FWS
Flight Weather Station

Projektpraktikum

Johannes Kasberger
0616782

Markus Klein
0726101

Hereby we declare that this work has been written autonomously, that all used sources and utilities are denoted accordingly and that these points of the work - including tables, maps and figures - which where taken from other creations or the Internet have been marked as borrowing by quoting the original sources. This document at hand will not be submitted to any other course.

_____ _____

Contents

List of Figures

List of Tables

Acknowledgement

In first place our special thanks go to Ao.Univ.Prof.Dr. Wolfgang Kastner who had an open ear for our project idea and agreed to be advisor for this project.

Furthermore we want to thank Guido Socher from tuxgraphics.org and the customer service team of ThiesKlima for their support.

Another person should also be named here is Roman Kreuzhuber who always was on spot with helping hands when we had questions concerning electronics.

Chapter 1

Introduction

1.1 Project idea

The idea to this project came up when I was flying my model helicopter. I was wondering how nasty the wind was these days and that it would have been better not to take off into the skies.

But where should I know the conditions from? The wind speed is always different than the values provided by online weather services.

That was the point I decided to build something on my own, submitting all the weather data to our website so one can decide beforehand whether to drive to our airfield or not.

1.2 Available products on the market

Having a look on the Internet reveals that there are plenty of weather stations available. Nearly all of them provide the basic features like temperature and air pressure measurement and an LCD for presenting the current conditions and/or a small forecast.

Some more expensive devices even provide a wind sensor and can be connected to a computer via USB. The software shipped with these devices also supports uploading of data to a website.

So why spending so much time for creating our own solution? Comparing the specifications of the wind speed sensors for instance shows that these very cheap sensors only have limited quality. Also detecting the wind direction is not available in most products, which is essential for airfield applications.

Another thing is extensibility. We needed a system that can be extended by new sensors easily and that is flexible enough to generate graphs and diagrams as desired.

The result of this investigation is the present project called Flight Weather Station (abbr. FWS).

1.3 About FWS

As depicted in Figure 1.1, FWS consists of the following parts:

- Sensors: These sensors capture the parameters: Wind speed, wind direction, temperature. Data are transmitted via cable.
- Slave: The slave collects and processes the data from the sensors.
- Master: This software collects the processed data of the slave over Ethernet LAN. It generates diagrams and graphs which are sent to the webserver by a separate script.
- Webserver (Presentation): The generated diagrams are integrated into the website with a plugin for the CMS TYPO3[1] also written specifically for this project.

The transfer of the data from the master to the webserver isn't part of this project. We recommend using cronjobs or similar approaches to accomplish this task.

[1] http://typo3.org

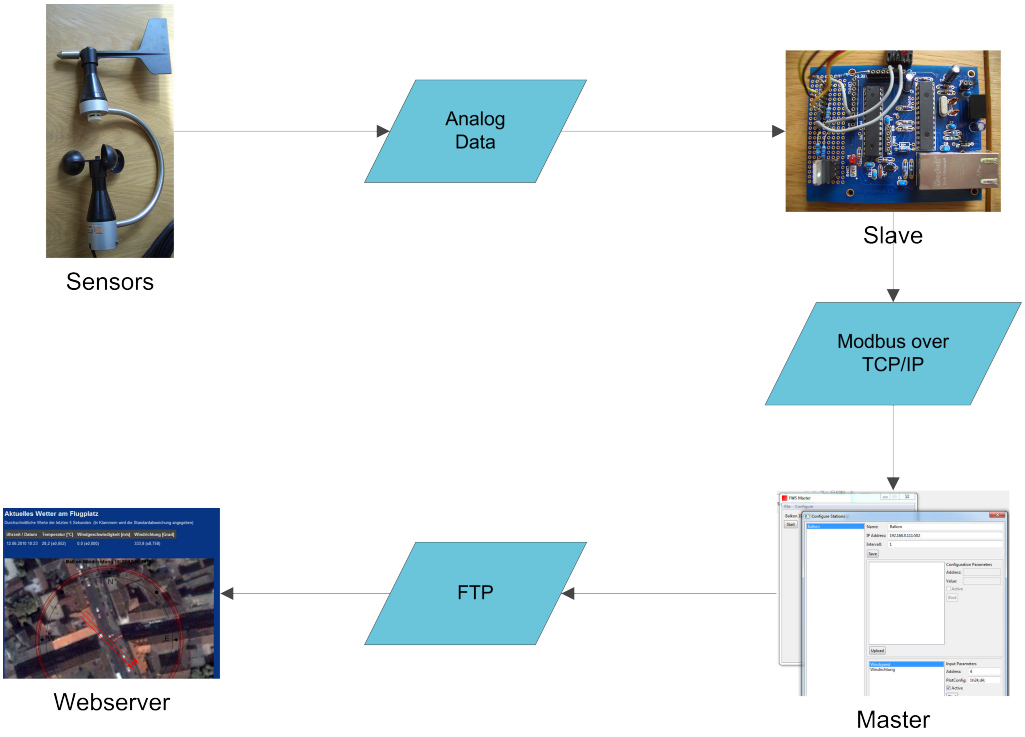
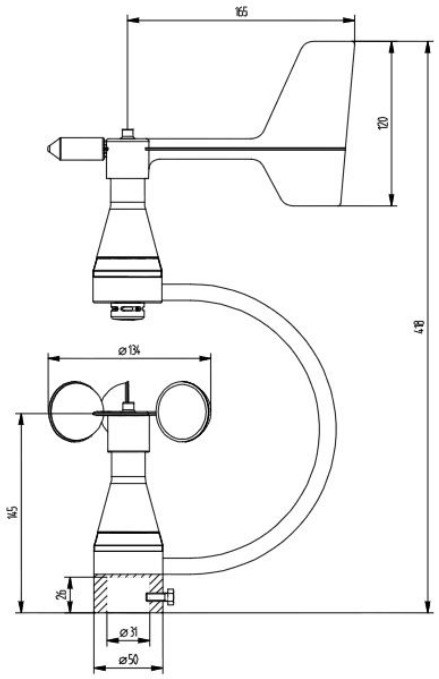


Figure 1.1: Module and protocol overview of FWS

Chapter 2

Sensors

(a) Wind sensor from ThiesKlima



(b) Dimensions of wind sensor ©ThiesKlima

For capturing the monitored parameters wind speed, wind direction and temperature, a product of ThiesKlima is used combining all needed sensors into one device. The connection between the sensor device and the microcontroller board is established by a 6-pin RJ-25 plug.

