-bit) / Raspberry Pi Imager v1.7.3

Raspberry Pi Imager was used to install the Raspberry Pi OS **Rasbian** on a 32GB SD-card[14].

https://www.raspberrypi.com/software/

10.1.2 MariaDB 10.5.15

The database used for this project. Can be accessed via SQL-statements (used in DHT2SQL.py and WeatherStation2023).

https://mariadb.com/

10.1.3 MariaDB Connector for Python 1.0.7

Needed to connect to MariaDBs via Python scripts. Thus the python script DHT2SQL.py can read sensor data and write it to the database.

https://mariadb.com/de/resources/blog/how-to-connect-python-programs-to-mariadb/

10.1.4 Python 3.9.2

Is already installed with Rasbian Linux (see 10.1.1). Necessary to start DHT2SQL.py via Cron job.

https://www.python.org/

10.1.5 pigpio / pigpio Daemon

Pigpio is the library used to access the GPIOs of the RasPi. It is used by DHT2SQL.py to access the sensors.

14 An SD-card is not recommended for production use, only for setup and programming. Consider to install the root system of linux (“/”) on an external USB-SSD or hard drive. See 6.6.

The Daemon is used to start pigpio automatically after rebooting the RasPi, thus it is an essential prerequisite for the Cron job.

```
> sudo apt install pigpio python-pigpio python3-pigpio
```

Library: http://abyz.me.uk/rpi/pigpio/index.html

Daemon: http://abyz.me.uk/rpi/pigpio/pigpiod.html

10.1.6 PHP 8.2.x

Always use the most recent version of *PHP* for web server projects. As this is not the main topic of this project, please check more reliable sources on the internet for this topic.

https://www.php.net/

10.1.7 Web server lighttpd

Apache might be the most commonly used web server on the internet, but for the purpose of this project a lightweight web server like *lighttpd* is absolutely sufficient. It needs to be installed along with PHP.

In the root directory of the web server index.php will be used to show current sensor data.

Some rights must be set on the web server's directory, as the PHP script needs to be executable.

```
sudo groupadd www-data
sudo usermod -G www-data -a pi
sudo chown -R www-data:www-data /var/www/html
sudo chmod -R 775 /var/www/html
```

There is much more to know about web server configuration, but as with PHP this is not the topic of this documentation.

https://www.lighttpd.net/

10.2 Windows

10.2.1 DaVinci Resolve

DaVinci Resolve is a good start for learning video editing on a (semi-)professional level. It provides a lot of functions and a lot of good tutorials are available on the internet (YouTube).

https://www.blackmagicdesign.com/de/products/davinciresolve

10.2.2 Win32DiskImager

Used to backup SD-cards from the RasPi. Also used for installation of images on new drives.

https://win32diskimager.org/

10.2.3 PuTTY

PuTTY is used to connect via SSH to the RasPi. There are other ways, but the author uses *PuTTY* since years.

https://www.chiark.greenend.org.uk/~sgtatham/putty/latest.html

10.2.4 Microsoft Visual Studio 2022

Used to create the desktop application Weather Station 2023. Direct support of Github .

https://visualstudio.microsoft.com/de/downloads/

10.2.5 Paint.NET 5.0.2

All graphics in this document (and for the YouTube channel) are done with this free application. There was no need for commercial products during the course of this project[15].

https://www.getpaint.net/download.html

15 The author believes that it is always a good idea to start with free software and upgrade only to commercial projects if special needs evolve. Normally, there is always a good free alternative available.

11 Sources

11.1 DHT2SQL.py

This is also available in the Github repository. It is considered as *final* and works without problems in this project. If someone uses this script for own projects, the database access needs to be configured accordingly. Critical code of this script which was essential for the described project, has been marked in red color.

