Arttu Räsänen

Measurement system in a wooden apartment building

Data acquisition system and web user interface

Thesis

Spring 2015

School of Engineering

Electrical Automation Engineering

Seinäjoen ammattikorkeakoulu

SEINÄJOKI UNIVERSITY OF APPLIED SCIENCES

SEINÄJOKI UNIVERSITY OF APPLIED SCIENCES

Thesis abstract

Faculty: School of Engineering

Degree programme: Automation Engineering

Specialisation: Electrical Automation

Author: Arttu Räsänen

Title of thesis: Measurement system in a wooden apartment building: Data acquisition system and web user interface

Supervisor: Ismo Tupamäki

Year: 2015 Number of pages: 83 Number of appendices: 19

The main goal of this thesis was to design an automated data acquisition system for measuring various environmental variables in a wooden apartment building. Measurement data is collected from the structures, rooms and outside the building via a distributed I/O-system containing several types of environmental sensors. Sensor data is acquired from the automation devices via Ethernet automation bus and OPC UA data transfer protocol. After that, the data is transferred into a terminal computer, where it is inserted into a relational database. Finally, the visualisation of the data is performed by an HTML 5 web application running on the terminal computer of the system.

Based on these requirements, a small-scale demonstration of the data acquisition system was realised. The demo system was built with a single Beckhoff CX -series PLC device equipped with Beckhoff KL -series input-/output modules, a terminal computer and sensors manufactured by Wika, Siemens and Vaisala. Software was developed for the PLC device and the terminal computer to demonstrate the functioning of the measurement system. This thesis includes an introduction to some basic concepts and technologies related to various kinds of measurements, relational database management systems and the OPC UA data transfer protocol.

Keywords: building automation, measurement technology, OPC UA, R, Python

SEINÄJOEN AMMATTIKORKEAKOULU

Opinnäytetyön tiivistelmä

Koulutusyksikkö: Tekniikan yksikkö

Koulutusohjelma: Automaatiotekniikan koulutusohjelma

Suuntautumisvaihtoehto: Sähköautomaatiotekniikan suuntautumisvaihtoehto

Tekijä: Arttu Räsänen

Työn nimi: Puukerrostalon mittausjärjestelmä: Tiedonkeruu ja web-käyttöliittymä

Ohjaaja: Ismo Tupamäki

Vuosi: 2015 Sivumäärä: 83 Liitteiden lukumäärä: 19

Opinnäytetyön tavoitteena oli suunnitella puukerrostalon rakenteisiin sijoitettava, automatisoitu mittausjärjestelmä erilaisten ympäristömuuttujien mittaamiseen. Mittaustietoja kerätään erilaisilla antureilla puukerrostalon rakenteista, huoneista sekä rakennuksen ulkopuolisista muuttujista väyläliitäntäisten hajautettujen tuloyksikköjen kautta. Mittaustietojen siirto järjestelmän pääte-PC:lle tapahtuu Ethernet-automaatioväylän sekä OPC UA -tiedonsiirtoyhteyskäytännön välityksellä. Pääte-PC:llä mittaustiedot syötetään ja taltioidaan relaatiotietokantaan. Pääte-PC:llä on HTML 5 -pohjainen käyttöliittymä järjestelmän tuottamien mittaustulosten tarkastelemiseen.

Varsinaisen mittausjärjestelmän laitteistoa vastaava pienen mittakaavan esittelyjärjestelmä rakennettiin tiedonkeruun toimivuuden esittelemiseksi. Esittelyjärjestelmä rakennettiin käyttäen Beckhoffin CX-sarjan PLC-laitetta, Beckhoffin KL-sarjan analogiatuloyksiköitä, pääte-PC:tä sekä Wikan, Siemensin ja Vaisalan valmistamia lämpötila- ja ilmankosteusantureita. Opinnäytetyön tuloksena kehitettiin ohjelmat PLC-laitteelle sekä pääte-PC:lle mittaustietojen keräämiseen, taltioimiseen sekä esittämiseen. Tämän opinnäytetyön teoriaosuus sisältää johdatuksen mittaustekniikkaan, SQL-kyselykieleen, relaatiotietokantoihin sekä OPC UA -tiedonsiirtotekniikkaan.

Avainsanat: kiinteistöautomaatio, mittaustekniikka, OPC UA, R, Python

Table of contents

Tables and figures

Abbreviations

OPC UA	OPC Unified Architecture; a communication protocol used in industrial automation.
Python	General-purpose, interpreted high-level programming language.
R	An extensible, open-source programming language for statistical computation and for producing graphics.
RDBMS	Relational Database Management System; a software application for storing and accessing large amounts of data.
SQL	Structured Query Language; a declarative programming language for interacting with relational database management systems.
PLC	Programmable Logic Controller; a digital computer typically used for controlling machines and processes in industrial automation systems.
IEC 61131-3	International standard for PLC programming languages.
FBD	Function Block Diagram, an IEC 61131-3 -compliant graphical programming language for PLC devices
ST	Structured Text, an IEC 61131-3 -compliant textual programming language for PLC devices
IL	Instruction List, an IEC 61131-3 -compliant textual programming language for PLC devices
UML	Unified Modeling Language, a graphical language to visualise the structure, interactions and activities of software systems

1 Acknowledgments

First of all I wish to thank Mr. Ismo Tupamäki, lecturer at Seinäjoki University of Applied Sciences, for his advice and guidance during this thesis project. I am particularly grateful for the invaluable assistance given by Mr. Niko Ristimäki, lecturer at SeAMK.

I also wish to acknowledge the help given by Mr. Teppo Lepistö, the area sales manager at Beckhoff Automation Oy. My special thanks are extended to my wife for her great support and encouragement for finishing this project.

2 Introduction

Wood is a building material with many good properties. Wood is a light-weight material, yet dense and capable of bearing considerable loads. It also features favourable heat insulating and acoustical properties. All these properties make wood a worthy material to be used in many kinds of building and structural engineering projects. By combining different wood species in the structures of a building, it is possible to build stable, economical houses with good acoustical, fire safety and heat-insulating properties.

Long-term measurements in wooden buildings would not enable only the investigation of the building physical properties of existing wooden structures and buildings, but also the research of building physics and development of new structural engineering methods. This measurement data could be used for validating the mathematical models and calculation programs that are currently used in the structural engineering of wooden structures. This would facilitate the adoption of wood, an environmentally friendly and natural material, as a building material of choice for buildings.

2.1 Goals

The main goal of this thesis was to design an automated measurement system for a wooden apartment building. Another goal was to think of a user-friendly, graphical way to present measurement results for the users.

This thesis was produced as a part of a project “Wooden apartment house as a study environment, stage 1”. It is a joint project and its parties are Seinäjoki University of Applied Sciences, Tampere University of Technology and Lakea Oy (construction company). The goal of this project is to produce a complete plan and cost estimating documents for a measurement system that will perform long-term measurements of several structural and environmental parameters inside a wooden building. The construction of the wooden apartment building, called “Mäihä” will begin in Seinäjoki, Finland in 2016.

2.2 Structure of the thesis

This thesis is divided into three parts. In chapter 3, the reader is given a general overview about the theoretical background of the technologies and processes that were used during the development of the thesis. This chapter introduces measurement technology in automation systems, measuring devices and background information related to various types of common physical measurements, such as temperature, humidity and solar radiation measurements. The reader is given an introduction to the SQL query language and two popular relational database management systems (RDBMS). Finally, the main characteristics of the OPC Unified Architecture data transfer protocol are presented and illustrated.

Chapter 4 focuses on the structure and design of an automated measurement system. It acquaints the reader with the characteristics and components of the measurement system. As the whole measurement system consists of several interconnected hardware and software components, the structure and functionalities of each of these components are explained with a set of simplified block diagrams. The main functionalities of the data collection and data acquisition programs and the web-based user interface are shown. The essential parts of the source code of the programs that were developed during this project are presented in the appendices.

3 Theory

Performing measurements is an essential task especially in the fields of technology, engineering and physics. Because numerous branches of technology and engineering are based on experimental studies instead of exact science, they rely on the measurements performed with a wide variety of measuring devices as their main source of new information. The spectrum of different measuring devices begins from simple liquid-filled thermometers all the way to the arrays of GPS satellites which are, in principle, sophisticated and complex measuring systems. (Aumala 1989, 1-2.)

The reasons for using measuring devices are numerous. Their main purpose is to augment the basic senses of a human with electronic or electro-mechanical devices that are highly specialised in measuring a certain physical variable. Typically, such measuring device can measure more accurately, more quickly and with a far greater measuring range and reliability than any human could. In addition, certain measuring devices can also measure quantities at which the human senses are not susceptible at all. (Aumala 1989, 1-2.)

Measuring devices can be made to perform measurements continuously and uninterruptedly. The signals produced by measuring devices can be utilised to affect industrial processes or the environments of humans. (Aumala 1989, 1-2.)

According to Aumala (1989, 5-7), industrial automation technology is typically divided into three parts:

- process automation
- manufacturing automation
- building automation.

When performing engineering measurements it is necessary to build a measuring system that has appropriate functions and is fit for the measuring task. Appropriate measuring devices must be selected and correct measurement procedures must be carried out to receive correct measurement results. It is important to notice that the processing or manipulation of measuring signals or the results of the measurement are also a part of measuring technology. It is necessary to process the measurement results to be able to draw conclusions from them. (Aumala 1989, 5-7.)

The structure of a general measuring system is shown in Figure 1. The parts of a measuring system can be categorised into six parts. According to Aumala (1989, 5-7) a typical measurement system consists of:

- a measurement target such as a physical variable
- measurement environment consisting of a sensor, its environment and its signalling
- both local and remote signal transfers
- signal pre-processing and processing
- signal storage
- usage of measurement data for various purposes.

In industrial automation systems the measurement data is processed hierarchically. The physical variable to be measured is converted into an electrical signal by the sensor and transmitter inside a measuring device. A local transfer link is used to transfer the electrical signals into the automation system, where the signal is processed, stored and possibly manipulated. The processed signal can then be used for different purposes, such as diagnostics, data logging, control or alarms. (Aumala 1989, 5-7.)

Various measuring devices are often combined into measuring systems. It is important to choose and use measuring devices and sensors that have output signal types suitable for interfacing with the rest of the measuring system. Electrical signals are the most common output signals in the measuring devices used in industrial automation. In analogue measurements either voltage or current signals are used. Current signaling is very common, especially in industrial environments where reliability and noise-free operation over great distances is important. A typical current output signal type is the 4..20 mA standard signal. (Aumala 1989, 5-7.)

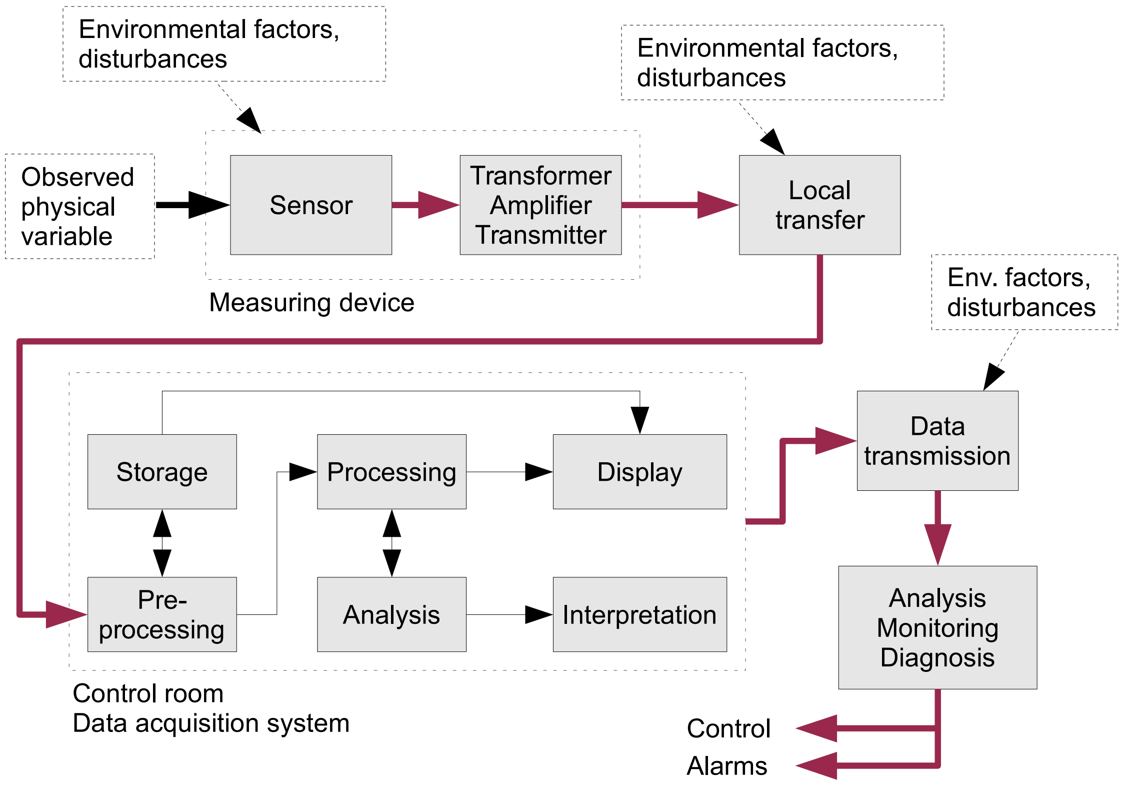

Environmental factors, disturbances
Environmental factors, disturbances
Observed physical variable
Sensor
Transformer Amplifier Transmitter
Local transfer
Measuring device
Env. factors, disturbances
Storage
Processing
Display
Data transmission
Pre-processing
Analysis
Interpretation
Analysis Monitoring Diagnosis
Control room
Data acquisition system
Control
Alarms

Figure 1. General structure of a measuring system. (Aumala 1989, 5-7.)

A typical modern measuring device used in industrial automation applications consists of a sensor element for detecting the measured physical variable and a digital signal processing system. The digital signal processing system converts the sensor signal into a standard electrical signal that is suitable for interfacing with automation systems. Standard signals in automation technology are current and voltage signals. The block diagram in Figure 2 shows the general structure of an electronic measuring device. (Aumala 1989, 5-7.)

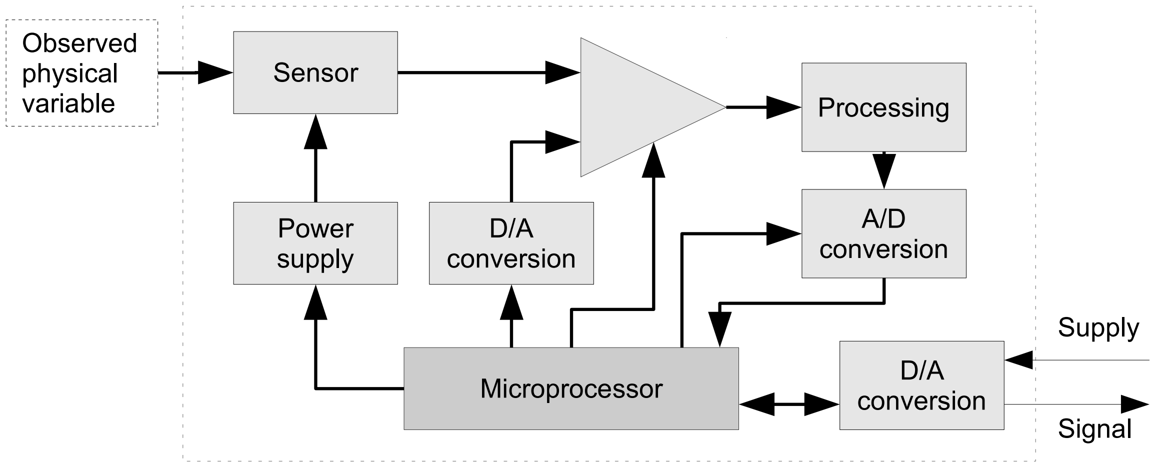


Figure 2. Structure of a measuring device transmitter. (Aumala 1989, 5-7.)

3.1 Temperature measurements

Temperature measurements offer essential information about physical processes in various branches of science, physics, engineering and also in industrial applications. The unit of temperature in the SI system is kelvin (K). The temperature differential of 1 K equals 1 degree Celsius (ºC). (Aumala 1989, 137-139.)

Several types of devices have been developed for measuring temperature. Such devices include bi-metal sensors, gas pressure sensors, thermocouples, resistance sensors and optical pyrometers. This section covers perhaps the most important type of temperature sensors: resistance sensors. Resistance sensor is a very common type of temperature sensor in many applications, including industrial, process and building automation. (Aumala 1989, 137-139.)

Resistance temperature sensors can be categorised into two main groups based on their construction and working principle. Temperature-dependent resistors, also known as thermistors form the first main group. Platinum temperature sensors are the second main group. Thermistors are resistors that are made of ceramic or polymer materials and they feature either a positive (PTC) or negative (NTC) temperature coefficient. Negative temperature coefficient means that the resistance of the thermistor decreases with increasing temperature. (The Resistor Guide 2015.)

In Figure 3, the temperature/resistance characteristic of a typical NTC thermistor is presented. As can be seen from the logarithmic Y-axis, NTC thermistors have a highly non-linear temperature-resistance relationship which is also highly sensitive to the changes in temperature.

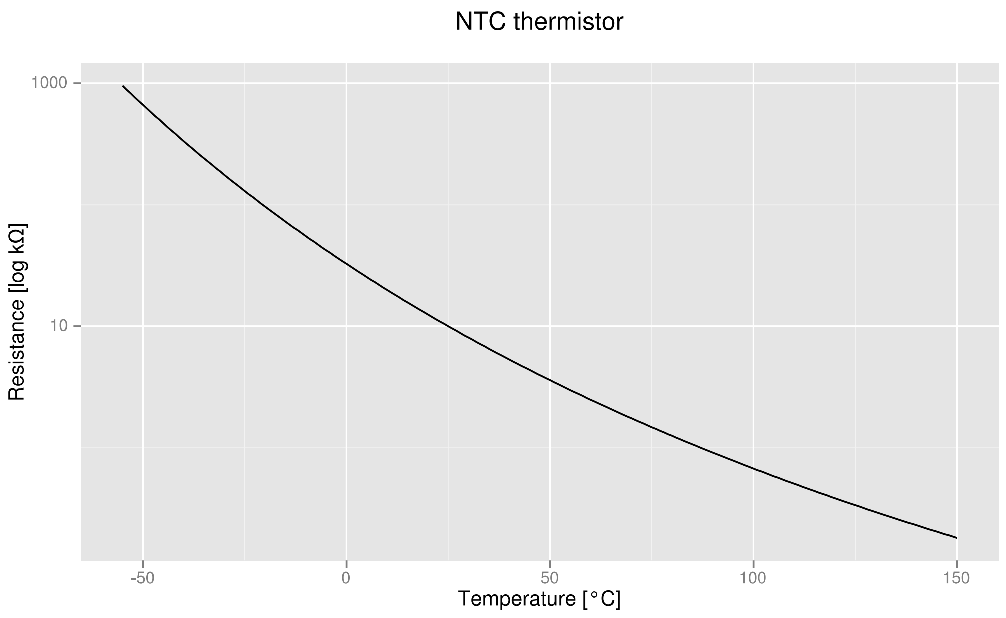


Figure 3. Characteristic curve of an NTC thermistor. (With data from: Vishay [Ref. 10 February 2015].)

NTC thermistors are used in wide variety of applications. They can be used in temperature measurements or temperature control applications. NTC thermistors are also useful as current-limiting devices and temperature compensation due to their highly non-linear behaviour. They are also used in automotive tasks, domestic appliances and telecommunication applications. (Vishay 2012.)

NTC thermistors are manufactured in a wide variety of package styles. Figure 4 shows just one of the many package styles that the manufacturer (Vishay) has to offer from its standard-precision NTC thermistor product line. (Vishay 2012.)

Figure 4. NTC thermistor. (Vishay 2012.)

The second group of temperature-dependent resistors are platinum sensors. They are commonly used in industrial applications and precision measurements. The most common types of platinum temperature sensors are the Pt100 and Pt1000 sensors. The sensor elements of these types of temperature sensors have a resistance value of 100 Ω and 1000 Ω at the temperature of 0 °C. (Aumala 1989, 137-139.)

Platinum temperature sensor elements have a positive temperature coefficient that has more linear temperature-dependent change than that of NTC thermistors. This makes them suitable for precision measurements. Figure 5 shows the resistance/temperature characteristic of a typical Pt100 temperature sensor element. Platinum temperature sensors can be used for measuring temperatures from -25 °C up to +850 °C. Certain specially constructed platinum sensors allow measurements up to +1000 °C. (Aumala 1989, 137-139.)

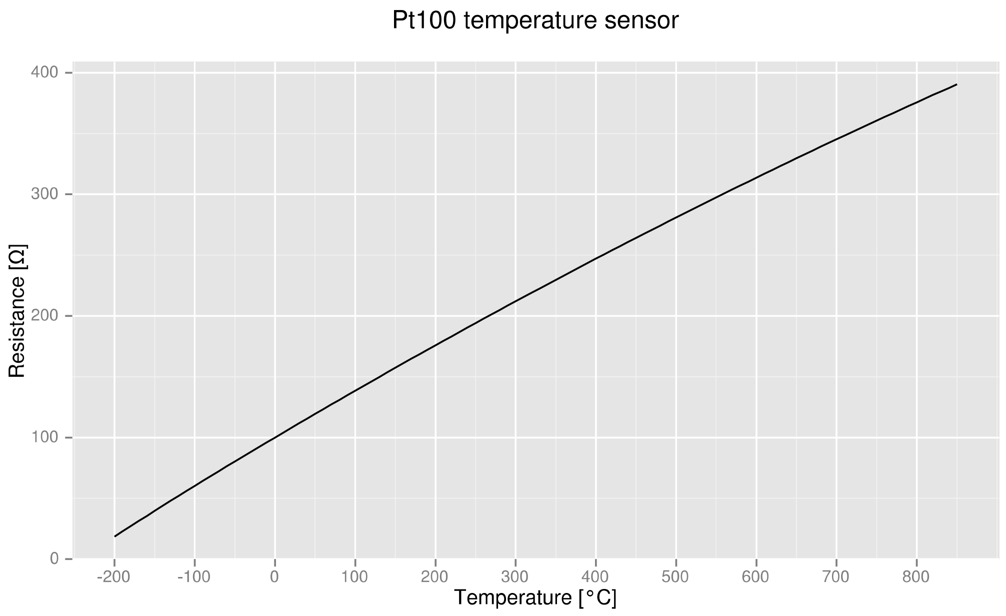


Figure 5. Characteristic curve of a Pt100 sensor. (With data from: Heraeus Holding 2015.)

Figure 6 shows a temperature sensor with a Pt1000 sensing element and an integrated temperature display. Its standard measuring range is -20..+80 °C. It features an analogue 4..20 mA current output signal and a switch output. (WIKA Group 2014.)

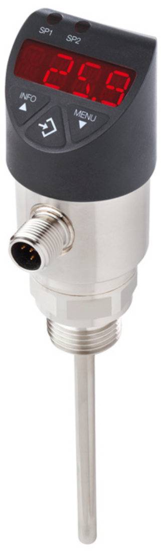


Figure 6. WIKA TSD-30 Pt1000 temperature sensor. (WIKA Group 2014.)

The varying resistance values of temperature-dependent resistors must be converted into measurement signal by means of an external power supply. This conversion is usually done in the transmitter of a sensor unit by means of a resistor bridge circuit. An example of a bridge circuit is the Wheatstone bridge shown on Figure 7. (Aumala 1989, 156-157.)

The resistance values of resistors R_1 and R_2 are fixed, while R is adjustable. R_x is the unknown resistance, such as the sensing element of a temperature-dependent resistance sensor. U is a voltage source. G is a galvanometer, an instrument that is used for measuring small electrical currents. (Aumala 1989, 156-157.)

In normal situation the resistance value of the adjustable resistor R is adjusted to match the resistance value of the unknown resistance R_x. In this case, the currents I_1 and I_2 will be equal. Because of this, the nodes “a” and “b” will be in the same potential and no current will flow between the nodes and through the galvanometer. When the value of R_x changes, the circuit will be shifted off from the equilibrium and currents I_1 and I_2 will no longer be equal. a potential difference is developed between the nodes “a” and “b”. As a result, an electrical current will flow through the galvanometer, indicating a change in resistance R_x. (Aumala 1989, 156-157.)

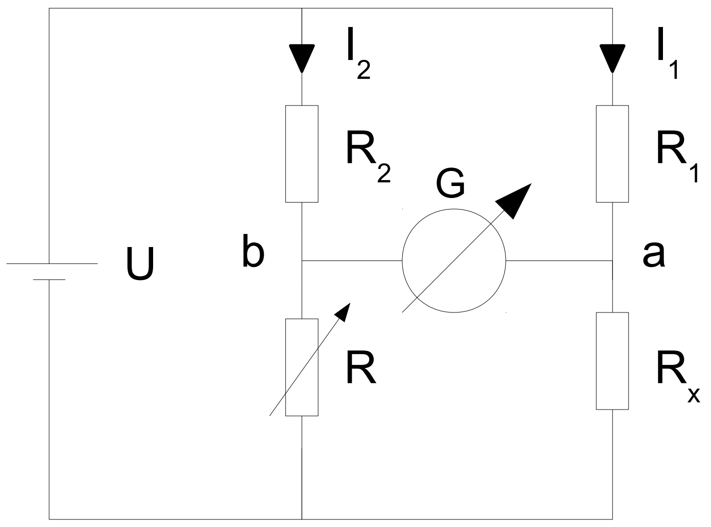


Figure 7. Wheatstone bridge. (Aumala 1989, 156-157.)

The voltage difference U_{ab} between the nodes “a” and “b” can be calculated with the following equation. (Aumala 1989, 156-157.)

$$U_{ab} = U \cdot \frac{R}{R + R_2} - U \cdot \frac{R_x}{R_x + R_1} \quad (1)$$

Another way to use the Wheatstone bridge circuit is the circuit shown in Figure 8. In this circuit, an operational amplifier is used to amplify the potential difference between nodes “a” and “b”. (Aumala 1989, 159.)

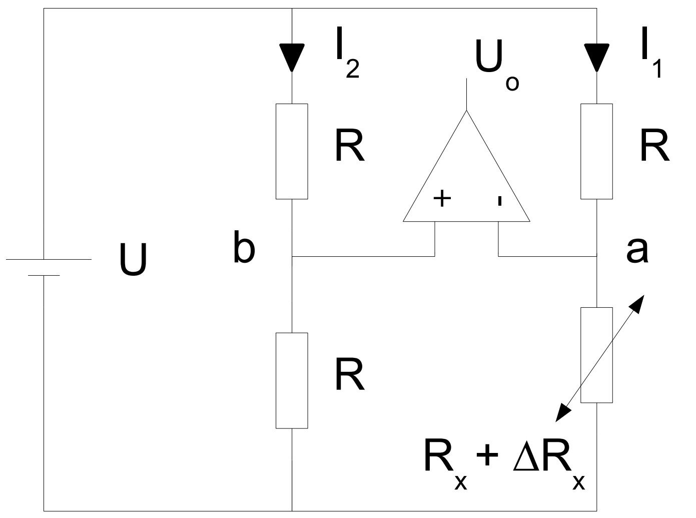


Figure 8. Transmitter circuit of a temperature sensor. (Aumala 1989, 159.)

The output voltage U_o of the operational amplifier in Figure 8 is calculated with the following equation. (Aumala 1989, 159.)

$$U_o = -\frac{AU}{4R} \cdot \Delta R_x \tag{2}$$

3.2 Humidity measurements

Humidity measurements give essential information about the amount of water contained by gaseous, liquid and solid materials. Determining the water content is significant in many chemical and technical processes as it may have severe negative effects in the materials used in said processes. It may induce corrosion and rusting in certain metal materials. High levels of air humidity may also promote the growth of micro-organisms, such as mould, yeast and fungi in wooden structures. Acceptable air humidity levels are also important for ensuring comfortable and healthy living conditions. (Härkönen, S. & Lähteenmäki, I. & Välimaa, T. 1992, 55-57.)