Komb. Windgeber		
Windgeschwindigkeit	Messbereich	1... 40 m/s
	Messprinzip	1 Reedkontakt / 2 Magnete
	Ausgangssignal	Potentialfreier Kontakt, typ. 2,3 Hz / ms^{-1}
	Auflösung	0,4 m Windweg
Windrichtung	Messbereich	0... 360°
	Messprinzip	Optoelektronische Abtastung einer Codescheibe
	Ausgangssignal	0 V... 4,69 V = 0... 337,5 <° (Vcc = 5 V)
	Auflösung	0,31 V = 22,5 Grad
	Ausgangswiderstand	ca. 10 kΩ
Temperatur	Messbereich	- 30 ... + 60 °C
	Sensor	NTC 10 kΩ
	Messschaltung	Spannungsteiler
Allgemein	Betriebstemperatur	-25... +60 °C (eisfrei)
	Anschlussart	Fest angeschlossenes 20 m Kabel mit anzeigeseitigem 6- poligen Westernstecker
	Spannungsversorgung (Vcc)	5 V DC (die Versorgung erfolgt vom Anzeigegerät)
	Stromverbrauch	3 mA
	Abmessung	Siehe Maßbild
	Gewicht	1 Kg
	Montageart	Auf Rohrstutzen von Ø 30mm und mindestens 30 mm Länge
	Schutzart	IP 54

Figure 2.1: Specification of wind sensor ©ThiesKlima

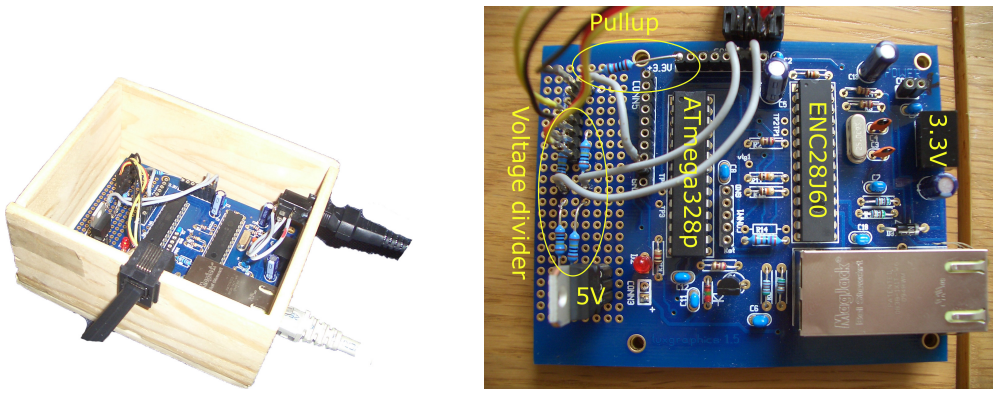
Figure 2.2: Pin layout of sensor plug ©ThiesKlima

Chapter 3

Slave

The term slave subsumes all parts of the sensor capturing unit. The following explanations will discuss hardware and software individually to provide a more specific view of these parts.