Version used: **DHT11/21/22/33/44 Sensor – 2019-11-07**

http://abyz.me.uk/rpi/pigpio/examples.html#Python_code/DHT.py

The original script **DHT22.py** should be used to test the sensors and GPIOs used. This is essential for finding the appropriate command which will be used for the Cron job.

```
#!/usr/bin/env python

# DHT2SQL.py / original: DHT.py
# 2019-11-07 / Ants.Cologne: 2022-12-02
# Public Domain
# Modified for use with ANTSTATION by Ants.Cologne 2022 (I only
changed __main__)

import time
import pigpio

DHTAUTO=0
DHT11=1
DHTXX=2

DHT_GOOD=0
DHT_BAD_CHECKSUM=1
DHT_BAD_DATA=2
DHT_TIMEOUT=3

class sensor:
   """
   A class to read the DHTXX temperature/humidity sensors.
   """
   def __init__(self, pi, gpio, model=DHTAUTO, callback=None):
      """
      Instantiate with the Pi and the GPIO connected to the
      DHT temperature and humidity sensor.
```

```
      Optionally the model of DHT may be specified.  It may be one
      of DHT11, DHTXX, or DHTAUTO.  It defaults to DHTAUTO in
which
      case the model of DHT is automtically determined.

      Optionally a callback may be specified.  If specified the
      callback will be called whenever a new reading is available.

      The callback receives a tuple of timestamp, GPIO, status,
      temperature, and humidity.

      The timestamp will be the number of seconds since the epoch
      (start of 1970).

      The status will be one of:
      0 DHT_GOOD (a good reading)
      1 DHT_BAD_CHECKSUM (receieved data failed checksum check)
      2 DHT_BAD_DATA (data receieved had one or more invalid val-
ues)
      3 DHT_TIMEOUT (no response from sensor)
      """
      self._pi = pi
      self._gpio = gpio
      self._model = model
      self._callback = callback

      self._new_data = False
      self._in_code = False

      self._bits = 0
      self._code = 0

      self._status = DHT_TIMEOUT
      self._timestamp = time.time()
      self._temperature = 0.0
      self._humidity = 0.0

      pi.set_mode(gpio, pigpio.INPUT)
      self._last_edge_tick = pi.get_current_tick() - 10000
      self._cb_id = pi.callback(gpio, pigpio.RISING_EDGE,
self._rising_edge)

   def _datum(self):
      return ((self._timestamp, self._gpio, self._status,
```

```
            self._temperature, self._humidity))

   def _validate_DHT11(self, b1, b2, b3, b4):
      t = b2
      h = b4
      if (b1 == 0) and (b3 == 0) and (t <= 60) and (h >= 9) and (h
<= 90):
         valid = True
      else:
         valid = False
      return (valid, t, h)

   def _validate_DHTXX(self, b1, b2, b3, b4):
      if b2 & 128:
         div = -10.0
      else:
         div = 10.0
      t = float(((b2&127)<<8) + b1) / div
      h = float((b4<<8) + b3) / 10.0
      if (h <= 110.0) and (t >= -50.0) and (t <= 135.0):
         valid = True
      else:
         valid = False
      return (valid, t, h)

   def _decode_dhtxx(self):
      """
            +-------+-------+
            | DHT11 | DHTXX |
            +-------+-------+
      Temp C| 0-50  |-40-125|
            +-------+-------+
      RH%   | 20-80 | 0-100 |
            +-------+-------+

               0      1      2      3      4
            +------+------+------+------+------+
      DHT11 |check-| 0    | temp |  0   | RH%  |
            |sum   |      |      |      |      |
            +------+------+------+------+------+
      DHT21 |check-| temp | temp | RH%  | RH%  |
      DHT22 |sum   | LSB  | MSB  | LSB  | MSB  |
      DHT33 |      |      |      |      |      |
      DHT44 |      |      |      |      |      |
            +------+------+------+------+------+
```

```
      """
      b0 =  self._code        & 0xff
      b1 = (self._code >>  8) & 0xff
      b2 = (self._code >> 16) & 0xff
      b3 = (self._code >> 24) & 0xff
      b4 = (self._code >> 32) & 0xff

      chksum = (b1 + b2 + b3 + b4) & 0xFF

      if chksum == b0:
         if self._model == DHT11:
            valid, t, h = self._validate_DHT11(b1, b2, b3, b4)
         elif self._model == DHTXX:
            valid, t, h = self._validate_DHTXX(b1, b2, b3, b4)
         else: # AUTO
            # Try DHTXX first.
            valid, t, h = self._validate_DHTXX(b1, b2, b3, b4)
            if not valid:
               # try DHT11.
               valid, t, h = self._validate_DHT11(b1, b2, b3, b4)
         if valid:
            self._temperature = t
            self._humidity = h
            self._status = DHT_GOOD
         else:
            self._status = DHT_BAD_DATA
      else:
         self._status = DHT_BAD_CHECKSUM
      self._new_data = True

   def _rising_edge(self, gpio, level, tick):