The moisture content of gases, such as air, is described with the following concepts and physical units:

- absolute and maximum humidity f [g/m³]
- relative humidity φ [%]
- condensation temperature [ºC]
- water content [dimensionless].

Absolute humidity describes the amount of water vapour contained by a certain mass of gas or gas mixture. It is expressed as grams per cubic metre. (Härkönen, S. et al. 1992, 55-57.)

$$f = \frac{m}{V} \tag{3}$$

where m = mass of water vapour

V = volume of gas mixture.

Maximum humidity describes the maximum amount of water vapour that the gas or gas mixture is able to contain at the saturation temperature. Maximum humidity depends on the temperature and it can be determined by measurements. (Härkönen, S. et al. 1992, 55-57.)

Relative humidity is the ratio between absolute humidity value and the maximum humidity value. It is most commonly expressed as percentage. (Härkönen, S. ym. 1992, 55-57.)

$$\varphi = \frac{f}{f_s} \cdot 100\ \% \tag{4}$$

where f = absolute humidity

f_s = maximum humidity

The condensation temperature describes a certain temperature where a mass of gas or gas mixture becomes saturated by water vapour. In other words, the relative humidity of the gas is 100 % at the condensation temperature. If the temperature of the gas is lowered further from the condensation point, the moisture will condensate into water droplets. Water content is the ratio of the mass of moisture contained by a gas or gas mixture and the mass of dry gas. (Härkönen, S. et al. 1992, 55-57.)

Several technologies have been developed for measuring the relative humidity levels of gases. The oldest device that is still sees wide-spread use in humidity measurements is the psychrometer. It is also called “wet-and-dry-bulb thermometer” due to its working principle. A psychrometer is basically a combination of so-called “wet” and “dry” temperature sensors. These sensors are usually built into a single measuring device. The “dry” temperature sensor is in direct contact with the ambient air, while the “wet” temperature sensor is covered by a piece of cloth or cotton that is kept wet at all times when the psychrometer is in operation. While the ambient air flows past the “wet” temperature sensor, it becomes saturated with water vapour, causing the water in the wet cloth to evaporate. Evaporation of water consumes energy and causes the temperature of the “wet” temperature sensor to decrease by a certain amount. The humidity of the ambient air can then be calculated by using the difference of the temperature values between the “dry” and “wet” sensors. (Härkönen, S. et al. 1992, 58-60.)

Humidity measurement is also possible by means of electrical techniques. Two common techniques of electrical humidity measurement are resistive and capacitive methods. Resistive humidity measuring device shown on Figure 9 contains a resistive element that is placed on a non-conductive surface. When humidity of the ambient air condenses on the resistive element, the resistance value between the connecting leads of the element changes accordingly. Capacitive humidity measuring device consists of two strips of conductive material placed on a non-conductive surface, in a similar structure as resistive devices. These two strips of conductive material form a capacitor whose capacitance is affected by the humidity of the ambient air. (Härkönen, S. et al. 1992, 58-60.)

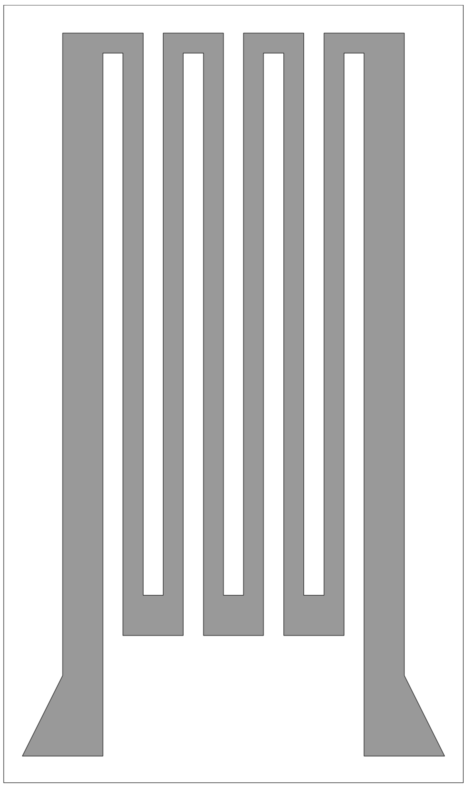


Figure 9. Resistive humidity sensor. (Härkönen, S. ym. 1992, 58-60.)

Both resistive and capacitive techniques have the disadvantage of being susceptible to deposits and impurities on the sensor surface. These impurities are a source of measurement error. (Härkönen, S. et al. 1992, 58-60.)

The types of capacitive humidity sensors HUMICAP® and INTERCAP® developed by Vaisala Oy are based on thin-film technology. Structure of a HUMICAP® humidity sensor is shown on Figure 10. HUMICAP® sensors are known as accurate measuring instruments that feature good long-term stability and negligible measuring hysteresis. HUMICAP® humidity sensor consists of four main parts (Vaisala Oy 2012 .):

- a substrate made out of metallised glass or ceramic material
- polymer thin-film layer coated with porous metal material
- conductive electrodes
- conductor leads for signal output.

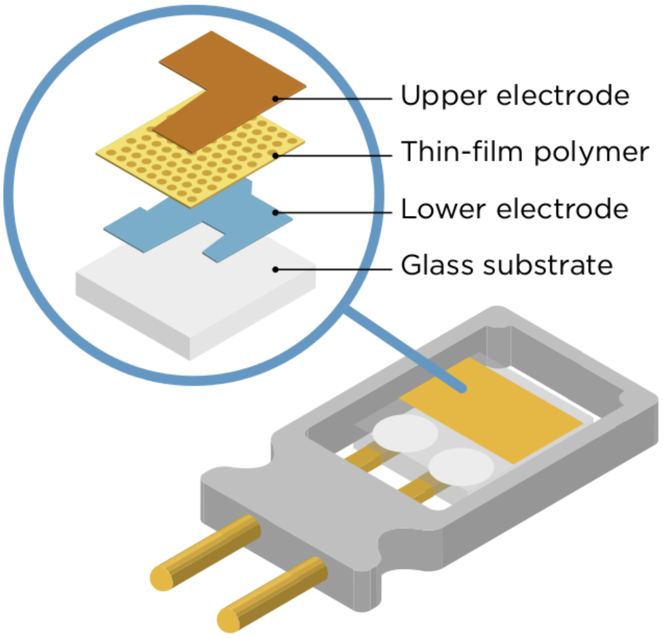


Figure 10. Structure of Vaisala HUMICAP® humidity sensors. (Vaisala 2012.)

HUMICAP® sensors can be used to measure relative humidities at a range of 0..100 % and their nominal accuracy is ±1 % RH. HUMICAP® sensor devices are insensitive to dust and other impurities due to their structure and working principle. The sensors are equipped with various filters, further protecting the sensor element from dust particles and contaminants. HUMICAP® sensors can also be equipped with electric heating elements, allowing operation in condensing environments. (Vaisala 2012.)

Vaisala INTERCAP® humidity sensor (Figure 11) is based on the same working principles as HUMICAP® sensors. They come pre-calibrated at the factory, thus eliminating the need for calibration before commissioning the device. INTERCAP® sensors are fully interchangeable. They are used in applications that require relative humidity measurement accuracy of ±3 % RH. (Vaisala 2012.)

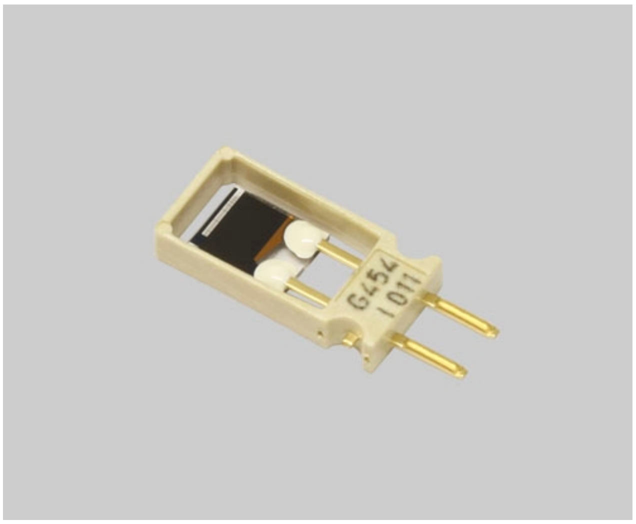


Figure 11. Vaisala INTERCAP® humidity sensor element. (Vaisala 2015.)

Assembled HUMICAP® and INTERCAP® sensor units feature different measuring probes for different uses. The units meant for industrial automation applications feature several types of signal outputs, such as various levels of voltage signals and also optional outputs for digital signals and dew point measurement. A separate current loop converter module is available for use with 4..20 mA current signals. (Vaisala 2015.)

3.3 CO_2 measurements

CO_2 (carbon dioxide) measurements are usually done by using indirect measurement methods. In such methods, the carbon-dioxide content of a gas or gas mixture is sensed indirectly from some other property of the gas that is also affected by it. One common indirect CO_2 measurement method is to utilise a physical phenomenon called infra-red absorption. The absorption of infra-red (IR) radiation in different media depends heavily on the molecular content of a particular gas. The absorption is also affected by the wavelength of the radiation. Infra-red absorption is characterised by the Lambert-Beer law, which describes the attenuation of the radiation as it passes through medium. (Aumala 1989, 147-148.)

$$I = I_0 \, e^{-\varepsilon(\lambda)\rho l} \tag{5}$$

where I = intensity of radiation

$\varepsilon(\lambda)$ = coefficient of attenuation at a wavelength of λ

ρ = density of medium

l = length of the material

I_0 = original intensity of radiation

Different gas molecules have different infra-red absorption characteristics. Figure 12 represents the absorption characteristic of carbon dioxide gas.

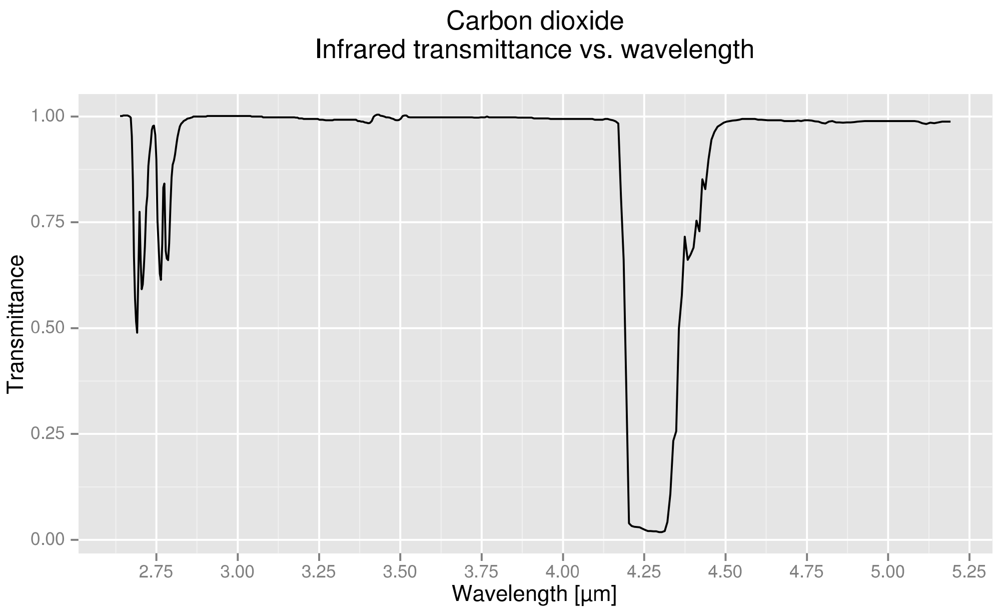


Figure 12. Infrared transmittance of carbon dioxide gas. (With data from: National Institute of Standards and Technology 2011.)

As can be seen from the shape of the characteristic curve in Figure 12, the transmittance of infra-red radiation through carbon dioxide gas is very low at a wavelength of 4,26 µm. This means that a large portion of the radiation is absorbed by the gas molecules. Hence, the amount of infra-red radiation transmitted through the gas or gas mixture depends on the amount of carbon dioxide molecules present in the gas. This attenuation of the infra-red radiation can then be measured by a special infra-red sensor. (Vaisala 2012.)

The CARBOCAP® carbon-dioxide sensor was developed by Vaisala Oy in 1992. It is a silicon-based infra-red absorption sensor with a special Fabry-Perot interferometer that can be adjusted by an electric signal. It features high reliability, accuracy and stability due to a continuously active built-in reference measurement at a wavelength where infra-red transmittance is high. In this way, possible changes in the intensity of the infra-red radiation source caused by contaminants can be automatically adjusted. Figure 13 shows the structure of a Vaisala CARBOCAP® infra-red carbon dioxide sensor. (Vaisala 2012.)

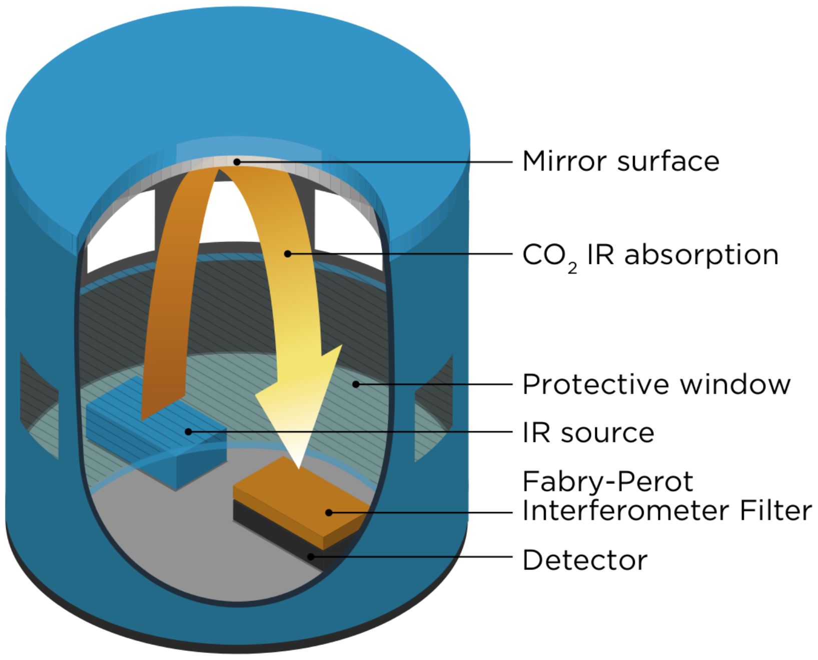


Figure 13. Structure of a Vaisala CARBOCAP® carbon dioxide sensor. (Vaisala 2012.)

CARBOCAP® sensors are available for measuring carbon dioxide content of gases for different gas concentrations and for different use cases. CARBOCAP® technology is used in various fields of technology, such as building and industrial automation, security systems, life science applications and in environmental research. (Vaisala 2012.)

3.4 Precipitation measurements

Precipitation describes the amount of rainfall falling on a certain area during a certain time period. Usually in European countries precipitation is expressed in units of millimetres. One millimetre of precipitation equals one litre of rainwater, distributed evenly over an area of one square metre. (Suomen Ilmatieteen Laitos [Ref. 6 March 2015].)

Precipitation can be measured electronically by using a number of mechanical measuring instruments. The most common type of mechanical measuring instrument is called a tipping-bucket rain gauge. Its working principle is illustrated in Figure 14. This type of device contains a funnel or cone which collects the rainfall and two small water containers, that are carefully balanced and mounted on a pivoting "see-saw" mechanism. The area of the mouth of the funnel must be known and the capacity of the containers is typically 0.1 or 0.2 millimetres. (Belfort Instrument [Ref. 3 February 2015].)

When a certain amount of precipitation is collected into one of the containers of the mechanism in Figure 14, the weight of the water causes the pivoting "see-saw" mechanism to tip over, emptying the first container and lifting the second (currently empty one) directly below the output orifice of the funnel. This process is repeated as long as enough precipitation falls into the funnel of the measuring device. (Belfort Instrument [Ref. 3 February 2015].)

Also attached into the see-saw mechanism is a small permanent magnet, which passes by a stationary reed relay with each swinging motion of the mechanism. The magnetic field of the permanent magnet swinging nearby the reed relay causes the relay's electrical contacts to make connection, causing a short electrical signal. This electrical signal indicates that a certain amount of precipitation has been collected by the device. (Belfort Instrument [Ref. 3 February 2015].)

The electrical signal produced by the reed relay can be used as an input into a weather station or weather data acquisition system. For example, a digital input module with capability of recording high-speed events could be used for counting the total tipping motions made by the “see-saw” mechanism per time period.

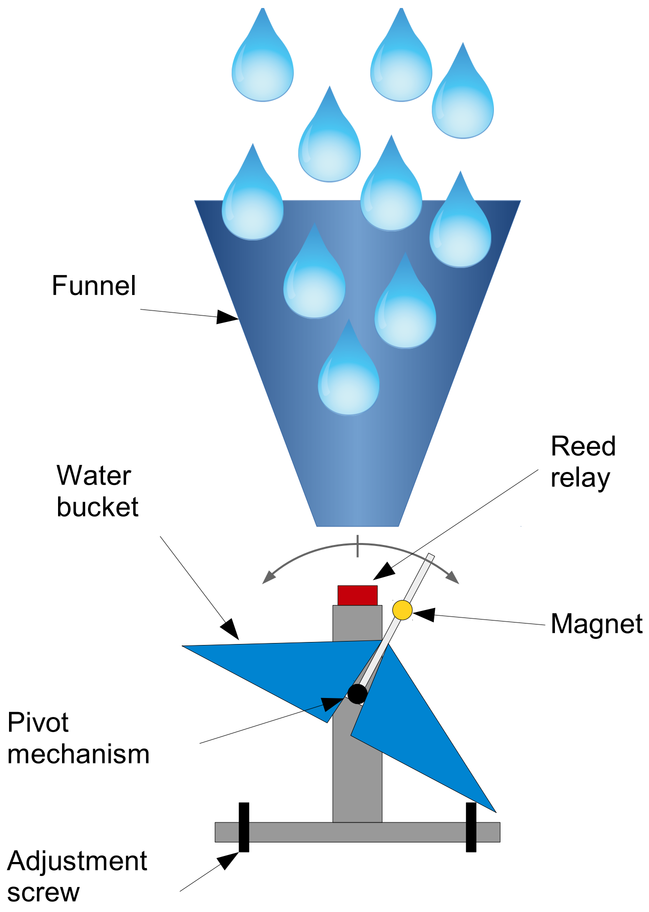
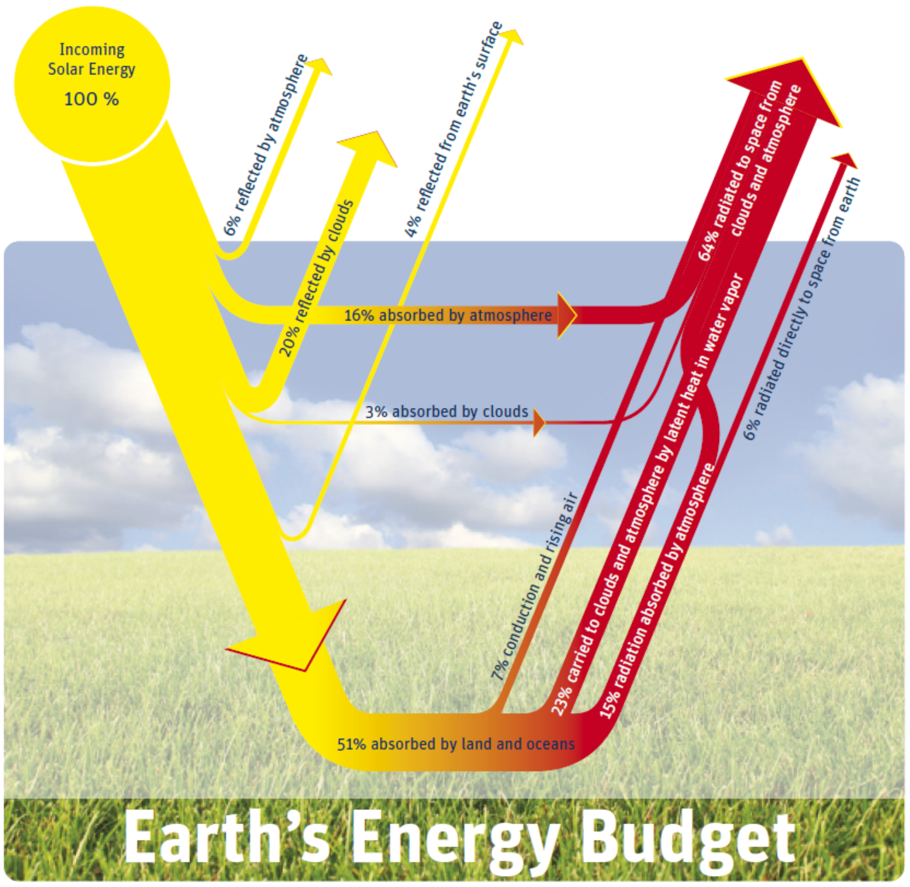


Figure 14. The working principle of a tipping-bucket precipitation sensor. (Belfort Instrument [Ref. 3 February 2015].)

Tipping-bucket precipitation sensors can be equipped with heating resistors which allow operation in freezing conditions. An example of such device is the RG13J/H rain gauge produced by Vaisala Oy. This tipping-bucket precipitation sensor features a heating resistor with a rated power of 33 Watts. The heater can be used with power supplies with an output voltage of either 24 V (RG13J) or 48 V (RG13H). (Vaisala 2014.)

3.5 Solar radiation measurements

The amount and type of solar radiation falling on the surface of the Earth depends on the atmospheric conditions. The wavelengths falling on the Earth range from long-range infra-red radiation (λ = 780 nm..50 µm), visible light (λ = 400..780 nm) to ultraviolet radiation (λ = 100..780 nm). The total irradiance of the solar radiation falling on the Earth's surface is called the Solar Constant and its measured value is approximately 1367 W/m^2. Figure 15 shows a schematic view of the incoming and radiated solar energy of the Earth. (Kipp & Zonen B.V. [Ref. 9 March 2015].)


Incoming Solar Energy 100 %
6% reflected by atmosphere
20% reflected by clouds
4% reflected from earth's surface
16% absorbed by atmosphere
3% absorbed by clouds
64% radiated to space from clouds and atmosphere
6% radiated directly to space from earth
7% conduction and rising air
23% carried to clouds and atmosphere by latent heat in water vapor
15% radiation absorbed by atmosphere
51% absorbed by land and oceans
Earth's Energy Budget
Figure 15. Schematic representation of the energy balance of the Earth. (Kipp & Zonen B.V. [Ref. 9 March 2015].)

The global horizontal irradiance (GHI) that falls on to the surface of the Earth consists of two parts. The first part is called diffuse horizontal irradiance (DHI), which is the portion of the solar radiation that is scattered by the Earth's atmosphere. The second part is direct normal irradiance (DNI), which is telling about the amount of radiation that is coming directly from the Sun. When the radiation is falling on the Earth's surface at an oblique angle, the radiation covers larger area of land and it must be corrected by a cosine function. (Kipp & Zonen B.V. [Ref. 9 March 2015].)

The relationship of the two main parts of the solar radiation is described by the following formula: (Kipp & Zonen B.V. [Ref. 9 March 2015].)

$$GHI = DHI + DNI \cdot \cos(\Theta) \tag{6}$$

where	GHI: global horizontal irradiance
	DHI: diffuse horizontal irradiance
	DNI: direct normal irradiance

GHI is measured by using a horizontally mounted pyranometer. Pyranometer is an instrument for measuring the global short-wave (300..2800 nm) solar radiation. (Kipp & Zonen B.V. [Ref. 9 March 2015].)

DNI is measured by an instrument called pyrheliometer. It is an instrument with a narrow (5º) field of view and as such, it must be installed into an automatic sun tracking device seen on Figure 16. (Kipp & Zonen B.V. [Ref. 9 March 2015].)

DHI can be measured by installing a second pyranometer on the sun tracking device, fitted with a suitable sun shade that blocks the direct beam radiation from reaching the sensor of the second pyranometer. (Kipp & Zonen B.V. [Ref. 9 March 2015].)

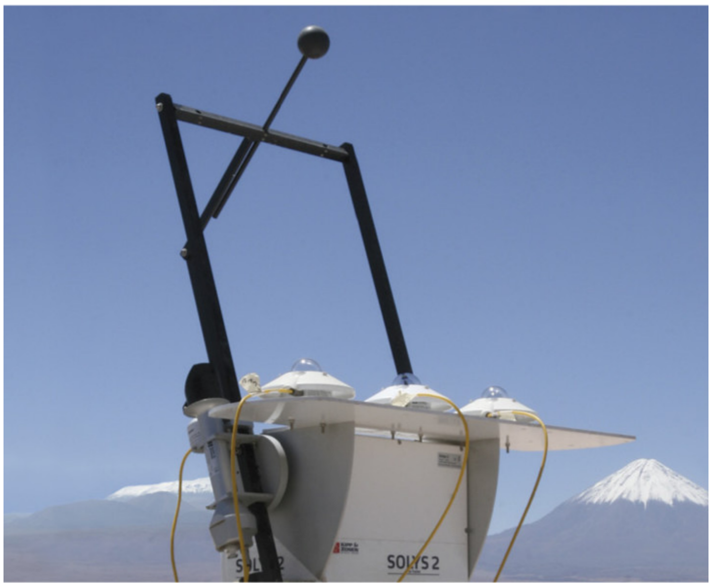


Figure 16. Complete solar measurement setup with measuring instruments mounted on a solar tracking device. (Kipp & Zonen B.V. [Ref. 9 March 2015].)

Another solar radiation measurement device is a pyrgeometer. Pyrgeometers are used for measuring the long-range (4500..42000 nm) solar radiation from the atmosphere. (Kipp & Zonen B.V. [Ref. 9 March 2015].)

3.6 Relational Database Management System (RDBMS)

The structure of relational database is based on the simple and logical structure called the Relational Model of data. The foundations of the Relational Model of data is based on concept of relations, attributes and tuples. (Connolly & Begg 2005, 71-74.)

A relational database consists of data that is logically structured within relations. **Relation** is a named two-dimensional table-like data structure, with certain number of columns and rows. Each relation in a database has an individual name and it contains one or more columns, which are also known as **attributes**. The elements, or rows, of a relation are called **tuples** or records and they contain the individual values of each attribute for that particular record. An example is shown in Table 1.

This small example consists of one **relation** (Branch) containing four **attributes** (columns) and five **tuples** (rows). (Connolly & Begg 2005, 71-74.)

Table 1. Example database relation (table).
(Connolly & Begg 2005, 71-74.)

Branch

branchNo	street	city	postcode
B005	22 Deer Rd	London	SW1 4EH
B007	16 Argyll St	Aberdeen	AB2 3SU
B003	163 Main St	Glasgow	G11 9QX
B004	32 Manse Rd	Bristol	BS99 1NZ
B002	56 Clover Dr	London	NW10 6EU

Relations, or the tables of a database, have the following properties (Connolly & Begg 2005, 76-77.):

- Each relation must have a distinct name in the relational database.
- Each of the cells of the table-like structure of a relation must contain only single or atomic values.
- Each attribute (column) must have a distinct name.
- Each tuple (row) must be distinct. Two tuples with the same content cannot exist in a relation.
- The order of the attributes or tuples has no significance.

Because each tuple in a relation must be distinct and duplicated tuples are not allowed, it is necessary to be able to uniquely identify all of the tuples contained in a relation. This can be achieved by using relational keys. A relational key defined as a **primary key** uniquely identifies all of the tuples in a relation. A simple but useful primary key would be an integer number that is distinct for each tuple. Relational keys defined as **foreign keys** also make it possible to represent relationships between the tuples of different relations (tables) in a database. For example, the tuples (persons) of a Staff relation are related to certain tuples of the Branch relation via a primary/foreign key relationship, such as in the example shown in 1. (Connolly & Begg 2005, 78-79.)

Table 2. Relational key example. (Connolly & Begg 2005, 78-79.)