3.1 Hardware



(a) Sensor device with all cables connected

(b) Mainboard with position of parts

Figure 3.1: Pictures of the hardware

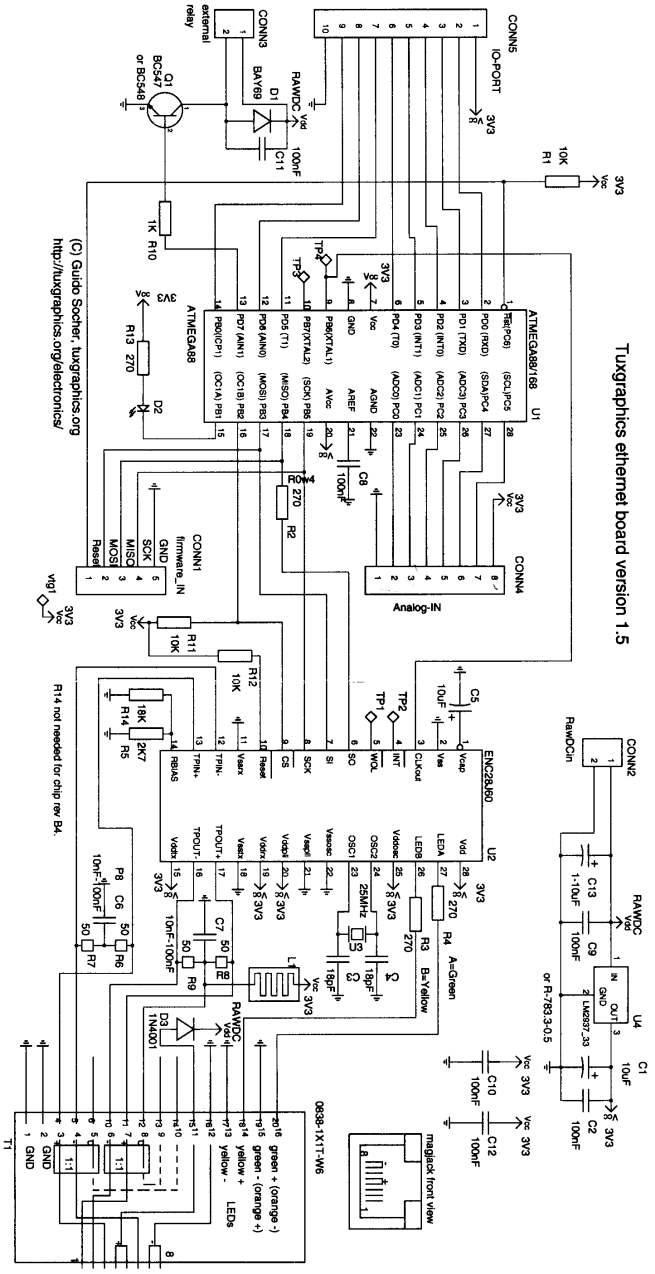


Figure 3.2: Schematics of mainboard ©tuxgraphics.org

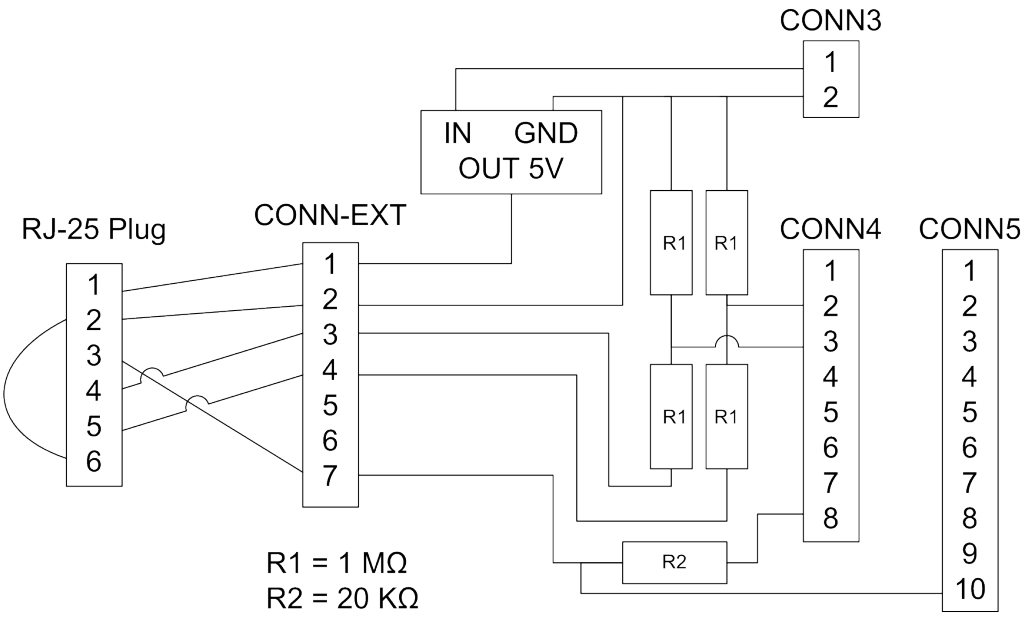


Figure 3.3: Schematics of additional hardware for sensor communication

The measurement data acquisition and TCP/IP handling is done by an ATMEL ATmega328p chip, whereas the basic Ethernet communication is handled by an ENC28J60 chip from Microchip. Both chips are communicating via an SPI connection.[1]

The mainboard is operated at $3.3V$ via a voltage regulator. Due to the fact that the sensor device requires a supply voltage of $5V$, a separate voltage regulator has been added. This regulator is sourced by CONN3 of the mainboard which can be switched on and off by software via the transistor connected upstream.

To be able to process the analog signals from the sensor device at the ATmega328p, two 1:1 voltage dividers have been installed. This limits the maximum input voltage at the microcontroller's side to $2.5V$.

The signal generated by the Reed contacts for capturing the wind speed is processed by using the Input Capture Unit of the microcontroller. The necessary pullup resistor is provided externally to ensure a fixed voltage reference. The internal pullup resistor may be too big due to production variations and may lead to undetectable signals.

The whole system runs at $12.5MHz$ system clock speed.

[1]The mainboard is a product of http://tuxgraphics.org

3.2 Software

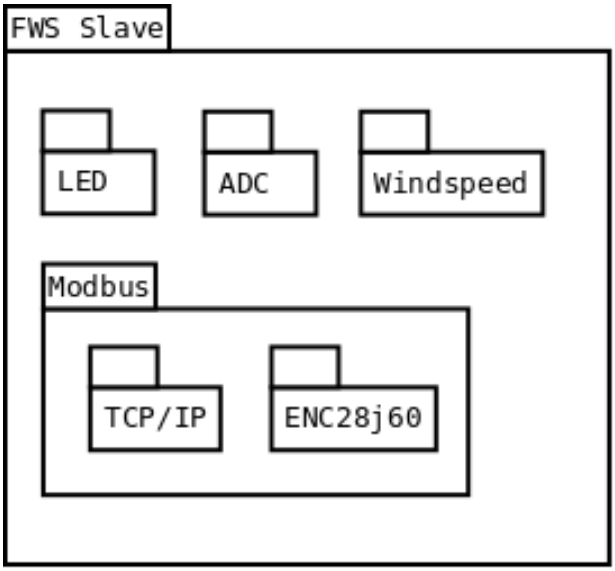


Figure 3.4: Overview of software modules

Main FWS Slave module

The main loop of the program uses polling to check for new data arriving from the Ethernet chip.

All other functionalities are implemented in submodules which are using interrupts and callback routines to return the results. In general, problems of parallel processes are avoided as interrupt routines cannot be interrupted.

Modbus module

In respect to the TCP/Modbus jargon the slave acts as a Server.

This module consists of two submodules which encapsulate the access to the ENC28J60 via SPI as well as the implementation of the protocols TCP/IP, ICMP and ARP. These submodules where taken from „tuxgraphics TCP/IP stack, 3rd generation"[2,3] and adapted as needed.

[2] http://tuxgraphics.org/common/src2/article09051/

[3] http://tuxgraphics.org/electronics/200905/embedded-tcp-ip-stack.shtml

The Modbus module itself is an in-house development. It currently supports the Modbus function codes for writing and reading single registers. It also has the possibility to request the device identifier.[4]

Registers (variables located in the main module) that should be available for Modbus transfers are registered using the module's API.

Analog signals - Temperature and wind direction

The Analog-Digital-Converter unit is encapsulated by the ADC sub module designed for subsequent capturing of multiple analog channels. Since the ATmega328p has only one ADC unit just one channel can be converted at once.

As this application needs to sample two channels this has to be done one after the other. To ensure safe channel switching, the ADC unit is driven using single conversions with highest possible prescaler providing maximum resolution.

Naturally, analog signals are subject to interference leading to non-precise results. In order to achieve a better data quality, the ADC module samples the current channel 256 times and calculates an average value before triggering the callback function and before switching to the next channel.

Index signal - Wind speed

The wind speed is triggered by a Reed contact in the sensor device generating a frequency of $2.53Hz$ per $1m/s$ wind speed.

This signal is captured using the Input Capture Unit of the 16-bit Timer 1. Counting the elapsed time between two index pulses allows to calculate the actual wind speed. This wind speed module also includes specific limits for valid data. For instance, in cases where there's hardly any wind and the wind wheel comes to stop right on top of the Reed contact, input capture interrupts are triggered permanently causing the calculation result to reach unfeasible values. These cases can be avoided by dropping all results not matching the specified values of the wind speed sensor.

3.3 Modbus Write Commands

All described parameters are permanently saved to internal EEPROM. In case of an EEPROM error they are reset to their default values. To modify

[4]This function has not been implemented completely as the Java-Modbus-Library at the master does not provide this feature.

the parameters, they can be transferred via Modbus commands (see address reference Table 3.1). All changes are stored in EEPROM automatically.

Address	Description	Valid values	Data format
0	IP address high segments	16-bit unsigned	65535
1	IP address low segments	16-bit unsigned	65535
2	Sensor enable	0 (disable), 1 (enable)	65535
3	k high word	16-bit unsigned	65535
4	k low word	16-bit unsigned	65535
5	d high word	16-bit unsigned	65535
6	d low word	16-bit unsigned	65535
7	div high word	16-bit unsigned	65535
8	div low word	16-bit unsigned	65535

Table 3.1: Modbus Write Addresses

Network

The mainboard supports $10Mbit/s$ T-Base Ethernet LAN. The user can change the IP address according to his needs. The port 502 is fixed and is reserved for Modbus communication.

- Possible values: any IP address (IPv4)
- Default value: 192.168.0.111

Write address: 0

- Value high byte: First segment of IP address (e.g. 192)
- Value low byte: Second segment (e.g. 168)

Write address: 1

- Value high byte: Third segment of IP address (e.g. 0)
- Value low byte: Fourth segment (e.g. 111)

IP address change sequence: First write to address 0 then to address 1. Changes will take effect when the complete address has been received and the Modbus response has been sent with the old IP address.

