intab

# WIRELESS SYSTEM

Modbus

2018-12-21

# 1 Table of contents

## 1.1 Tables

# 2 SCOPE

## 2.1 PURPOSE

This document describes the Modbus implementation for the I/O Board of the WiSensys system. In this document the limitations and possible deviations to the Modbus Application Protocol Specification will be listed as well as explained why this is done.

## 2.2 DOCUMENT OVERVIEW

The document is divided into a number of sections:

- an overview of MODBUS;
- how to configure the interface
- communication examples.

## 2.3 DOCUMENT HISTORY

| Rev. | Date | Author | Fault/Chance |
|---|---|---|---|
| PA1 | 2005-12-06 | GePl | Initial Version |
| PA2 | 2005-12-09 | JoTu | Formatting and small changes |
| R1A | 2006-02-14 | GePl | Some minor changes |
| PB1 | 2006-02-15 | GePl | |
| P1B2 | 2006-07-14 | HaHo | - Restructuring of the document<br>- Better description of the implementation limitation chosen for the design |
| R1B | 2006-07-19 | HaHo | Made an R release after review comments on P1B2 |
| P1C1 | 2008-10-09 | HaHo | Added new sensor types<br>- Low voltage sensor - $CO_2$ sensor<br>- Energy sensor - Pulse sensor<br>- Energy Multi sensor - Pulse Energy sensor |
| P1C2 | 2010-08-05 | HaHo | Added new sensor types |
| P1C3 | 2011-06-26 | HaHo | Register descriptions for the mili-voltage sensor was wrong, it only described a single register, but the mili-voltage sensor uses 2 registers |
| P1C4 | 2017-04-16 | WiJo | - Added function codes 0x03 and 0x10 to write sensor parameters<br>- Added new sensor types |
| P1C5 | 2017-06-21 | WiJo | - Changed register definitions and examples for function codes 0x03 and 0x10 |
| P1C6 | 2017-06-30 | WiJo | Chapter 7.3.2:<br>- Changed ID output active time from 4096 to 8192<br>- Changed Sensor type Current to Wireless Motor Control<br>- Added sensor type Wireless Valve Control<br>- Added and updated Examples<br>Chapter 8 and 9: Updated examples |
| P1C7 | 2017-09-30 | WiJo | Chapter 7.2.3 changed bit 1 to reserved |
| P1C8 | 2017-12-04 | WiJo | Chapter 7.2.2 and 7.3.2:<br>- Added sensor type tap (Wireless Valve Control)<br>- Updated tap sensor register definitions<br>- Added "MODBUS extended feature" which increases the registers per sensor from 10 to 20.<br>- Added RSSI value to modbus register 19. |
| P1C9 | 2018-04-03 | WiJo | Added sensor type pressure |
| P1C10 | 2018-11-28 | WiJo | Added sensor type all-in-one<br>Corrected device type for Temperature 2 decimals |

# 3 APPLICABLE DOCUMENTS

[1] Title: MODBUS Application Protocol specification
Author: www.MODBUS-IDA.org
Doc. no.:
Revision: v1.1a

[2] Title: MODBUS over serial line specification and implementation guide
Author: www.MODBUS.org
Doc. no.:
Revision: v1.0

[3] Title: General WiV Radio Protocol
Author: Han Hoekstra
Doc. no:
Revision: P1A15

# 4 NOTES

## 4.1 GLOSSARY

## 4.2 ABBREVIATIONS

| | |
|---|---|
| BE | Big Endian |
| FAQ | Frequently Asked Questions |
| LE | Little Endian |
| LSB | Least Significant Byte |
| LSW | Least Significant Word |
| MSB | Most Significant Byte |
| MSW | Most Significant Word |
| WiV | Wireless Value |
| Word | 16-bit value |

# 5 General

This section gives a global overview of the MODBUS protocol as used for the WiSensys sensor system.

The WiSensys MODBUS IO board has the possibility to connect to the MODBUS using an RS485 connection. The implementation is an implementation of the MODBUS slave protocol and is a subset of the protocol as described in [1]. The RTU version of the protocol has been implemented; this means only none ASCII binary data will be used.

# 6 MODBUS protocol overview

This section is taken from [1] and is presented here as a reference only, not as a complete specification of the Modbus protocol.

## 6.1 Physical Layer

The communication will use RS-485 as physical layer. The communication is in an asynchronous format with one start bit, eight data bits, one parity bit, and one stop bit. Even and odd parity is supported. If no parity is specified, the number of stop bits can be user configured for either one or two stop bits. Baud rates supported are 1200 – 115K2 baud.

## 6.2 Datalink Layer

The MODBUS protocol defines a simple protocol data unit **(PDU)** independent of the underlying communication layers. The mapping of MODBUS protocol on specific buses or network can introduce some additional fields on the application data unit **(ADU)**.

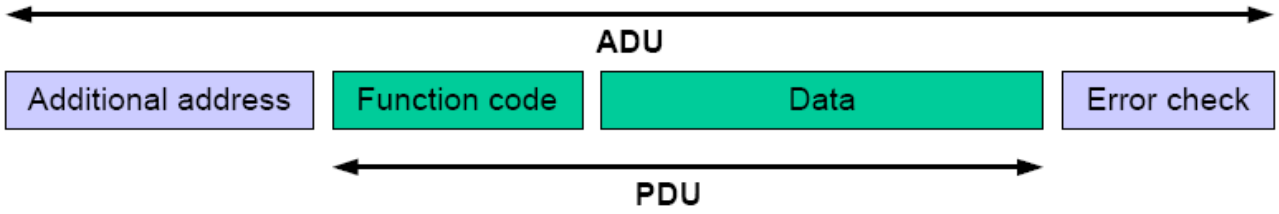



**Figure 1: General MODBUS frame**

The MODBUS application data unit is built by the client that initiates a MODBUS transaction. The function indicates to the server what kind of action to perform. The MODBUS application protocol establishes the format of a request initiated by a client.

The function code field of a MODBUS data unit is coded in one byte. Valid codes are in the range of 1 ... 255 decimal (128 – 255 reserved for exception responses). When a message is sent from a Client to a Server device the function code field tells the server what kind of action to perform. Function code "0" is not valid.

Sub-function codes are added to some function codes to define multiple actions.

The data field of messages sent from a client to server devices contains additional information that the server uses to take the action defined by the function code. This can include items like discrete and register addresses, the quantity of items to be handled and the count of actual data bytes in the field.

The data field may be nonexistent (of zero length) in certain kinds of requests, in this case the server does not require any additional information. The function code alone specifies the action.

If no error occurs related to the MODBUS function requested in a properly received MODBUS ADU the data field of a response from a server to a client contains the data requested. If an error related to the MODBUS function requested occurs, the field contains an exception code that the server application can use to determine the next action to be taken.

For example a client can read the ON/OFF states of a group of discrete outputs or inputs or it can read/write the data contents of a group of registers.

When the server responds to the client, it uses the function code field to indicate either a normal (error-free) response or that some kind of error occurred (called an exception response). For a normal response, the server simply echoes to the request the original function code.