Branch

branchNo	street	city	postcode
B005	22 Deer Rd	London	SW1 4EH
B007	16 Argyll St	Aberdeen	AB2 3SU
B003	163 Main St	Glasgow	G11 9QX
B004	32 Manse Rd	Bristol	BS99 1NZ
B002	56 Clover Dr	London	NW10 6EU

Staff

staffNo	fName	lName	position	sex	dob	salary	branchNo
SL21	John	White	Manager	M	1945-10-01	30000	B005
SG37	Ann	Beech	Assistant	F	1960-11-10	12000	B007
SG14	Mary	Howe	Assistant	F	1970-02-19	9000	B003

Primary key: branchNo

Foreign key: branchNo

Relational Database Management System is a software application that provides interactions between the user's applications and the database. It also enables the user to define, create, maintain and control access to the database. In addition, relational database management systems have a host of other features and facilities. These facilities enable users to define the database by defining the data types and structures to be stored into the database through a Data Definition Language (DDL). The user may also update, delete and fetch data records from a database through a Data Manipulation Language or DML. (Connolly & Begg 2005, 20-21.)

RDBMS lets users to enter (or insert) new data into a database and also manipulate the data contained by a database in different ways. It provides error recovery control system, which allows the database to be restored to a previous consistent state in case of error in hardware or users' software applications. RDBMS also provides integrity system, which maintains the internal consistency of the data contained by the database at all times. It may also provide other security features, such as granting database access to a certain person or group of persons. (Connolly & Begg 2005, 20-21.)

Figure 17 shows the main components of a typical database management system and how it interfaces with other software components. It also shows the methods for accessing and managing files and retrieving users' queries from the database. The components of DBMS are enclosed by a gray dashed line.

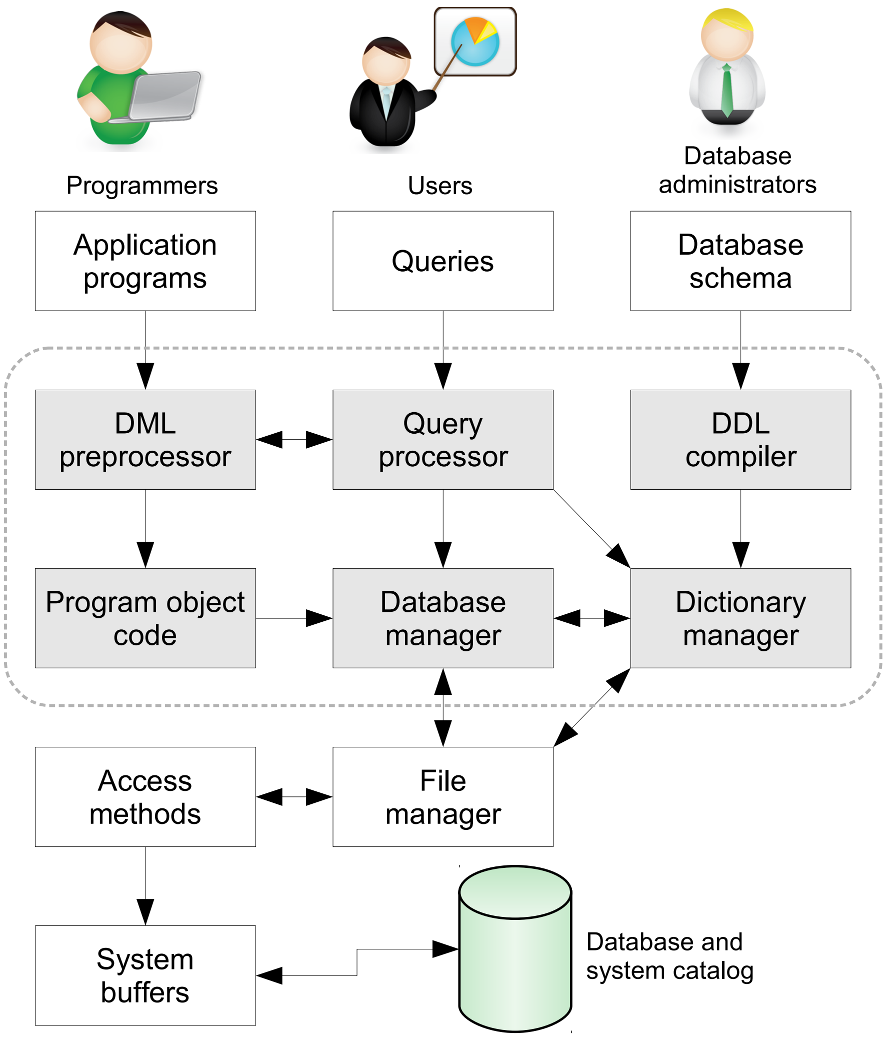


Figure 17. Major software components of DBMS. (Connolly & Begg 2005, 20-21.)

Because a database system conjoins the description of the data (metadata) and the data itself into a central repository, it is possible for the relational database management system to provide access to the data through a more generalised query language. The most common and most widely-used query language for interacting with databases is the Structured Query Language (SQL). Users may interact with databases with application programs. These application programs may be developed with some third-generation programming language, such as C, Java, Python or R and they may feature some fourth-generation language such as SQL embedded in them for accessing the database. (Connolly & Begg 2005, 20-21.)

The SQL language itself is divided into two or four main parts. According to Connolly & Begg (2005, 114-115), the two most commonly used parts of the SQL language are Data Definition Language (DDL) and Data Manipulation Language (DML).

In addition, two additional parts of the SQL language are defined. They are Transaction Control Language (TCL) and Data Control Language (DCL). (Kreibich 2010, 34-35)

The Data Definition Language is used for defining the structure of the database and also for controlling the access rights and other security features of a database. The Data Manipulation Language is used for inserting, updating and deleting data from a database. Modern version of the SQL standard also contains many other features such as the possibility of using IF..THEN..ELSE constructs familiar from procedural programming languages such as C. (Connolly & Begg 2005, 114-115.)

Transaction Control Language is used to ensure the reliability and integrity of the database operations. Transaction Control Language is typically used automatically in conjunction with both DDL and DML operations. Transactions are a way of grouping several low-level DDL or DML database operations into a single logical transaction. By utilising transactions, the integrity and reliability of the database is ensured. (Kreibich 2010, 51.)

Data Control Language is used for granting and revoking access rights to database objects. Different permissions can be applied to both DDL and DML languages. (Kreibich 2010, 51.)

As SQL is a non-procedural language, the user does not have to specify *how* he/she wants to access the data in a database, but instead he/she can focus on *what* information he/she wants to access. The command structure of SQL consists of standard English words, such as SELECT, FROM and WHERE. By convention, SQL code is typically written in upper-case letters even though SQL itself is a case-insensitive programming language. (Connolly & Begg 2005, 114-115.)

The SQL Data Manipulation Language consists of the following statements (Connolly & Begg 2005, 116-118.):

- SELECT: for making queries of the data present in the database
- INSERT: for inserting new data into a table
- UPDATE: for updating previously inserted data in a table
- DELETE: for deleting data from a table.

The SELECT SQL statement is the most powerful of the SQL DML statements. It can be used for building both simple and highly complex database queries. It is capable of performing several different kinds of data processing and relational algebra's set operations in a single SQL statement. Complex queries may be constructed by adding modifiers for filtering, grouping, ordering and aggregating the query results. (Connolly & Begg 2005, 116-118.)

The INSERT SQL statement is used for inserting new rows of data values into a database table. INSERT statements insert the data into each of the database table's columns in the same order as the table was created. It can also be used for copying data rows from table to table. (Connolly & Begg 2005, 116-118.)

The UPDATE SQL statement is used for modifying the data in the database. The updated data must have compatible data types for the corresponding table columns. DELETE statements is used for deleting specific rows from the database table. (Connolly & Begg 2005, 116-118.)

The Data Definition Language in SQL allows database objects to be created and deleted. Such objects may be database schemas, indexes and perhaps most commonly, tables. The SQL standard also contains other database object types. The major DDL statements of SQL are (Connolly & Begg 2005, 168-169.):

- CREATE: for creating database objects
- ALTER: for changing the properties of database objects
- DROP: for deleting database objects.

The CREATE statement is used with the TABLE statement for creating new database tables. The CREATE TABLE statement provides facilities for defining the column names and data types, integrity features, relational keys and other constraints. It should be noted that there are significant differences in the way that the CREATE and CREATE TABLE statements are used in different RDBMS software. (Connolly & Begg 2005, 169-171.)

The ISO SQL standard contains the data types shown on 3. These data types are available for use with CREATE TABLE statements when defining the data types of the columns of a database. There are significant variations in supported data types among different RDBMS software products. For example, the BIT and CHARACTER data types are often referred to as **string** data types. Similarly, the exact and approximate numeric types are often referred to as **numeric** data types by many RDBMS software packages. (Connolly & Begg 2005, 159.)

Table 3. ISO SQL data types. (Connolly & Begg 2005, 159.)

Data type	**ISO SQL declarations**			
Boolean	BOOLEAN			
Character	CHAR	VARCHAR		
Bit	BIT,	BIT VARYING		
Exact numeric	NUMERIC	DECIMAL	INTEGER	SMALLINT
Approximate numeric	FLOAT	REAL	DOUBLE PRECISION	
Datetime	DATE	TIME	TIMESTAMP	
Interval	INTERVAL			
Large objects	CHARACTER LARGE OBJECT		BINARY LARGE OBJECT	

The exact numeric data types are used for storing numbers that have an exact representation. The number may be a standard or small integer or it may have an optional decimal point and sign. Approximate numeric data types, such as FLOAT, may be used for storing very small or very large numeric values. The desired precision of the numeric values can be defined by the database administrator. (Connolly & Begg 2005, 160-161.)

Datetime data can be used for defining points of time in a specified accuracy. The date-time data is processed by the RDBMS software in the ISO standard notation, which divides the data into years, months, days, hours, minutes and seconds and time zone information. (Connolly & Begg 2005, 160-161.)

Slightly similar to the date-time data, interval data types may represent certain periods of time. Interval data consists of two different data types: year-month intervals and day-time intervals. The first one may contain only the fields for years and months, while the latter may only include fields for the days, hours, minutes and seconds. (Connolly & Begg 2005, 162.)

SQL Data Control Language provides facilities for security and access control features. The GRANT statement is used by the database administrator for allowing other users to access the database. The administrator may grant all or partial access rights for particular users. Certain users can be allowed to only execute SELECT statements on certain database tables, while another users may also perform INSERT or UPDATE operations. Each database object created in SQL language has an owner. When a user with sufficient privileges creates a database object, such as table, that user also becomes the owner of the object and he/she receives full access privileges on that particular object. The user may then either GRANT other users the necessary privileges to use the object, or REVOKE them. The GRANT statement may also be used with the modifier PUBLIC to make the database object accessible for all present and future users. (Connolly & Begg 2005, 189-191.)

3.6.1 SQLite

SQLite is a lightweight software package that provides a versatile and full-featured Relational Database Management System (RDBMS). Even though SQLite supports many of the features defined in the SQL92 programming language standard, it has certain special characteristics that differentiate it from the other popular RDBMS software tools. Its hardware and storage requirements are very modest, requiring one megabyte of disk space and approximately four megabytes of memory. SQLite is also a cross-platform program and it can be used under practically any computer operating system and also in many embedded devices. (Kreibich 2010, 1-2.)

Traditional RDBMS software applications such as MySQL or Microsoft SQL Server are based on client/server architecture which necessitates many installation, configuration and administration tasks until the database system can be used. Such database engines are highly complex software packages, consisting of many processes which manage client connections, file input/output and caching, query processing, among other things. For these reasons and for performance concerns, the database instances typically consist of countless files and directories scattered throughout the server computer's file system. To access and use the database, all these components need to be in place and in order. Thus, the database instances may be difficult to move or backup. (Kreibich 2010, 1-2.)

For these reasons it is customary to dedicate a server computer system solely for running the database management system. Suitable software libraries provided by the vendor of the database management system must be loaded into any client application that needs to communicate with the database. The structure of a traditional RDBMS architecture is shown on Figure 18. (Kreibich 2010, 1-2.)

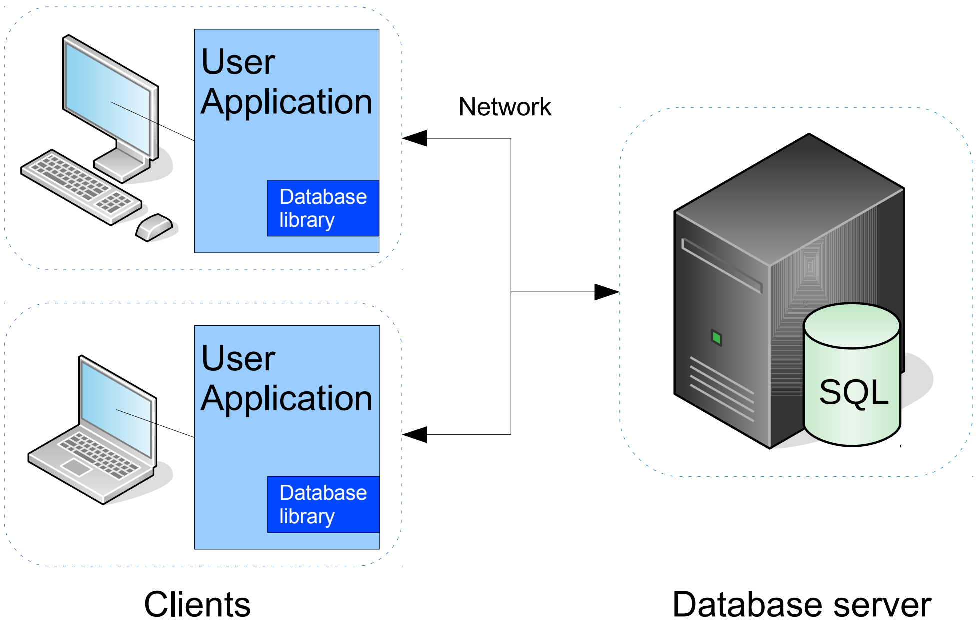


Figure 18. Traditional RDBMS client/server architecture. (Kreibich 2010, 3.)

The general architecture of SQLite is shown on Figure 19. When compared to the complex structure of traditional database solutions, SQLite is based on a very different software architecture. It is also meant for different use cases. While the database engines of traditional RDBMS software applications are typically meant to be run on a separate computer system, the entire database engine of SQLite is integrated into any application that needs to access the database. With SQLite, there is no clear distinction between clients and the database server because the client software application itself works as the database server engine. This “serverless” software architecture has many advantages over traditional SQL databases for certain usage scenarios, but also certain disadvantages. Using SQLite in multi-user, networked applications with per-user access control is difficult or impossible. It is also not suitable for handling high transaction rates or extremely large, multiple gigabyte-datasets. For such applications, a traditional client/server RDBMS software solution is preferred. (Kreibich 2010, 1-2.)

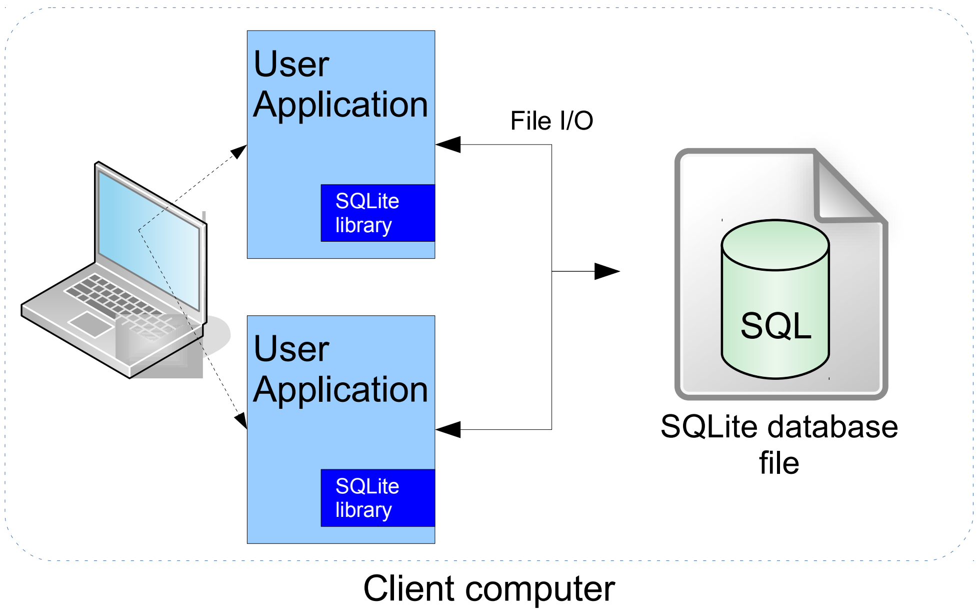


Figure 19. SQLite database architecture. (Kreibich 2010, 3.)

The serverless architecture used by SQLite dramatically reduces the complexity of the database system. With SQLite, it is possible to embed the entire database management engine directly into the user applications that require access to the database. The SQLite database itself resides on the file system of the user's computer as a single regular file. It is very straightforward to move or make a back up of a self-contained SQLite database file, especially when comparing to the steps required to perform similar operations with other full-featured RDBMS software solutions. This simplicity also enables SQLite to be ported into almost any kind of device and environment that is capable of reading and writing regular files. Because SQLite packages the database schema and the data itself into a single file, it is also possible to use them as a way of sharing complete datasets with other users. (Kreibich 2010, 4.)

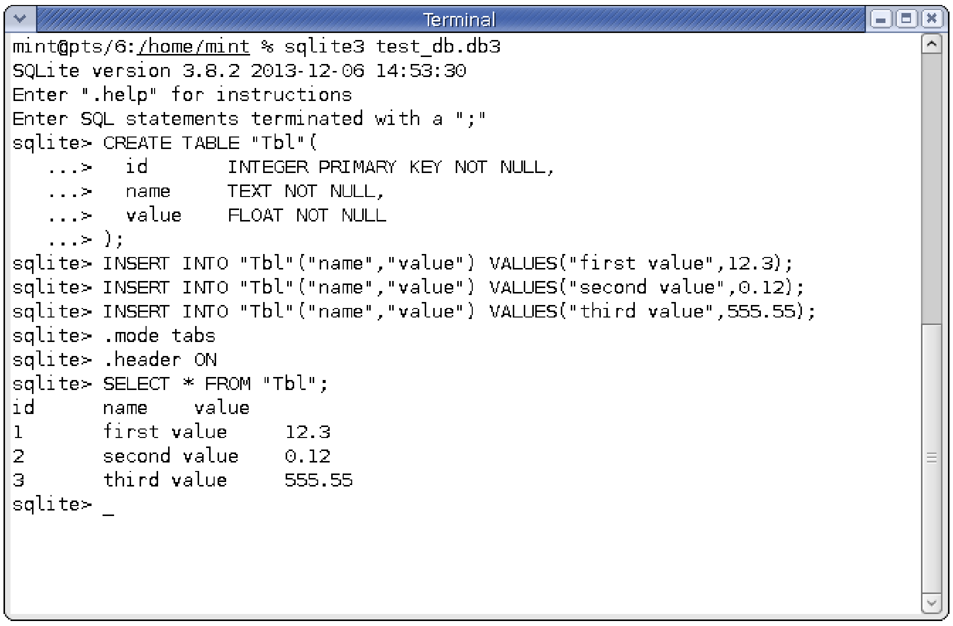
SQLite is mostly compatible with the SQL92 programming language standard, but it also includes its own features, extensions and deviations from the standard. Perhaps the most important difference in SQLite is the use of a dynamic column types. In standard RDBMS software packages the type of information that a column of a database table is able to store is fixed to a certain type. This means that a database column that is defined to store information as integer numbers can only be used to hold integer numbers. Errors would be produced if the user would attempt to insert floating-point data or character strings into a strictly SQL92-compliant database table that is defined to store integer numbers.

In SQLite a total of five column types are also defined, but they are not enforced. This allows the insertion of almost any type of information into any SQLite database table, regardless of the type definitions that the columns were assigned to. Despite this, it is always a good idea to define correct data types for the table columns when designing an SQLite database schema. The data types supported by SQLite are listed on 4. (Kreibich 2010, 36-37.)

Table 4. Data types supported by SQLite. (Kreibich 2010, 36-37.)

Data type affinity	SQLite column storage class
NULL	NULL
Text	TEXT
Integer	INTEGER
Float	FLOAT
Large objects	BLOB

Normal SQLite installations include a program *sqlite* (or *sqlite3*, depending on the version) for interacting with SQLite databases. It is a simple command-line program that is used for creating databases and database tables, reading and writing information from/to databases, performing maintenance tasks and fine-tuning settings of databases. It can be used in interactive mode, where the user enters SQL commands directly into the database engine via command-line interface, or in batch mode where SQL commands are read from SQL script files and executed automatically. Figure 20 shows a sample *sqlite3* session in interactive mode. The *sqlite* or *sqlite3* command-line program is a very useful and simple way to access SQLite databases.



```
mint@pts/6:/home/mint % sqlite3 test_db.db3
SQLite version 3.8.2 2013-12-06 14:53:30
Enter ".help" for instructions
Enter SQL statements terminated with a ";"
sqlite> CREATE TABLE "Tbl"(
   ...>   id       INTEGER PRIMARY KEY NOT NULL,
   ...>   name     TEXT NOT NULL,
   ...>   value    FLOAT NOT NULL
   ...> );
sqlite> INSERT INTO "Tbl"("name","value") VALUES("first value",12.3);
sqlite> INSERT INTO "Tbl"("name","value") VALUES("second value",0.12);
sqlite> INSERT INTO "Tbl"("name","value") VALUES("third value",555.55);
sqlite> .mode tabs
sqlite> .header ON
sqlite> SELECT * FROM "Tbl";
id      name    value
1       first value     12.3
2       second value    0.12
3       third value     555.55
sqlite> _
```

Figure 20. Example *sqlite3* session.

3.6.2 MySQL Server

MySQL Server is the world's most popular relational database management system (RDBMS) that is available under free software license. It is well-suited for heavily loaded production systems and mission-critical applications, but it can also be embedded into mass-produced software applications via MySQL libraries that are linked into the user's application. MySQL Server is the main workhorse behind many high-traffic websites on the internet. (Oracle 2015.)

MySQL Server is very fast, reliable and offers good scalability. It can be configured to work as a personal database server for a single user laptop computer all the way to large database clusters with up to 255 networked server computers. One of the original design goals of MySQL was to develop a database management system for handling large databases much faster than existing solutions. MySQL Server is capable of handling databases containing hundreds of thousands of tables and from 50 million up to five billion records. (Oracle 2015.)

Several different database storage engines exist in MySQL Server, such as InnoDB, MyISAM, MEMORY and ARCHIVE, to name a few. MySQL Server features a pluggable storage engine subsystem that allows storage engines to be loaded and unloaded from a running MySQL Server instance. (Oracle 2015.)

MyISAM was the default storage engine in MySQL Server versions prior to 5.5.5. MyISAM is an engine that is well-suited for many kinds of tasks, such as web services and data warehousing. It is a general-purpose engine that offers the all major SQL features except transactions and allows large database sizes of up to 256 terabytes. (Oracle 2015.)

InnoDB is a more modern engine that has full transactional capabilities (commit, roll back, crash-recovery). It allows databases of up to 64 terabytes to be created and used. (Oracle 2015.)

The ARCHIVE engine is suited for storing extremely large amounts of historical data that is seldom accessed. This engine uses compression to reduce the storage requirements of the data. (Oracle 2015.)

MEMORY is a fast storage engine which uses the system random access memory as its storage space. It is typically used as a temporary data store. When MySQL Server is restarted or shut down, all data that was stored in MEMORY database tables will be irreversibly lost. (Oracle 2015.)

MySQL Server supports most of the ISO SQL data types shown on 3, such as numeric and character data and date and time information. In addition, MySQL includes a few other data types, such as TINYINT, MEDIUMINT and BIGINT for storing very small, medium-size and large integer values. (Oracle 2015.)

The access control and security system in MySQL Server is highly flexible, customisable and secure and it features host-based verification. MySQL Server also encrypts all password traffic between the server and user when the user connects to it via network. (Oracle 2015.)

Users can connect to a MySQL database server via several protocols. All clients may connect to the server via standard TCP/IP connection, and clients running a Unix operating system may also connect via standard Unix domain socket files. Windows clients may connect via named pipes. (Oracle 2015.)

A standard MySQL Server installation also includes *mysql*, a full-featured command-line application that enables the database administrator to access the database interactively or non-interactively. It works in a similar way as the *sqlite3* application shown on Figure 20. (Oracle 2015.)

3.7 OPC Unified Architecture

OPC Unified Architecture (abbreviated as OPC UA) is a data transfer protocol enabling communication between various kinds of automation devices from different device manufacturers. The specification for the OPC UA protocol is provided by the OPC Foundation. The acronym “OPC” originally stood for “OLE for Process Control”. (Mahnke, W. & Leitner, S-H. & Damm, M. 2008, 8-9.)

OPC UA is an evolution from the previous OPC standards, such as the popular OPC Classic that is based on Microsoft's COM/DCOM (Distributed Common Object Model) communication technology. Instead of relying to vendor-specific proprietary technologies such as DCOM, the OPC UA standard specifies two open transport protocols, UA/TCP and SOAP/HTTP that are platform-independent and suited for different purposes. The SOAP/HTTP transport protocol defined in the OPC UA standard is based on state-of-the-art Web technologies, allowing OPC UA applications to communicate and transfer data through networks in a similar way as Web browsers communicate with Web servers. UA/TCP is a fast and simple transport protocol, especially suited for efficient Intranet data transfer. (Mahnke, W. et al. 2008, 199-200.)

3.7.1 Comparison with OPC Classic

The widely-used Classic OPC standard contains three main specifications for transferring information between different devices and systems. These specifications standardise the flow of information from the automation devices all the way to the management level. These specifications are (Mahnke, W. et al. 2008, 3-4.):

- OPC DA (Data Access)
- OPC HDA (Historical Data Access)
- OPC A&E (Alarm & Events).

OPC Security extends the main three OPC Classic specifications. It was designed for controlling the client access permissions and protecting against unauthorised tampering of process data. Three extensions to the OPC Classic DA specification exist (Mahnke, W. et al. 2008, 6-7.):

- OPC Batch
- OPC Complex Data
- OPC Data eXchange (DX).

An interface called OPC Commands was also introduced but never released. All of the functionalities of the OPC Classic specifications, together with the three DA extensions, were incorporated into the OPC UA standard. (Mahnke, W. et al. 2008, 6-7.)

Classic OPC and OPC UA technology utilise the client/server model for communication. It is also possible to run both client and server application in the same device. In that case the device may not only act as a receiver of data, but it can also provide it to other client devices. (Mahnke, W. et al. 2008, 3-4.)

In Classic OPC, an OPC Server software runs on a network-connected device that needs to provide information about certain process data to other devices. The OPC server encapsulates the source of data, such as a Programmable Logic Controller (PLC) and then makes the data available through one or more of its interfaces via COM/DCOM technology present in most versions of the Microsoft Windows® operating system. (Mahnke, W. et al. 2008, 3-4.)

A concrete example of a typical use case for the OPC UA technology is shown on Figure 21. This simple example shows a compact Programmable Logic Controller (PLC) device on the left, providing raw process data from a machine, automation process or environment for the OPC UA client applications on the right. The PLC device is equipped with an OPC UA server software that is providing access to the process data.

Because there are no dependencies to COM/DCOM technology in the OPC UA protocol, the server application can be run inside the PLC device itself. This can eliminate the need of the PLC device being connected to a Windows® PC for OPC connectivity. Also for the same reason the OPC UA server is able to communicate with a wider range of client devices when compared with OPC Classic.

