MODBUS Client

MODBUS Client communication protocol

Supported device types and versions
Communication line configuration
Line protocol parameters
Station configuration
I/O tag configuration
Note to FloBoss 103 device
Note to Honeywell
Literature
Changes and modifications
Document revisions

Supported device types and versions

The protocol executes client (master) communication with arbitrary devices which supports a standard MODBUS RTU and ASCII in the versions of serial communication as well as MODBUS over TCP/IP. Moreover, it supports two extensions:

- **Byte mode** - allows working with devices that get back the values of registers as 1 byte (in contrast with Modbus standard in which the register value is 2 bytes).
- **Variable mode -** allows working with devices that get back values of registers with different sizes than standard 2 bytes. It was implemented because of support the flowmeter FloBoss 103 made by Fisher Controls International (at this time a part of Emerson Process Management): 1-byte variables, 4-byte unsigned/signed integers, text strings of length 10,12,20,40 characters, 6-byte time stamp, and other.

Communication line configuration

- Line category Serial (serial communication)
- Line category SerialOverUDP Device Redundant (serial communication).
- Line category RFC2217 Client (serial communication).
- Line category TCP/IP-TCP and TCP/IP-TCP Redundant (MODBUS over TCP/IP). Reserved TCP port 502 is used in common, but it is possible to use any other one according to the setting of the device. The line number is not used, set the value e.g. to 1.
  Note: For redundant systems, it is possible to enter multiple names addresses separated by commas.
  Note: In the case of WAGO 750-8100 type PLC and communication via MODBUS TCP, it was necessary to set a small polling period (e.g. 1 second) in the time parameters of the station. In the case of a longer period (5 seconds), the connection was closed quite often by the PLC.

Line protocol parameters

A dialog window of communication line configuration - **Protocol parameters** tab.
They influence some optional protocol parameters.

The line protocol contains the following parameters:

Parameter	Meaning	Unit	Default value
Immediate Disconnect	The parameter is implemented only for TCP/IP-TCP and TCP/IP-TCP Redundant line categories. The parameter activates the disconnection of the TCP connection after the execution of each read cycle, or after the value is written. The parameter was implemented due to problems with connection stability on mobile GPRS networks.	YES /NO	NO
Tcp No Delay	Setting *Tcp No Delay* parameter to YES causes low-level socket option TCP_NODELAY being set, thus turning off the default packet coalesce feature. The parameter is implemented only for TCP/IP-TCP and TCP/IP-TCP Redundant line categories.	YES /NO	NO

Station configuration

- Communication protocol "**Modbus Client**".
- The station address is a decimal number mostly in the range of 1 up to 247. Address 0 is reserved as broadcast.

Station protocol parameters

Configuration dialog box - tab **Parameter**.
They influence some optional parameters of the protocol. The following station protocol parameters can be set:

Table 1

Parameter	Meaning	Unit	Default value
Retry Count	Maximum count of request retries. If no reply returns after a request had been sent, the station will be in the status of a communication error.	s	2
Retry Timeout	Timeout before resending a request if no reply had not received.	s	0.1
Wait First Timeout	Delay after sending the request before reading the response.	s	0.1
Wait Timeout	Timeout between reading the reply.	s	0.1
Max. Wait Retry	The maximum number of retries of the reply reading.	-	20
Start Silent Interval	"Start silent interval" before the beginning of the transmission in RTU mode.	ms	50
Stop Silent Interval	"Stop silent interval" after ending of the transmission in RTU mode.	ms	50
Little Endian Mode	Byte order in Little-endian mode for 4-byte variables. The individual options indicate in which bytes (1-lowest, 4-highest) the individual bytes from the communication will go: • **2143** - first the lower word is received, then the higher word (higher byte within the word is always first) • **3412** - first the higher word is received, then the lower word (lower byte within the word is always first) • **1234** - bytes are received from lowest to highest (direct opposite of big-endian)	-	2143
Byte mode	Special byte mode of transmission in which the values of registers have a length of 1 byte and not 2 bytes as it is defined in Modbus protocol specification.	YES /NO	NO
Variable mode	Special variable mode of transmission in which the values of registers have variable lengths. The setting of *Variable mode*: Little endian = the lowest bytes are sent first Big endian = the highest bytes are sent first OFF = variable mode is switched off **Note 1:** Variable and byte mode are incompatible and only one of them can be enabled. **Note 2:** Emerson FloBoss 103 device: text strings and time stamps of 6-byte are sent always from the lowest byte. **Note 3:** Variable mode is implemented only for Protocol Mode=*RTU*. **Note 4**: A data encoding *big-endian* is used automatically, according to the default parameter values *Byte mod=NO* and *Variable mode=OFF* (i.e. according to MODBUS protocol specification).	OFF Little endian Big endian	OFF
Full debug	Logging of detailed debug information about communication in the line log.	YES /NO	NO
Protocol mode	Protocol mode: *RTU* or *ASCII*. **Note:** In the case of "MODBUS over TCP/IP", the parameter value is ignored and Protocol Mode=*RTU* is used.	RTU ASCII	RTU
Addressing model	Sets an address model of MODBUS protocol: **MODBUS PDU** data are addressed from *0* up to *65535*. **MODBUS data Model** data are addressed from *1* up to *65536*. **Note:** *MODBUS PDU* is a default value. If the *MODBUS data Model* is set, the object with the address X is addressed as X-1 in *MODBUS PDU*. After you change this parameter, restart a proper communication process.	MODBUS PDU MODBUS data Model	MODBUS PDU
TCP/IP protocol variant	Select a variant of the protocol in case of TCP/IP communication: **"MODBUS TCP"** is a variant of communication without control checksum. Safeguarding is done by the underlying TCP protocol. **"MODBUS over TCP"** is a variant where a payload is MODBUS RTU data containing a checksum.	"MODBUS TCP" "MODBUS over TCP"	"MODBUS TCP"
Max. Registers	Maximum count of registers that are read by one request.	-	100
Max. Bytes	Maximum count of bytes that are required by one request (only in "Byte mode").	-	100
Skip Unconfigured	To read the values from addresses that are not configured is not allowed. **Description and example:** The requests for data, which are limited by protocol parameter "Max. Registers" or "Max. Bytes", are sent as standard. If I/O tags with addresses "Holding Registers" 1, 2, and 5 have been configured, one request reading 5 registers starting with the address 1 is sent although the I/O tags with addresses 3 and 4 are not configured. It is more efficient to obtain the required data by one request than by two ones even if the unnecessary data are also read. If the parameter "Skip Unconfigured" is set on YES, two requests are sent, the first one reads two registers from address 1 and the second one reads one register from address 5.	YES /NO	NO

Check Receive Length	If this parameter is set to YES, then an extra check is performed when receiving a response to a read request: the length of received data is checked whether it matches the number of registers in a read request: • if Byte mode is on (Byte mode=YES), the length of received data must equal to the number of registers • if both Byte mode and variable mode are off, the length of received data must equal to double of the number of registers • if the variable mode is on (Variable mode=little-endian or big-endian), check has not been implemented yet This extra check is reasonable on high-latency and variable-latency lines - e.g. GPRS networks - to detect and avoid the situation when read request (#1) is repeated due to timeouts and then two responses are received, the second of which could be considered to be an answer to another read request (#2), thus causing wrong values being assigned to I/O tags addressed by this read request #2.	YES /NO	NO

I/O tag configuration

Possible types of I/O tag values for invariable mode: **Ai**, **Ao**, **Di**, **Do**, **Ci**, **Co, TxtI**.

Possible types of I/O tag values for variable mode: **Ai**, **Ao**, **Di**, **Do**, **Ci**, **Cout**, **TxtI**, **TxtO**, **TiA**.

I/O tag address:

The main address space in the protocol MODBUS is divided into the following registers:

- Coils type (reading/writing)
- Discrete Inputs (reading)
- Holding Registers (reading/writing)
- Input Registers (reading)

Independent addressing with the address size of 2 bytes, i.e. addresses from 0 up to 65535 (so-called *MODBUS PDU* addressing model), is in an address space of each type of register. Some devices work with address space starting with 1 (so-called *MODBUS Data Model*). In this case, it is necessary to deduct 1 in the address at configuration I/O tags in the D2000 system or change the setting of the parameter Addressing model to the *MODBUS data Model*.