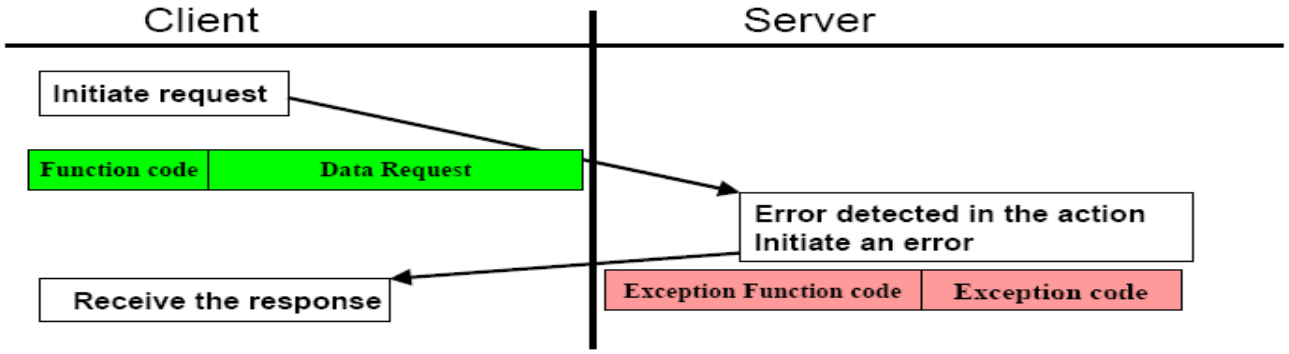
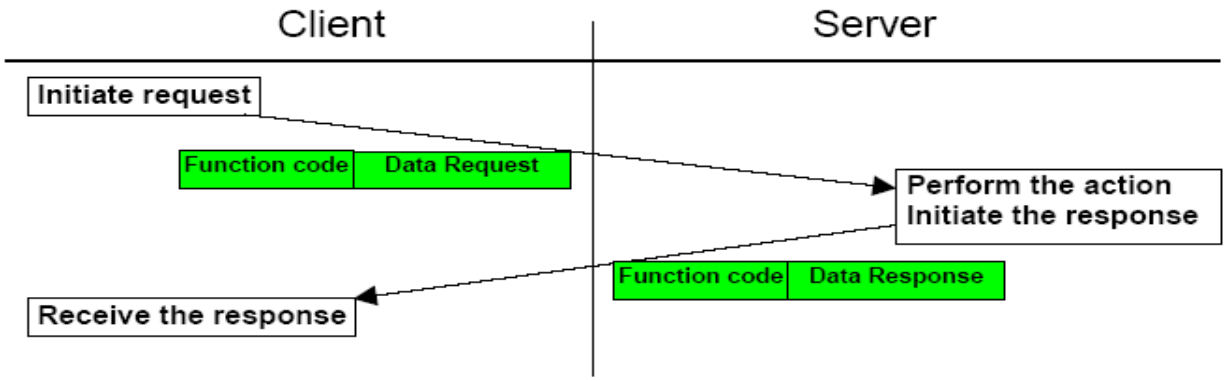



**Figure 2: MODBUS transaction (error free)**

For an exception response, the server returns a code that is equivalent to the original function code from the request PDU with its most significant bit set to logic 1.




The size of the MODBUS PDU is limited by the size constraint inherited from the first MODBUS implementation on Serial Line network (max. RS485 ADU = 256 bytes).

Therefore:

**MODBUS PDU for serial line communication =** 256 - Server address (1 byte) - CRC (2 bytes) = **253 bytes**.

The MODBUS protocol defines three PDUs. They are:

- MODBUS Request PDU, mb_req_pdu
- MODBUS Response PDU, mb_rsp_pdu
- MODBUS Exception Response PDU, mb_excep_rsp_pdu

The Modbus PDU also contains a CRC checksum, this is a standard CRC16, meaning the initial value is 0xFFFF and polynomial value is 0xA001 see [2].

### 6.2.1 Timing

| | |
|---|---|
| MessageTimeout | A request shall be answered within 100 mSec |
| InterCharTimeout | The Time between two consecutive characters shall be less than 3 character times. At @9600 this is 3.1 mSec |

## 6.3 Data Encoding

### 6.3.1 Normal data

MODBUS uses a 'big-Endian' representation for addresses and data items. This means that when a numerical quantity larger than a single byte is transmitted, the most significant byte is sent first. So for example:

| Register size | Value | |
|---|---|---|
| 16-bits | 0x1234 | the first byte sent is 0x12 then 0x34 |

### 6.3.2 CRC

Unlike normal data, CRC values are transmitted as 'little-Endian'. This means that the least significant byte is transmitted first (LSB first). See also [2].

## 6.4 MODBUS Data model

MODBUS bases its data model on a series of tables that have distinguishing characteristics. The four primary tables are:

| Primary tables | Object type | Type of | Comments |
|---|---|---|---|
| Discrete Input | Single bit | Read-Only | This type of data can be provided by an I/O system. |
| Coils | Single bit | Read-Write | This type of data can be alterable by an application program. |
| Input Registers | 16-bit word | Read-Only | This type of data can be provided by an I/O system. |
| Holding Registers | 16-bit word | Read-Write | This type of data can be alterable by an application program. |

The distinctions between inputs and outputs, and between bit-addressable and word addressable data items, do not imply any application behavior. It is perfectly acceptable, and very common, to regard all four tables as overlaying one another, if this is the most natural interpretation on the target machine in question.

For each of the primary tables, the protocol allows individual selection of 65536 data items, and the operations of read or write of those items are designed to span multiple consecutive data items up to a data size limit which is dependent on the transaction function code. It's obvious that all the data handled via MODBUS (bits, registers) must be located in device application memory. But physical address in memory should not be confused with data reference. The only requirement is to link data reference with physical address.

MODBUS logical reference numbers, which are used in MODBUS functions, are unsigned integer indices starting at zero.

# 7 WiSensys implementation

## 7.1 The physical interface on the RS-485 I/O Board

The physical interface on the RS-485 I/O board can either be configured as a 4-wire (RS-422) or 2-wire (RS-485) interface. When configured as RS-422 the user also has the option of setting the interface to full- or half-duplex mode.
By default the interface is configured to work in half-duplex mode.