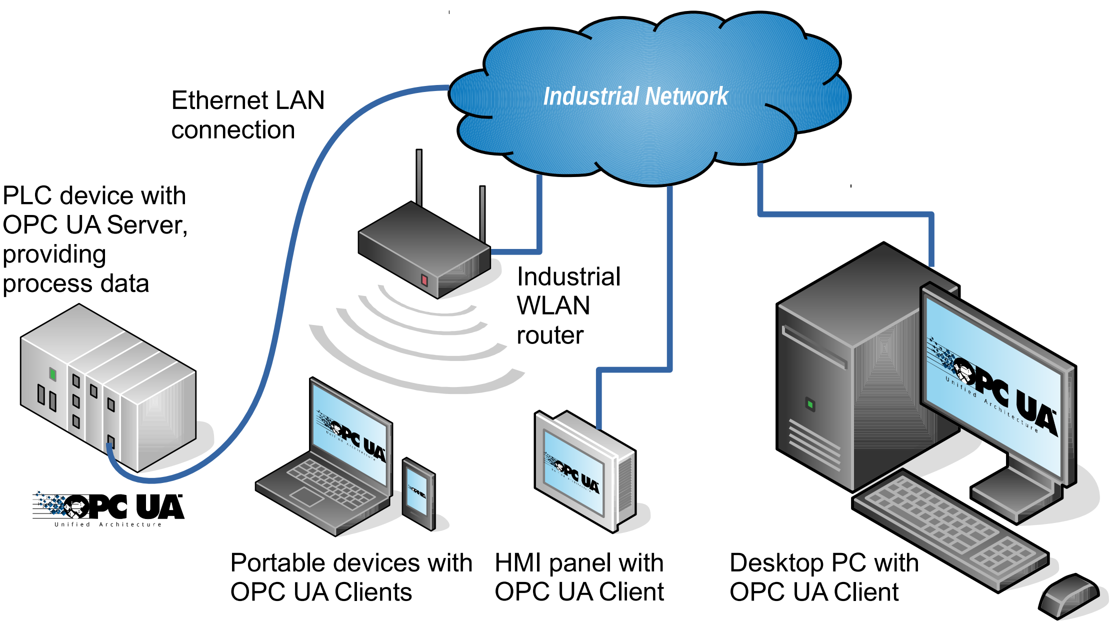

Industrial Network
Ethernet LAN connection
PLC device with OPC UA Server, providing process data
Industrial WLAN router
OPC UA
Portable devices with OPC UA Clients
HMI panel with OPC UA Client
Desktop PC with OPC UA Client

Figure 21. OPC UA usage example.

Possible OPC UA clients (and also servers) include devices such as computers, user interface panels, PLCs, smart field devices and other automation devices, tablets, smartphones and embedded systems. OPC UA enables true hardware-independent communications in industrial automation. (Mahnke, W. et al. 2008, 8-9.)

3.7.2 Structure of OPC UA

The OPC Unified Architecture standard was designed from ground up to eliminate the problems experienced by OPC Classic. It was designed to be fully compatible with the earlier OPC Classic standard and to provide the same functionality and performance, whilst offering platform-independence and improved scalability. (Mahnke, W. et al. 2008, 8-9.)

Because OPC Classic has a strong dependence for Microsoft's COM/DCOM technology, it is problematic to be implemented in certain resource-limited devices. For instance, it may be difficult for embedded systems with low processing power to accommodate the necessary software components to implement the OPC Classic functionalities. (Mahnke, W. et al. 2008, 8-9.)

The OPC UA standard builds on several layers of components. The most basic and fundamental components of OPC UA are transport mechanisms and data modelling. (Mahnke, W. et al. 2008, 10-11.)

Two main transport mechanisms are defined in the OPC UA standard. The first is UA TCP Binary and the second is Web Services. UA TCP Binary is a highly optimised binary transport protocol that is well-suited for high-performance Intranet communication. OPC UA can also utilise modern internet standards such as Web Services, XML and HTTP that facilitate communication through the internet and firewalls. (Mahnke, W. et al. 2008, 198-200.)

The data modelling component provides a set of basic rules for defining an information model with OPC UA. This base information model can be extended by additional information models built on top of the base model. It also defines the base data types that are used when building a type hierarchy. (Mahnke, W. et al. 2008, 10.)

OPC UA Services are an interface between the servers working as providers of information model and clients that are consuming the information model. The Services are using the transport mechanisms to allow clients and servers to exchange data between each other. (Mahnke, W. et al. 2008, 201.)

This component-based structure shown in Figure 22 allows OPC UA clients to only access the data that they require, without having to understand the entire information model structure exposed by complex systems. On the other hand, it also allows OPC UA to be used with highly complex systems or systems that have very specialised needs. (Mahnke, W. et al. 2008, 10-11.)

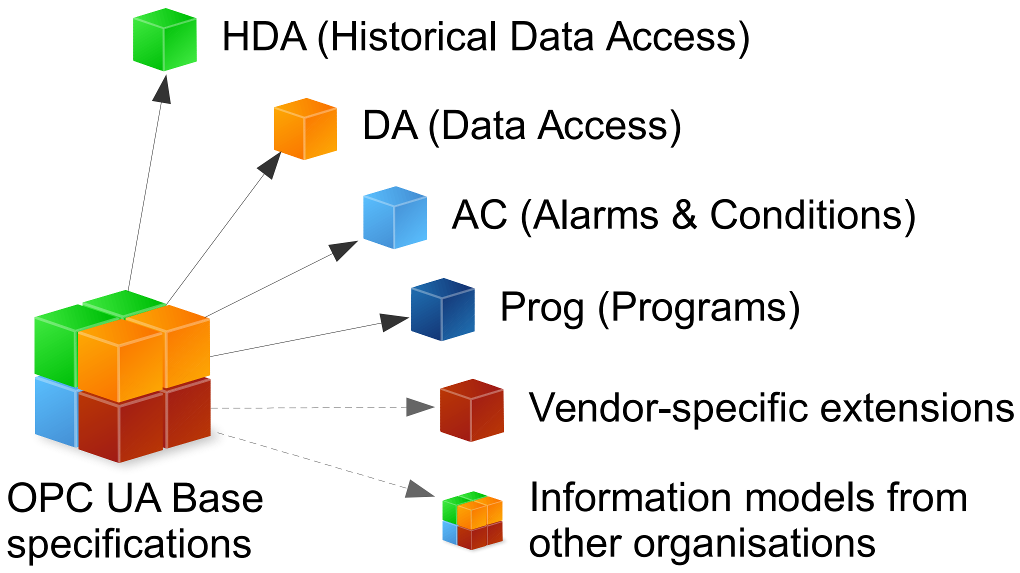

HDA (Historical Data Access)
DA (Data Access)
AC (Alarms & Conditions)
Prog (Programs)
Vendor-specific extensions
OPC UA Base specifications
Information models from other organisations

Figure 22. The layered architecture of OPC UA. (Mahnke, W. et al. 2008, 11.)

Several extensions to the base OPC UA specifications are developed to cover all successful features of the OPC Classic standard. Other organisations may also build their own information models on top of the base OPC UA model. An example of such extension is PLCopen, which is a standard for PLC device programming languages. (Mahnke, W. et al. 2008, 11.)

3.7.3 OPC UA software structure

OPC UA uses a similar client/server model for communication as OPC Classic. Applications that expose and provide the information that a device contains or produces are called OPC UA servers and applications that receive and consume the information are called OPC UA clients. (Mahnke, W. et al. 2008, 255-256.)

Typical OPC UA applications consist of three layers of software. These layers are presented in Figure 23.

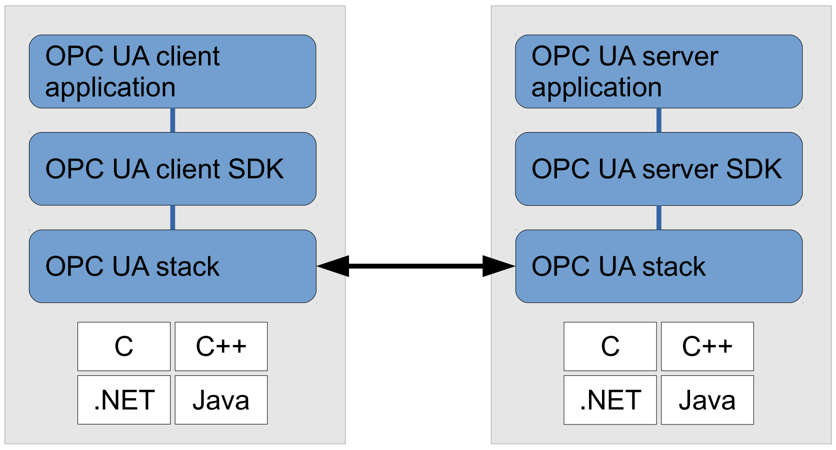


Figure 23. Software layers of OPC UA applications. (Mahnke, W. et al. 2008, 255-256.)

The OPC UA stack forms the lowest level of the three software layers. The stack is responsible for low-level functionalities. The stack is further divided into several parts, such as platform, transport, security and encoding layers and client and server APIs (Application Programming Interfaces). (Mahnke, W. et al. 2008, 255-256.)

The SDK (Software Development Kit) layer is responsible for high-level functionalities. These functionalities may be directly related to OPC UA operations or general functionality such as logging, security or configuration. (Mahnke, W. et al. 2008, 258-259.)

Depending on the requirements of the use case, the application layer may consist of either OPC UA client, server or client/server applications. The main task of an OPC UA client may be to present a user-friendly user interface for the end user and to communicate with the OPC UA stack by utilising the functionalities present in the SDK layer. (Mahnke, W. et al. 2008, 258-259.)

OPC UA server consists of applications that manage the OPC UA address space and applications that act as a front-end for a system of DCS- or PLC devices. In the latter case, the server application is mainly responsible for providing methods for exchanging data between OPC UA applications and the underlying system. (Mahnke, W. et al. 2008, 261.)

SDKs for developing OPC UA -enabled applications are available for several programming languages. The official OPC UA SDKs that are made by the OPC Foundation are available for C, C#, C++ and Java programming languages. They are available for the members of the OPC Foundation. Several unofficial, free software implementations (with various levels of functionality and supported features) also exist for other programming languages, such as Python and JavaScript.

3.7.4 DA (Data Access)

The Data Access interface is the most successful and the most widely-deployed interface in OPC UA and OPC Classic. The DA interface is incorporated in the majority of all products utilising OPC technologies, making it the most important OPC UA interface. Usually other interfaces are implemented to extend the basic functionality provided by DA. (Mahnke, W. et al. 2008, 4-5.)

The DA interface enables automation devices to access variables that contain the current data and state of automation processes. If the OPC UA server is configured to do so, the variables can be read, written and monitored by OPC UA clients utilising the DA interface. In this way real-time data can be moved from the field- and automation level devices, such as PLCs, PACs (Programmable Automation Controllers) and DCS (Distributed Control Systems) to the management level, to display clients and HMIs. This is useful for evaluating and monitoring the performance and functioning of the automation system. (Mahnke, W. et al. 2008, 5.)

3.7.5 HDA (Historical Data Access)

The Historical Data Access interface was designed to provide clients a way to access not just current, real-time data but previously stored data. Historical data can be accessed in an uniform manner by using the HDA interface. Typical usage applications may include various kinds of data logging systems and SCADA (Supervisory Control And Data Acquisition) applications. An HDA client can access historical data values stored in the OPC UA server in three different ways. (Mahnke, W. et al. 2008, 6.)

The first way is to define the desired time range and variables that the client wants to access. The server processes the query and returns all data values from its archive that satisfy the client's request. The second way is to query the server for one or more variables for the specific timestamps. The third way is to let the server compute aggregate values from the data stored in the archive and then return the computed values back to the client. It is also possible for the clients to insert, replace and delete data values by using the HDA interface. (Mahnke, W. et al. 2008, 6.)

3.7.6 A&E (Alarm & Events)

The Alarm & Events interface of the OPC UA standard provides clients a flexible way to convey information about alarms and events happening in the automation system. This information could be used, for example, to warn the user about a malfunction in the system. (Mahnke, W. et al. 2008, 5-6.)

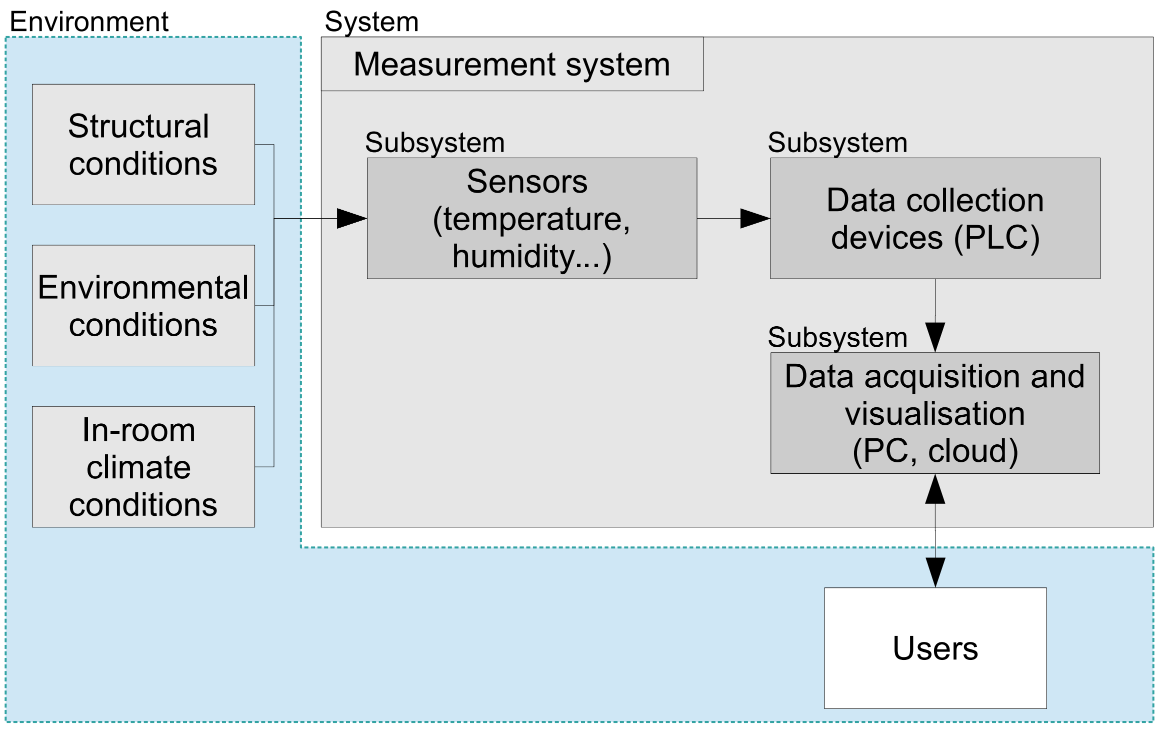
An event is a notification of a single event that has occurred in the system. An alarm is a notification that is produced when the state of a certain condition is changed in the automation system. This change could be a limit switch being activated, a strain gauge deforming past a certain limit or a temperature switch reaching its set-point value. An alarm must be transmitted to the client without delays so that it could have a chance to react to it. (Mahnke, W. et al. 2008, 5-6.)

When a client is required to receive alarm and event information from the automation process, it connects to the A&E interface of the OPC UA server and subscribes for notifications. The client will then receive all notifications that have been triggered in the server. Filtering rules can be written to control the flow and amount of event information. (Mahnke, W. et al. 2008, 5-6.)

4 Demonstration measurement system

This small-scale measurement system was designed and built to act as a proof of concept for the full-scale measurement system that is planned to be placed into a wooden building. The design of a measurement system is one part of the project “Wooden apartment house as a study environment, stage 1”, which is a joint project between Seinäjoki University of Applied Sciences (SeAMK), Tampere University of Technology (TUT) and Lakea Oy (a construction company).

The measurement system consists of three major subsystems. The general structure of the system is shown in the form of a block diagram in Figure 24. It is presented in the form of a technical system or a block diagram, with its environment marked with light blue colour and its subsystems marked as rectangles with a darker grey background colour. The arrows convey the general direction of the flow of information or data. Starting from chapter 4.3, each of the three subsystems of the measurement system are described in a similar manner.


Environment
System
Measurement system
Structural conditions
Environmental conditions
In-room climate conditions
Subsystem
Sensors (temperature, humidity...)
Subsystem
Data collection devices (PLC)
Subsystem
Data acquisition and visualisation (PC, cloud)
Users
Figure 24. Block diagram of the measurement system.

The measurement system works by autonomously collecting information from its environment by the sensor subsystem. The sensors of the sensor subsystem are placed at various locations inside and outside the wooden building, gathering structural and environmental information. The sensor subsystem of the full-scale measurement system consists of approximately hundred sensor components of several different types.

Measurement information is transferred into the data collection subsystem via cabling as analogue and digital signals. The data collection subsystem is responsible for performing analogue-to-digital conversion and processing the measurement information by filtering and scaling it into correct physical units. The functions of the data collection subsystem are realised by utilising PLC (Programmable Logic Controller) devices as its active components.

Next, the filtered and scaled measurement information is transferred into the data acquisition and visualisation subsystem. This subsystem includes necessary components for allowing it to communicate with the data collection subsystem, acquiring the processed measurement data and storing it into its data storage. This subsystem includes a component which allows it to present both the current and historical measurement data for the users via a simple web user interface.

4.1 Objective

The objective of the demo system was to test and demonstrate the working of the full-scale measurement system. The testing was done by building a proof of concept set-up that is based on similar components and structure as the full-scale system.

The data produced by the measurement system will be used for research purposes, particularly in the research programme “High Performance Building (HPB18)” initiated by Tampere University of Technology (TUT). Long-term thermal, moisture and weather measurements offer important information about the physical properties of wooden structures. Concrete, measurement-based information is beneficial for the development of simplified structural design methods. Such design methods could be formulas, tables and guidelines that structural engineers and designers could use in their future building projects.

4.2 System description

Both this small-scale demonstration system and the full scale measurement system consist of standard components used in building automation and industrial automation. Components such as PLC devices, analogue input modules and sensors with standard voltage and current signal levels are used.

The task of choosing the sensor components was assigned to the researchers of Tampere University of Technology. The system will have sensors for measuring temperature and humidity levels from the structures, in-room carbon dioxide levels and the amount of wind-driven rain falling on the building's walls. Measured weather parameters include wind speed and direction, precipitation (amount, duration, intensity), atmospheric pressure, air temperature and relative humidity.

Programmable Logic Controller devices with distributed input/output capabilities were researched from several manufacturers. Out of the potential manufacturers, Beckhoff was selected as its products' technical specifications either met or exceeded all of the requirements of this application. In addition, Beckhoff has a range of embedded PLC devices with built-in Ethernet automation bus ports and software for the OPC UA communication protocol.

A standard x86-based PC is used for running the software components that are performing the data acquisition and visualisation tasks. In the small-scale demonstration system, the software components of the subsystem are running inside of a GNU/Linux-based virtual machine image. The host of the virtual machine is a standard office desktop computer.

4.3 Sensor subsystem

The sensor subsystem consists of various sensors that are placed at different locations both inside and outside of the wooden building. The sensors gather physical information from the environment such as temperature and humidity values, convert them into electrical signals that are transferred them into the data collection subsystem. The sensor subsystem of the full-scale measurement system is shown in Figure 25 in detail. The small-scale demonstration system features only temperature and humidity sensor components.

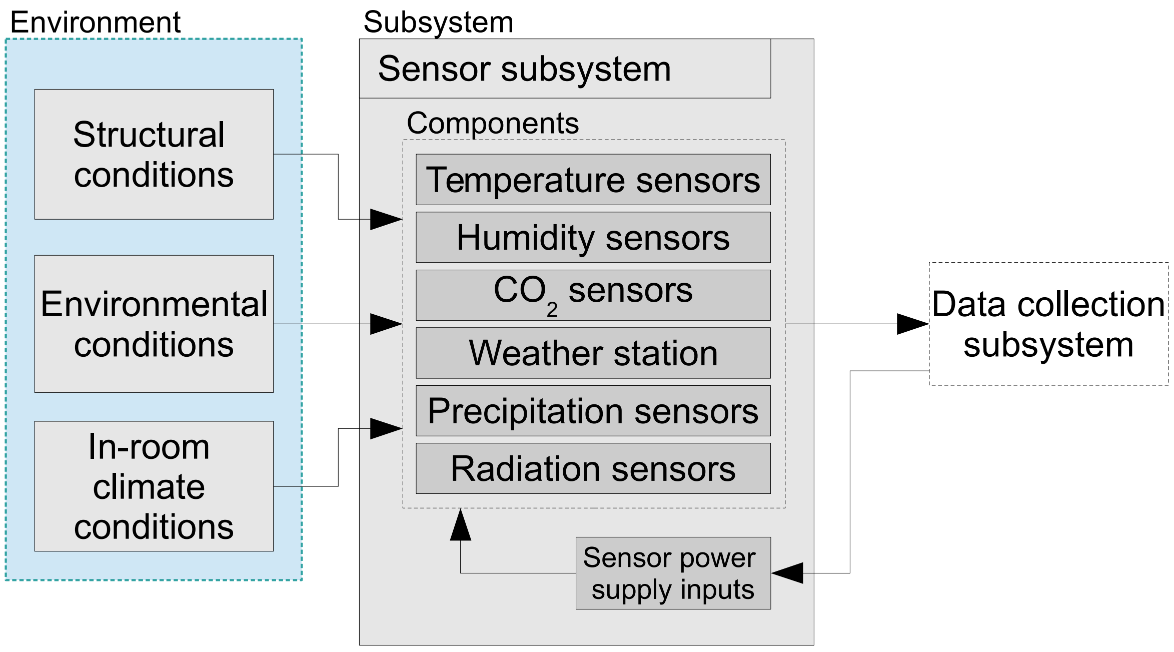


Figure 25. Sensor subsystem of the full-scale measurement system in detail.

The sensor subsystem of the small-scale system consists of a total of four sensor components:

- two Vaisala HMP60 temperature and humidity sensors
- one WIKA TSD-30 temperature sensor
- one Siemens SITRANS TS 500 temperature sensor.

All sensors in the system output analogue electrical signals. The WIKA and Siemens temperature sensors output 4..20 mA current signals. The output signal types of the Vaisala temperature/humidity sensors are 0..5 V voltage signals.

4.4 Data collection subsystem

The data collection subsystem in both full-scale and small-scale measurement systems is constructed by utilising fieldbus-connected PLC (Programmable Logic Controller) devices and analogue input modules (also called “bus terminals” in Beckhoff's terminology).

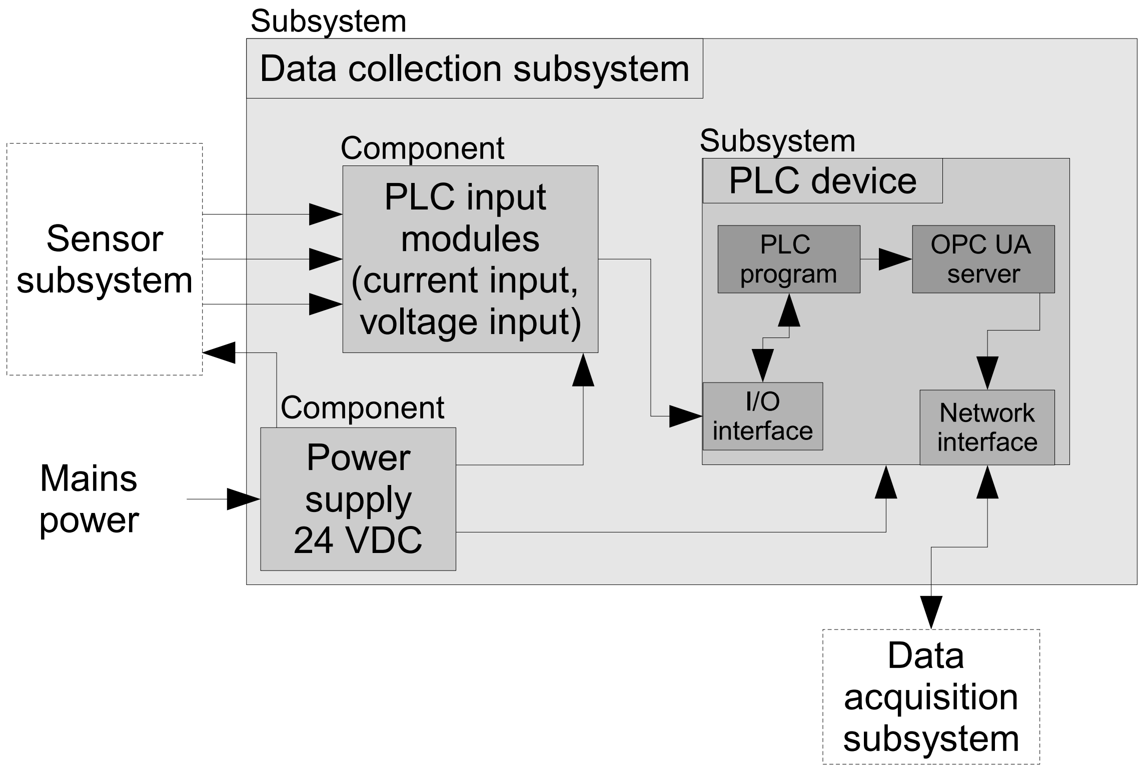


Figure 26. Data collection subsystem in detail.

Figure 26 shows the main components of the data collection subsystem as a block diagram. The PLC device, with its many hardware and software components forms another subsystem inside the main data collection subsystem.

4.4.1 Data collection in the full-scale system

Beckhoff's CX 8091 embedded PLC shown in Figure 27 was chosen to work as the data collection component in the full-scale measurement system. It was chosen because it has built-in support for the OPC UA protocol, good extendibility thanks to its standard Ethernet bus and a wide selection of Beckhoff's K- and E-bus terminals. The CX 8091 has sufficient processing power capabilities for the task and its low electrical power consumption is also an advantage. High-speed versions of the CX-series PLCs with fast Intel processors were also considered. Those devices consume up to 15..42 Watts of power during operation, while the power consumption of the CX 8091 is just 3 Watts.

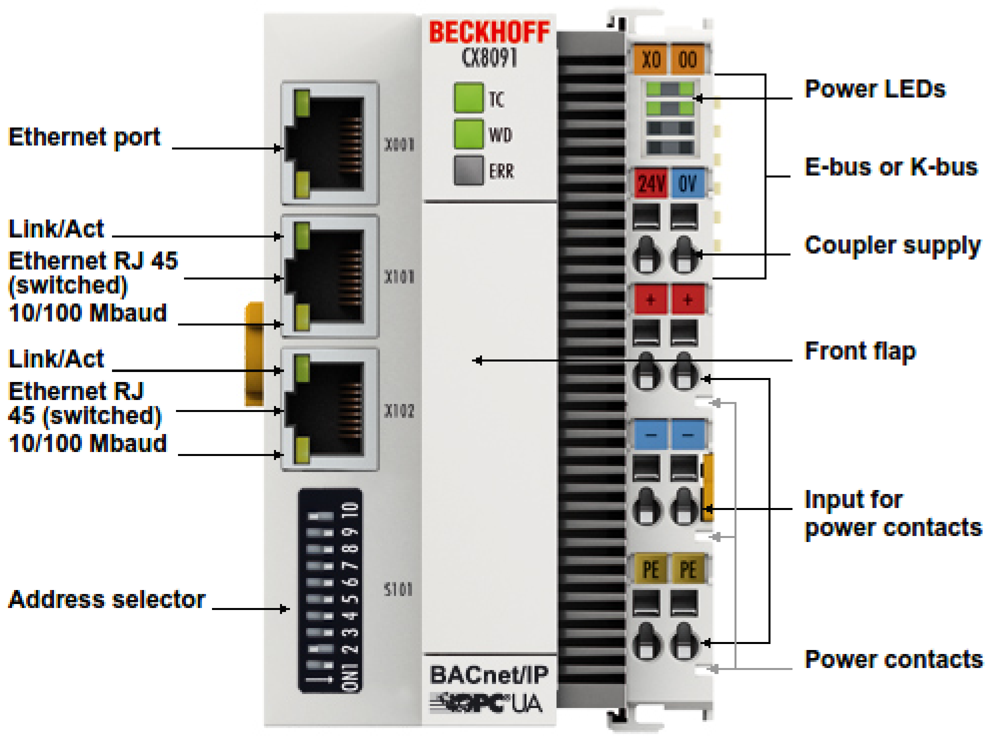


Figure 27. Beckhoff CX 8091 embedded PLC. (Beckhoff 2014.)