I/O tag address can be in a basic or extended format (for a variable mode).

Basic format of I/O tag address:
Address format is *[/|U|Uu|Ul|f|F|L|Ll|S|Sl|B|X|sn.|an.|An.][d|D][b][s]RdFn[-WrFn[d]].Address[.BitNr]* in which:

- First character defines a type of I/O tag:
  - **I** - Integer16 (default) - one register is read, signed
  - **U** - Unsigned16 - one register is read, unsigned
  - **Uu** - Unsigned16 - one register is read, unsigned, only upper byte is considered (1st in sequence)
  - **Ul** - Unsigned16 - one register is read, unsigned, only lower byte is considered (2nd in sequence)
  - **f** - Float (4 bytes = 2 registers) - two registers with *Address* and *Address+1* are read and transmitted as big-endian (see Note).
  - **F** - Float (4 bytes = 2 registers) - two registers with *Address* and *Address+1* are read and transmitted as little-endian (so-called Modicon format), (see Note)
  - **L** - Unsigned long (4 bytes = 2 registers) - two registers with *Address* and *Address+1* are read, unsigned, and transmitted as big-endian (see Note)
  - **Ll** - Unsigned long (4 bytes = 2 registers) - two registers with *Address* and *Address+1* are read and transmitted as little-endian, unsigned (see Note)
  - **S** - Signed long (4 bytes = 2 registers) - two registers with *Address* and *Address+1* are read, signed, and transmitted as big-endian (see Note)
  - **Sl** - Signed long (4 bytes = 2 registers) - two registers with *Address* and *Address+1* are read and transmitted as little-endian, signed (see Note)
  - **B** - Byte unsigned, only the upper 8 bits of the register value
  - **X** - Byte unsigned, only the lower 8 bits of the register value
  - **sn.** - Text string with the length of *n* characters, one register is one character, ***n*** registers with *Address* up to *Address+n-1* are read
  - **an.** - Text string with the length of ***2*n*** characters, one register is two ASCII characters, characters are transmitted in the same order as they appear in the string, *n* registers with *Address* up to *Address+n-1* are read
  - **An.** - Text string with the length of ***2*n*** characters, one register is two ASCII characters, characters are transmitted in big-endian order (i.e . "1234" is transmitted as "2143"), *n* registers with *Address* up to *Address+n-1* are read
- Modifier ***d*** indicates that a number is an 8-byte number (4 consecutive registers). It can be used for types *L*, *Ll*, *S*, *Sl*, *F*,*f,* and it is used for configuration of signed/unsigned 8-byte integer as well as an 8-byte float (big-endian <B8>..<B1> and little-endian <B1>..<B8> formats).
  Modifier ***D*** indicates that a number is an 8-byte number (4 consecutive registers). It can be used for types *Ll*, *Sl*, *F* and it is used for configuration of signed/unsigned 8-byte integer as well as an 8-byte float (little-endian format <B2><B1><B4><B3><B6><B5><B8><B7>).
  Note: when using a modifiers ***d*** *or* ***D***, the I/O tag must be of Analog type (Ai), because Integer type (Ci) in D2000 is implemented as a 4-byte variable and overflow might occur. Writing of Integer type (Co) as an 8-byte number is supported.
- Modifier ***b*** indicates that figure is coded by BCD. It can be used for I/O tags of *I*, *U*, *B, L, Ll* types.
- Modifier ***s*** indicates that a status register (Unsigned16) located on address *Address* is followed by a big-endian Float value located on address *Address+1 .. Address+2*. This indicator is used for type *f* and it is implemented for calorimeter Endress+Hauser RMS621. The following table shows the values of the status register and their mapping to D2000 attributes.