Sensor enable

The microcontroller has a configuration parameter for completely enabling/disabling the the sensor device by switching the connected transistor.

Read and Write address: 2

Possible values:

- 0 = disable
- 1 = enable (default)

Temperature calibration

The temperature calculation uses the following formula:

$$Temperature = (k * Value_{sensor} + d)/div$$

Where the factors k, d and div are responsible for calibration. Sensor values are between 0 and 720. All factors are 32-bit signed values. This means 2 transmission per factor are necessary.

Write addresses: High word of k starts at address 3. Last address is 8 containing the low word of div.

Possible values: any 32 bit signed value

Default values:

- $k = -1134$
- $d = 690000$
- $div = 1000$

Change sequence: Always write all 3 factors with the high word in the order k, d and finally div. Changes will take effect when the lower word of div has been received.

3.4 Modbus Read Commands

The registers listed in Table 3.2 show the available sensor data.

Address	Description	Valid values	Data format
2	Sensor enable	0 (disable), 1 (enable)	65535
3	Wind direction	0..3375 (steps of 225); 32767 = no value	6553.5
4	Wind speed	0..1000 (steps of 1); 65535 = no value	6553.5
5	Temperature	16-bit signed; 32767 = no value	6553.5

Table 3.2: Modbus Read Addresses

Sensor enable

Please refer to Section 3.3.

Read address: 2

Wind direction

This register holds the current wind direction in a range from 0 to 337.5 degrees. As the sensor detects 16 positions the delivered value also has a step size of 22.5 degrees.

The transferred value is multiplied by 10 to transmit the first decimal place.

The value 32767 is transferred if the slave has been disabled (refer to Section 3.3).

Read address: 3

Wind speed

The current wind speed is denoted by values within the range from 0 to 100.0.

The transferred value is multiplied by 10 to transmit the first decimal place.

The value 65535 is transferred if the slave has been disabled (refer to Section 3.3).

Read address: 4

Temperature

This register holds the current temperature. The range is only limited by the sensor (refer to Figure 2.1).

The transferred value is multiplied by 10 to transmit the first decimal place.

Read address: 5

Chapter 4

Master

4.1 About the Master

The master is written in Java and uses the jamod Library[1] for the communication with the slaves using the Modbus protocol. The view is created with the SWT Library[2].

Before the collection process of the measurement values can be started it's necessary to create a configuration. In this configuration all slaves are defined. A slave must be added to the master before it can be used. Each slave has an individual name and IP address. In the configuration phase it's necessary to specify the kind of data the slaves collect. The data that is transferred from the slave to the master is called Input Parameter. To configure the sensor, it's possible to define Configuration Parameters. These values are transferred from the master to the slave. To map the parameters to the memory in the slave, they must be bound to addresses. This happens individually for each slave. So it's possible to collect different sensor data from different slaves. It's also possible to configure the kind of plots that are generated (see Section 4.3).

After the configuration of the slaves the collection process can be started. While each slave is polled in its own thread, a single collector thread collects the data, generates a text file with the information about the current values and draws the plots. The time between polling of the slaves and generation of the output files can also be configured. As soon as the current day changes the values from the last day are aggregated to a history value. The values from the slave are kept for two days so it's possible to build a plot based on hourly data from the last day.

[1] http://jamod.sourceforge.net/

[2] http://www.eclipse.org/swt/

4.2 Implementation

The source code can be downloaded at github[3]. The doc folder[4] contains the javadoc generated source code documentation. In the following list a short overview on the classes and their functionality is provided. For a class diagram, see Figure 4.1.

- Configuration: Parameter, InputParameter, ConfigParameter, SlaveInputBinding, SlaveConfigBinding, Slave
- Save the configuration: PersistencePreferences, all classes with ContentHandler in their name
- Data collection: Slave, MeasurementCollector, Measurement, SlaveInputBinding, MeasurementHistoryController, MeasurementHistory, MeasurementHistoryEntry
- Plotting and Output generation: MeasurementCollector, PlotBase, TimePlot, CurrentPlot
- ModBus: ModbusWrapper, Slave
- View: All classes that start with view (not shown in class diagram)

[3] http://www.github.com/schugabe/fws

[4] http://github.com/schugabe/FWS/tree/master/master/FWS_Master/doc/

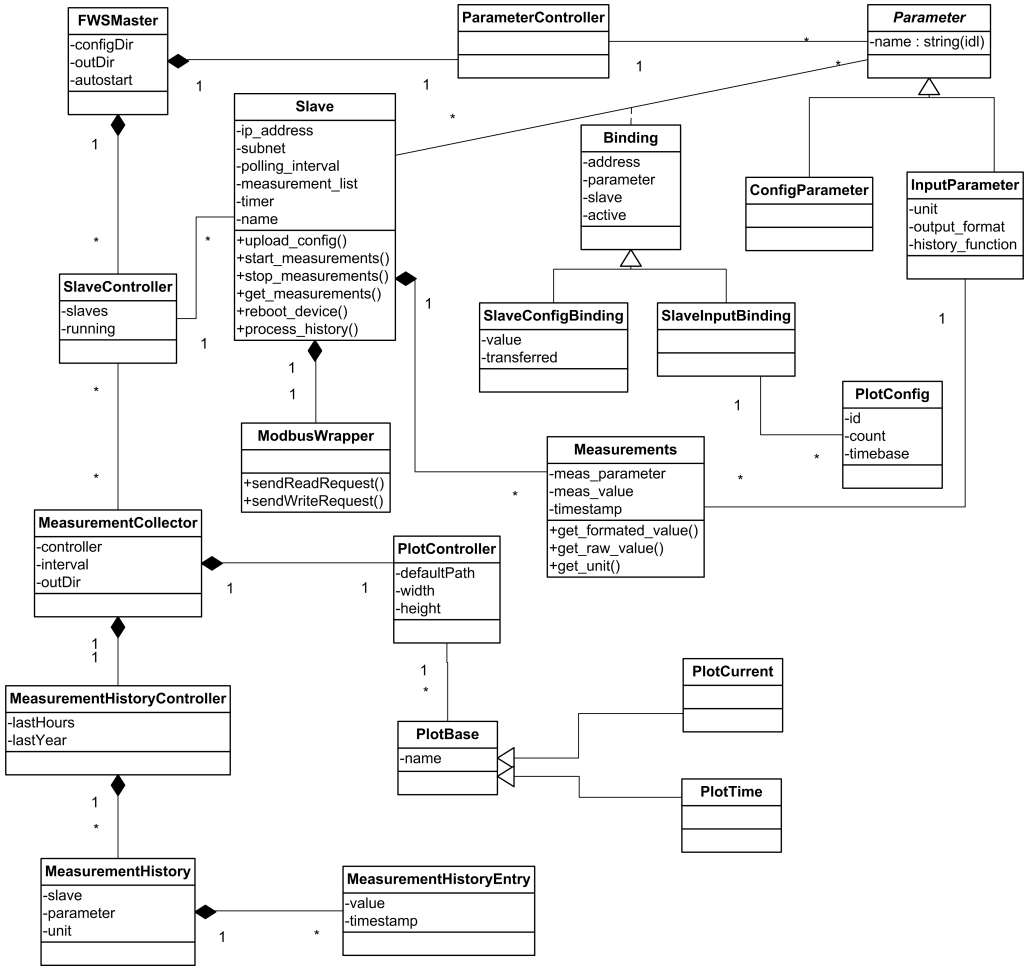


Figure 4.1: Class diagram of the master

4.3 Configuration

The views that support the configuration are shown in Figures 4.6c, 4.6b, 4.5 and 4.7.