There are two more properties that must be set for the interface, these are baud rate and parity. The default baud rate is 9600 baud and the parity is by default set to none. There are also a number of serial properties that can't be set, these are fixed, they are:

- Number of databits : 8
- Number of startbits : 1
- Number of stopbits : 1

To summarise, the default communication is set to:
Half-duplex, 9600 baud, 8 databits, no parity, 1 startbit and 1 stopbit

## 7.2 Sensor information

### 7.2.1 General

To read sensor information the Modbus capability to read a input registers will be implemented. We use the command "read input registers'. Because some sensors have the ability to measure multiple quantities and or the measurement value exceeds the Modbus 16-bits register width, a block of 10 Modbus registers will be reserved per sensor. When the IO board "MODBUS extended feature" is available and enabled this will be 20. The explanation and examples below assume the "MODBUS extended feature" is disabled. The following table lists the purpose of each of these registers.

| Register number | Description |
|---|---|
| 0 | LSB is the sensor type, see [3] for a description of the sensor types<br>MSB is a flag byte, see 7.2.3 |
| 1 | The first 16-bit measurement value |
| 2 | The second 16-bit measurement value (if required) |
| 3 | The third 16-bit measurement value (if required) |
| 4 | The fourth 16-bit measurement value (if required) |
| 5 | The fifth 16-bit measurement value (if required) |
| 6 | The sixth 16-bit measurement value (if required) |
| 7 | The seventh 16-bit measurement value (if required) |
| 8 | The eighth 16-bit measurement value (if required) |
| 9 | The ninth 16-bit measurement value (if required) |
| 10-18 | etc.. (Only implemented when "MODBUS extended feature" is enabled) |
| 19 | The RSSI value of the sensor<br>(Only implemented when "MODBUS extended feature" is enabled) |

**Table 1: Modbus input register mapping**

Modbus allows up to 65536 input registers to be read, but since every sensor uses 10 registers, it will only be possible to read data from sensor 1 to 6552, because the register numbering will translate directly to the sensor address device ID.
Data for sensor 1 start at register 10 and ends at register 19. Sensor 2 will be from register 20 to 29 etc.. The last sensor is 6552 which can be read from register 65520 to 65529.

It is only possible to read the registers belonging to a single sensor with one command. So when the Modbus command specifies either a number of registers greater than 10 or a start register address and number of registers that would mean reading the last registers of one sensor and the first registers of the next, the I/O board will report back an error. For example when the start address is 15 and the number of registers to read is 8 then an error is reported back. This is because register 15 to 19 belong to sensor 1 and register 20 to 22 belong to sensor 2 and it is not allowed to read registers of more than 1 sensor in a single command.

When a request is done to read either a reserved register or the 2nd measurement value for a sensor that only supports measuring a single quantity, the I/O board will report the value 0x7FFF for that register. The value 0x7FFF is chosen, because this is the maximum value for a signed 16-bit integer and most measurements will be signed 16-bit integers.

### 7.2.2 Sensor measurements

The Modbus protocol is based on 16-bit registers, so the sensor data is mapped onto those 16-bit registers. For most sensors, the 16-bit register is enough to represent its measurement values, but a number of sensors have 32-bit measurement values, these are mapped onto 2 consecutive registers, with the first register holding the least significant word and the second register the most significant word.

The table below lists all known sensor types, the quantities they can measure, the range of the measurement value and the number of decimals included in the value. For sensors that have multiple measurement quantities, the first listed quantity is in register 1, the second in register 2 etc..

| Sensor type | Device Type | Register | Measurement Quantity | Value range | No. of decimals |
|---|---|---|---|---|---|
| Temperature | 0x01 | 1 | Temperature | -32768..32767 | 1 |
| Humidity | 0x02 | 1 | Humidity | -32768..32767 | 1 |
| Humidity/Temp | 0x03 | 1 | Humidity | -32768..32767 | 1 |
| | | 2 | Temperature | -32768..32767 | 1 |
| Switch | 0x04 | 1 | Open/Close | 0 or 1 | 0 |
| Voltage | 0x06 | 1 | Voltage | -32768..32767 | 2 |
| Current | 0x07 | 1 | Current | -32768..32767 | 2 |
| Energy | 0x08 | 1 | Energy LSW | | |
| | | 2 | Energy MSW | 0.. 2147483647 | 0 |
| | | 3 | Power | 0.. 65535 | 0 |
| Low Voltage | 0x0A | 1 | Voltage | -32768..32767 | 3 |
| Energy Multi | 0x0B | 1 | Energy LSW | | |
| | | 2 | Energy MSW | 0.. 2147483647 | 0 |
| | | 3 | Power | 0.. 65535 | 0 |
| $CO_2$/Hum/Temp | 0x0C | 1 | $CO_2$ | 0..32767 | 0 |
| | | 2 | Humidity | -32768..32767 | 1 |
| | | 3 | Temperature | -32768..32767 | 1 |
| Pulse | 0x0D | 1 | Pulses LSW | | |
| | | 2 | Pulses MSW | 0.. 2147483647 | 0 |
| Pulse Energy | 0x0E | 1 | kJ Energy LSW | | |
| | | 2 | kJ Energy MSW | 0.. 2147483647 | 0 |
| Dendrometer | 0x10 | 1 | Percentage | 0..65535 | 2 |
| Mili-voltage | 0x11 | 1 | Voltage LSW | -2147483648.. 2147483647 | 3 |
| | | 2 | Voltage MSW | | |
| RMS | 0x12 | 1 | Voltage | -32768..32767 | 3 |
| Contact Change | 0x13 | 1 | Open/Close | 0 or 1 | 0 |
| | | 2 | Regular measurement[1] | 0 or 1 | 0 |
| Contact Percentage | 0x14 | 1 | Open/Close | 0 or 1 | 0 |
| | | 2 | Percentage closed | 0 .. 10000 | 2 |
| RSSI | 0x15 | 1 | RSSI value | 0 .. 255 | 0 |
| SDI-12 5 Channel | 0x16 | 1 | Channel 1 LSW | -2147483648.. 2147483647 | 3 |
| | | 2 | Channel 1 MSW | | |
| | | 3 | Channel 2 LSW | -2147483648.. 2147483647 | 3 |
| | | 4 | Channel 2 MSW | | |
| | | 5 | Channel 3 LSW | -2147483648.. 2147483647 | 3 |
| | | 6 | Channel 3 MSW | | |
| | | 7 | Channel 4 LSW | -2147483648.. 2147483647 | 3 |
| | | 8 | Channel 4 MSW | | |
| | | 9 | Channel 5 LSW[2] | -2147483648.. 2147483647 | 3 |
| | | 10 | Channel 5 MSW[2] | | |
| SDI-12 10 Channel | 0x17 | 1 | Channel 1 | -32768..32767 | 1 |
| | | 2 | Channel 2 | -32768..32767 | 1 |
| | | 3 | Channel 3 | -32768..32767 | 1 |
| | | 4 | Channel 4 | -32768..32767 | 1 |
| | | 5 | Channel 5 | -32768..32767 | 1 |
| | | 6 | Channel 6 | -32768..32767 | 1 |
| | | 7 | Channel 7 | -32768..32767 | 1 |

[1] 0 means a measurement at the normal sensor measurement, a 1 means that the sensor has detected a contact state change and send the change immediately, not at the normal measurement time.