      edge_len = pigpio.tickDiff(self._last_edge_tick, tick)
      self._last_edge_tick = tick
      if edge_len > 10000:
         self._in_code = True
         self._bits = -2
         self._code = 0
      elif self._in_code:
         self._bits += 1
         if self._bits >= 1:
            self._code <<= 1
            if (edge_len >= 60) and (edge_len <= 150):
               if edge_len > 100:
                  # 1 bit
                  self._code += 1
```

```
            else:
               # invalid bit
               self._in_code = False
         if self._in_code:
            if self._bits == 40:
               self._decode_dhtxx()
               self._in_code = False

   def _trigger(self):
      self._new_data = False
      self._timestamp = time.time()
      self._status = DHT_TIMEOUT
      self._pi.write(self._gpio, 0)
      if self._model != DHTXX:
         time.sleep(0.018)
      else:
         time.sleep(0.001)
      self._pi.set_mode(self._gpio, pigpio.INPUT)


   def cancel(self):
      """
      """
      if self._cb_id is not None:
         self._cb_id.cancel()
         self._cb_id = None

   def read(self):
      """
      This triggers a read of the sensor.

      The returned data is a tuple of timestamp, GPIO, status,
      temperature, and humidity.

      The timestamp will be the number of seconds since the epoch
      (start of 1970).

      The status will be one of:
      0 DHT_GOOD (a good reading)
      1 DHT_BAD_CHECKSUM (receieved data failed checksum check)
      2 DHT_BAD_DATA (data receieved had one or more invalid val-
ues)
      3 DHT_TIMEOUT (no response from sensor)
      """
      self._trigger()
```

```
      for i in range(5): # timeout after 0.25 seconds.
         time.sleep(0.05)
         if self._new_data:
            break
      datum = self._datum()
      if self._callback is not None:
          self._callback(datum)
      return datum

if __name__== "__main__":
   import sys
   import pigpio
   import DHT
   import mariadb        #added by Ants.Cologne
   import sys            #added by Ants.Cologne

   def callback(data):
      print("{:.3f} {:2d} {} {:3.1f} {:3.1f} *".
         format(data[0], data[1], data[2], data[3], data[4]))

   argc = len(sys.argv) # get number of command line arguments

   if argc < 2:
      print("Need to specify at least one GPIO")
      exit()

   pi = pigpio.pi()
   if not pi.connected:
      exit()

   #ANTSTATION SQL
   try:
      conn = mariadb.connect(
         user="root",              # We're using root as this
script runs only on localhost
         password="password", # I'm using a different pwd, of
course ;)
         host="127.0.0.1",         # localhost, no access from
outside
         port=3306,               # same port as from outside
         database="antstation"     # Ants.Cologne: in this db
we're storing our sensor data!
      )
   except mariadb.Error as e:
      print(f"Error connecting to MariaDB Platform: {e}")
```

```
      sys.exit(1)

   # Get Cursor
   cur = conn.cursor()
   #print("SQL works")

   # Instantiate a class for each GPIO
   # for testing use a GPIO+100 to mean use the callback
   S = []
   for i in range(1, argc): # ignore first argument which is com-
mand name
      g = int(sys.argv[i])
      if (g >= 100):
         s = DHT.sensor(pi, g-100, callback=callback)
      else:
         s = DHT.sensor(pi, g)
      S.append((g,s)) # store GPIO and class

   while True:
      try:
         for s in S:
            if s[0] >= 100:
               s[1].read() # values displayed by callback
            else:       # Only this is relevant for ANTSTATION
               d = s[1].read()
               print("{:.3f} {:2d} {} {:3.1f} {:3.1f}".
                  format(d[0], d[1], d[2], d[3], d[4]))
               if d[2] == 0:          # We only write to db if sen-
sor data was OK
                  try:
                     cur.execute(
                        "INSERT INTO raw_data
(temp,hygro,sensor_id) VALUES (?,?,?)",
                        (d[3],d[4],d[1]))
                     print("Data written to database. Sensor: " +
str(d[1]))
                  except mariadb.Error as e:
                     print(f"error: {e}")

                  conn.commit()
         #time.sleep(2)     #commented out as we're not using the
infinite-loop here (which will be done by CRON)
         break              #added to leave the while-loop
      except KeyboardInterrupt:
         break
```

```
# Close SQL connection
conn.close()

for s in S:
   s[1].cancel()
   #print("cancelling {}".format(s[0]))        #removed
pi.stop()
```