The result of the configuration phase is an XML file with all settings in it (Listing 4.1). The location of this file is OS dependent.

- **OS X**: /Users/{username}/Library/Application Support/FWSMaster/settings.xml
- **Linux**: ~/.fws_master/settings.xml

- **Windows**: C:\{User Dir} \.fws_master \settings.xml

Listing 4.1: Sample settings file

```
<?xml version="1.0" encoding="UTF—8" standalone="no"?>
<fws_config>
<path>/home/user/FWSMaster/output</path>
<generatortime>5</generatortime>
<autostart>false</autostart>
<plotwidth>800</plotwidth>
<plotheight>600</plotheight>
<parameter typ="config">
<name>Messintervall</name>
</parameter>
<parameter typ="input">
<name>Temperatur</name>
<unit>°C</unit>
<format>6553.6</format>
<history>MAX</history>
</parameter>
<parameter typ="input">
<name>Windrichtung</name>
<unit>Richtung</unit>
<format>65536</format>
<history>AVG</history>
</parameter>
<station intervall="2" ip="192.168.1.3:30000" name="Dach">
<binding active="true" address="3" parameter="Messintervall"
    transfered="false" type="config" value="1000"/>
<binding active="true" address="0" parameter="Temperatur" plotconfig="
    d4;1h24;" type="input"/>
<binding active="true" address="1" parameter="Windrichtung" plotconfig
    ="d4;c1;h24;" type="input"/>
</station>
</fws_config>
```

Parameters

There are two types of parameters. Input Parameters and Configuration Parameters. Each parameter has a unique name and a boolean value if it's enabled. The name is the identifier of the parameter so it has to be unique.

Beside of these common attributes the Configuration and Input Parameters have further attributes. The parameters must be assigned to slave addresses. This process is referred to as binding a parameter to a slave. To bind a parameter to a slave, the user must define the address on the slave where the parameter is saved.

Input Parameter

Additional attributes of an Input Parameter:

- Unit
- Format
- History Function

Unit The unit is just used for the description in the generated files. Available units are:

- speed $\frac{m}{s}$
- speed $\frac{km}{h}$
- frequenzy Hz
- direction
- temperature $°C$

Format The transferred value from the slave is a 16 bit integer value. To be able to display floating point numbers, it's possible to set the desired output format. Example: The temperature equals $23.3°C$. The slave measures the temperature and converts it to 233. This integer value is then transmitted to the master. The master converts the number back to the desired format.

History Function There are three different history functions available. For more details about these functions and how they are used refer to Section 4.5.

- average
- minimum
- maximum

Configuration Parameter

Additional attributes of a Configuration Parameter:

- value

Value A configuration parameter is a value that is transferred from the master to the slave. The value transferred is saved in the attribute value. This value must be a 16 bit integer value.

Plots

The plots are generated with the help of the jFreeChart Library [5]. The plots can be particularly configured for the input parameters. Each binding can have several plots assigned. It's also possible to plot more than one data into one plot. The plots are configured with a string. The simplest configuration looks like `h24;` With that configuration one plot is generated and the data for this plot are the values from the last 24 hours. The plots are generated in the output directory that can be set by the user.

It's possible to use different data ranges for the plots. To allow the user to choose one range there are three different time bases available:

- c - current
- h - hours
- d - days

Current represents the newest values for the plot. Currently this is only implemented for the wind direction. Figure 4.2 provides an example.

Hours represents the values of the last hours for the plot. The amount of the hours is specified in the plot configuration. It's the number after the timebase. Figure 4.3 provides an example of a 24 hour plot.

Days plots the values from the last days. For more details about the history, see Section 4.5.

Configuration Syntax

Each plot is specified by three different parts: [Number]CharacterNumber;

[5] http://www.jfree.org/jfreechart/

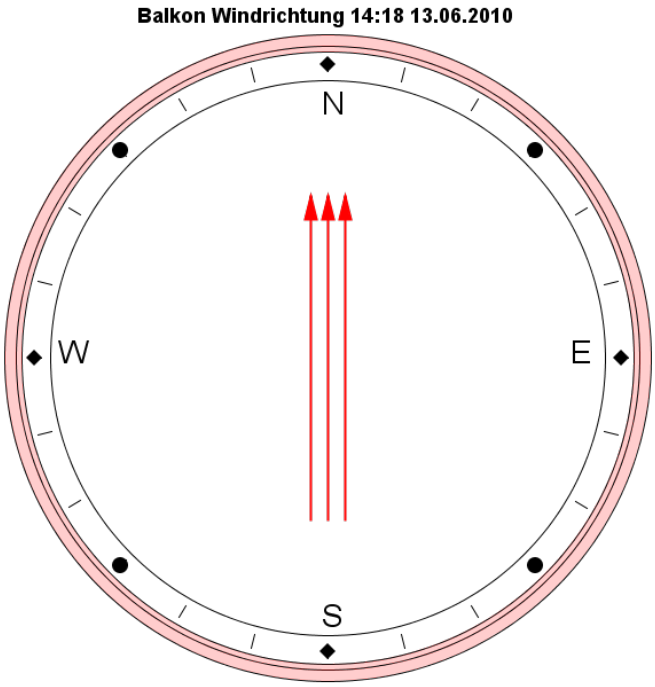


Figure 4.2: Current wind direction plot

First Number This number is optional and controls if more than one input parameter is drawn in one plot. All configured plots with the same first number (ID) are drawn in the same plot.

Character Defines the timebase.

Second Number The second number defines how much data will be in the plot (amount of data). `d4;` will plot the last four days.

End of Configuration Each configuration must end with an `';'`.