[2] Use only when the IO board “MODBUS extended feature” is available and enabled

| | | | | | |
|---|---|---|---|---|---|
| | | 8 | Channel 8 | -32768..32767 | 1 |
| | | 9 | Channel 9 | -32768..32767 | 1 |
| | | 10 | Channel 10[2] | -32768..32767 | 1 |
| Temperature 2 decimals | 0x19 | 1 | Temperature | -32768..32767 | 2 |
| Extended voltage | 0x1A | 1 | Voltage | -32768..32767 | 2 |
| Temperature 3 Channel | 0x1C | 1 | Temperature 1 | -32768..32767 | 1 |
| | | 2 | Temperature 2 | -32768..32767 | 1 |
| | | 3 | Temperature 3 | -32768..32767 | 1 |
| Temperature 5 Channel | 0x1D | 1 | Temperature 1 | -32768..32767 | 1 |
| | | 2 | Temperature 2 | -32768..32767 | 1 |
| | | 3 | Temperature 3 | -32768..32767 | 1 |
| | | 4 | Temperature 4 | -32768..32767 | 1 |
| | | 5 | Temperature 5 | -32768..32767 | 1 |
| RS232 5 Channel 32 bit | 0x1E | 1 | Channel 1 LSW | -2147483648.. 2147483647 | 0 |
| | | 2 | Channel 1 MSW | | |
| | | 3 | Channel 2 LSW | -2147483648.. 2147483647 | 0 |
| | | 4 | Channel 2 MSW | | |
| | | 5 | Channel 3 LSW | -2147483648.. 2147483647 | 0 |
| | | 6 | Channel 3 MSW | | |
| | | 7 | Channel 4 LSW | -2147483648.. 2147483647 | 0 |
| | | 8 | Channel 4 MSW | | |
| | | 9 | Channel 5 LSW[2] | -2147483648.. 2147483647 | 0 |
| | | 10 | Channel 5 MSW[2] | | |
| Modbus 5 channel 32 bit | 0x1F | 1 | Channel 1 LSW | -2147483648.. 2147483647 | 0 |
| | | 2 | Channel 1 MSW | | |
| | | 3 | Channel 2 LSW | 2147483648.. 2147483647 | 0 |
| | | 4 | Channel 2 MSW | | |
| | | 5 | Channel 3 LSW | 2147483648.. 2147483647 | 0 |
| | | 6 | Channel 3 MSW | | |
| | | 7 | Channel 4 LSW | 2147483648.. 2147483647 | 0 |
| | | 8 | Channel 4 MSW | | |
| | | 9 | Channel 5 LSW[2] | 2147483648.. 2147483647 | 0 |
| | | 10 | Channel 5 MSW[2] | | |
| Modbus 10 channel 16 bit | 0x20 | 1 | Channel 1 | -32768..32767 | 0 |
| | | 2 | Channel 2 | -32768..32767 | 0 |
| | | 3 | Channel 3 | -32768..32767 | 0 |
| | | 4 | Channel 4 | -32768..32767 | 0 |
| | | 5 | Channel 5 | -32768..32767 | 0 |
| | | 6 | Channel 6 | -32768..32767 | 0 |
| | | 7 | Channel 7 | -32768..32767 | 0 |
| | | 8 | Channel 8 | -32768..32767 | 0 |
| | | 9 | Channel 9 | -32768..32767 | 0 |
| | | 10 | Channel 10[2] | -32768..32767 | 0 |
| RS232 1 Channel 32 bit | 0x21 | 1 | Channel 1 LSW | -2147483648.. 2147483647 | 0 |
| | | 2 | Channel 1 MSW | | |
| RS232 2 Channel 32 bit | 0x22 | 1 | Channel 1 LSW | -2147483648.. 2147483647 | 0 |
| | | 2 | Channel 1 MSW | | |
| | | 3 | Channel 2 LSW | -2147483648.. 2147483647 | 0 |
| | | 4 | Channel 2 MSW | | |
| RS232 3 Channel 32 bit | 0x23 | 1 | Channel 1 LSW | -2147483648.. 2147483647 | 0 |
| | | 2 | Channel 1 MSW | | |
| | | 3 | Channel 2 LSW | -2147483648.. 2147483647 | 0 |
| | | 4 | Channel 2 MSW | | |
| | | 5 | Channel 3 LSW | -2147483648.. 2147483647 | 0 |
| | | 6 | Channel 3 MSW | | |

| | | | | | |
|---|---|---|---|---|---|
| RS232 4 Channel 32 bit | 0x24 | 1 | Channel 1 LSW | -2147483648.. 2147483647 | 0 |
| | | 2 | Channel 1 MSW | | |
| | | 3 | Channel 2 LSW | -2147483648.. 2147483647 | 0 |
| | | 4 | Channel 2 MSW | | |
| | | 5 | Channel 3 LSW | -2147483648.. 2147483647 | 0 |
| | | 6 | Channel 3 MSW | | |
| | | 7 | Channel 4 LSW | -2147483648.. 2147483647 | 0 |
| | | 8 | Channel 4 MSW | | |
| RS232 1 Channel 16 bit | 0x25 | 1 | Channel 1 | -32768..32767 | 0 |
| RS232 2 Channel 16 bit | 0x26 | 1 | Channel 1 | -32768..32767 | 0 |
| | | 2 | Channel 2 | -32768..32767 | 0 |
| RS232 3 Channel 16 bit | 0x27 | 1 | Channel 1 | -32768..32767 | 0 |
| | | 2 | Channel 2 | -32768..32767 | 0 |
| | | 3 | Channel 3 | -32768..32767 | 0 |
| RS232 4 Channel 16 bit | 0x28 | 1 | Channel 1 | -32768..32767 | 0 |
| | | 2 | Channel 2 | -32768..32767 | 0 |
| | | 3 | Channel 3 | -32768..32767 | 0 |
| | | 4 | Channel 4 | -32768..32767 | 0 |
| RS232 5 Channel 16 bit | 0x29 | 1 | Channel 1 | -32768..32767 | 0 |
| | | 2 | Channel 2 | -32768..32767 | 0 |
| | | 3 | Channel 3 | -32768..32767 | 0 |
| | | 4 | Channel 4 | -32768..32767 | 0 |
| | | 5 | Channel 5 | -32768..32767 | 0 |
| RS232 6 Channel 16 bit | 0x2A | 1 | Channel 1 | -32768..32767 | 0 |
| | | 2 | Channel 2 | -32768..32767 | 0 |
| | | 3 | Channel 3 | -32768..32767 | 0 |
| | | 4 | Channel 4 | -32768..32767 | 0 |
| | | 5 | Channel 5 | -32768..32767 | 0 |
| | | 6 | Channel 6 | -32768..32767 | 0 |
| RS232 7 Channel 16 bit | 0x2B | 1 | Channel 1 | -32768..32767 | 0 |
| | | 2 | Channel 2 | -32768..32767 | 0 |
| | | 3 | Channel 3 | -32768..32767 | 0 |
| | | 4 | Channel 4 | -32768..32767 | 0 |
| | | 5 | Channel 5 | -32768..32767 | 0 |
| | | 6 | Channel 6 | -32768..32767 | 0 |
| | | 7 | Channel 7 | -32768..32767 | 0 |
| RS232 8 Channel 16 bit | 0x2C | 1 | Channel 1 | -32768..32767 | 0 |
| | | 2 | Channel 2 | -32768..32767 | 0 |
| | | 3 | Channel 3 | -32768..32767 | 0 |
| | | 4 | Channel 4 | -32768..32767 | 0 |
| | | 5 | Channel 5 | -32768..32767 | 0 |
| | | 6 | Channel 6 | -32768..32767 | 0 |
| | | 7 | Channel 7 | -32768..32767 | 0 |
| | | 8 | Channel 8 | -32768..32767 | 0 |
| RS232 9 Channel 16 bit | 0x2D | 1 | Channel 1 | -32768..32767 | 0 |
| | | 2 | Channel 2 | -32768..32767 | 0 |
| | | 3 | Channel 3 | -32768..32767 | 0 |
| | | 4 | Channel 4 | -32768..32767 | 0 |
| | | 5 | Channel 5 | -32768..32767 | 0 |
| | | 6 | Channel 6 | -32768..32767 | 0 |
| | | 7 | Channel 7 | -32768..32767 | 0 |
| | | 8 | Channel 8 | -32768..32767 | 0 |
| | | 9 | Channel 9 | -32768..32767 | 0 |
| RS232 10 Channel 16 bit | 0x2E | 1 | Channel 1 | -32768..32767 | 0 |
| | | 2 | Channel 2 | -32768..32767 | 0 |

