

MODBUS Client

MODBUS Client communication protocol

[Supported device types and versions](#)

[Communication line configuration](#)

[Line protocol parameters](#)

[Station configuration](#)

[I/O tag configuration](#)

[Note on FloBoss 103 device](#)

[Note on Honeywell](#)

[Literature](#)

[Changes and modifications](#)

[Document revisions](#)

Supported device types and versions

The protocol implements client (master) communication with arbitrary devices that support MODBUS RTU or MODBUS ASCII standards (serial communication) as well as MODBUS over TCP/IP. Moreover, it supports two extensions:

- **Byte mode** - allows working with devices that implement the registers as 1-byte variables (in contrast with Modbus standard in which the register value is 2 bytes).
- **Variable mode** - allows working with devices that implement the registers with different sizes than standard 2 bytes. It was implemented because of support of 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, a 6-byte time stamp, and other.
Note: this mode enables work with devices implementing the so-called Enron Modbus or Daniel Modbus.
- **Passive (scanning) mode** allows to work in eavesdropping mode. This applies especially to serial communication when the communication port of the device is already used to communicate with another Master device. Due to the nature of the Modbus protocol, it is necessary to receive both requests and responses in this mode.

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 commonly used, 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.

Forced disconnection: If all stations on the [TCP/IP-TCP](#) or [TCP/IP-TCP Redundant](#) line are in the simulation mode or the communication is stopped for them, the line will be disconnected (the communication socket will be closed). If the simulation is disabled for at least one station and the communication is not stopped for it (the [Parameters](#) tab of the *Station* type object), the line will be connected again.

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
Passive Mode	This parameter activates the passive (scanning) mode. In this mode, requests are not sent and writing does not work. Only received packets are parsed. The nature of the Modbus protocol implies that it is necessary to receive both the requests and responses of existing communication.	YES /NO	NO
TCP No Delay	Setting the <i>TCP No Delay</i> parameter to YES causes the low-level socket option TCP_NODELAY to be 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 response returns after a request has been sent, the station's status will change to a communication error.	-	2
Retry Timeout	Timeout before resending a request if no response has been received.	s	0.1
Wait First Timeout	The delay after sending the request and before reading the response.	s	0.1
Wait Timeout	The delay between the response readings.	s	0.1
Max. Wait Retry	The maximum number of retries of the response 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
Read After Write	After writing to the I/O tag, the reading immediately follows. By setting the parameter to the NO value, it is possible to reduce the load on communications (especially serial ones) with a large number of writes.	YES /NO	YES
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: <ul style="list-style-type: none">• 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 <i>Variable mode</i> : 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 modes 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= <i>RTU</i> . Note 4: A data encoding <i>big-endian</i> is used automatically, according to the default parameter values <i>Byte mod=NO</i> and <i>Variable mode=OFF</i> (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: <i>RTU</i> or <i>ASCII</i> . Note: In the case of "MODBUS over TCP/IP", the parameter value is ignored and Protocol Mode= <i>RTU</i> 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: <i>MODBUS PDU</i> is a default value. If the <i>MODBUS data Model</i> is set, the object with the address X is addressed as X-1 in the <i>MODBUS PDU</i> . After you change this parameter, a restart of the respective communication process was required in the past (KOM binaries older than May 27, 2021).	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