Example For the configuration string `h24;1h24;d30;c1;d365;` the following plots are generated:

- Last 24 hours
- Last 24 hours with other data with ID 1
- Last 30 days

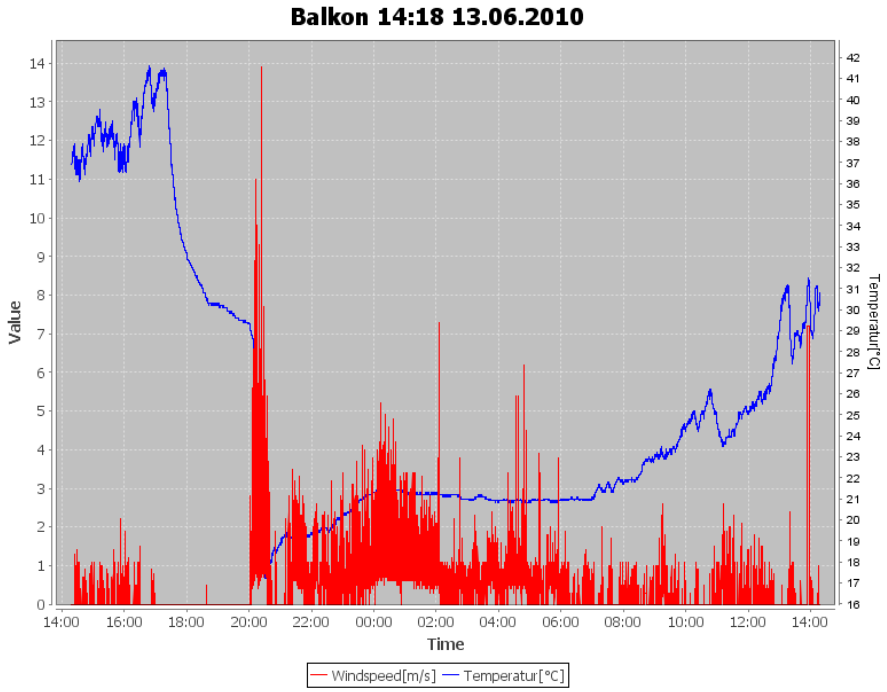


Figure 4.3: Last 24 hours of two separate parameters in one plot

- Current value
- Last 365 days

Detailed Information about the plots

If two different datasets are plotted in one diagram two axes with separate scales will be generated. When more than two data sets should be drawn in one diagram they share one axis. If more plots with more than two datasets have to be generated it is advisable each to keep the diagrams tidy.

The direction parameters values are not connected with a line. Otherwise the diagrams would get unreadable if the wind direction changes a lot.

The direction parameter values are mapped to a description of the direction. 0° results in N(orth) (see Figure 4.4)

4.4 Slaves

After the slaves are configured they are added to the slave controller. This controller takes care for starting and pausing the specific threads.

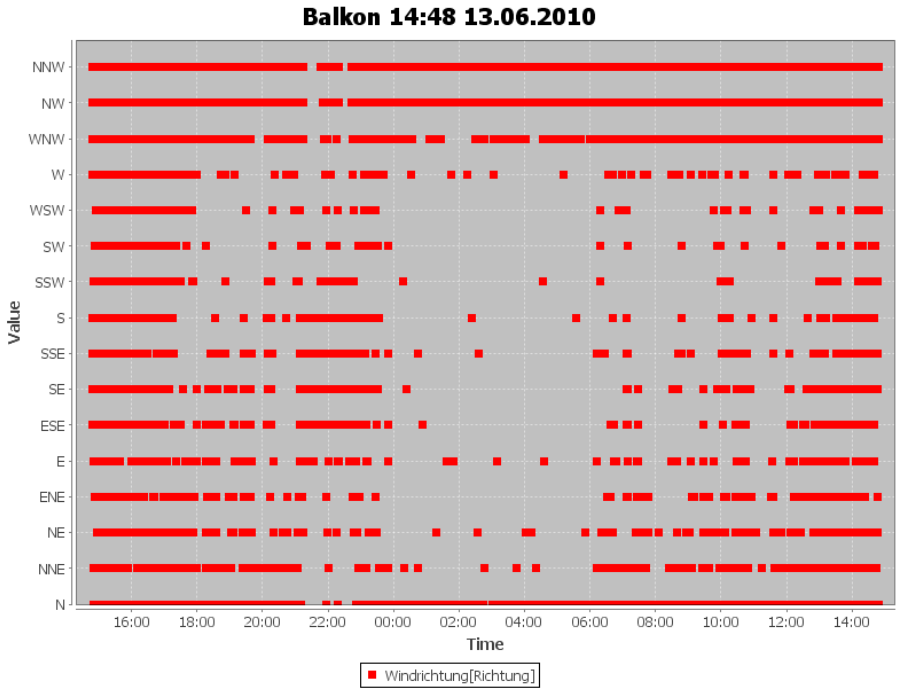


Figure 4.4: Axis description of direction parameter

Transferring the Data

Each slave has its own thread assigned. This thread polls in its own interval the data values from the slave. For each transmission a new TCP connection is opened. After the value has been transferred the connection gets closed. This is necessary as the TCP stack of the slave doesn't support keep alive signals. Leaving the connection open would otherwise result in timeouts. Each slave has a list of measurements. A measurement consists of a timestamp and a value. These lists are collected from the Output Generation thread. If a slave is paused its thread gets suspended and waits for the wake up signal from the controller.

The Modbus function that is used to transfer the data is named sendReadRequest.

Transferring the configuration

When the user wants to upload a new configuration the slave will call the method sendWriteRequest for each Configuration Parameter and reads back the answer of the slave. If the answer equals the previously sent value the

value is marked as transferred. A transferred value isn't uploaded again if it is unchanged.

Change the IP Address

The IP Address represents a special configuration parameter. It is always mapped to the address 0 and 1 on the slave. These two 16 bit registers are used to save the IP address. When the IP address has been changed it's transferred to the slave. After the successful transmission of the two values the new IP is saved. Otherwise the old IP is kept.

Collecting the Data

A data collector thread runs in background an collects the data from all the slaves. For each slave/parameter combination a distinct list is kept with all the recent values in it. Before saving the measurements they are converted into a simpler data type that can be serialized and does not have references to other classes.

In the collector thread the text file is generated that contains the information about the current status of the slaves. Each slave is represented in this file. The information written in this file are the current values of the Input Parameters bound to that slave. This value is the average of the measurements since the last run of the collector. To be able to see how the value changed the standard deviation is calculated and written to the output. An example of a result file is given in Listing 4.2. For each parameter a new line is started in the file. The syntax is `name[unit]:value;standard deviation;`

Listing 4.2: Sample result file

```
16:53:36 13.06.2010
Balkon
Windspeed[m/s]:0.0;0.0;
Temperatur[°C]:25.474999999999998;0.0452267016866652;
Windrichtung[Richtung]:232.5;138.14518054963375;
====
Dach
Windspeed[m/s]:0.0;0.0;
Temperatur[°C]:26.672999999999998;0.03345864;
Windrichtung[Richtung]:232.5;138.14518054963375;
====
```

4.5 History

The collector adds the measurements to the history controller. This controller converts the values to the entries in the history. Each input parameter has two histories: one short term history and one long term history.