| | | | | | |
|---|---|---|---|---|---|
| | | 3 | Channel 3 | -32768..32767 | 0 |
| | | 4 | Channel 4 | -32768..32767 | 0 |
| | | 5 | Channel 5 | -32768..32767 | 0 |
| | | 6 | Channel 6 | -32768..32767 | 0 |
| | | 7 | Channel 7 | -32768..32767 | 0 |
| | | 8 | Channel 8 | -32768..32767 | 0 |
| | | 9 | Channel 9 | -32768..32767 | 0 |
| | | 10 | Channel 10[2] | -32768..32767 | 0 |
| Modbus 5 channel float | 0x2F | 1 | Channel 1 | 32-bit float IEEE 754 | 0 |
| | | 2 | Channel 2 | | |
| | | 3 | Channel 3 | 32-bit float IEEE 754 | 0 |
| | | 4 | Channel 4 | | |
| | | 5 | Channel 5 | 32-bit float IEEE 754 | 0 |
| | | 6 | Channel 6 | | |
| | | 7 | Channel 7 | 32-bit float IEEE 754 | 0 |
| | | 8 | Channel 8 | | |
| | | 9 | Channel 9[2] | 32-bit float IEEE 754 | |
| | | 10 | Channel 10[2] | | |
| Temperature 2 Channel | 0x30 | 1 | Temperature 1 | -32768..32767 | 1 |
| | | 2 | Temperature 2 | -32768..32767 | 1 |
| Temperature 4 Channel | 0x31 | 1 | Temperature 1 | -32768..32767 | 1 |
| | | 2 | Temperature 2 | -32768..32767 | 1 |
| | | 3 | Temperature 3 | -32768..32767 | 1 |
| | | 4 | Temperature 4 | -32768..32767 | 1 |
| Temperature 6 Channel | 0x32 | 1 | Temperature 1 | -32768..32767 | 1 |
| | | 2 | Temperature 2 | -32768..32767 | 1 |
| | | 3 | Temperature 3 | -32768..32767 | 1 |
| | | 4 | Temperature 4 | -32768..32767 | 1 |
| | | 5 | Temperature 5 | -32768..32767 | 1 |
| | | 6 | Temperature 6 | -32768..32767 | 1 |
| Load Cell | 0x33 | 1 | Weight | -32768..32767 | 3 |
| Tap | 0x34 | 1 | Close/Open (Tap) | 0 or 1 | 0 |
| | | 2 | Open/Close (Contact) | 0 or 1 | 0 |
| | | 3 | Voltage | -32768..32767 | 2 |
| Pressure | 0x35 | 1 | Pressure | -32768..32767 | 1 |
| | | 2 | Temperature | -32768..32767 | 1 |
| All-In-One | 0x36 | 1 | Temperature | -32768..32767 | 1 |
| | | 2 | Humidity | -32768..32767 | 1 |
| | | 3 | $CO_2$ | 0..32767 | 0 |
| | | 4 | Lux LSW | -2147483648.. 2147483647 | 2 |
| | | 5 | Lux MSW | | |
| | | 6 | Movement | 0 or 1 | 0 |

**Table 2: Sensor measurements**

**Examples:**

Humidity/Temperature sensor with 16-bit measurements
Let's assume we have a combined humidity/temperature sensor with device ID 23 and the humidity is 45.4% and the temperature is 29.7 °C. Note: the flag byte is set to 0 for this example.

This means that the Modbus request must start reading from input register 230, read 10 registers and the reply will indicate the registers having the following values :

| Register | 230 | 231 | 232 | 233 | 234 | 235 | 236 | 237 | 238 | 239 |
|---|---|---|---|---|---|---|---|---|---|---|
| Value Hex | 0x0003 | 0x01C6 | 0x0129 | 0x7FFF | 0x7FFF | 0x7FFF | 0x7FFF | 0x7FFF | 0x7FFF | 0x7FFF |
| Dec | 3 | 454 | 297 | 32767 | 32767 | 32767 | 32767 | 32767 | 32767 | 32767 |

**Table 3: Humidity/Temperature sensor example**

Pulse sensor with a 32-bit measurement value
Let's assume we have a pulse sensor with device ID 12 and a pulse count of 19088743 (which is 0x01234567 hexadecimal). Note: the flag byte is set to 0 for this example.

This means that the Modbus request must start reading from input register 120, read 10 registers. The reply will indicate the registers having the following values:

| Register | 120 | 121 | 122 | 123 | 124 | 125 | 126 | 127 | 128 | 129 |
|---|---|---|---|---|---|---|---|---|---|---|
| Value Hex | 0x000D | 0x4567 | 0x0123 | 0x7FFF | 0x7FFF | 0x7FFF | 0x7FFF | 0x7FFF | 0x7FFF | 0x7FFF |
| Dec | 13 | 17767 | 291 | 32767 | 32767 | 32767 | 32767 | 32767 | 32767 | 32767 |

**Table 4: Pulse sensor example**

Register 121 contains the LSW part of the pulse count and 122 contains the MSW part of the pulse count. The device receiving the Modbus reply must combine these 2 register values to get the original pulse count value.

### 7.2.3 Flag byte

The flag byte gives information about the sensor, each bit has a specific meaning, which is described in the table below.

| Bit | Name | Description |
|---|---|---|
| 0 | In reach | Indicates if the sensor is in reach. 1 = in reach, 0 = out of reach |
| 1 | Reserved | Reserved |
| 2 | Alarm | Sensor alarm flag. 1 = sensor signals an alarm, 0 = no alarm |
| 3 | | |
| 4 | | |
| 5 | | |
| 6 | | |
| 7 | | |

**Table 5: Flag byte**

In reach flag

This flag is used to indicate if a sensor is still in reach of the basestation responsible for servicing it. When the flag is set, the sensor is still transmitting data to the basestation. When the flag is 0, the base has not received any measurement messages from the sensor for a period of time. The time-out period depends on the sensors measurement- and transmit interval as well as a programmable number of measurement messages that can be missed. The actual time-out period can be calculated using this formula :

Time-out = MI * TI * MM

MI measurement interval in seconds
Tit Transmit interval, this is the number of measurements that must be done before the data is send
MM the number of messages that can be missed before a sensor is considered out or range

Reserved

This flag is reserved.