4.4.2 Data collection in the small-scale demonstration system

In the small-scale demonstration system, Beckhoff's CX 1001 embedded PLC was used. When compared to the CX 8091, it is a previous-generation product with less processing power and without built-in support for OPC UA transfer protocol. Nevertheless, this CX 1001 PLC device was chosen for the demo system because both devices have a similar structure, despite some technological differences. The CX 1001 PLC features fewer connectivity options, less RAM memory and a slower processor than the CX 8091.

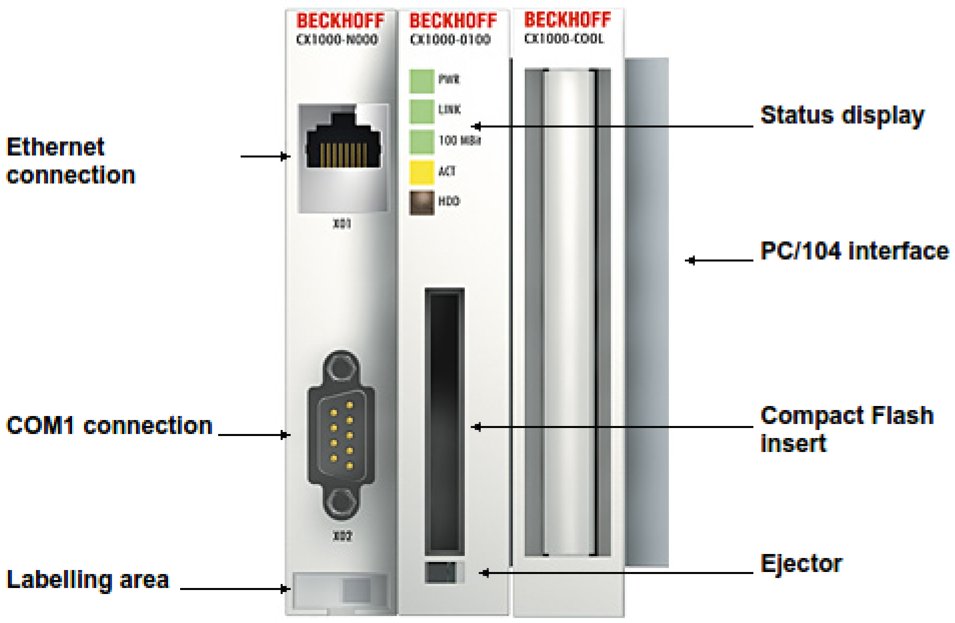


Figure 28. Beckhoff CX 1001 embedded PLC. (Beckhoff 2014.)

The CX 1001 CPU module unit shown on Figure 28 features built-in Ethernet connectivity for network access and communicating with other automation devices. The Ethernet port is also used for transferring programs for the PLC runtime environment. It features a connector for attaching Beckhoff's KL-series input/output modules or bus terminals.

4.4.3 Comparison of PLCs

The most important characteristics of both PLC devices are compared in Table 5. It was found that the features of the CX 1001 PLC were sufficient to meet the requirements of the small-scale demonstration system. A single CX 1001 unit, together with analogue input bus terminals, temperature and humidity sensors was provided by Seinäjoki University of Applied Sciences.

Table 5. Technical specifications of PLC devices. (Beckhoff 2014 - 2015.)

Technical data	**Beckhoff CX 1001**	**Beckhoff CX 8091**
Processor	Intel Pentium MMX, 266 MHz	ARM 32-bit, 400 MHz
System RAM	32 MB, expandable to 128 MB	64 MB, not expandable
Internal flash memory	64 MB, CompactFlash	512 MB, MicroSD
Communication interfaces	1 x Ethernet, 1 x RS-232	1 x Ethernet, 1 x USB
Input/output connectivity	Beckhoff KL-bus terminals	Beckhoff EL- and KL-bus terminals
Fieldbus connectivity	None (separate modules available)	2 x RJ45 (BACnet/OPC UA)
Operating system	Windows XP Embedded or CE	Microsoft Windows CE 6
Programming environment	TwinCAT 2	TwinCAT 2
Programming languages	IEC 61131-3 languages	IEC 61131-3 languages
Max. power consumption	4 W	3 W

Time-limited trial version of Beckhoff's OPC UA server software was installed into the CX 1001 PLC device. The OPC UA server software allows communication between the PLC device and the data acquisition subsystem via Ethernet bus. The measurement data is transferred into the data acquisition PC using the Data Access function of the OPC UA protocol.

4.4.4 PLC bus terminals

All the bus terminals that are used in the small-scale measurement system are Beckhoff's KL-series analogue input terminals. Four bus terminals of type KL 3061 and one of type KL 3022 are used for receiving the electrical signals from the sensor subsystem. In addition, one KL 2134 bus terminal with four digital outputs is installed into the system, but it is not used. The bus terminals are terminated with a KL 9010 end terminal.

KL 3061 bus terminal shown in Figure 29 is a bus terminal with a single analogue voltage signal input. It has an input voltage range of 0..10 Volts and its analogue-to-digital conversion resolution is 12 bits.

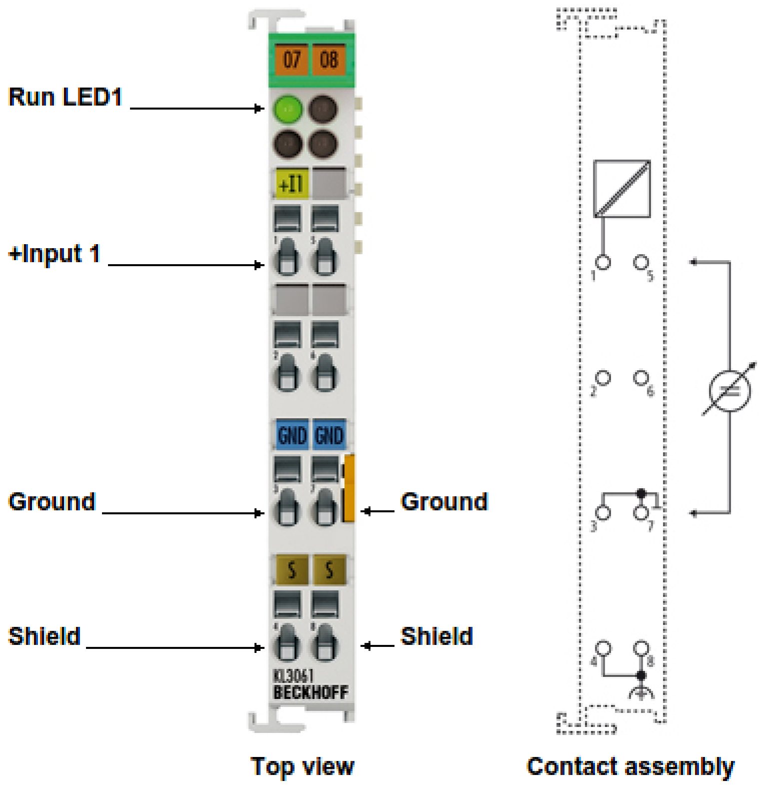


Figure 29. Beckhoff KL 3061 analogue voltage-input bus terminal. (Beckhoff 2015.)

KL 3022 bus terminal has two analogue current signal inputs. It has input signal values of 4..20 mA and it features a 12-bit analogue-to-digital converter. Figure 30 shows an overview of the KL 3022 bus terminal.

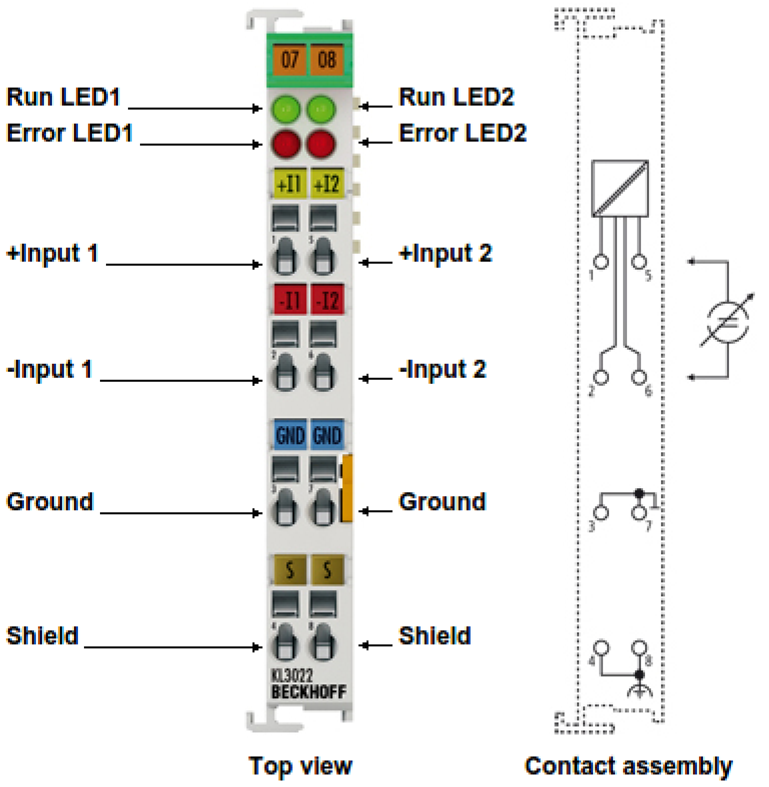


Figure 30. Beckhoff KL 3022 dual analogue current-input bus terminal. (Beckhoff 2015.)

KL 2134 is a digital-output bus terminal with four outputs. It is also installed into the small-scale measurement system, but is not used. Figure 31 shows an overview of the KL 2134 output module.

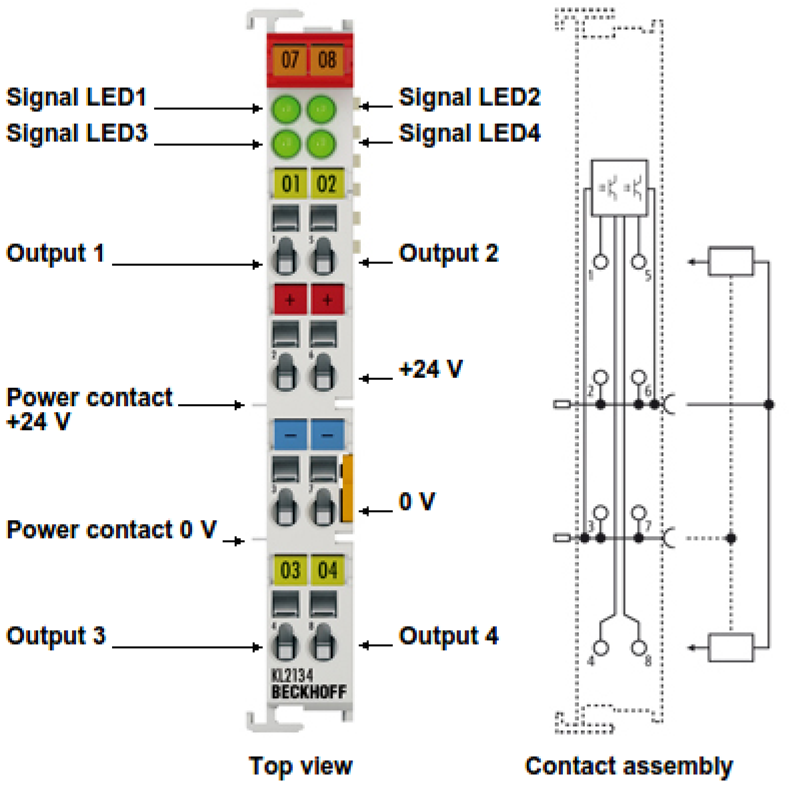


Figure 31. Beckhoff KL 2134 bus terminal with quad digital outputs. (Beckhoff 2015.)

The bus terminal stack of the system is terminated with a KL 9010 end terminal. Its only function is to enable the communication between the PLC and the bus terminals. The end terminal is shown in Figure 32.

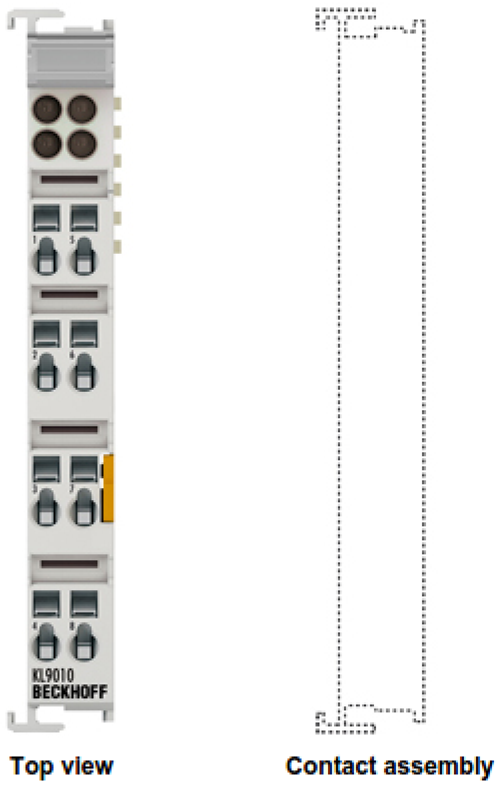


Figure 32. Beckhoff KL 9010 end terminal. (Beckhoff 2015.)

4.4.5 Power supply

One of the components of the data collection subsystem is a switching power supply unit. The power supply provides power for the PLC devices, its auxiliary equipment and the sensor subsystem.

Figure 33. MURR Elektronik switching power supply unit. (MURR Elektronik [Ref. 21 March 2015].)

The power supply unit is pictured in Figure 33. The unit converts the mains voltage into a voltage level of 24 VDC.

Table 6. Power supply technical specifications. (MURR Elektronik [Ref. 21 March 2015].)

Technical data	**MURR Elektronik MPS3-230/24**
Input voltage range	95..265 VAC
Input frequency	50..60 Hz
Max. input current	1.5 A (115 VAC), 0.75 A (230 VAC)
Output voltage	24 VDC
Max. output current	3 A
Output voltage ripple	20 mV (max. r.m.s.)
Short-circuit protection	Yes
Overload protection	Yes
Efficiency	84 %

The device is short-circuit protected and it features a wide input voltage range; it can accommodate input voltages from 95 VAC to 265 VAC. The technical specifications of the power supply unit are listed in Table 6. (MURR Elektronik [Ref. 21 March 2015].)

4.5 Data collection program

A program was developed for the data collecting PLC of the small-scale measurement system for processing the electrical signals received from the sensor subsystem and sending them into the data acquisition subsystem. The main functionality of the program is to convert the electrical signals into a form that is suitable for storage on the computer and later use (numerical values).

The program was developed with Beckhoff TwinCAT 2 PLC programming environment in IEC 61131-3 programming languages. The program was written mostly in FBD (Function Block Diagram) graphical language and partially in ST (Structured Text) language. In addition, a time delay function block was copied from the help file of TwinCAT 2. This function block is written in the IL (Instruction List) language and it provides time delay functionality for the filtering function block.

4.5.1 Structure of program

The structure of the PLC program consists of function blocks (FB). Function block is a POU (program organisational unit) which maintains an internal state variables in addition of its input and output variables and functionalities. This is the main difference between function blocks and POUs of type “function”. Function POUs always return values based only on their input values without maintaining internal state variables. Once a function block is created by one of the IEC 61131-3 programming languages, instances of that function block can be instantiated anywhere in the user's PLC program without having to re-write any program code again.

By putting both the functionalities and state variables inside a function block definition, the same program code can be re-used from program to program and project to project. Hence, use of function blocks can be thought to be a basic form of object-oriented programming in the context of PLC programming.

The general structure of the PLC program is shown in Figure 34. It consists of three function blocks that perform the tasks of scaling, timing and filtering. The input values received from the bus terminals are read into the scaling function block. The scaling function block performs linear interpolation of the raw input data. It returns an output value that is linearly proportional to the raw input value. In addition to the input value, it requires the minimum and maximum value of the input values and the measurement range (range of output value) as its parameters.

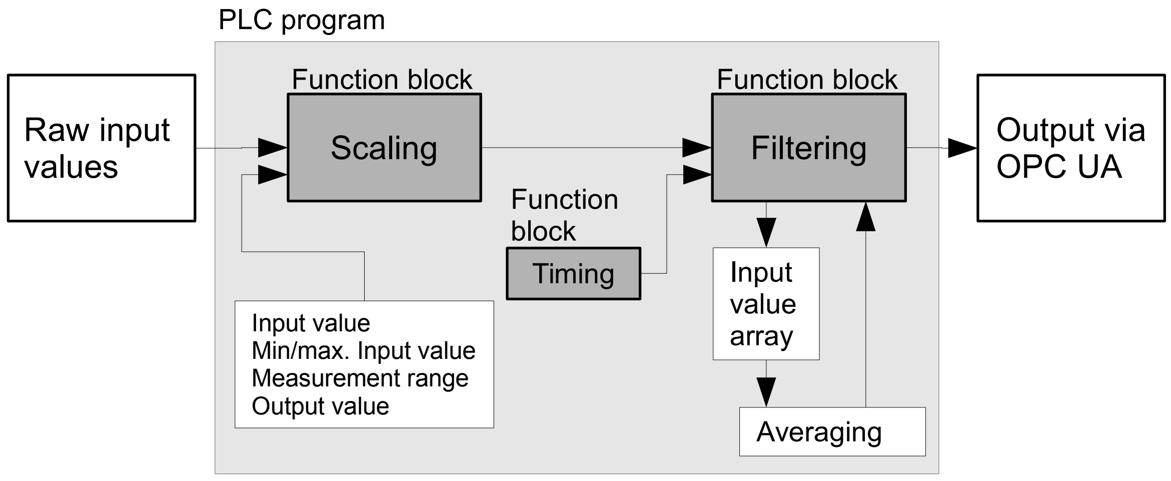
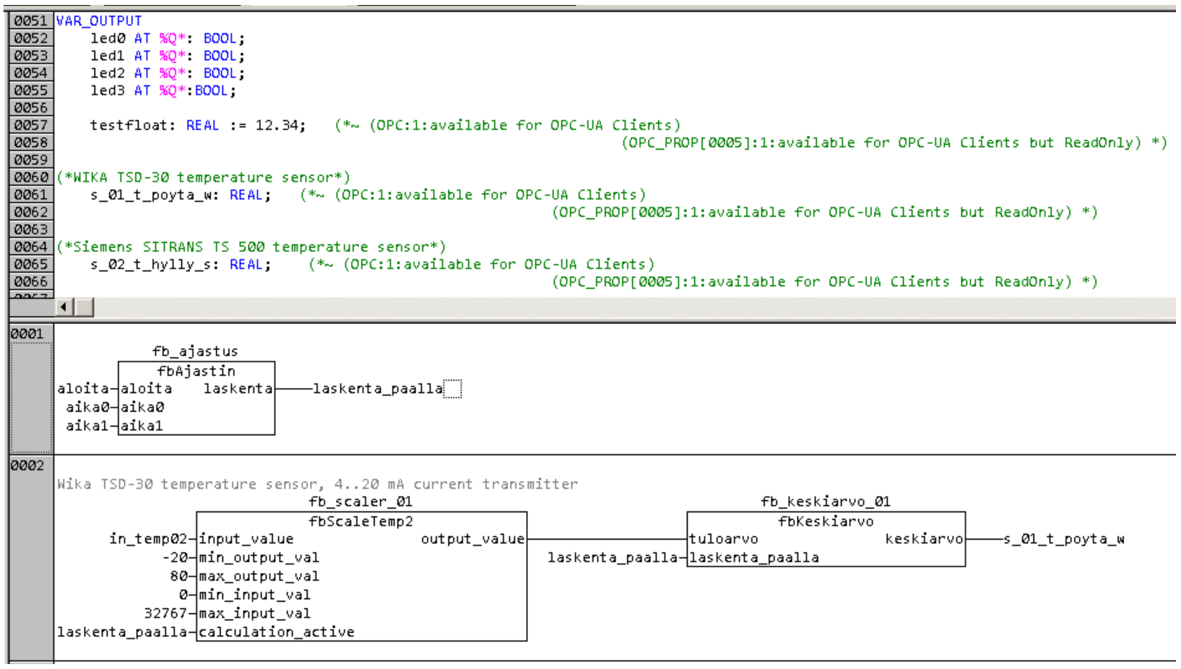


Figure 34. General structure of the PLC program.

The scaled measurement values are fed into the filtering function block which performs a simple low-passing operation to the data. The filtering block maintains an numeric array of instantaneous measurements that is updated at regular intervals, such as every second. The interval is set by providing necessary timing signal into one of the block's inputs. The timing function block is used for this purpose. The filtering block reads a new scaled value from the scaling function block and stores the values into each element of its internal array. When the array becomes full of measurement values, an average value from the array contents is computed and the result is assigned as the output variable of the block. The output variable is then defined to be read via OPC UA connection by adding a special comment line into the variable declaration in TwinCAT 2 programming environment.

In this way, possible high-frequency fluctuation or noise in the measurements is attenuated. The disadvantage of such filtering method is that the capability of recording quick changes in the environmental conditions is decreased.

The timing function block (*fbAjastin)* outputs a binary signal with user-defined durations for the on/off signal states. It is used to determine how frequently the filtering function block (*fbKeskiarvo*) samples the measurement values scaled by the scaling block (*fbScaleTemp2*). One timing block may be used to control any amount of filtering blocks using the same sampling period. The function blocks were programmed in IEC 61131-3 languages FBD, ST and IL. The source code for the *fbAjastin*, *fbTauko*, *fbKeskiarvo* and *fbScaleTemp2* function blocks are given in the appendices 16, 17, 18 and 19, respectively.



```
VAR_OUTPUT
    led0 AT %Q*: BOOL;
    led1 AT %Q*: BOOL;
    led2 AT %Q*: BOOL;
    led3 AT %Q*:BOOL;

    testfloat: REAL := 12.34;    (*~ (OPC:1:available for OPC-UA Clients)
                                                  (OPC_PROP[0005]:1:available for OPC-UA Clients but ReadOnly) *)

(*WIKA TSD-30 temperature sensor*)
    s_01_t_poyta_w: REAL;    (*~ (OPC:1:available for OPC-UA Clients)
                                                  (OPC_PROP[0005]:1:available for OPC-UA Clients but ReadOnly) *)

(*Siemens SITRANS TS 500 temperature sensor*)
    s_02_t_hylly_s: REAL;    (*~ (OPC:1:available for OPC-UA Clients)
                                                  (OPC_PROP[0005]:1:available for OPC-UA Clients but ReadOnly) *)
```

Figure 35. PLC data collection program example.

In Figure 35 a TwinCAT 2 screen capture of instances of the scaling, filtering and timing function blocks are shown. In the example, the scaled and averaged measurement value is stored into the output variable ”s_01_t_poyta_w”. This variable is then made accessible to OPC UA clients by placing a special comment tag in the variable declaration. The TwinCAT 2 development environment supports the Data Access, Historical Access and Alarm & Conditions OPC UA profiles. In this project only the Data Access profile of OPC UA was used.

4.6 Data acquisition and visualisation subsystem

The data acquisition and visualisation subsystem consists of a standard x86-based personal computer, Ethernet cabling and local area network equipment. In the small-scale demonstration system the PC is equipped with the GNU/Linux operating system and it includes all of the necessary software components for data acquisition and visualisation tasks.

This subsystem is presented in the form of a block diagram in Figure 36. In this block diagram, the desktop computer inside the main subsystem is thought to form another subsystem that includes its own software and hardware components. As a computer is a highly complex system, only those components that are immediately related to the working of the measurement system are shown in this simplified diagram.

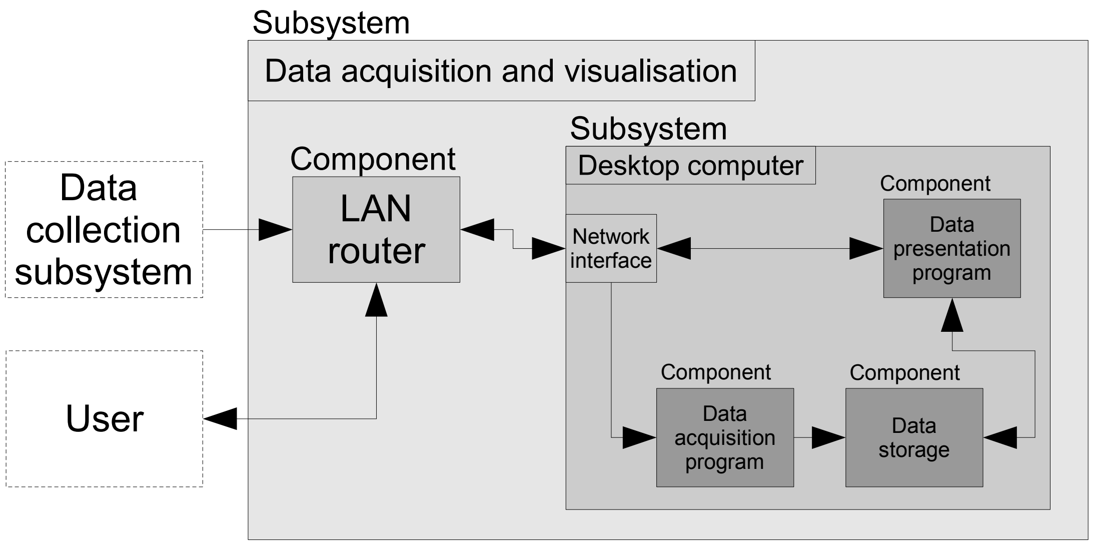


Figure 36. Data acquisition and visualisation subsystem in detail.

Both the user and the subsystems are networked together via Ethernet cabling and a LAN (local area network) router. The router used was a standard rack-mounted router for office networking. It was manufactured by Allied Telesyn and it supports 10/100 Mbps Ethernet connections.

4.6.1 Overview of the data acquisition program

A simple command-line demonstration program for data acquisition and storage tasks was developed with the Python programming language. The Python language was chosen because of its good availability of database, scheduling and communication extension modules. Development in Python is particularly easy and straightforward because Python is an interpreted ”scripting” language. Programs developed in interpreted programming languages often have significantly lower processing performance when compared to to those developed in compiled languages such as C. In the case of this particular program, rapid processing was not a high priority.

The data acquisition program is run on regular intervals, such as every five or ten minutes and sensor data is transferred into a measurement database. MySQL and SQLite database management systems were considered to be used in this project. SQLite was chosen to be used in the demonstration system due to its light system requirements, easy interfacing with the Python and R programming languages and convenience. Most GNU/Linux distributions come equipped with an SQLite installation.

The program uses a configuration file which contains a list of monitored items, among other things. Not all items that are present in the configuration file were implemented. Particularly, scheduled data collection proved to be problematic with Python's scheduling module, resulting in malfunctions on rare occasions. The scheduling was then implemented by the *cron* scheduling facility that is provided by the operating system of the data acquisition computer. Currently the data acquisition program is capable of monitoring the measurement data produced by a single OPC UA -enabled PLC device.

4.6.2 Structure of the data acquisition program

The program consists of four classes of objects, each of which have their own functionalities and data. The class diagram of the program components and their general relationships is shown on Figure 37.

Several third-party Python software libraries, or “modules” were utilised during the program development. The most important module that was used is FreeOpcUa. It contains the necessary basic functionality for establishing connection to OPC UA servers and transferring information via a network connection. FreeOpcUa is available for the C++ and Python programming languages and it is a free software project. (Rykovanov [Ref. 16 April 2015].)

Another module that was highly valuable in the development of the data acquisition program is sqlite3. It is a Python module that allows SQLite database access by embedding SQL code into Python programs.

In the early phases of the development the program was written into a single Python .py source file. This kind of simple structure worked fairly well when the amount of code and functionality in the program was small. Soon after more functionalities, such as SQLite database operations were incorporated into the program, a single .py source file became difficult to use. This is the main reason why the functionalities of the program were distributed into several separate class constructs and finally into several separate .py source files. The class constructs residing in separate files are then imported into the main program file and then instantiated where needed. Instantiation is done by Python's object-oriented programming facilities.