11.2 Testing

The original script **DHT22.py** can be used for testing the GPIO connections in the console.

```
> python3 DHT.py <sensor(s)>
```

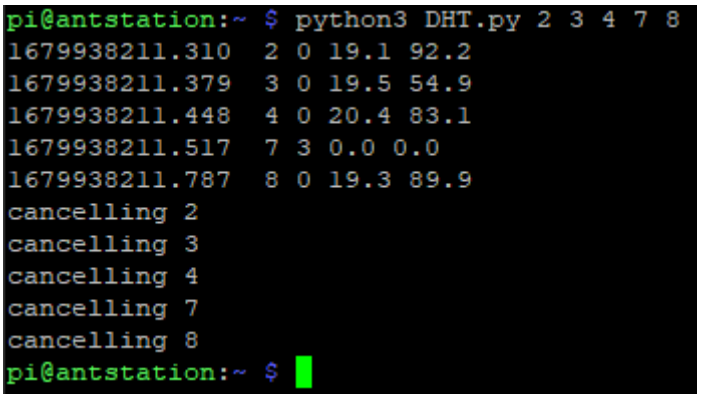


*Image 11.1: In this case sensor **7** shows an error, but the other sensors are tested successfully. This version of DHT.py runs only once and can be used to show data via a PHP web page.*

If the sensor GPIOs have been checked successfully, **DHT2SQL.py** can be tested:

```
> python3 DHT2SQL.py <sensor(s)>
```