Alarm flag

The alarm flag indicates that the current measurement (or one of the measurements for a sensor that measures multiple quantities at the same time) has caused an alarm. This can be either because the measured value is below a specified minimum trip value, or above a maximum trip value.

## 7.3 Sensor command write and read

### 7.3.1 General

To send sensor commands the Modbus capability to write a holding registers will be implemented. We use the command "write multiple registers" for writing holding registers. To read the written value back the command "read holding registers" will be implemented. To have the possibility to add more commands in the future, a block of 10 Modbus holding registers is reserved per sensor. When IO board "MODBUS extended feature" is available and enabled this will be 20. The explanation and examples below assume the "MODBUS extended feature" is disabled.

The holding registers are independent of the input registers which are read with command "read input registers". The following table lists the purpose of each of these holding registers.

| Holding Register number | Description |
|---|---|
| 0 | Reserved |
| 1 | Reserved |
| 2 | Command 1, the 16-bit ID to send, (Valid range is 8191 - 12287) |
| 3 | Command 1, the 16-bit data to send (Valid range is -32768 – 32767) |
| 4 | Command 2, the 16-bit ID to send, (Valid range is 8191 - 12287) |
| 5 | Command 2, the 16-bit data to send (Valid range is -32768 – 32767) |
| 6 | Command 3, the 16-bit ID to send, (Valid range is 8191 - 12287) |
| 7 | Command 3, the 16-bit data to send (Valid range is -32768 – 32767) |
| 8 | Command 4, the 16-bit ID to send, (Valid range is 8191 - 12287) |
| 9 | Command 4, the 16-bit data to send (Valid range is -32768 – 327670) |
| 10-19 | Reserved (Only implemented when "MODBUS extended feature" is enabled) |

**Table 6: Modbus holding register mapping**

Modbus allows up to 65536 holding registers to be written or read, but since every sensor uses 10 holding registers, it will only be possible to write and read data for sensor 1 to 6552, because the holding register numbering will translate directly to the sensor address device ID.
Data for sensor 1 starts at holding register 10 and ends at holding register 19. Sensor 2 will be from holding register 20 to 29 etc.. The last sensor is 6552 which uses holding register 65520 to 65529.

It is only possible to write or read the holding registers belonging to a single sensor with one command. So when the Modbus command specifies either a number of holding registers greater than 10 or a start holding register address and number of holding registers that would mean reading the last holding register of one sensor and the first holding registers of the next, the I/O board will report back an error. For example when the start address is 15 and the number of registers to read is 8 then an error is reported back. This is because holding register 15 to 19 belong to sensor 1 and holding register 20 to 22 belong to sensor 2 and it is not allowed to write or read registers of more than 1 sensor in a single command.

When a request is done to read a reserved register, the I/O board will report the value 0x7FFF for that register. When a request is done to write a reserved register the normal response is send back but the value is not stored.

When the Modbus command couldn't be send to the base board the register with the ID will be cleared. This could be one of the following reasons:

- Auto paired sensor not yet synchronized (Can be checked in Sensorgraph)
- ID value not supported by sensor

### 7.3.2 Sensor commands

The Modbus protocol is based on 16-bit registers, so the sensor data is mapped onto those 16-bit registers.

The table below lists all known sensor types that support the capability to write a sensor commands with Modbus protocol. The table also lists the register number, register quantity, the value or range and the number of decimals included in the value.

| Sensor type | Device Type | Register | Register Quantity | Value or value range | No. of decimals |
|---|---|---|---|---|---|
| **T.B.D** (Wireless Motor Control) | **T.B.D** | 2 | ID for output active time A | 8192 | 0 |
| | | 3 | Data for Output active time A | -32768..32767 (seconds) | 0 |
| Tap (Wireless Valve Control) | 0x34 | 2 | ID for output active time A | 8192 | 0 |
| | | 3 | Data for Output active time A | -32768..32767 (seconds) | 0 |
| | | 4 | ID for sample interval overrule (4s **T.B.D**) | 8196 | 0 |
| | | 5 | Data for sample interval overrule (4s **T.B.D**) | 0 = not active 1 = active | 0 |

**Table 7: Sensor commands**

**Notes:**

The registers must be written in pairs or multiple pairs, first ID and second Data register.
The register pair number where the ID and Data is written may be chosen freely once after power up as long as the ID & Data are written in pairs in register 2&3, 4&5, 6&7 or 8&9. Once a ID is written on a specific register you may not use this register to write another ID.
The same ID may not be written in two different ID registers

**Examples:**

Sensor with Output active time A
Let's assume we have a sensor with device ID 1 and we want to write the value 0x0200 for "Output active time A".

This means that the Modbus request must start writing at holding register 12 and write 2 registers. For writing 2 registers the write multiple registers command 0x10 will have the following values:

| Holding Register | 12 | 13 |
|---|---|---|
| Value Hex | 0x2000 | 0x0200 |
| Dec | 8192 | 512 |

**Table 8: Sensor 1 example**

Sensor with sample interval overrule to 1 second
Let's assume we have a sensor with device ID 12 and we want to write the value 0x0001 for "sample interval overrule to 1 second".

This means that the Modbus request must start writing at holding register 126 and write 2 registers. For writing 2 registers the write multiple registers command 0x10 will have the following values:

| Holding Register | 126 | 127 |
|---|---|---|
| Value Hex | 0x2004 | 0x0001 |
| Dec | 8196 | 1 |

**Table 9: Sensor 12 example**

Writing multiple sensor commands
Let's assume we have a sensor with device ID 1 and we want to write the value 0x0003 for output active time A, 0x0002 for output active time B and 0x0001 for sample interval overrule to 1 second

This means that the Modbus request must start writing at holding register 126 and write 6 registers. For writing 6 registers the write multiple registers command 0x10 will have the following values:

| Holding Register | 12 | 13 | 14 | 15 | 16 | 17 |
|---|---|---|---|---|---|---|
| Value Hex | 0x2000 | 0x0003 | 0x2001 | 0x0002 | 0x2004 | 0x0001 |
| Dec | 8192 | 3 | 8193 | 2 | 8196 | 1 |

**Table 10: Sensor 1 example**

### 7.4 Supported commands/responses/errors

#### 7.4.1 Read Input Registers

The WiSensys Modbus I/O Board supports the Modbus 'Read Input Registers' command, which look as specified in the table below

| | | |
|---|---|---|
| Slave address | 1 Byte | 1 to 247 |
| Function code | 1 Byte | 0x04 |
| Start Address | 2 Bytes | 0x0010 to 0xFFF9 |
| Number of Input Registers | 2 Bytes | 0x0001 to 0x000A |
| CRC | 2 Bytes | 0xXXXX |

**Table 11: Read Input Registers command**

The reply to this command can either be a normal response message or an exception response in case there is a problem with the command. First a normal response, see the table below for the message layout.

| | | |
|---|---|---|
| Slave address | 1 Byte | 1 to 247 |
| Function code | 1 Byte | 0x04 |
| Byte count | 1 Byte | 2 x N |
| Input Registers | N x 2 Bytes | |
| CRC | 2 Bytes | 0xXXXX |

**Table 12: Normal Read Input Registers response**

In the reply message N represents the number of registers as specified in the original command.