Status register	D2000 attributes

0 : Invalid value	Weak
1 : Measured value valid	Valid
2 : Overflow warning 3 : Overflow error 4 : Underflow warning 5 : Underflow error 6 : Saturated steam alarm 7 : Error in differential pressure calculation 8 : Wrong medium for DP calculation 9 : Wrong value range - DP calculation inaccurate 10 : Differential pressure - general error 11 : Range overshoot (Tsat > 350 etc.) on 12 : Change in state of aggregation 26 : Differential pressure --> general error 99 : No measured value is assigned to the register in the setup of the ModBus	Weak

- Parameter ***RdFn*** is a function of the Modbus protocol for a data reading. The following functions are implemented:
  - **1** - Read Coils: binary status reading
  - **2** - Read Discrete Inputs: binary input reading
  - **3** - Read Holding Registers: status register reading (Integer16/Unsigned16 and Float32 - reads two successive registers)
  - **4** - Read Input Registers: input register reading (Integer16/Unsigned16 and Float32 - reads two successive registers)
  - **0** - A value is not read, it is only written. The function for writing (WrFn) must be set.
- Parameter ***WrFn*** is the function of the Modbus protocol for data writing. The following functions are implemented:
  - **5** - Write Single Coil: binary status writing (default for *Read Coils*)
  - **6** - Write Single Register: status register writing (default for *Read Holding Registers*)
  - **16** - Write Multiple registers: multiple registers writing, it must be used when 2-register type is written (e.g. Float, Unsigned long, etc.).
    Note: function can be used to write more than two registers at once if a text string is used. Example:
    if we have an I/O tag with address a3.0-16.#8A00 (i.e. text string covering 3 registers, having length of 6 characters) and we write a string '123456', then hexadecimal values 0x3132, 0x3334 and 0x3536 (ASCII code for '1' is 0x31, for '2' is 0x32 etc) will be written to registers 0x8A00, 0x8A01 and 0x8A02.
  - **22** - Mask Write Register: write affects only the value of the particular bit *BitNr* of the status register. It is usable only for *Do* value types with the address parameter *BitNr*.
- Parameter ***d*** activates the function "delayed write". Sending of the value is delayed until the request to write the value of the object without parameter *d* comes. All accumulated requests waiting to be written are sent. If the function *WrFn* is set to "Write Multiple Registers", the values are sent in one packet.
- Parameter ***Address*** is a 2-byte address of register (0-65536). See also the protocol parameter Addressing model.
  Note: address can be specified as a hexadecimal number using a number sign (#), e.g. #50CE
- Parameter ***BitNr*** is a number of bit in a word. The values 0-7 are allowed to be used for binary statuses and inputs, values 0-15 are allowed to be used for reading of bit from 16-bit status or input registers.

Note about the byte and register order

1. MODBUS protocol uses the big-endian, i.e. the most significant byte (MSB) is transmitted as first. Examples:

Received bytes of MSB-LSB	I/O tag type	Value
0x00 0x01	I, U	1
0xFF 0xFE	I	-2
0xFF 0xFE	U	65534
0x01 0x02	B	1
0x01 0x02	X	2

2. When values are read from two registers as big-endian the received bytes are analysed in this way:

Most significant register (ADR address)		Least significant register (ADR+1 address)	
MSB	LSB	MSB	LSB

Examples:

Received bytes of register (MSB-LSB)	Received bytes of register + 1 (MSB-LSB)	I/O tag type	Value
0x00 0x00	0x00 0x01	L, S	1
0xFF 0xFF	0xFF 0xFE	S	-2
0x00 0x01	0x00 0x02	L, S	65538
0x3F 0x80	0x00 0x00	f	1.0
0xC0 0x00	0x00 0x00	f	-2.0

3. When values are read from two registers as little-endian the received bytes are analysed in this way (if Little Endian Mode=**2143**):

Least significant register (ADR address)		Most significant register (ADR+1 address)	
MSB	LSB	MSB	LSB

Examples:

Received bytes of register (MSB-LSB)	Received bytes of register + 1 (MSB-LSB)	I/O tag type	Value
0x00 0x01	0x00 0x00	LI, SI	1
0xFF 0xFE	0xFF 0xFF	SI	-2
0x00 0x02	0x00 0x01	LI, SI	65538
0x00 0x00	0x3F 0x80	F	1.0
0x00 0x00	0xC0 0x00	F	-2.0

Example of configuration:

- *1.10* - the function *Read Coils* reads the binary status value with address 10.
- *3.1* - a signed 16-bit number, it is read by the function *Read Holding Registers* from the address 1 (it can be also in the form *I3.1*).
- *U3.1* - an unsigned 16-bit number that is read by the function *Read Holding Registers* from address 1.
- *I3-6.1000* - signed 16-bit number that is read by the function *Read Holding Registers* from address 1000 and written by the function *Write Single Register* (as this function is the default, the address could be also *I3.1000)*.
- *S3.321* - a signed 32-bit number, it is read by the function *Read Holding Registers* from the registers 321 and 322.
- *B1.20.0* - a bit that is read by function *Read Coils* from address 20 as 0-bit in a byte.
- *s10.3.123* - a text string, length 10 characters (2 bytes per character), it is read by the function *Read Holding Registers* from the address 123.
- *a5.3.123* - a text string, length 10 characters (1 byte per character), it is read by the function *Read Holding Registers* from the address 123.
- *U0-6.456* - an unsigned 16-bit number, is written to the register 456, it is written by *Write Single Register*, a register reading is not performed.

Extended format of I/O tag address:

Address format is *[xN][I|U|F|B|C|T][b]RdFn[-WrFn].Address[.BitNr]* in which:

- *xN* indicates the number of bytes that read or write. Valid values for N are 1, 2, 4 (in combination with *I, U, F*), 6 for *T* type, and an arbitrary number for *C* type.
- A letter defines the type of I/O tag. Besides standard *I, U, F, B,* two extra types have been added:
  - **C** - text string of fixed length (e.g. x10.C3.1001 is a 10-character string on address 1001)
  - **T** - time stamp with length of 6 bytes (ss:mi:hh dd:mm:yy)
- The meaning of other parameters is in compliance with the standard mode.

See the example of the configuration in the next section.

Note to FloBoss 103 device

- configuration software ROCLINK800
- default login LOI, password 1000
- logging in FloBoss 103: click on DirectConnect (connection through COM1, on the side of FloBoss 103 it is connected to LOI-local interface)
- menu *Configure->Modbus->Configuration*
  set the parameter "Variable Mode" on station in D2000 according to setting "Byte Order":
  - if "Least Significant Byte first" then "Little endian"
  - if "Most Significant Byte first" then "Big endian"
- I/O tags are configured through menu *Configure -> Modbus -> Registers* on FloBoss 103
- following types are supported (*n* means 16-bit address):
  - Binary input:
    - address in D2000: 1.*n*, e.g. 1.1001, variable of Di/Dout type
    - address in FloBoss 103: variable of *BIN* type
      Function: 1
      Starting/ending register: *n*
  - Binary output:
    - address in D2000: 1.*n*, e.g. 1.1001, variable of Dout type
    - address in FloBoss 103: variable of *BIN r/w*
      Function: 1 (for reading)
      Starting/ending register: *n*
      Function: 5 (for reading)
      Starting/ending register: *n*
  - Unsigned Int 8 bits input:
    - address in D2000: x1.B3.*n*, e.g. x1.B3.1003, variable of Ci/Co type