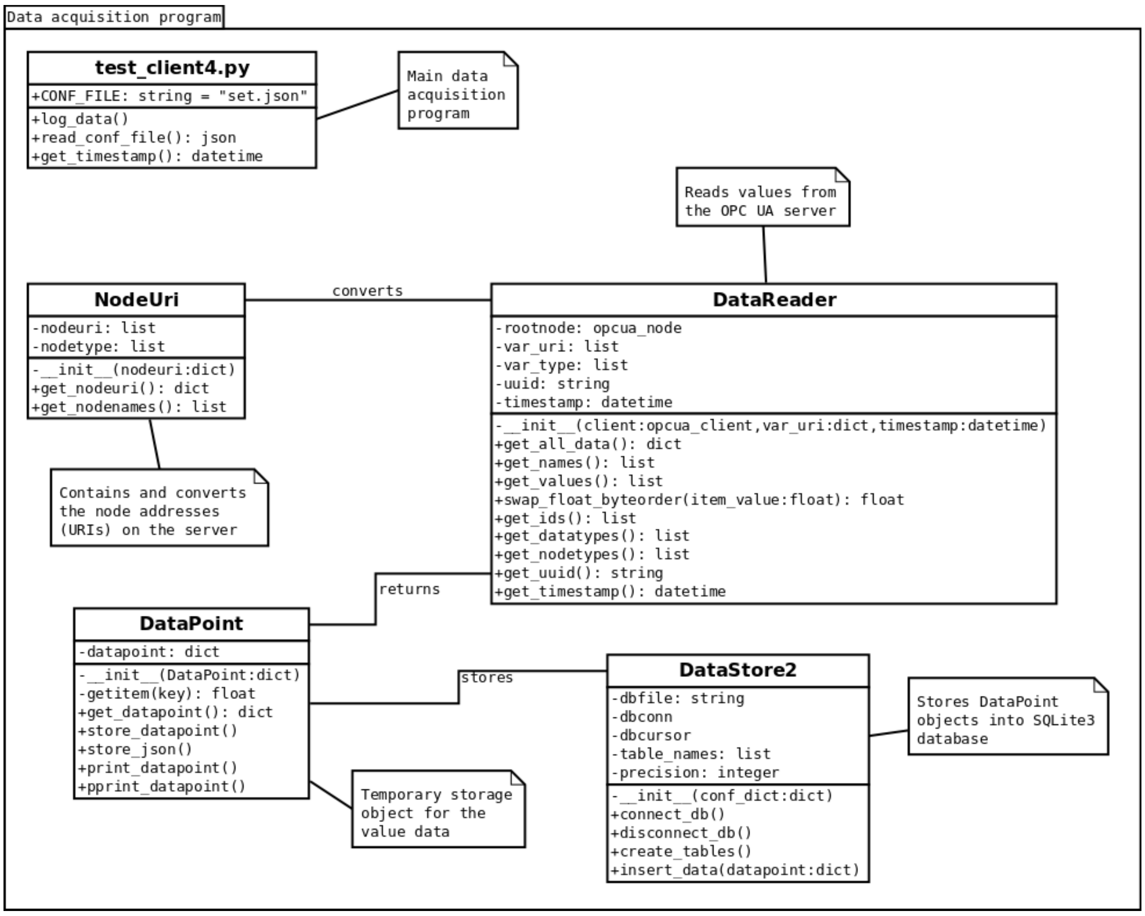


Figure 37. General structure of the data acquisition program.

4.6.3 Functionality of the data acquisition program

The data acquisition program consists of several classes and functions. An UML sequence diagram was created which shows the functions and flow of information in the data acquisition program. The diagram is shown in appendix 1.

The main program features a function for reading a configuration file and storing its contents into a dictionary data structure. The format of the configuration file is JSON, which can be parsed directly into a dictionary data structure in Python. The configuration file includes some configuration options, such as a list of nodes whose values are to be acquired from the remote OPC UA server and stored in a data store, and the location of the SQLite database file. An example configuration file with several nodes is shown in appendix 13. All features present in the configuration file were not implemented in the data acquisition program.

An OPC UA node could be any variable in a program organisational unit (POU) that the PLC device is running. For example, a temperature sensor could be represented by a single OPC UA node. Its value can be read by calling a suitable function of an OPC UA client instance in the program code.

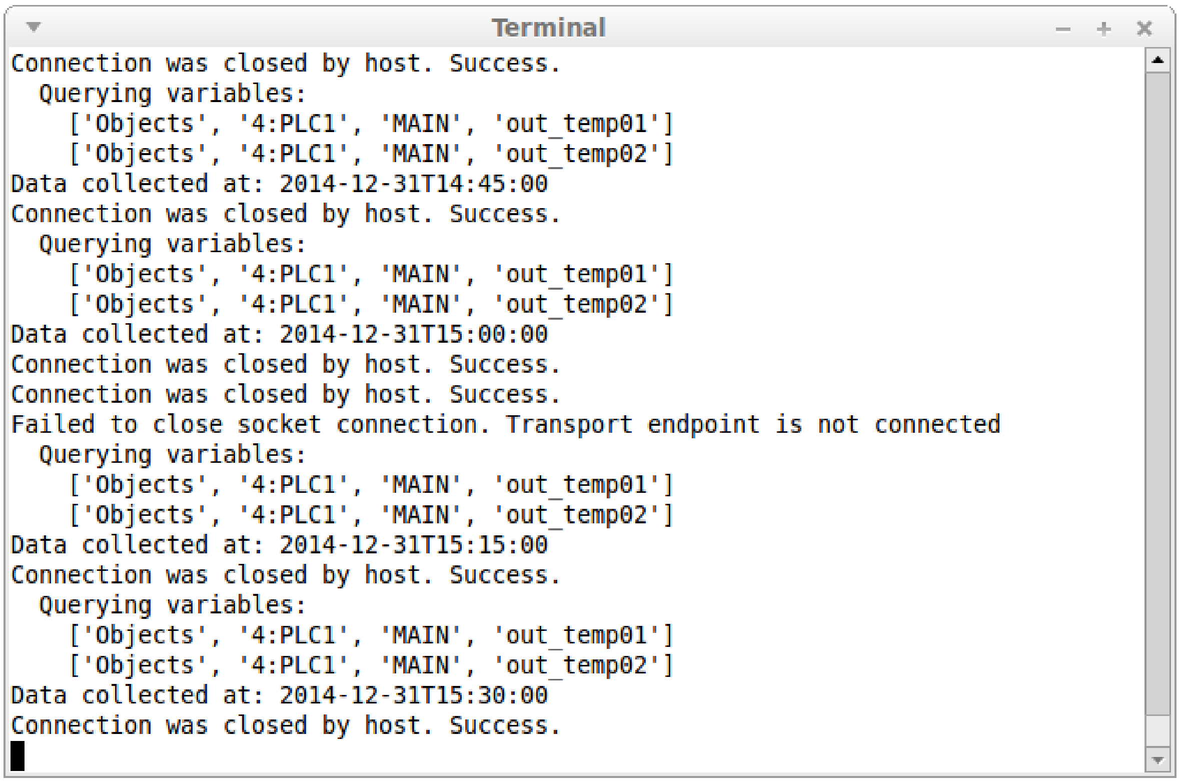
During the execution of the data logging function of the main program, instances of objects of the classes NodeUri, DataReader, DataPoint and DataStore2 are instantiated. A NodeUri object is instantiated for reading the node URI list and node type list from the configuration data dictionary. It converts the node URI and node type lists into a form that can be used by the DataReader object. The source code for node URI conversion object is shown in appendix 14.

A DataReader object is constructed with the OPC UA client instance, node URI list, node type dictionary object and current timestamp. The DataReader instance loops through the node URI list and queries the OPC UA server for each item's name, ID and value. The resulting data is packaged into a data dictionary which contains all data for each monitored node, timestamp and other information. The (partial) source code for the DataReader object is shown in appendix 9.

A DataPoint object is instantiated in the main program. It is constructed with the data dictionary produced by the DataReader instance and it is simply a kind of temporary container for the data that is used by the DataStore2 object. A DataPoint object also features some simple functions, such as printing the data object or fetching a single value from it. It also contains a function for storing the data object as a JSON-formatted text file. This function was used during the early developments of the data acquisition program. An example of a data structure received from the OPC UA server is shown in appendix 15.

A DataStore2 (or DataStore) object is then instantiated and constructed with the configuration data dictionary. DataStore2 object stores all information into a single database table, which simplifies its use. During its construction it reads the SQLite database file name, lists of node names and node types and the precision of the stored numerical values from the configuration data dictionary. It contains functions for connecting and disconnecting to/from the database, creating the suitable tables on the database and for inserting data into the tables. The data insertion function takes a DataPoint object (which contains a data dictionary) as its parameter, extracts all relevant information from it and stores the extracted data as temporary list objects. Then it creates an SQL transaction, prepares the data to be inserted into the database, inserts the data in the temporary lists and ends the transaction. Finally, the database connection is closed. The source code for the database storage class is shown in appendices 10, 11 and 12.

After the data is inserted into the SQLite database, the execution of the program returns to the main program's data logging function, which prints some information on the command-line terminal. When the execution of the data logging function is finished, the main program waits for 15 seconds for the OPC UA client to finish all active communications and finally disconnects the client from the OPC UA server. An example of program output is shown in Figure 38.



```
Connection was closed by host. Success.
  Querying variables:
    ['Objects', '4:PLC1', 'MAIN', 'out_temp01']
    ['Objects', '4:PLC1', 'MAIN', 'out_temp02']
Data collected at: 2014-12-31T14:45:00
Connection was closed by host. Success.
  Querying variables:
    ['Objects', '4:PLC1', 'MAIN', 'out_temp01']
    ['Objects', '4:PLC1', 'MAIN', 'out_temp02']
Data collected at: 2014-12-31T15:00:00
Connection was closed by host. Success.
Connection was closed by host. Success.
Failed to close socket connection. Transport endpoint is not connected
  Querying variables:
    ['Objects', '4:PLC1', 'MAIN', 'out_temp01']
    ['Objects', '4:PLC1', 'MAIN', 'out_temp02']
Data collected at: 2014-12-31T15:15:00
Connection was closed by host. Success.
  Querying variables:
    ['Objects', '4:PLC1', 'MAIN', 'out_temp01']
    ['Objects', '4:PLC1', 'MAIN', 'out_temp02']
Data collected at: 2014-12-31T15:30:00
Connection was closed by host. Success.
```

Figure 38. Terminal output of the data acquisition program.

4.6.4 Overview of the data visualisation program

A web application for monitoring the environmental data gathered by the sensor subsystem was developed. The application is presented to the user as an interactive web site, with certain control widgets that are familiar from the user interfaces of regular desktop applications. By using the web user interface, the user may conveniently select a certain time range and pick the sensors whose data he/she would like to view as a line chart and/or download.

The application was developed with the web development tools offered by RStudio, an integrated development environment for the R programming language. In addition, a line chart visualisation written in JavaScript was embedded in the R user interface program code. Free software version of the Shiny Server software is used for serving the web application in the Intranet of Seinäjoki University of Applied Sciences. Figure 39 shows the general structure of the data presentation program.

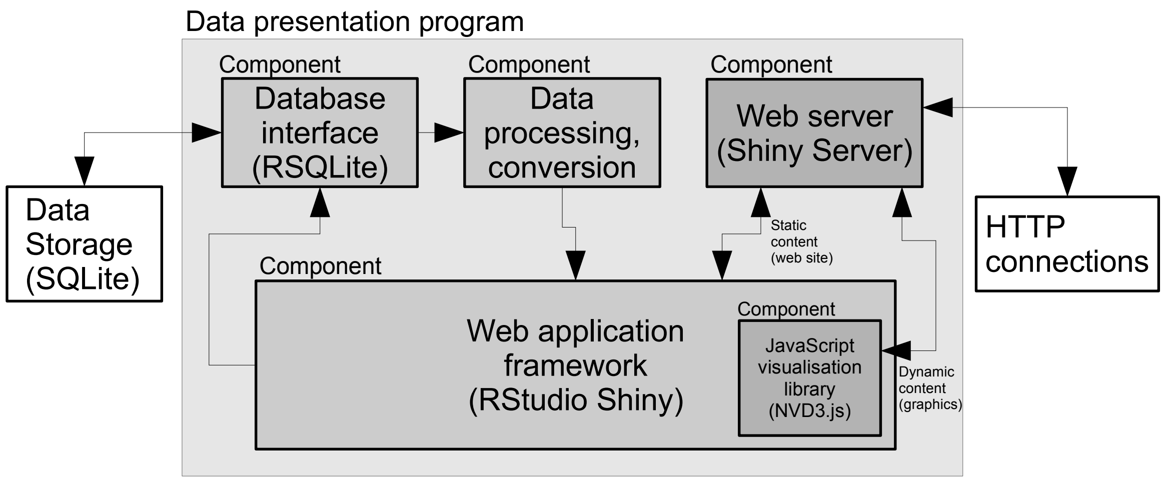

Data presentation program
Component
Database interface (RSQLite)
Component
Data processing, conversion
Component
Web server (Shiny Server)
Data Storage (SQLite)
HTTP connections
Static content (web site)
Component
Web application framework (RStudio Shiny)
Component
JavaScript visualisation library (NVD3.js)
Dynamic content (graphics)

Figure 39. Structure of the data presentation program (web application).

4.6.5 Structure of the data visualisation program

The web user interface application consists of several software components that are responsible for different parts of the application. The components were developed mostly in R and SQL programming languages. The chart visualisation components uses the JavaScript language.

The RSQLite database interface accesses the SQLite database, which contains the measurement data gathered by the sensor subsystem. It contains a group of functions that perform SQL queries and fetch the data that the user wants to view from the database. Source code for the database access is shown in Appendix 4. Appendix 7 shows the reactive functions for accessing and rendering the data in a database.

If the user selects a large number of sensors and/or a long time range of measurements, the visualisation program and the JavaScript chart visualisation may start performing sluggishly. This is due to the fact that the table-like data structures of R must be converted into a form that the JavaScript visualisation program supports. This conversion is done by a function which currently does not utilise any of the performance optimisations present in R, such as vectorisation. The data structure conversion function is shown in appendix 8.

Converting data structures with the un-optimised conversion function (and rendering them with the JavaScript visualisation program) will become a time-consuming process if the structures are very large. Aggregate functions provided by the SQL database language were used to improve the performance in the demonstration system. As SQLite is particularly optimised for handling large amounts of data, the database interface can efficiently compute smaller subsets or aggregate values from a large data set containing even hundreds of thousands of numeric data points. Appendix 5 shows an R function that generates aggregated SQL queries from its input parameters.

Based on user's input, the database interface automatically computes either hourly, daily, monthly or yearly aggregate values of the raw measurement data. This significantly reduces both the computational load and the amount of data transferred between the web server and user's computer. It also protects the user's web browser from becoming unresponsive when a long time range is selected. The user may still download raw, non-aggregated measurement data via the web user interface as a standard .csv (Comma Separated Values) file. The source code for the selection of aggregate function is shown in appendix 6.

The user interface is written in R language by using the Shiny web development framework. It enables building of interactive web applications entirely with R programming language. An example view on the interface is shown in Figure 40. A short excerpt of the user interface code is shown in appendix 3.

Basic Web applications made with RStudio Shiny framework generally consist of at least two files, *ui.R* and *server.R*. The user interface of the application is defined in the *ui.R* file and the application logic and functionalities are defined in *server.R*. Other R program files used in this project are *global.R*, which loads and starts the necessary R software libraries, *linechart.R*, which is a "binding" between R and JavaScript visualisation programs and *helperfunctions.R*, which contains functions for database access. The file *linechart.R* is based on Joe Cheng's example code. (Cheng [Ref. 16 April 2015)

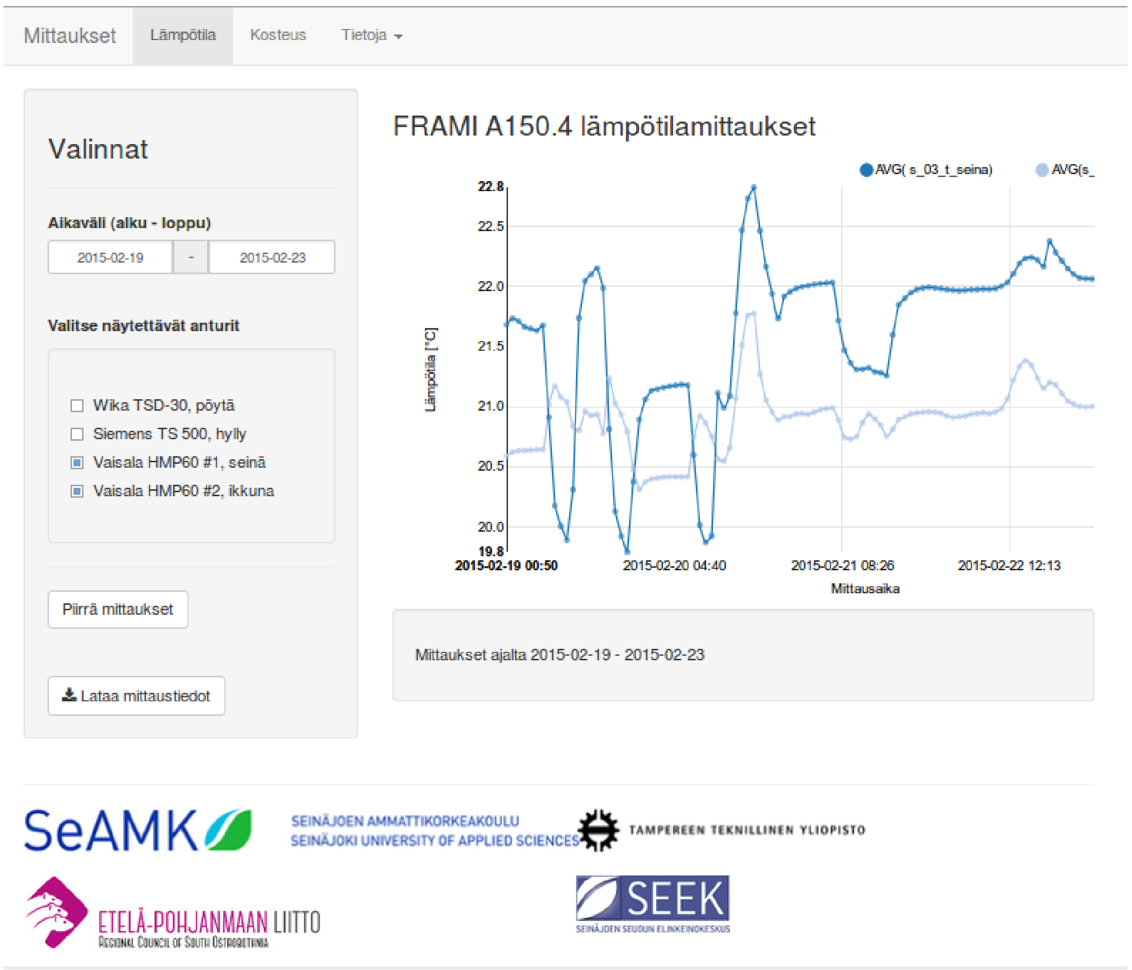


Figure 40. Web user interface for the measurement system.

4.6.6 Functionality of the data visualisation program

The main function of the data visualisation program is to work as an interface between the user and the measurement system. The environmental data that the measurement system has collected is presented to the user via web browser. The user may choose the time range and which sensors' measurements are shown on the website. The user may also view general and technical information about the measurement system via the “Tietoja” menu item on top of the page.

Using the user interface is straightforward. When the user selects a time range, one or more sensors, and presses the plotting button “Piirrä mittaukset” on the left pane of the web site, measurement data is fetched from the database and the line chart is updated to visualise the data.

To enable interactivity, a JavaScript visualisation library *NVD3.js* was used for plotting the measurement data as a line chart. The library enables the user to view the exact values at any specific point of time simply by pointing the mouse over the data point on the line chart area.

The information pages were writted as *Markdown* documents. Markdown is a simple and light-weight document markup language that can easily be converted into many other formats, such as HTML, LaTeX or PDF. The Markdown documents were embedded into the data visualisation web application with the R Markdown extension package for R. The R Markdown package provides functions for rendering Markdown documents on RStudio Shiny web applications.

A sequence diagram of the data visualisation program was made with an UML (Unified Modeling Language) drawing program Dia. The diagram is shown in appendix 2.

5 Summary

This thesis was done as a part of the research project “Wooden apartment house as a study environment, stage 1” of Seinäjoki University of Applied Sciences. The main goal of the project was to examine possible solutions for implementing an automated measurement system for measuring several parameters both inside and outside of a wooden apartment building. This kind of real-world measurement data would allow building physics researchers and scientists to investigate the physical properties of the construction materials that are currently used in the wood building industry.

A plan, along with estimated summary of expenses was outlined for a networked measurement system that is able to fulfill the requirements of the project. The summary of expenses was calculated by the author's colleague Arthur Lenkiewicz, who was also responsible for the design of the measurement system. The measurement system consists of standard components that are commonly used in industrial and building automation. An embedded PLC device from Beckhoff was selected to form the “back bone” of the measurement system. Then, suitable input modules were selected from Beckhoff's line-up for receiving the analogue signals from the sensors in the building for further processing.

A small-scale demonstration system was assembled for demonstrating the working principles of this system. The necessary software components for collecting the measurement data from the field devices, transferring it via network, storing it in a database and visualising the data were developed during this project. The data collection component consists of standard IEC 61131-3 -compliant PLC program. The data acquisition and storage program consists of a program developed in Python and SQL programming languages. The measurement data is visualised with a web application that combines R, SQL and JavaScript languages.

This project was highly interesting and offered many chances to learn new things from programmable logic controller systems, automation buses, software development and measurement technology. Seeing the system performing temperature and humidity measurements automatically and visualising historical measurements was the most rewarding part of the project.

Bibliography

Aumala, O. 1989. Mittaustekniikan perusteet. Espoo: Otatieto Oy.

Beckhoff. 2015. Analog input KL3021, KL3022. [Web page]. Beckhoff Automation. [Ref. 21 March 2015]. Available at: http://beckhoff.fi/english/bus_terminal/kl3021_kl3022.htm.

Beckhoff. 2015. Analog input KL3061, KL3062. [Web page]. Beckhoff Automation. [Ref. 21 March 2015]. Available at: http://beckhoff.fi/english/bus_terminal/kl3061_kl3062.htm.

Beckhoff. 2014. CX100x-0xxx | Basic CPU module. [Web page]. Beckhoff Automation. [Ref. 21 March 2015]. Available at: http://www.beckhoff.pl/english.asp?embedded_pc/cx100x_0xxx.htm.

Beckhoff. 2015. Digital Output KL2114, KL2134. [Web page]. Beckhoff Automation. [Ref. 21 March 2015]. Available at: http://beckhoff.fi/english/bus_terminal/kl2114_kl2134.htm.

Beckhoff. 2014. Embedded PC series CX8000. [Web page]. Beckhoff Automation. [Ref. 21 March 2015]. Available at: http://beckhoff.fi/english.asp?embedded_pc/embedded_pc_series_cx8000.htm.

Beckhoff. 2015. System Terminals KL9010. [Web page]. Beckhoff Automation. [Ref. 21 March 2015]. Available at: http://beckhoff.fi/english/bus_terminal/kl9010.htm.

Belfort Instrument. Undated. Application note: Tipping bucket precipitation sensors. [PDF document]. Belfort Instrument. [Ref. 3 February 2015]. Available at: http://belfortinstrument.com/wp-content/uploads/2013/09/TIPPING-BUCKET-AUTOMATED-PRECIPITATION-GAUGE.pdf.

Cheng, J. Undated. NVD3 line chart output. [Web page]. RStudio, Inc. [Ref. 16 April 2015]. Available at: http://shiny.rstudio.com/gallery/nvd3-line-chart-output.html.

Connolly, T. & Begg, C. 2005. Database Systems: A Practical Approach to Design, Implementation, and Management. Essex: Pearson Education Limited.

Heraeus Holding. 2015. General Technical Information. [Web page]. Heraeus Holding GmbH. [Ref. 16 February 2015]. Available at: http://heraeus-sensor-technology.com/en/produkte_1/allgemeineinformationen/infromations.aspx.

Härkönen, S., Lähteenmäki, I. & Välimaa, T. 1992. Teollisuuden mittaustekniikka/Analyysimittaukset. Helsinki: Oy Edita Ab.

Kipp & Zonen B.V. 2013. Introduction to Solar Radiation. [Web page]. Kipp & Zonen B.V. [Ref. 9 March 2015]. Available at: http://www.kippzonen.com/Knowledge-Center/Theoretical-info/Solar-Radiation/Introduction.

Kipp & Zonen B.V. 2013. SMP-3 Pyranometer. [PDF document]. Kipp & Zonen B.V. [Ref. 9 March 2015]. Available at: http://www.kippzonen.com/Product/201/SMP3-Pyranometer.

Kipp & Zonen B.V. Undated. Solar Energy Guide. [PDF document]. Kipp & Zonen B.V. [Ref. 9 March 2015]. Available at: http://www.kippzonen.com/Download/415/Solar-Energy-Guide-English.

Kreibich, J. A. 2010. Using SQLite. California: O'Reilly Media, Inc.

Mahnke, W., Leitner, S-H. & Damm, M. 2008. OPC Unified Architecture. Berlin-Heidelberg: Springer-Verlag.

MURR Elektronik. Undated. MPS Power Supply 1-phase. [Web page]. MURR Elektronik. [Ref. 21 March 2015]. Available at: http://onlineshop.murrelektronik.com/mediandoweb/index.php?ID_O_PRODUCT=12897&ID_O_TREE_GROUP=332&BEGIN=1&sLanguage=English&pageturning=10.

National Institute of Standards and Technology. 2011. Carbon dioxide. [Web page]. National Institute of Standards and Technology. [Ref. 3 February 2015]. Available at: http://webbook.nist.gov/cgi/cbook.cgi?ID=C124389&Type=IR-SPEC&Index=1#IR-SPEC.

Oracle. 2015. MySQL 5.5 reference manual. [PDF document]. Oracle Corporation. [Ref. 18 March 2015]. Available at: http://downloads.mysql.com/docs/refman-5.5-en.a4.pdf.

Rykovanov, A. Undated. FreeOpcUa. [Web page]. Rykovanov, A. [Ref. 16 April 2015]. Available at: http://freeopcua.github.io/.

Suomen Ilmatieteen Laitos. Undated. Sademäärä. [Web page]. Suomen Ilmatieteen Laitos. [Ref. 6 March 2015]. Available at: http://ilmatieteenlaitos.fi/sade.

The Resistor Guide. 2015. NTC thermistor. [Web page]. Mrmak, N., Oorschot P. & Pustjens, J-W. [Ref. 10 February 2015]. Available at: http://www.resistorguide.com/ntc-thermistor/.

Vaisala. 2013. Application note: Hiilidioksidin mittaaminen. [PDF document]. Vaisala Oy. [Ref. 3 March 2015]. Available at: http://www.vaisala.fi/Vaisala%20Documents/Application%20notes/CEN-TIA-Parameter-How-to-measure-CO2-Application-note-B211228FI-A.pdf.

Vaisala. 2012. Instrument Catalogue. [PDF document]. Vaisala Oy. [Ref. 18 February 2015]. Available at: http://www.vaisala.com/Vaisala%20Documents/Common%20Files/Instrument%20Catalog-B210974EN_web.pdf.

Vaisala. 2014. Product datasheet: RG13/RG13H Rain Gauges. [PDF document]. Vaisala Oy. [Ref. 6 March 2015]. Available at: http://www.vaisala.com/Vaisala%20Documents/Brochures%20and%20Datasheets/WEA-MET-G-RG13-RG13H-datasheet-B010195EN-F-LOW-v1.pdf.

Vaisala. 2012. Technology description: CARBOCAP® sensor. [PDF document]. Vaisala Oy. [Ref. 3 February 2015]. Available at: http://www.vaisala.com/Vaisala%20Documents/Technology%20Descriptions/CEN-TIA-G-Carbocap-Technology-description-B210780FI-C.pdf.