#### 7.4.2 Write multiple registers

The WiSensys Modbus I/O Board supports the Modbus 'Write multiple registers' command, which look as specified in the table below

| | | |
|---|---|---|
| Slave address | 1 Byte | 1 to 247 |
| Function code | 1 Byte | 0x10 |
| Start Address | 2 Bytes | 0x0010 to 0xFFF9 |
| Number of Registers | 2 Bytes | 0x0001 to 0x000A |
| Byte count | 1 Bytes | 2 x N |
| Registers | N x 2 Bytes | |
| CRC | 2 Bytes | 0xXXXX |

**Table 13: Write Multiple Registers command**

The reply to this command can either be a normal response message or an exception response in case there is a problem with the command. First a normal response, see the table below for the message layout.

| | | |
|---|---|---|
| Slave address | 1 Byte | 1 to 247 |
| Function code | 1 Byte | 0x10 |
| Start Address | 2 Bytes | 0x0010 to 0xFFF9 |
| Number of Registers | 2 Bytes | 0x0001 to 0x000A |
| CRC | 2 Bytes | 0xXXXX |

**Table 14: Normal Write Multiple Registers response**

In the reply message N represents the number of registers as specified in the original command.

### 7.4.3 Read holding registers

The WiSensys Modbus I/O Board supports the Modbus 'Read holding registers' command, which look as specified in the table below

| | | |
|---|---|---|
| Slave address | 1 Byte | 1 to 247 |
| Function code | 1 Byte | 0x03 |
| Start Address | 2 Bytes | 0x0010 to 0xFFF9 |
| Number of Registers | 2 Bytes | 0x0001 to 0x000A |
| CRC | 2 Bytes | 0xXXXX |

**Table 15: Read Holding Registers command**

The reply to this command can either be a normal response message or an exception response in case there is a problem with the command. First a normal response, see the table below for the message layout.

| | | |
|---|---|---|
| Slave address | 1 Byte | 1 to 247 |
| Function code | 1 Byte | 0x03 |
| Byte count | 1 Byte | 2 x N |
| Registers | N x 2 Bytes | |
| CRC | 2 Bytes | 0xXXXX |

**Table 16: Normal Holding Registers response**

In the reply message N represents the number of registers as specified in the original command.

### 7.4.4 Exception response

When the Modbus I/O board detects a problem with a received command, it can reply with an exception response, which is shown in the next table.

| | | |
|---|---|---|
| Slave address | 1 Byte | 1 to 247 |
| Function code | 1 Byte | Command function code + 0x80 (bit 7) |
| Exception code | 1 Byte | See table below |
| CRC | 2 Bytes | 0xXXXX |

**Table 17: Exception response**

The function code is the same as from the original command, but bit 7 is set to indicate an exception. The exception code indicates to the receiver what the problem with the original command was. The exception codes are listed below.

| Exception code | Description |
|---|---|
| 0x01 | Illegal function, the received command is not supported |
| 0x02 | Illegal data address, the start address and/or number of registers is invalid. The problems could be :<br>• the start address is invalid<br>• the length is invalid, not between 0x01 and 0x0A<br>• the combination of start address and number of registers means returning data for more than 1 sensor |
| 0x03 | Illegal data value, something in the data is wrong. The problems could be :<br>• Sensor not found, or not supported<br>• Request would read registers for more than one sensor<br>• The Number of Registers is wrong |
| 0x04 | Slave device failure, this indicates an internal error in the I/O Board |

**Table 18: Supported Modbus exception codes**

# 8 Serial Communication examples

### 8.1.1 Read Input Registers

A typical MODBUS RTU 'Read Input Registers' command (request) may look like this:
0x01 0x04 0x00 0x0A 0x00 0x02 0x51 0xC9

The actual format of the data depends on the type of command desired. The example above is the MODBUS 'Read Input Registers' function. The command above is a request to read 2 input registers starting from address 10.

0x01 The address of the slave device (WiSensys I/O Board) that should handle the command. Each slave device has its own unique address.

0x04 Byte code for the MODBUS 'Read Input Registers' function.

0x00 0x0A The starting address of the registers to be read, 0x000A means the 1st register of sensor number 1.

0x00 0x02 The number of registers to be read. In this case '0 02' indicates two register are to be read.

0x51 0xC9 The final two characters of the command string make up the CRC, used to check for errors in the message.

A typical response to this example command could be:
0x01 0x04 0x04 0x00 0x01 0x00 0xD9 0x6B 0xDE

0x01 The address of the slave device (WiSensys I/O Board) that sends the response

0x04 This is the function to which this message is a reply

0x04 The length byte indicating the number of bytes to follow, this is 2 x the number of register to read as specified in the request

0x00 0x01 The value read from the 1st register. 00 01 means that this is a temperature sensor, because this is the 1st register of a sensor block.

0x00 0xD9 The value read from the 2nd register. Register data is read back as 16 bits. 00 D9 means the temperature is 21.7 °C.

0x6B 0xDE The CRC for the response message.

### 8.1.2 Write multiple registers

A typical MODBUS RTU 'Write multiple registers' command (request) may look like this:
0x01 0x10 0x00 0x0C 0x00 0x02 0x04 0x20 0x00 0x04 0xD2 0x7A 0xA7

The actual format of the data depends on the type of command desired. The command above is a request to write 2 holding starting from address 12.

0x01 The address of the slave device (WiSensys I/O Board) that should handle the command. Each slave device has its own unique address.

0x10 Byte code for the MODBUS 'Write multiple register' function.

0x00 0x0C The starting address of the holding registers to be written, 0x000C means the 3th holding register of sensor number 1.

0x00 0x02 The number of holding registers to be written. In this case '00 02' indicates two holding registers are to be written.

0x04 Byte count, the number of bytes to be written.

0x20 0x00 The value to write to the 3th holding register of sensor one. 0x20 0x00 means that value 8192 is written in this holding register.

0x04 0xD2 The value to write to the 4th holding register of sensor one. 0x04 0xD2 means that value 1234 is written in this holding register.

0x7A 0xA7 The final two characters of the command string make up the CRC, used to check for errors in the message.

A typical response to this example command could be:
0x01 0x10 0x00 0x0C 0x00 0x02 0x81 0xCB

0x01 The address of the slave device (WiSensys I/O Board) that sends the response

0x10 This is the function to which this message is a reply

0x00 0x0C The starting address of the holding registers written, 0x000C means the 3th holding register of sensor number 1.

0x00 0x02 The number of holding registers written. In this case '00 02' indicates that two holding register are written.

0x81 0xCB The CRC for the response message.