Short term history

In the short term history entries from the last two days are kept. This history is queried when plotting a diagram with the 'h' timebase. To this list all new measurements are added as soon as new data arrive from the collector. If the day has changed the previous day is transferred to the long term history. There are at least the last 24 hours in the short term history.

Long term history

As soon as a new day has started the past day gets transferred to the long term history. To avoid too many values in this list one representing value is calculated for the day. This happens with one of the history functions listed in Section 4.3. During the calculation of the history value all measurements older than the last day are removed from the short day history. The representing value is added to the long term history.

Storage of the history

The short and long term history are serialized and transferred to the hard disk. After new data has been added to the history it's saved to the hard disk. To prevent the user closing the master during the I/O operations a semaphore is used. [6]

An old history is kept to prevent loosing the history if the master crashes during writing the history. Before saving the history the thread locks the semaphore. Then, it renames the current history to another filename. Finally, the new history is saved, and the semaphore is released.

If an exception arises during loading the history the master tries to load the old history file. If that results in an exception a new history is created.

[6] Without that semaphore we often had corrupted history files because the master closed at a critical instant.

4.6 View

The view is created with the help of SWT. SWT uses the OS drawing APIs to draw the widgets. The look and feel of the application is very good integrated in the OS it runs on.

Screenshots

Figures 4.5, 4.6, 4.7 (Page 31) and 4.8 (Page 32) show the screenshots of the master.

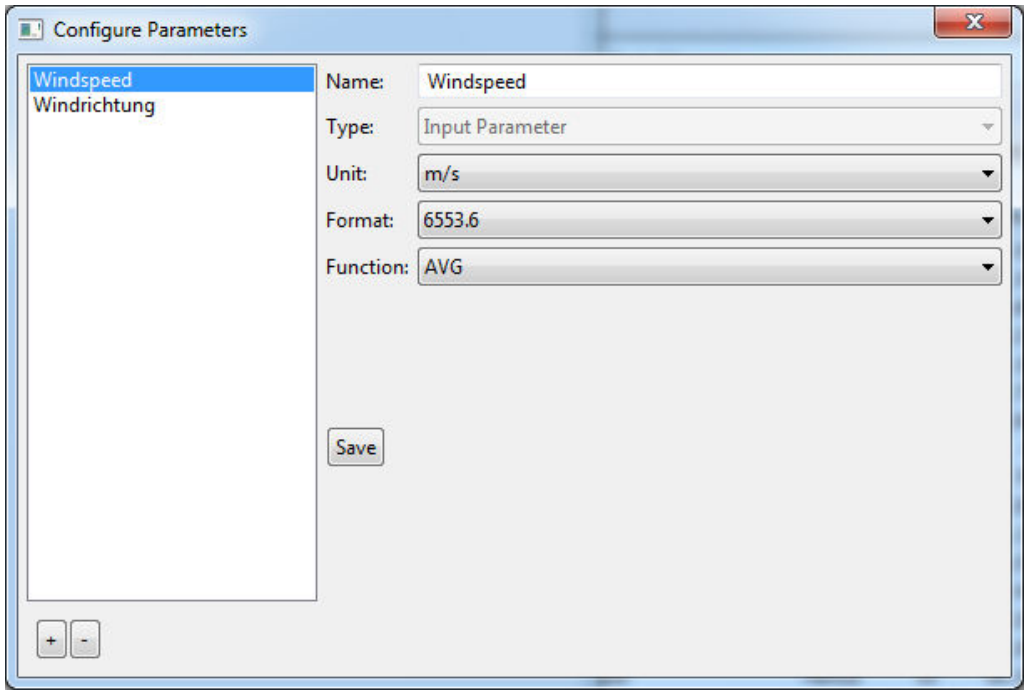
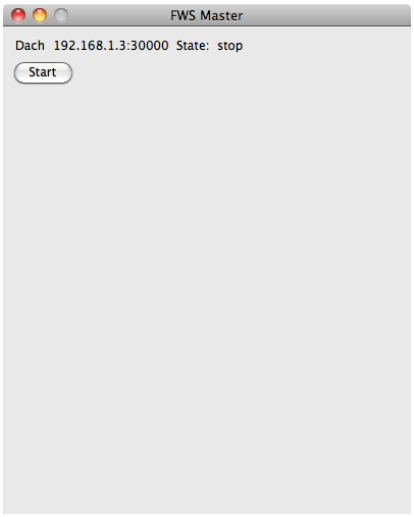
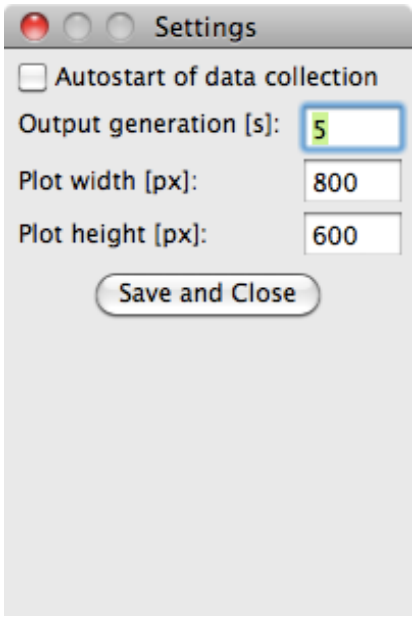


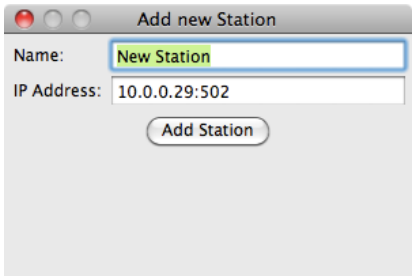
Figure 4.5: Adding/editing of the parameters