Vaisala. 2012. Vaisala HUMICAP Technology description. [PDF document]. Vaisala Oy. [Ref. 18 February 2015]. Available at: http://www.vaisala.fi/Vaisala%20Documents/Technology%20Descriptions/HUMICAP-Technology-description-B210781FI-C.pdf.

Vaisala. 2015. Vaisala INTERCAP® humidity sensor. [Web page]. Vaisala Oy. [Ref. 18 February 2015]. Available at: https://store.vaisala.com/eu/products/product/15778HM/vaisala-intercap-humidity-sensor.

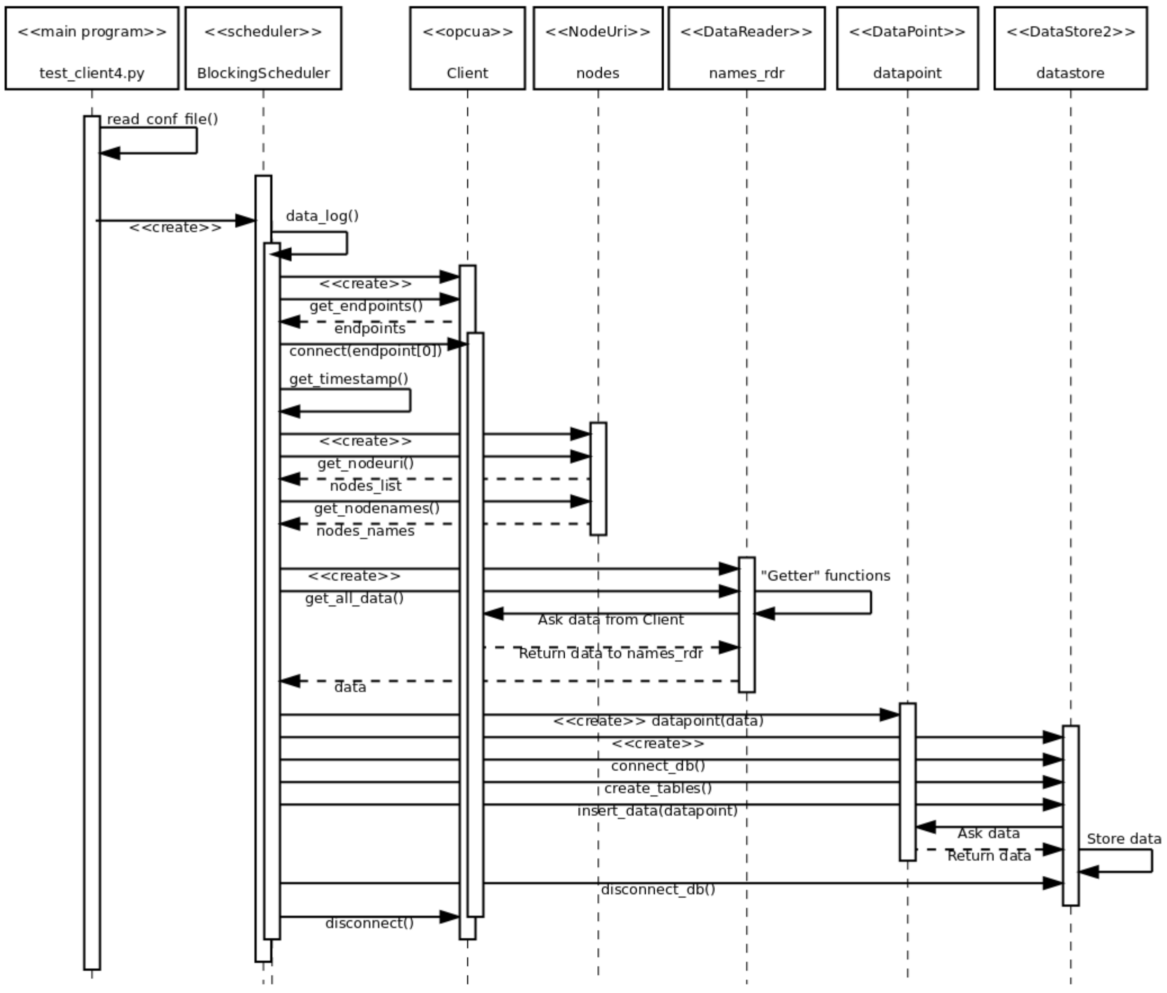
Vishay. 2012. Application note: NTC Thermistors. [PDF document]. Vishay BCCompoments. [Ref. 10 February 2015]. Available at: http://www.vishay.com/docs/29053/ntcintro.pdf.

Vishay. Undated. Curve Computation Program: NTC leaded and assemblies. [Web page]. Vishay BCCompoments. [Ref. 10 February 2015]. Available at: http://www.vishay.com/resistors-non-linear/curve-computation-list/.

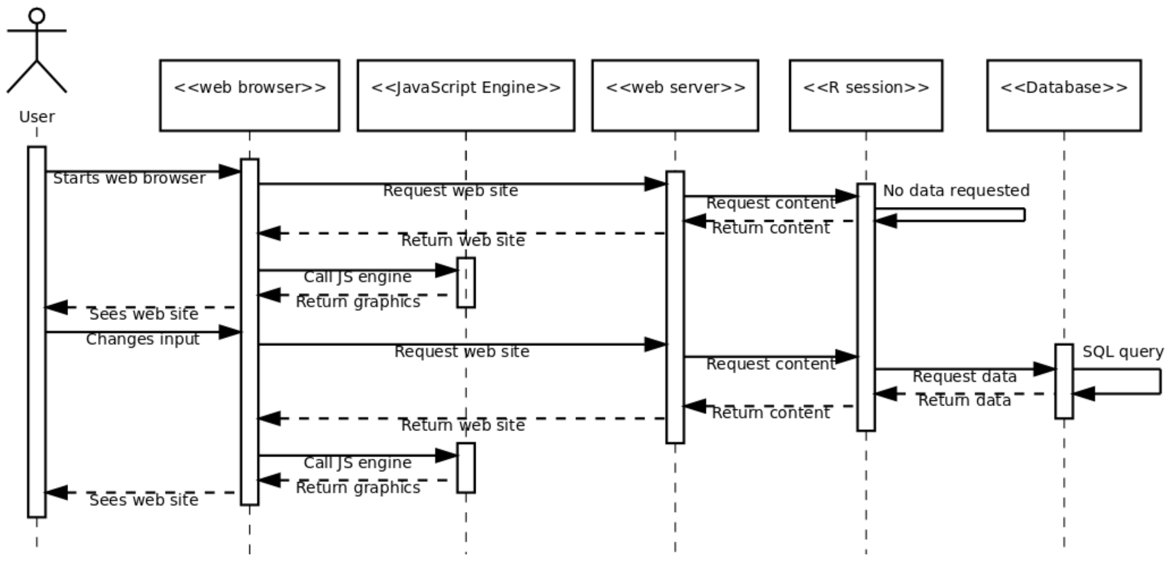
WIKA Group. 2014. Electronic temperature switch with display, model TSD-30. [PDF document]. WIKA Group. [Ref. 10 February 2015]. Available at: http://en-co.wika.de/upload/DS_TE6703_en_co_36632.pdf.

Appendices

Appendix 1: UML sequence diagram of data acquisition program.



Appendix 2: UML sequence diagram of data visualisation program.



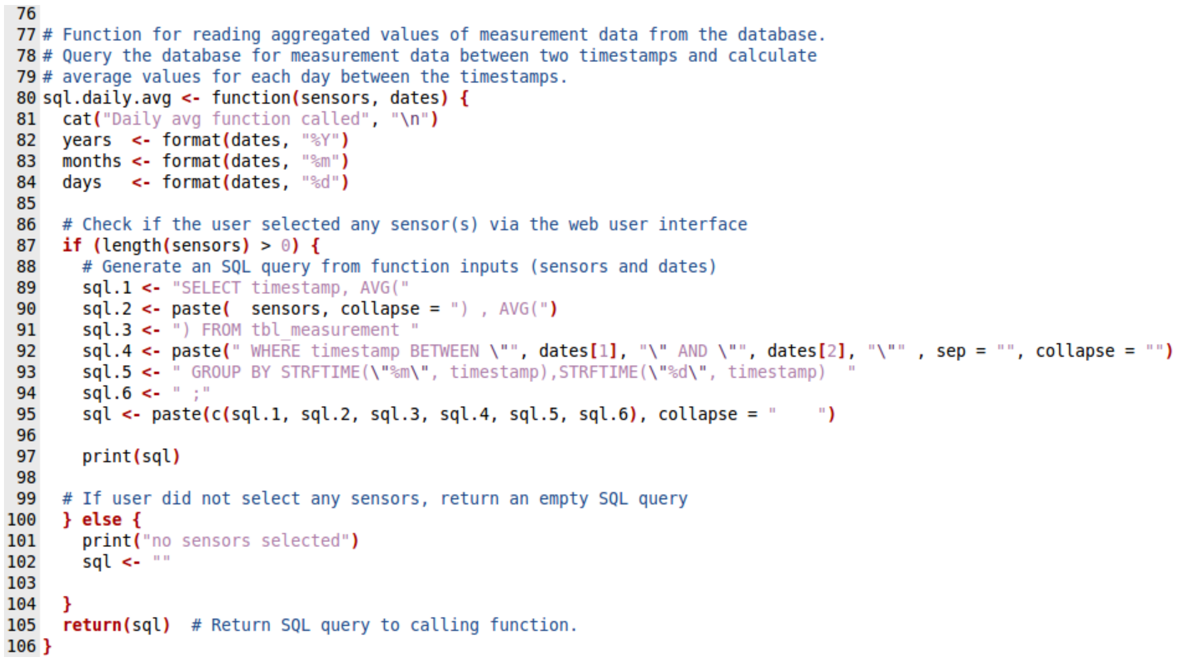
Appendix 3: User interface source code. (excerpt)

```
#Create navbar page
shinyUI(navbarPage(title = "Mittaukset",
                   id = "tab_selected",
                   fluid = FALSE,
                   #theme = "bootstrap.css",



  #Create Temperature tab panel
  tabPanel(title = "Lämpötila", value = "tab_temp",
    #Create sidebar layout for structural data
    sidebarLayout(position = "left",
                  fluid = FALSE,
      #Create left side panel for sidebar layout
      sidebarPanel(
        h3("Valinnat"),  #Title text for left side panel
        hr(),

        #Create date range input for data point selection
        #By default, select last 5 days' measurement data
        dateRangeInput(inputId = "temp_timeRange",
                       separator = "-",
                       label = strong("Aikaväli (alku - loppu)"),
                       weekstart = 1,
                       language = "fi",
                       end = Sys.Date(),
                       start = Sys.Date() - 1,
                       max = Sys.Date()
                       ),
```

Appendix 4: SQLite database access in R. (excerpt)

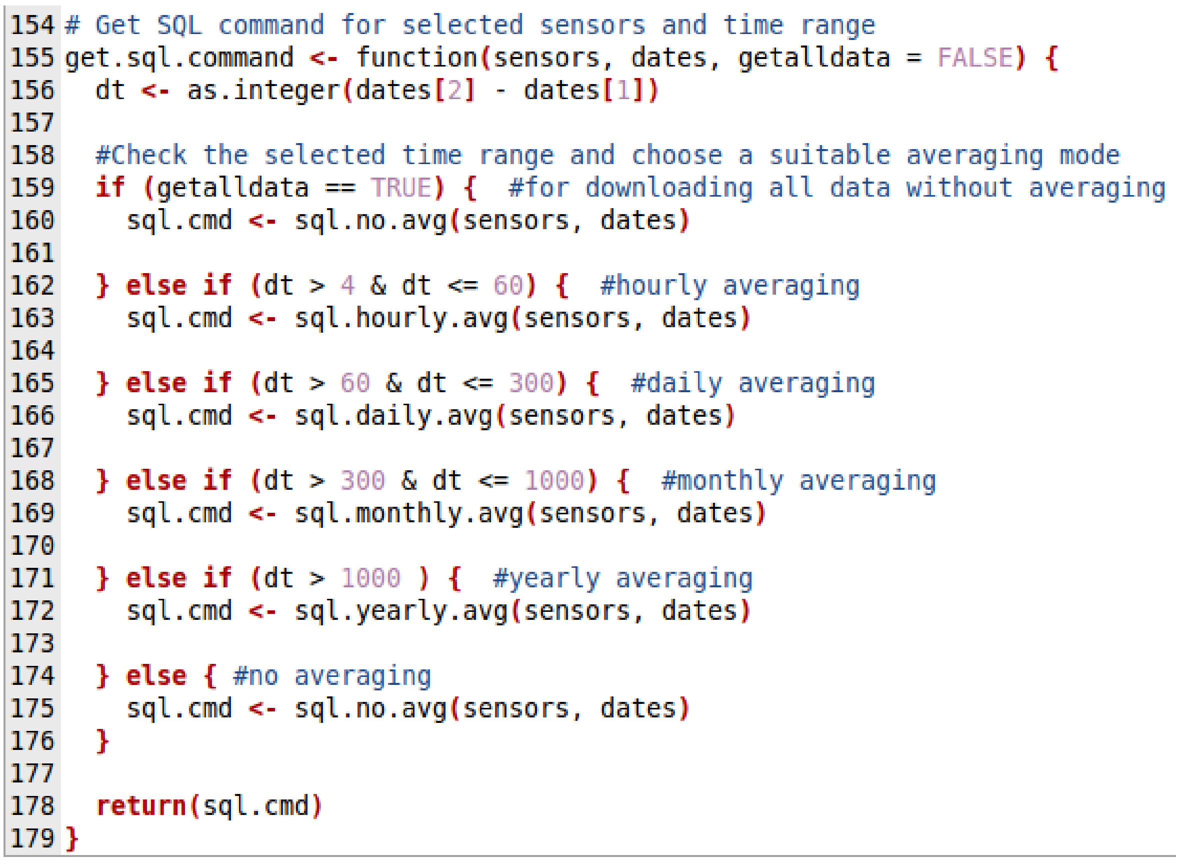
```r
#Function for reading measurement data values from database and returning them
values <- function(daterange, sensors, getalldata = FALSE)({
  # Database driver for SQLite
  drv <- dbDriver(drvName = "SQLite")

  #Define SQLite database file
  #dbname <- "./testdb.db3"
  dbname <- "/home/user/projects/freeopcua/cave_toimisto.db3"

  #Database connection
  dbconnection <- dbConnect(drv, dbname)

  #Convert start & end dates to text strings
  date.start <- as.character(daterange[1])
  date.end   <- as.character(daterange[2])
  timespan.days <- as.integer(daterange[2] - daterange[1])

  #Call SQL query generator function
  sql <- get.sql.command(sensors = sensors, dates = daterange, getalldata = getalldata)
  print(paste("SQL command is:",sql))

  #Query the database with RSQLite function
  #if(nrow(data.df) <= 1) {
  if (nchar(sql) > 0) {
    data.df <- dbGetQuery(conn = dbconnection, statement = sql)

  } else {  #If no data was returned, make an empty data.frame
    data.df <- data.frame()

  }

  # Return a data.frame object
  return(data.df)
})
```

Appendix 5: Aggregation of measurement data in R with embedded SQL. (excerpt)



```r

# Function for reading aggregated values of measurement data from the database.
# Query the database for measurement data between two timestamps and calculate
# average values for each day between the timestamps.
sql.daily.avg <- function(sensors, dates) {
  cat("Daily avg function called", "\n")
  years  <- format(dates, "%Y")
  months <- format(dates, "%m")
  days   <- format(dates, "%d")

  # Check if the user selected any sensor(s) via the web user interface
  if (length(sensors) > 0) {
    # Generate an SQL query from function inputs (sensors and dates)
    sql.1 <- "SELECT timestamp, AVG("
    sql.2 <- paste(  sensors, collapse = ") , AVG(")
    sql.3 <- ") FROM tbl_measurement "
    sql.4 <- paste(" WHERE timestamp BETWEEN \"", dates[1], "\" AND \"", dates[2], "\"" , sep = "", collapse = "")
    sql.5 <- " GROUP BY STRFTIME(\"%m\", timestamp),STRFTIME(\"%d\", timestamp)  "
    sql.6 <- " ;"
    sql <- paste(c(sql.1, sql.2, sql.3, sql.4, sql.5, sql.6), collapse = "    ")

    print(sql)

  # If user did not select any sensors, return an empty SQL query
  } else {
    print("no sensors selected")
    sql <- ""

  }
  return(sql)  # Return SQL query to calling function.
}
```

Appendix 6: Selection of aggregation mode in R.

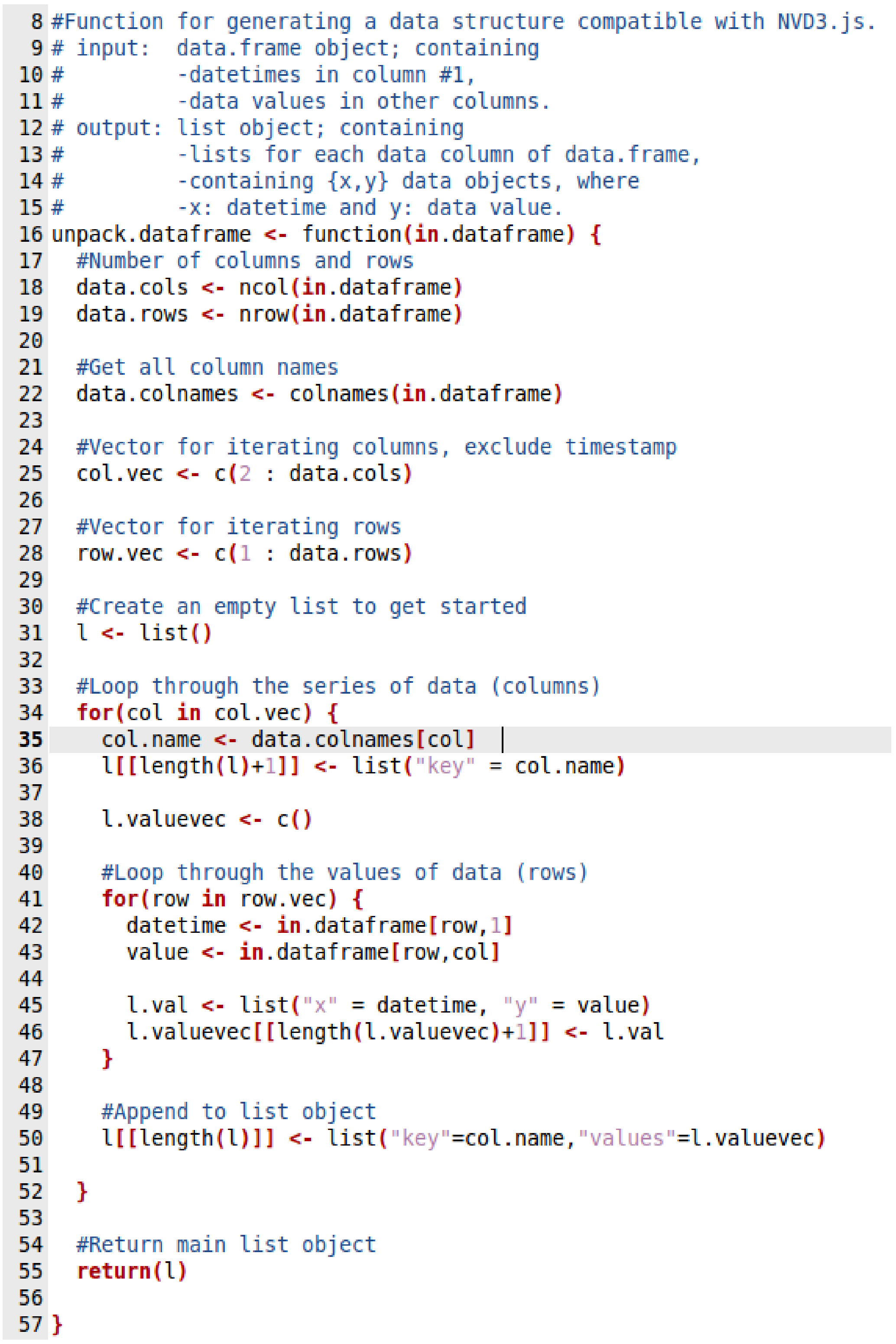


```r
# Get SQL command for selected sensors and time range
get.sql.command <- function(sensors, dates, getalldata = FALSE) {
  dt <- as.integer(dates[2] - dates[1])

  #Check the selected time range and choose a suitable averaging mode
  if (getalldata == TRUE) {  #for downloading all data without averaging
    sql.cmd <- sql.no.avg(sensors, dates)

  } else if (dt > 4 & dt <= 60) {  #hourly averaging
    sql.cmd <- sql.hourly.avg(sensors, dates)

  } else if (dt > 60 & dt <= 300) {  #daily averaging
    sql.cmd <- sql.daily.avg(sensors, dates)

  } else if (dt > 300 & dt <= 1000) {  #monthly averaging
    sql.cmd <- sql.monthly.avg(sensors, dates)

  } else if (dt > 1000 ) {  #yearly averaging
    sql.cmd <- sql.yearly.avg(sensors, dates)

  } else { #no averaging
    sql.cmd <- sql.no.avg(sensors, dates)
  }

  return(sql.cmd)
}
```

Appendix 7: Reactive functions for accessing the measurement data in R.

```r
#Reactive function for obtaining temperature data from database.
get_temp_data_react <- reactive({
  # Store measurement time range into Date object.
  daterange <- input$temp_timeRange
  daterange <- c( daterange[1], daterange[2]+1  )

  sensor.selection <- input$temp_select_sensor

  data.selection <- sensor.selection
  print(data.selection)

  #Use values() function to get the data.
  #Pass date range and selected sensors into the function.
  #Return a data.frame. Each column will be a series in the line chart.
  measurements.df <- values(daterange, data.selection)

  #return data frame to the calling function
  return(measurements.df)
})

#Function for rendering temperature data with NVD3.
output$mychart_temp <- renderLineChart({
  #Make dependency for plot button
  input$temp_button_plot

  #Make get_temp_data_react() function as reactive conductor,
  # store measurement data into a data.frame.
  measurements.df <- isolate(get_temp_data_react())
  return(measurements.df)
})
```

Appendix 8: Function for data structure conversion between R and JavaScript.



```r
#Function for generating a data structure compatible with NVD3.js.
# input:  data.frame object; containing
#         -datetimes in column #1,
#         -data values in other columns.
# output: list object; containing
#         -lists for each data column of data.frame,
#         -containing {x,y} data objects, where
#         -x: datetime and y: data value.
unpack.dataframe <- function(in.dataframe) {
  #Number of columns and rows
  data.cols <- ncol(in.dataframe)
  data.rows <- nrow(in.dataframe)

  #Get all column names
  data.colnames <- colnames(in.dataframe)

  #Vector for iterating columns, exclude timestamp
  col.vec <- c(2 : data.cols)

  #Vector for iterating rows
  row.vec <- c(1 : data.rows)

  #Create an empty list to get started
  l <- list()

  #Loop through the series of data (columns)
  for(col in col.vec) {
    col.name <- data.colnames[col]
    l[[length(l)+1]] <- list("key" = col.name)

    l.valuevec <- c()

    #Loop through the values of data (rows)
    for(row in row.vec) {
      datetime <- in.dataframe[row,1]
      value <- in.dataframe[row,col]

      l.val <- list("x" = datetime, "y" = value)
      l.valuevec[[length(l.valuevec)+1]] <- l.val
    }

    #Append to list object
    l[[length(l)]] <- list("key"=col.name,"values"=l.valuevec)

  }

  #Return main list object
  return(l)

}
```

Appendix 9: Reading data values from OPC UA server.

```python
import uuid
import opcua
#from array import array

class DataReader:

    #Initialise a DataReader object
    def __init__(self, client, var_uri, timestamp):
        #Get the root node of OPC UA namespace
        self.__rootnode = client.get_root_node()

        #Get node information
        self.__var_uri = var_uri["NODES_LIST"]
        self.__var_type = var_uri["NODES_TYPES"]

        #Generate unique identifier
        self.__uuid = str(uuid.uuid4())

        #A timestamp for data
        self.__timestamp = timestamp


    #Get names from node URI list
    def get_names(self):
        self.__names = []  #Initialise a list for names

        #For each node URI, get its name
        for item in self.__var_uri:
            self.__names.append(self.__rootnode.get_child(item).get_name().name)
        return(self.__names)


    #Get values from node URI list
    def get_values(self):
        self.__values = []  #Initialise a list for values

        #For each node URI, get its value
        for item in self.__var_uri:
            item_value = self.__rootnode.get_child(item).get_value()
            self.__values.append(item_value)

        return(self.__values)
```

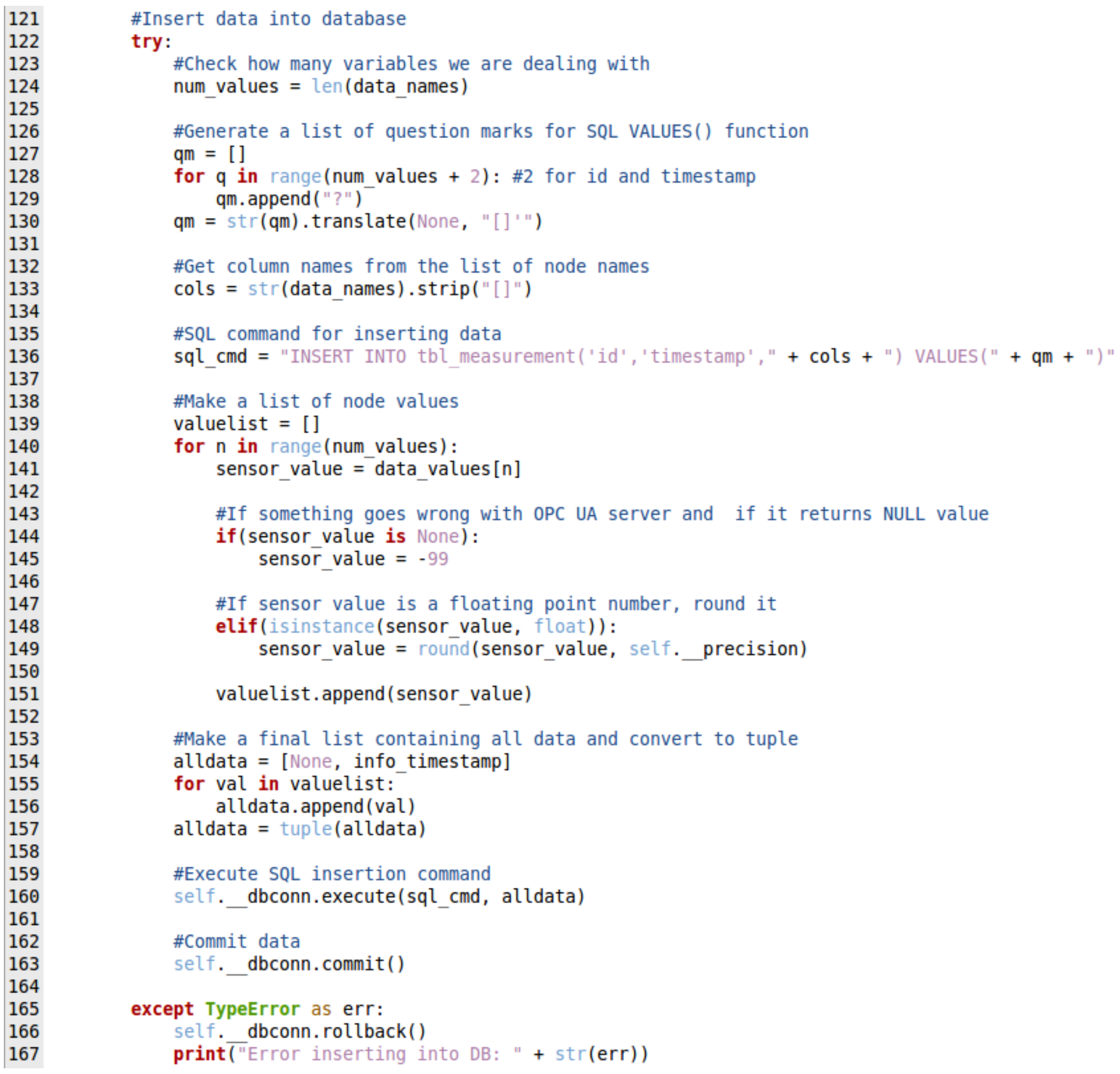
Appendix 10: Initialise and establish a connection to SQLite database.

```python
import sqlite3 as data_db
import os
import NodeUri

class DataStore:

    #Initialise a DataStore object with settings dictionary
    # and a list of node names
    def __init__(self, conf_dict, nodes_names):
        #Create SQLite3 database file
        self.__dbfile = conf_dict["STORAGE"]["SQLITE3_FILE"]
        self.__dbconn = None
        self.__dbcursor = None

        #Set DataStore settings from configuration dictionary
        #Create nodeuri
        #~ nodes = NodeUri.NodeUri(conf_dict["MONITORING"])
        self.__nodes_names = nodes_names
        self.__nodes_types = conf_dict["MONITORING"]["NODES_TYPES"]
        self.__precision = conf_dict["STORAGE"]["STORAGE_PRECISION"]

        #Check if database file exists already, connect if it does
        self.__dbexists = False
        db_exists = os.path.exists(self.__dbfile)
        if(db_exists):
            #~ print("DB '" + str(self.__dbfile) + "' was found.")
            self.__dbexists = True
        else:
            self.connect_db()


    #Connect to database file
    def connect_db(self):
        dbfile = self.__dbfile

        #Try connecting to the database
        try:
            #~ print("Connecting to database:" + str(self.__dbfile))
            self.__dbconn = data_db.connect(self.__dbfile)
            self.__dbcursor = self.__dbconn.cursor()
            self.__dbexists = True
        except data_db.OperationalError as err:
            print("Error connecting to DB '" + str(dbfile) + "': " + str(err))
            self.__dbconn = None
            self.__dbcursor = None

```

Appendix 11: Create SQLite database tables.