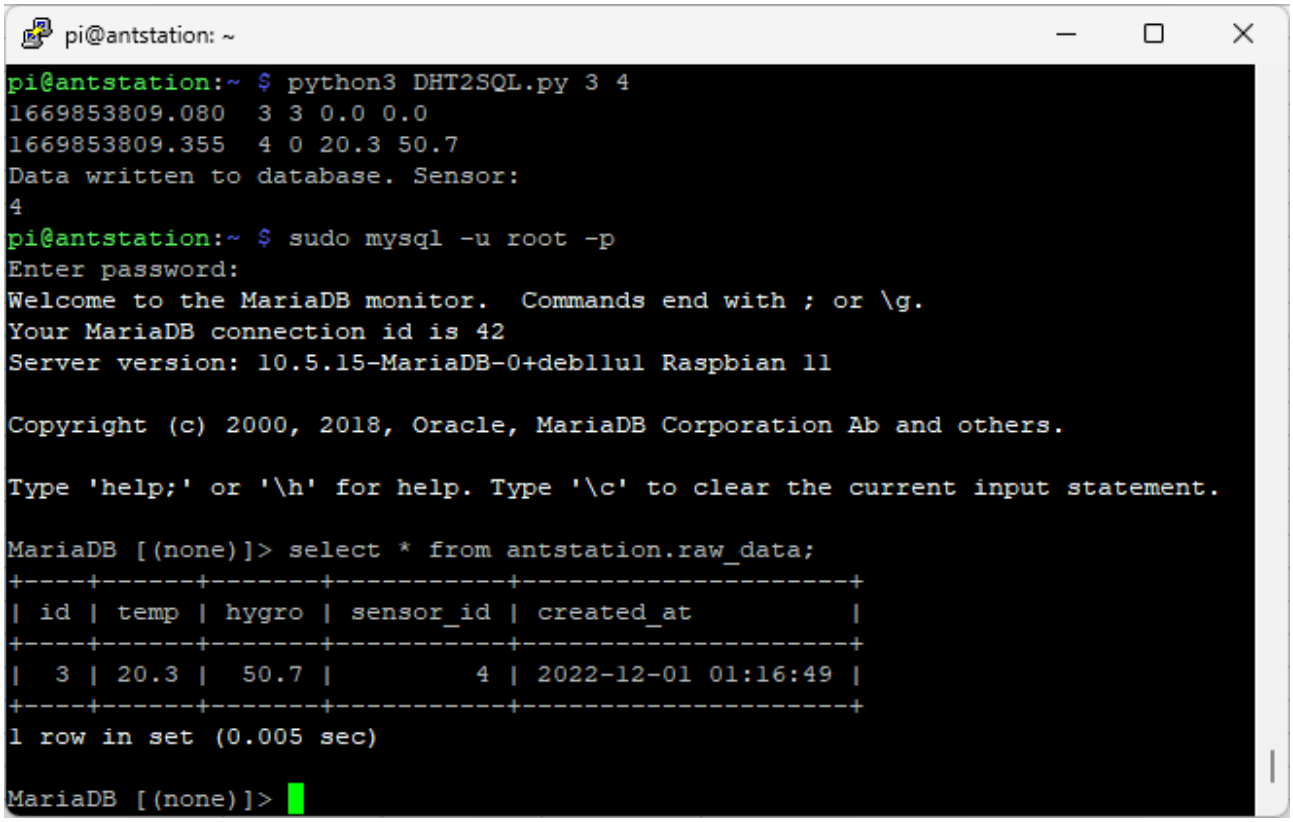


Image 11.2: There is only one sensor connected to the RasPi (sensor 4), but the script is tested with parameters 3 and 4. Only sensor 4 writes successfully to the database.

Only when everything has been tested successfully the Cron job should be configured.

11.3 DHT.py

This is a modification of the original script, only the `"__main__"` has changed. This script returns after the first while round, so each sensor is touched once. In the current version of this file there are actually two rounds of the while, whereas the second row for each sensor returns “cancelling” which needs to be ignored during further processing.

The main goal of this script is that it can be called by a PHP web page and display current sensor data in a web browser environment.

```
if __name__== "__main__":
   import sys
   import pigpio
   import DHT

   def callback(data):
      print("{:.3f} {:2d} {} {:3.1f} {:3.1f} *".
         format(data[0], data[1], data[2], data[3], data[4]))

   argc = len(sys.argv) # get number of command line arguments

   if argc < 2:
```

```
      print("Need to specify at least one GPIO")
      exit()

   pi = pigpio.pi()
   if not pi.connected:
      exit()

   # Instantiate a class for each GPIO
   # for testing use a GPIO+100 to mean use the callback
   S = []
   for i in range(1, argc): # ignore first argument which is command name
      g = int(sys.argv[i])
      if (g >= 100):
         s = DHT.sensor(pi, g-100, callback=callback)
      else:
         s = DHT.sensor(pi, g)
      S.append((g,s)) # store GPIO and class

   while True:
      try:
         for s in S:
            if s[0] >= 100:
               s[1].read() # values displayed by callback
            else:
               d = s[1].read()
               print("{:.3f} {:2d} {} {:3.1f} {:3.1f}".
                  format(d[0], d[1], d[2], d[3], d[4]))
         #time.sleep(2)
         break
      except KeyboardInterrupt:
         break

   for s in S:
      s[1].cancel()
      print("cancelling {}".format(s[0]))
   pi.stop()
```

11.4 /var/www/html/index.php

While the desktop application can only access values stored in the database, a PHP script can execute a Python script and show its results on a web page. This could be used to test sensors, e.g. by refreshing this page on a smartphone if sensors are connected to the circuit board.