(a) Quick overview of the slaves and their status



(b) Configuration of the basic settings



(c) Add new Slave

Figure 4.6: Screenshots of FWS Master

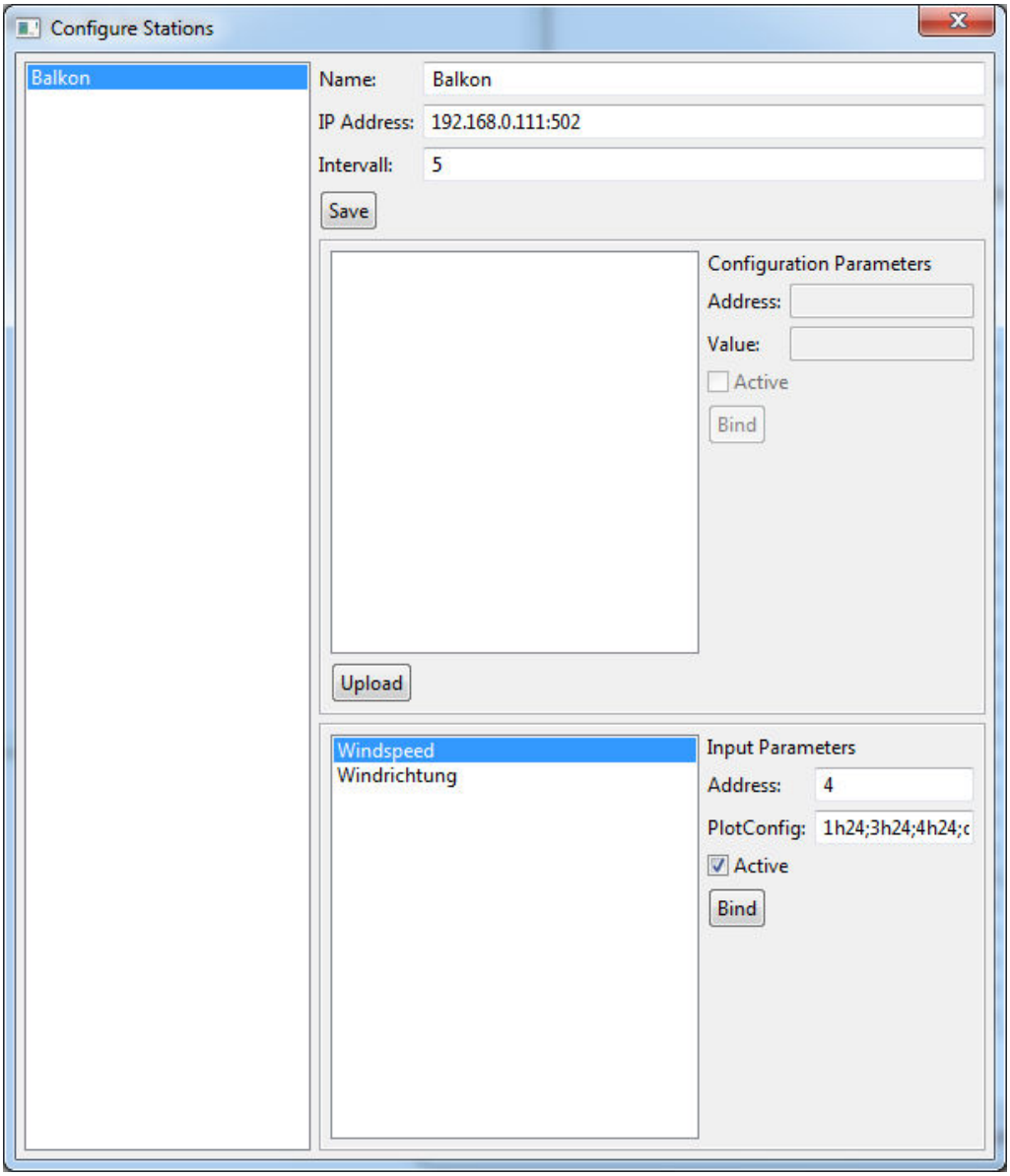


Figure 4.7: Editing slave specific settings. The parameter bindings are also set here

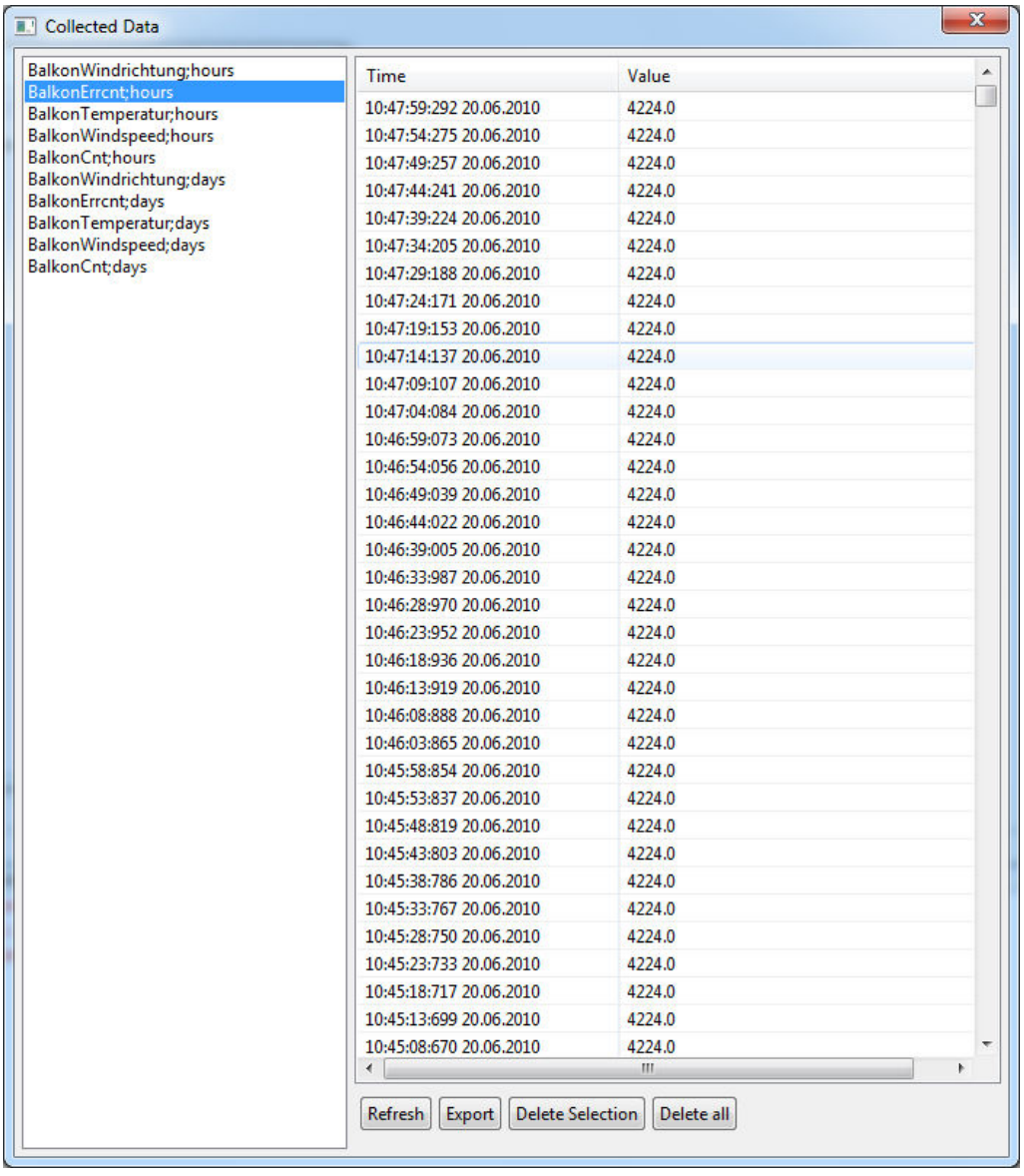


Figure 4.8: Shows the currently collected and saved data