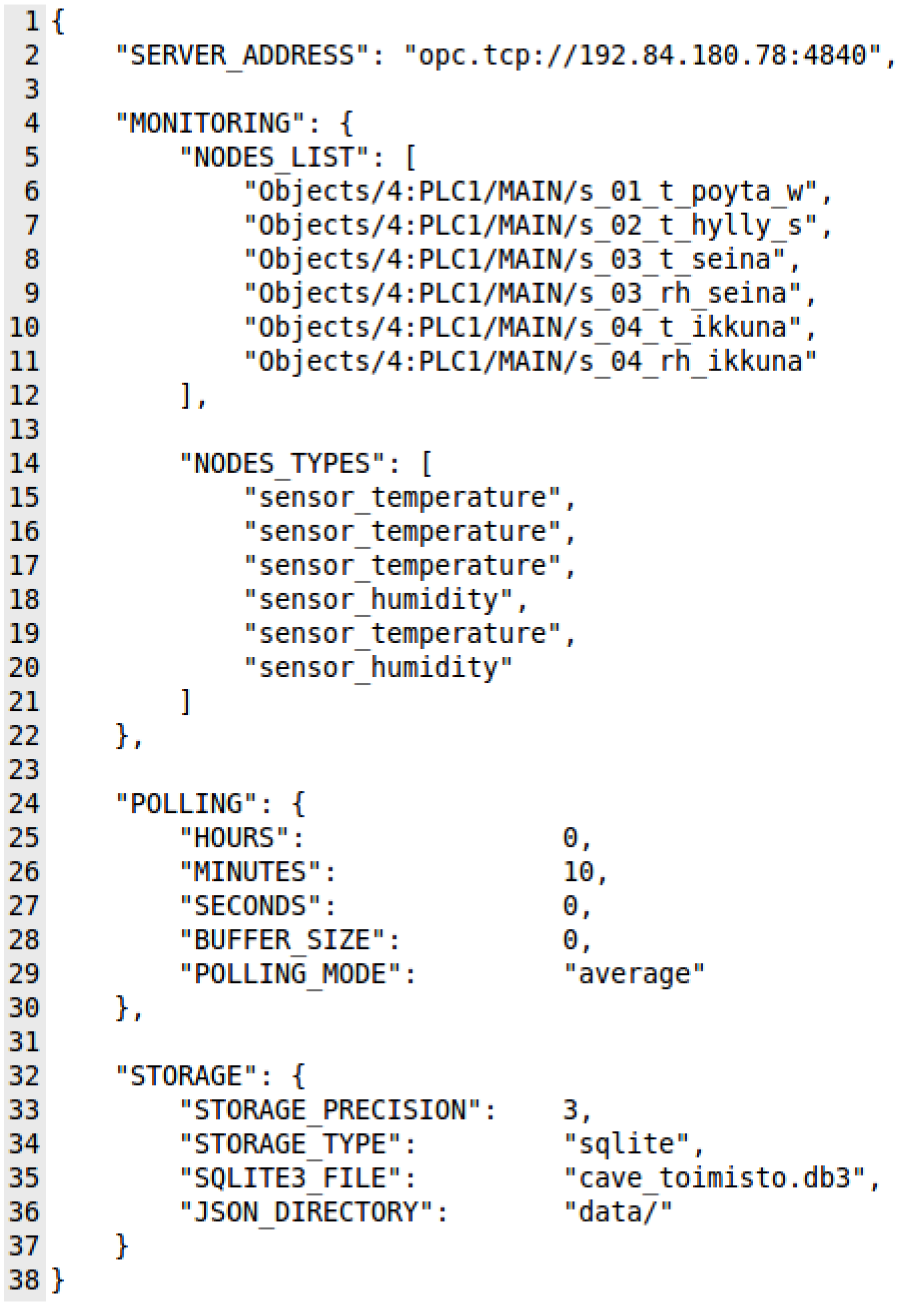
```
    #Prepare database tables
    def create_tables(self):
        if(self.__dbexists):
            try:
                #Create measurements table if it does not exist
                sql_cmd = "CREATE TABLE IF NOT EXISTS 'tbl_measurement'(" +\
                    "'id'            INTEGER PRIMARY KEY, " +\
                    "'timestamp'     DATETIME )"
                #~ print(sql_cmd)
                self.__dbconn.execute(sql_cmd)

                #Initialise a list for column names
                column_names = []

                #Get all columns present in measurements table
                sql_cmd = "PRAGMA table_info(tbl_measurement)"
                columns = self.__dbconn.execute(sql_cmd)

                #Get column name, encode to UTF-8 and add to list
                for col in columns:
                    #~ print(i[1].encode("utf-8"))
                    column_names.append(col[1].encode("utf-8"))

                #Create column for sensor if it does not exist
                for n in self.__nodes_names:
                    try:
                        column_names.index(n) #check if column exists

                    #If column 'n' does not exist, create it
                    except ValueError as err:
                        print("Adding column to database: " + str(n))
                        sql_cmd = "ALTER TABLE tbl_measurement " +\
                            "ADD COLUMN " + str(n) + " TEXT"
                        #~ print(sql_cmd)
                        self.__dbconn.execute(sql_cmd)


            except data_db.OperationalError as err:
                print("Error with database operation: " + str(err))

            finally:
                if(self.__dbconn):
                    self.__dbconn.close()
```

Appendix 12: Insert data values into SQLite database.



```python
        #Insert data into database
        try:
            #Check how many variables we are dealing with
            num_values = len(data_names)

            #Generate a list of question marks for SQL VALUES() function
            qm = []
            for q in range(num_values + 2): #2 for id and timestamp
                qm.append("?")
            qm = str(qm).translate(None, "[]'")

            #Get column names from the list of node names
            cols = str(data_names).strip("[]")

            #SQL command for inserting data
            sql_cmd = "INSERT INTO tbl_measurement('id','timestamp'," + cols + ") VALUES(" + qm + ")"

            #Make a list of node values
            valuelist = []
            for n in range(num_values):
                sensor_value = data_values[n]

                #If something goes wrong with OPC UA server and  if it returns NULL value
                if(sensor_value is None):
                    sensor_value = -99

                #If sensor value is a floating point number, round it
                elif(isinstance(sensor_value, float)):
                    sensor_value = round(sensor_value, self.__precision)

                valuelist.append(sensor_value)

            #Make a final list containing all data and convert to tuple
            alldata = [None, info_timestamp]
            for val in valuelist:
                alldata.append(val)
            alldata = tuple(alldata)

            #Execute SQL insertion command
            self.__dbconn.execute(sql_cmd, alldata)

            #Commit data
            self.__dbconn.commit()

        except TypeError as err:
            self.__dbconn.rollback()
            print("Error inserting into DB: " + str(err))
```

Appendix 13: Configuration file for the data acquisition program. (example)



```
{
    "SERVER_ADDRESS": "opc.tcp://192.84.180.78:4840",

    "MONITORING": {
        "NODES_LIST": [
            "Objects/4:PLC1/MAIN/s_01_t_poyta_w",
            "Objects/4:PLC1/MAIN/s_02_t_hylly_s",
            "Objects/4:PLC1/MAIN/s_03_t_seina",
            "Objects/4:PLC1/MAIN/s_03_rh_seina",
            "Objects/4:PLC1/MAIN/s_04_t_ikkuna",
            "Objects/4:PLC1/MAIN/s_04_rh_ikkuna"
        ],

        "NODES_TYPES": [
            "sensor_temperature",
            "sensor_temperature",
            "sensor_temperature",
            "sensor_humidity",
            "sensor_temperature",
            "sensor_humidity"
        ]
    },

    "POLLING": {
        "HOURS":                0,
        "MINUTES":              10,
        "SECONDS":              0,
        "BUFFER_SIZE":          0,
        "POLLING_MODE":         "average"
    },

    "STORAGE": {
        "STORAGE_PRECISION":    3,
        "STORAGE_TYPE":         "sqlite",
        "SQLITE3_FILE":         "cave_toimisto.db3",
        "JSON_DIRECTORY":       "data/"
    }
}
```

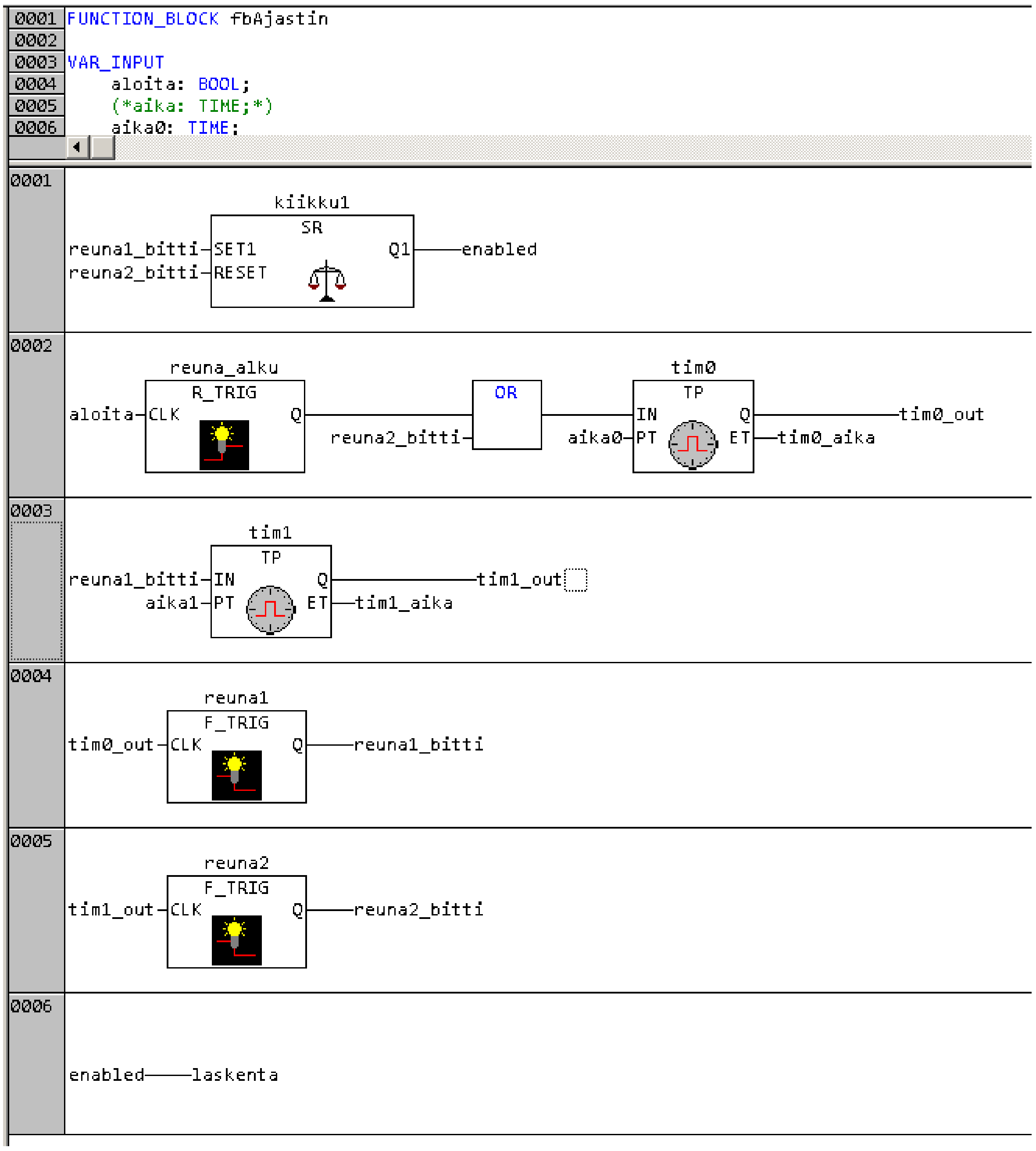
Appendix 14: Reading monitored node URIs from configuration file.

```python
class NodeUri:

    #Initialise nodes' URIs and types, read into list objects
    def __init__(self, nodeuri):
        self.__nodeuri = nodeuri["NODES_LIST"]
        self.__nodetype = nodeuri["NODES_TYPES"]

        #Check if length of nodeURI and nodetype lists are not same
        if(len(self.__nodeuri) != len(self.__nodetype)):
            print("Error, check node configuration file.")
            print("Using 'default' definition for all nodes.")
            self.__nodetype = []
            for n in range(len(self.__nodeuri)):
                self.__nodetype.append("default")


    #Read list of nodes from configuration file
    def get_nodeuri(self):
        #~ nodes = cd["MONITORING"]["NODES_LIST"]

        #Make dictionary containing node URI list and node types list
        nodes_dict = {}

        #Make a list containing lists of node path elements
        nodeslist = []  #initialise a list for nodes

        #Check if there are any nodes to monitor
        if(len(self.__nodeuri) > 0):
            for node in self.__nodeuri:
                node_str = []

                #split node path to elements separated by '/' character
                node = node.split("/")

                #encode node path elements to UTF-8 text strings
                for item in node:
                    node_str.append(item.encode("utf-8"))
                nodeslist.append(node_str)   #and add to list of nodes

        #Make a list containing node types
        typeslist = []
        for nodetype in self.__nodetype:
            #Encode to UTF-8 text and append to list
            typeslist.append(nodetype.encode("utf-8"))

        #Insert node information into dictionary
        nodes_dict = {"NODES_LIST":  nodeslist,
                      "NODES_TYPES": typeslist}

        #Return dictionary object containing node URIs and node types
        return(nodes_dict)
```

Appendix 15: Structure of the data received from OPC UA server. (example)

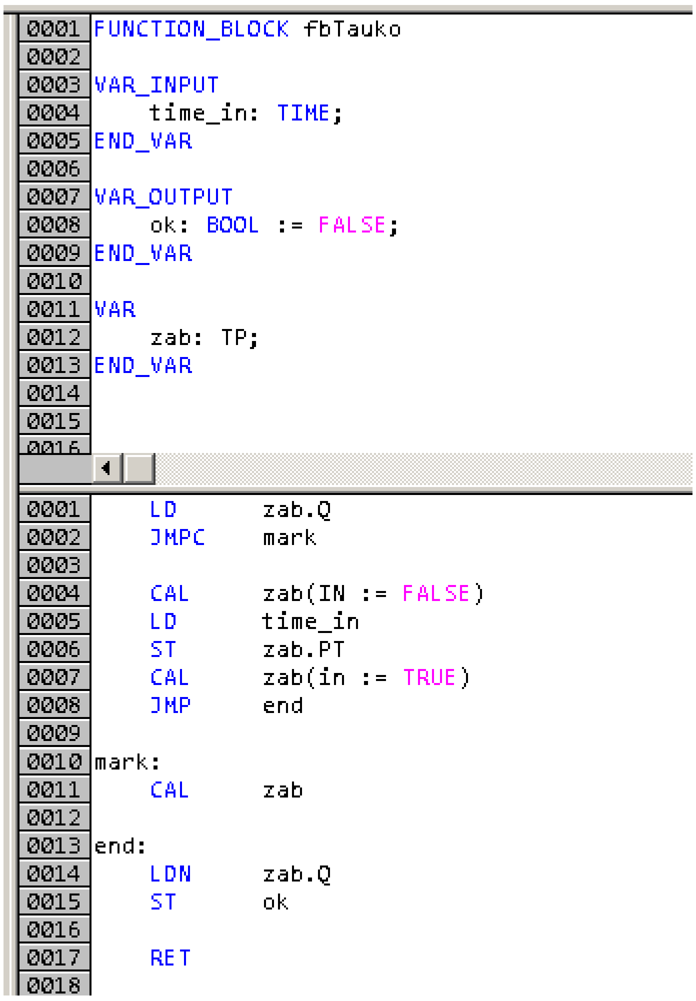
```
{
    "info": {
        "timestamp": "2014-12-18T10:38:22",
        "uuid": "17aa2266-212c-4dee-9b62-6d718d267210"
    },
    "datapoint": {
        "node_value": [
            7193,
            14915,
            true,
            true,
            25.518362045288086,
            22.06304931640625,
            12.34000015258789
        ],
        "node_type": [
            "int",
            "int",
            "bool",
            "bool",
            "float",
            "float",
            "float"
        ],
        "node_id": [
            "MAIN.temp01_integer_value",
            "MAIN.temp02_integer_value",
            "MAIN.led1_boolean_value",
            "MAIN.led2_boolean_value",
            "MAIN.temp01_float_value",
            "MAIN.temp02_float_value",
            "MAIN.testfloat"
        ],
        "node_name": [
            "temp01_integer_value",
            "temp02_integer_value",
            "led1_boolean_value",
            "led2_boolean_value",
            "temp01_float_value",
            "temp02_float_value",
            "testfloat"
        ]
    }
}
```

Appendix 16: Timing function block fbAjastin.



```
FUNCTION_BLOCK fbAjastin

VAR_INPUT
    aloita: BOOL;
    (*aika: TIME;*)
    aika0: TIME;
```

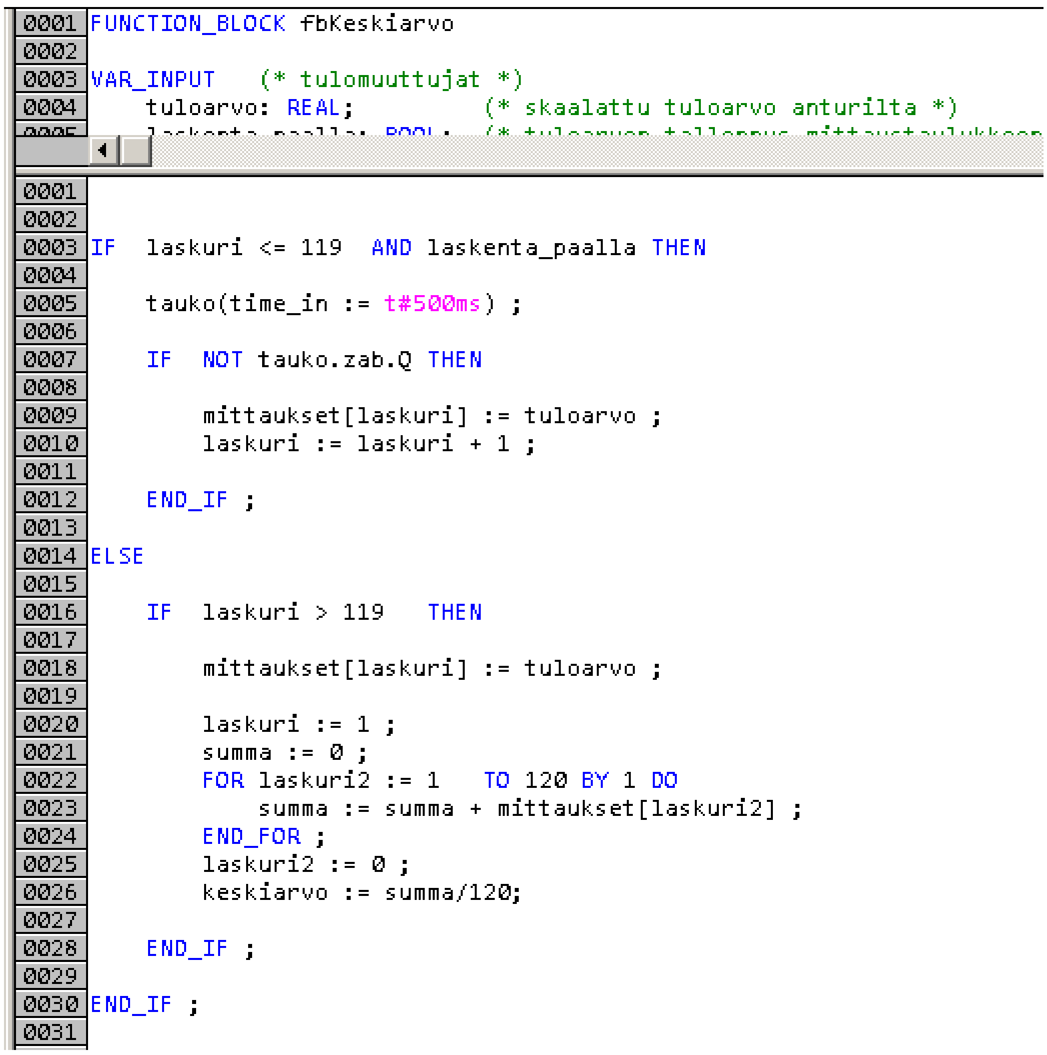
Appendix 17: Timing function block fbTauko (from Beckhoff TwinCAT 2 help file).



```
FUNCTION_BLOCK fbTauko

VAR_INPUT
    time_in: TIME;
END_VAR

VAR_OUTPUT
    ok: BOOL := FALSE;
END_VAR

VAR
    zab: TP;
END_VAR


```

```
    LD      zab.Q
    JMPC    mark

    CAL     zab(IN := FALSE)
    LD      time_in
    ST      zab.PT
    CAL     zab(in := TRUE)
    JMP     end

mark:
    CAL     zab

end:
    LDN     zab.Q
    ST      ok

    RET

```

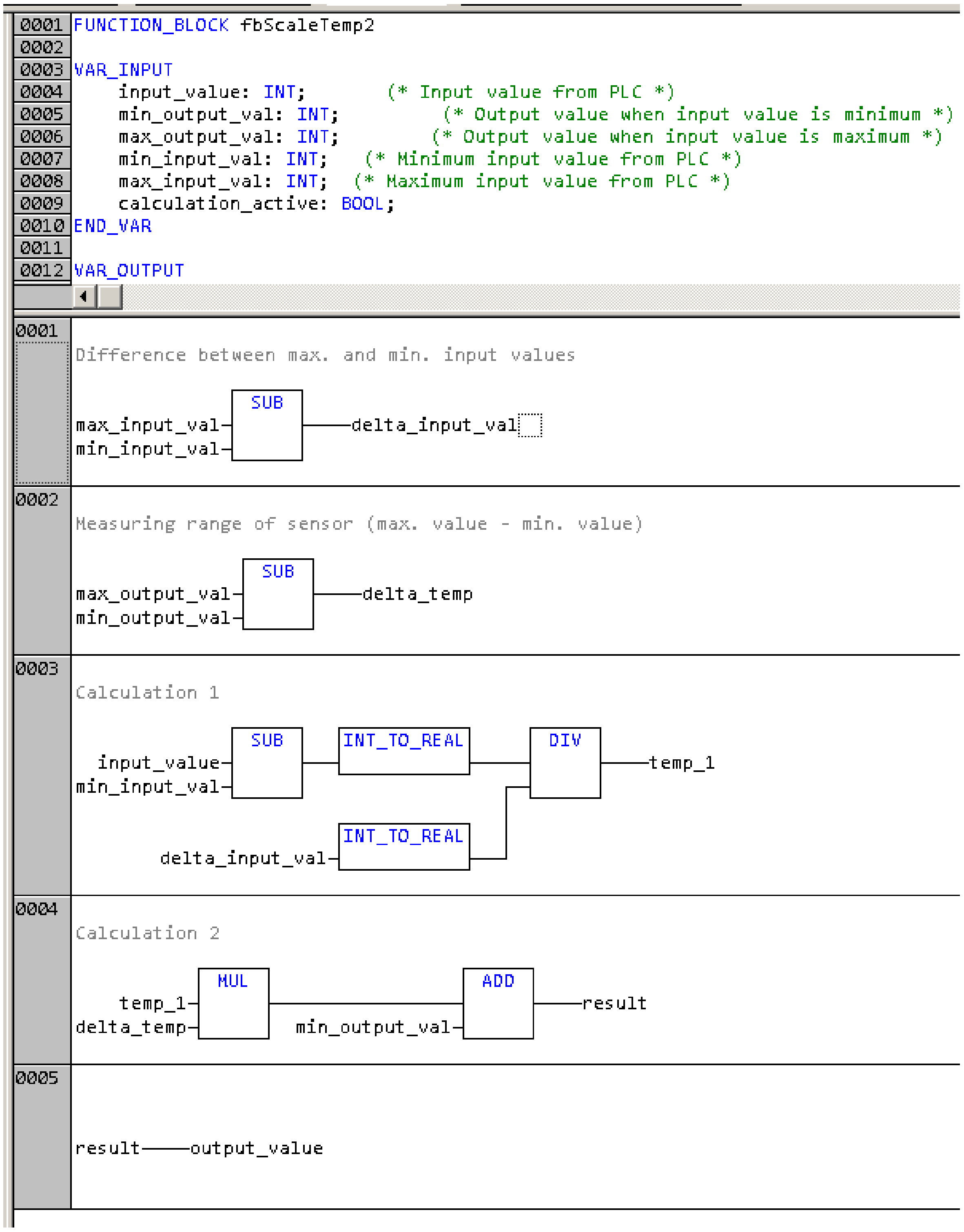
Appendix 18: Averaging function block fbKeskiarvo.



```
FUNCTION_BLOCK fbKeskiarvo

VAR_INPUT    (* tulomuuttujat *)
    tuloarvo: REAL;          (* skaalattu tuloarvo anturilta *)
    laskenta_paalla: BOOL;   (* tuloarvon tallennus mittaustaulukkoon
```

```


IF   laskuri <= 119   AND laskenta_paalla THEN

    tauko(time_in := t#500ms) ;

    IF  NOT tauko.zab.Q THEN

        mittaukset[laskuri] := tuloarvo ;
        laskuri := laskuri + 1 ;

    END_IF ;

ELSE

    IF  laskuri > 119   THEN

        mittaukset[laskuri] := tuloarvo ;

        laskuri := 1 ;
        summa := 0 ;
        FOR laskuri2 := 1   TO 120 BY 1 DO
            summa := summa + mittaukset[laskuri2] ;
        END_FOR ;
        laskuri2 := 0 ;
        keskiarvo := summa/120;

    END_IF ;

END_IF ;

```

Appendix 19: Scaling function block fbScaleTemp2.



```
FUNCTION_BLOCK fbScaleTemp2

VAR_INPUT
    input_value: INT;        (* Input value from PLC *)
    min_output_val: INT;          (* Output value when input value is minimum *)
    max_output_val: INT;        (* Output value when input value is maximum *)
    min_input_val: INT;    (* Minimum input value from PLC *)
    max_input_val: INT;  (* Maximum input value from PLC *)
    calculation_active: BOOL;
END_VAR

VAR_OUTPUT
```

Difference between max. and min. input values

Measuring range of sensor (max. value - min. value)

Calculation 1

Calculation 2