It is easy to change the web page to other languages. In this example the language can be changed by adding "**?lang=<language>**" to the browser's URL. Currently the default language is English, but if "**?lang=de**" is given, the page switches to German[16].

```
<html>
<head>
<?php
if ($_GET["lang"] == "de") {
   echo "<title>Aktuelle Daten ANTSTATION</title>\n";
}
else {
```

16 German is the mother tongue of the author of this document. Feel free to change to other languages if needed.

```
   echo "<title>Live data ANTSTATION</title>\n";
}
?>
<META HTTP-EQUIV="refresh" CONTENT="60">
</head>
<body>
<?php
if ($_GET["lang"] == "de") {
   echo "<h1>Aktuelle Daten ANTSTATION</h1>\n";
}
else {
   echo "<h1>Live data ANTSTATION</h1>\n";
}

$result = array();

exec("python3 /home/pi/DHT.py 2 3 4 7 8", $result);

foreach($result as $res)
{
   if (!str_contains($res,'cancelling')) {
      $arr1 = explode(' ', $res);
      if ($arr1[3] == "0") {
         if ($_GET["lang"] == "de") {
            echo "<p>Sensor  ".$arr1[2].': Temperatur: '.$arr1[4]."°C - Luft-
feuchtigkeit: $arr1[5]%\n";
         }
         else {
            echo "<p>Sensor  ".$arr1[2].': Temperature: '.$arr1[4]."°C - Humid-
ity: $arr1[5]%\n";
         }
      }
      else {
         if ($_GET["lang"] == "de") {
            echo "<p style=\"color:#FF0000\";>Sensor ".$arr1[2]." überprüfen!
Fehlerhafter Sensor.</p>\n";
         }
         else {
            echo "<p style=\"color:#FF0000\";>Check Sensor ".$arr1[2]."! Wrong
sensor data.</p>\n";
         }
      }
   }
}
?>
</body>
</html>
```

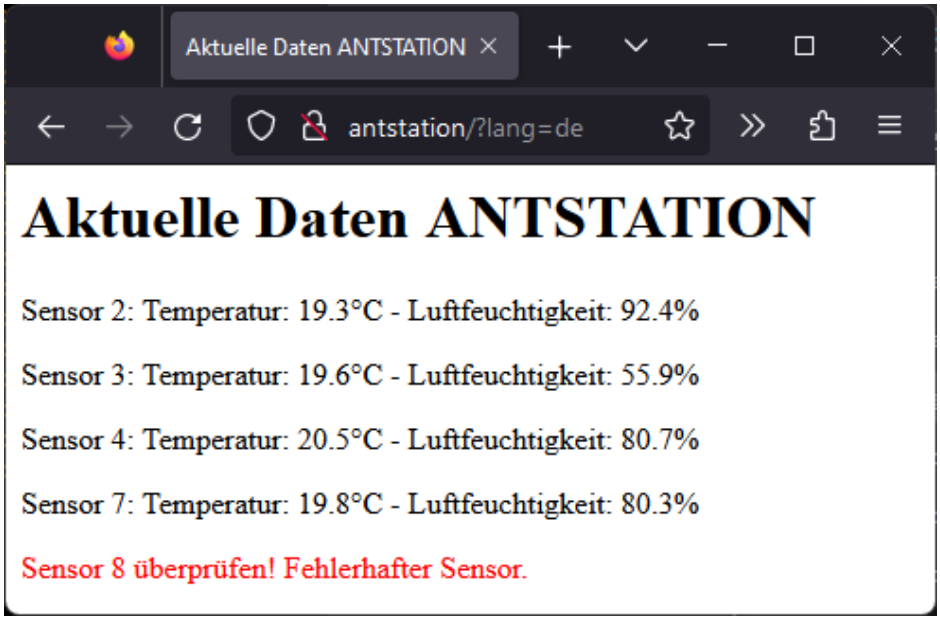


Image 11.3: Showing the web page in German.