### 8.1.3 Read holding Registers

A typical MODBUS RTU 'Read holding registers' command (request) may look like this:
0x01 0x03 0x00 0x0C 0x00 0x02 0x04 0x08

The actual format of the data depends on the type of command desired. The example above is the MODBUS 'Read holding registers' function. The command above is a request to read 2 holding registers starting from address 12.

0x01 The address of the slave device (WiSensys I/O Board) that should handle the command. Each slave device has its own unique address.

0x03 Byte code for the MODBUS 'Read holding registers' function.

0x00 0x0C The starting address of the registers to be read, 0x000C means the 3th register of sensor number 1.

0x00 0x02 The number of registers to be read. In this case '0 02' indicates two register are to be read.

0x04 0x08 The final two characters of the command string make up the CRC, used to check for errors in the message.

A typical response to this example command could be:
0x01 0x03 0x04 0x20 0x00 0x04 0xD2 0x68 0x6E

0x01 The address of the slave device (WiSensys I/O Board) that sends the response

0x03 This is the function to which this message is a reply

0x04 The length byte indicating the number of bytes to follow, this is 2 x the number of register to read as specified in the request

0x20 0x00 The value read from the 3nd register. 20 00 means the last written ID is 8192

0x04 0xD2 The value read from the 4th register. Register data is read back as 16 bits. 04 D2 means the last written Data (for ID 8192) is 1234

0x68 0x6E The CRC for the response message.

## 9 TCP/IP Communication examples

### 9.1.1 Read Input Registers

A typical Modbus TCP frame 'Read Input Registers' command (request) may look like this:
0x00 0x00 0x00 0x00 0x00 0x06 0x01 0x04 0x00 0x0A 0x00 0x02

The actual format of the data depends on the type of command desired. The example above is the MODBUS 'Read Input Registers' function. The command above is a request to read 2 input registers starting from address 10.

0x00 0x00 The Transaction identifier for synchronization between messages of server & client

0x00 0x00 The Protocol identifier, Zero for Modbus/TCP

0x00 0x06 The Length field, Number of remaining bytes in this frame

0x01 The address of the slave device (WiSensys I/O Board) that should handle the command. Each slave device has its own unique address.

0x04 Byte code for the MODBUS 'Read Input Registers' function.

0x00 0x0A The starting address of the registers to be read, 0x000A means the 1st register of sensor number 1.

0x00 0x02 The number of registers to be read. In this case '00 02' indicates two register are to be read.

A typical response to this example command could be:
0x00 0x00 0x00 0x00 0x00 0x07 0x01 0x04 0x04 0x00 0x01 0x00 0xD9

0x00 0x00 The Transaction identifier for synchronization between messages of server & client

0x00 0x00 The Protocol identifier, Zero for Modbus/TCP

0x00 0x07 The Length field, Number of remaining bytes in this frame

0x01 The address of the slave device (WiSensys I/O Board) that sends the response

0x04 This is the function to which this message is a reply

0x04 The length byte indicating the number of bytes to follow, this is 2 x the number of register to read as specified in the request

0x00 0x01 The value read from the 1st register. 00 01 means that this is a temperature sensor, because this is the 1st register of a sensor block.

0x00 0xD9 The value read from the 2nd register. Register data is read back as 16 bits. 00 D9 means the temperature is 21.7 °C.

### 9.1.2 Write multiple registers

A typical Modbus TCP frame 'Write multiple registers' command (request) may look like this:
0x00 0x00 0x00 0x00 0x00 0x09 0x01 0x10 0x00 0x0C 0x00 0x02 0x04 0x20 0x00 0x00 0x0A

The command above is a request to write 2 holding register starting from address 12.

0x00 0x00 The Transaction identifier for synchronization between messages of server & client

0x00 0x00 The Protocol identifier, Zero for Modbus/TCP

0x00 0x09 The Length field, Number of remaining bytes in this frame

0x01 The address of the slave device (WiSensys I/O Board) that should handle the command. Each slave device has its own unique address.

0x10 Byte code for the MODBUS 'Write multiple register' function.

0x00 0x0C The starting address of the holding registers to be written, 0x000C means the 3st holding register of sensor number 1.

0x00 0x02 The number of holding registers to be written. In this case '00 02' indicates two holding register is to be written.

0x04 Byte count, the number of bytes to be written.

0x20 0x00 The value to write to the 3th holding register of sensor one. 0x20 0x00 means that value 8192 is written in this holding register.

0x00 0x0A The value to write to the 4th holding register of sensor one. 0x00 0x0A means that value 10 is written in this holding register.

A typical response to this example command could be:
0x00 0x00 0x00 0x00 0x00 0x06 0x01 0x10 0x00 0x0C 0x00 0x02

0x00 0x00 The Transaction identifier for synchronization between messages of server & client

0x00 0x00 The Protocol identifier, Zero for Modbus/TCP

0x00 0x06 The Length field, Number of remaining bytes in this frame

0x01 The address of the slave device (WiSensys I/O Board) that sends the response

0x10 This is the function to which this message is a reply

0x00 0x0C The starting address of the holding registers written, 0x000C means the 3st holding register of sensor number 1.

0x00 0x02 The number of holding registers written. In this case '00 02' indicates that two holding register are written.

### 9.1.3 Read holding Registers

A typical Modbus TCP frame 'Read holding registers' command (request) may look like this:
0x00 0x00 0x00 0x00 0x00 0x06 0x01 0x03 0x00 0x0C 0x00 0x02

The actual format of the data depends on the type of command desired. The example above is the MODBUS 'Read holding Registers' function. The command above is a request to read 2 input registers starting from address 12.

0x00 0x00 The Transaction identifier for synchronization between messages of server & client

0x00 0x00 The Protocol identifier, Zero for Modbus/TCP

0x00 0x06 The Length field, Number of remaining bytes in this frame

0x01 The address of the slave device (WiSensys I/O Board) that should handle the command. Each slave device has its own unique address.

0x03 Byte code for the MODBUS 'Read holding registers' function.

0x00 0x0C The starting address of the registers to be read, 0x000C means the 3th register of sensor number 1.

0x00 0x02 The number of registers to be read. In this case '00 02' indicates two register are to be read.

A typical response to this example command could be:
0x00 0x00 0x00 0x00 0x00 0x07 0x01 0x03 0x04 0x10 0x00 0x00 0x10

0x00 0x00 The Transaction identifier for synchronization between messages of server & client

0x00 0x00 The Protocol identifier, Zero for Modbus/TCP

| | |
|---|---|
| 0x00 0x07 | The Length field, Number of remaining bytes in this frame |
| 0x01 | The address of the slave device (WiSensys I/O Board) that sends the response |
| 0x03 | This is the function to which this message is a reply |
| 0x04 | The length byte indicating the number of bytes to follow, this is 2 x the number of register to read as specified in the request |
| 0x20 0x00 | The value read from the 3th register. 20 00 means the last written ID is 8192 |
| 0x00 0x10 | The value read from the 4th register. Register data is read back as 16 bits. 00 10 means the last written data (for ID 8192) is 10 |