        - address in FloBoss 103: variable of *UINT8* type
          Function: 3A or 3B
          Starting/ending register: *n*
    - Unsigned Int 8 bits output:
        - address in D2000: x1.B3.*n*, e.g. x1.B3.1003, variable of Co type
        - address in FloBoss 103: variable of *UINT8 r/w* type
          Function: 3A or 3B
          Starting/ending register: *n*
          Function: 6
          Starting/ending register: *n*
    - Unsigned Int 16 bits input:
        - address in D2000: x2.U3.*n*, e.g. x2.U3.1004, variable of Ci/Co type
        - address in FloBoss 103: variable of *UINT16* type
          Function: 3A or 3B
          Starting/ending register: *n*
    - Unsigned Int 16 bits output:
        - address in D2000: x2.U3.*n*, e.g. x2.U3.1004, variable of Co type
        - address in FloBoss 103: variable of *UINT16 r/w* type
          Function: 3A or 3B
          Starting/ending register: *n*
          Function: 6
          Starting/ending register: *n*
    - Signed Int 16 bits input:
        - address in D2000: x2.I3.*n*, e.g. x2.I3.1005, variable of Ci/Co type
        - address in FloBoss 103: variable *INT16* type
          Function: 3A or 3B
          Starting/ending register: *n*
    - Signed Int 16 bits output:
        - address in D2000: x2.I3.*n*, e.g. x2.I3.1005, variable of Co type
        - address in FloBoss 103: variable of *INT16 r/w* type
          Function: 3A or 3B
          Starting/ending register: *n*
          Function: 6
          Starting/ending register: *n*
    - Unsigned Int 32 bits input:
        - address in D2000: x4.U3.*n*, e.g. x4.U3.1006, variable of Ci/Co type
        - address in FloBoss 103: variable of *UINT32* type
          Function: 3A or 3B
          Starting/ending register: *n*
    - Unsigned Int 32 bits output:
        - address in D2000: x4.U3.*n*, e.g. x4.U3.1006, variable of Co type
        - address in FloBoss 103: variable of *UINT32 r/w* type
          Function: 3A or 3B
          Starting/ending register: *n*
          Function: 6
          Starting/ending register: *n*
    - Float 32 bits input:
        - address in D2000: x4.F3.*n*, e.g. x4.F3.1008, variable of Ai/Ao type
        - address in FloBoss 103: variable of *FL* type
          Function: 3A or 3B
          Starting/ending register: *n*
    - Float 32 bits output:
        - address in D2000: x4.F3.*n*, e.g. x4.F3.1008, variable of Co type
        - address in FloBoss 103: variable of *FL r/w* type
          Function: 3A or 3B
          Starting/ending register: *n*
          Function: 6
          Starting/ending register: *n*
    - String (N bytes) input:
        - address in D2000: x1N.C3.*n*, e.g. x10.C3.1010, variable of TxtI/TxtO type
        - address in FloBoss 103: variable of *AC*m(*AC10,AC12,AC20,AC30,AC40*) type
          Function: 3A or 3B
          Starting/ending register: *n*
    - String (N bytes) output:
        - address in D2000: xN.C3.*n*, e.g. x10.C3.1010, variable of Co type
        - address in FloBoss 103: variable of *AC*N r/w type (*AC10,AC12,AC20,AC30,AC40*)
          Function: 3A or 3B
          Starting/ending register: *n*
          Function: 6
          Starting/ending register: *n*
    - Time and date 6 bytes input:
        - address in D2000: x6.T3.*n*, e.g. x6.T3.1010, variable of TiA/TxtI type
        - address in FloBoss 103: variable of *DT6* type
          Function: 3A or 3B
          Starting/ending register: *n*
        - **Note 1**: FloBoss 103 supports local and monotonous time - that is why the configuration of station in D2000 must correspond to configuration of FloBoss.
        - **Note 2**: It is possible to set time and date but it requires to configure extra the I/O tags for a second, minute, hour, day, month, and year as *Unsigned Int 8 bits* and after that to write into them.

Note on Honeywell controllers

The basic parameters and current data of these controllers are not normally read by means of functions 0x01 up to 0x04. It is necessary to use the function 0x14/0x15 Read/write configuration reference data. These controllers use "big-endian" byte order. Therefore, for proper functionality, it is not necessary to modify parameters that change byte mode and endianness.

Examples of I/O tag configuration:

```
20.039 - 16-bit number from address 39(0x27)
f20.040 - 32-bit real number from address 40(0x28)
```

Literature

- MODBUS APPLICATION PROTOCOL SPECIFICATION V1.1b, December 28, 2006. http://www.modbus.org

Blog

You can read sblog about Modbus protocol:

- Communication – Modbus protocol
- Communication - Modbus in practice

Changes and modifications

-

Document revisions

- Ver. 1.0 - November 27th, 2006 - document creating.
- Ver. 1.1 - November 21st, 2007 - document update.
- Ver. 1.2 - April 24th, 2009 - document update.
- Ver. 1.3 - November 3rd, 2010 - document update.
- Ver. 1.4 - December 6th, 2010 - document update.

Related pages:

Communication protocols