Bool Mask	<p>If a value of the integer type (Holding Registers, Input Registers) is assigned to an I/O tag of the <i>Di</i> or <i>Dout</i> type, this is done by comparing the read value with zero. If the value is zero, the value of the I/O tag is False, otherwise True. The <i>Bool Mask</i> parameter allows specific bits to be filtered out before the comparison is made, based on a bitmask specified as a hexadecimal number (the leftmost byte is the highest). The bitmask <i>FF FF FF FF</i> means that all bits are considered (for 1- and 2-register integer addresses).</p> <p>Bitmask 01 means that only the lowest bit is considered. If the address of the I/O tag specifies the use of only the lower/upper byte of the register, the lowest/second lowest byte of the mask is applied.</p>	-	FF FF FF FF
Skip Unconfigured	<p>This parameter is used to avoid reading the values from addresses that are not configured.</p> <p>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 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 to YES, two requests are sent, the first one reads two registers starting from address 1 and the second one reads one register from address 5. Some Modbus servers respond by exception to the reading of a range of registers that contains "unknown" registers (which e.g. they don't have mapped to internal memory).</p> <p>See an example of such communication: 14:55:31.532 20-06-2024 D MDB_IDAC> REQ St: 1, Fnc: 3, Addr: 1179, Len: 17 14:55:31.533 20-06-2024 T MDB_IDAC> RQ><12><3B><00><00><00><06><01><03><04><9B><00><11> 14:55:31.563 20-06-2024 T MDB_IDAC> IN><12><3B><00><00><00><03><01><83><04> 14:55:31.563 20-06-2024 E MDB_IDAC> Read Holding Registers: ExceptionCode 04 (ReadMultipleRegisters not OK)</p>	YES /NO	NO
Check Receive Length	<p>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:</p> <ul style="list-style-type: none"> if Byte mode is on (Byte mode=YES), the length of received data must be equal to the number of registers if both Byte mode and variable mode are off, the length of received data must be equal to double the number of registers if the variable mode is on (Variable mode=little-endian or big-endian), the check has not been implemented yet <p>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.</p>	YES /NO	NO
Dummy Request Mode	<p>If the parameter is set to YES, then a single request (which contains the total number of registers) is used. It is necessary that the I/O tags are defined for all addresses 0..N. This mode can be used for special devices that send all data (with variable size registers - 2, 4, 8 bytes - in a single response).</p>	YES /NO	NO
TCP Write Password	<p>On the TCP/IP-TCP line, immediately after the connection is established, it is possible to write a specific value to the selected address (TCP Password Address) using the selected function (TCP Password Function). In this way, the ComAp control unit enables the authorized Modbus client to be authenticated. The value is entered in hexadecimal in the order of the bytes as they will be transmitted (e.g. "01 0A BC D0". If an odd number of bytes is entered, a byte with the value 0 will be added at the end.</p> <p>Setting the parameter to an empty value causes the write to not be performed after the connection is established.</p>	-	
TCP Password Address	<p>Address for writing the password (TCP Write Password) on the TCP/IP-TCP line. If the password is longer than 2 bytes, it is the address of the first register. If the password has e.g. 6 bytes, written as 3 registers from the specified address.</p>	-	0
TCP Password Function	<p>The write function for writing the password (TCP Write Password) on the TCP/IP-TCP line. If the number of bytes of the password is greater than 2, function 16 (Write Multiple Registers) is used even if function 6 (Write Single Register) is specified.</p>	6 /16	6

I/O tag configuration

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

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

The I/O tag with an address starting with **%IGNORE** will be ignored.

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

The basic format of I/O tag address:

Address format is `[I|U|Uu|Ul|f|L|LI|S|Sl|B|X|sn.|an.|An.][d|D][b|s]RdFn[-WrFn[d]].Address[.BitNr] [,Items]` in which:

- The 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, and only the upper byte is processed (1st in sequence)
 - Ul** - Unsigned16 - one register is read, unsigned, and only the lower byte is processed (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](#))
 - LI** - 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*, *LI*, *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 LI, Sl, F and it is used for the 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>).`
- Modifier **b** indicates that the number BCD-coded. It can be used for I/O tags of *I*, *U*, *B*, *L*, *LI* 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 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 a 2-register type is written (e.g. Float, Unsigned long, etc.).
Note: The 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 a 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". The 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 the 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 bit's position in a word. The values 0-7 are allowed to be used for binary statuses and inputs, and values 0-15 are allowed to be used for reading bits from 16-bit status or input registers.
Note: coexistence of an I/O tag without a *BitNr* parameter and multiple I/O tags with a *BitNr* parameter having the same *Address* is possible.
- Parameter **Items** indicates the number of objects to read. This parameter is only meaningful if the [Destination Column](#) is configured. The parameter specifies the number of objects that will be read and written to the structured variable. If this parameter is not specified, the number of read objects is derived from the size of the structure, so this parameter allows to limit the number of read items (e.g. if values are to be read into the next rows of the structure using another I/O tag).

Note on writing: if only a part of the register (lower/higher byte or selected bit) is written and the I/O tag also has **RdFn** (function for a data reading) configured, then when writing, the value of the entire register is read first, and then the relevant part of the register is modified and the entire register is written.

Note about the byte and register order

1. MODBUS protocol uses the big-endian, i.e. the most significant byte (MSB) is transmitted 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 analyzed in this way:

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

Examples:

Received bytes of the register N (MSB LSB)	Received bytes of the register N+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 analyzed 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 the register N (MSB LSB)	Received bytes of the register N+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.
- 1.10, 4 - the function *Read Coils* reads the binary status values with addresses 10-13 to the [Destination Column](#).
- 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 the 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.
- *Ld3.3204* - reading the value of a 64-bit unsigned number, it is read by the function *Read Holding Registers* from addresses 3204 to 3207.

Extended format of I/O tag address:

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

- xN indicates the number of bytes that read or write. Valid values for N are 1, 2, 4, 8 (in combination with I, U, F, B), 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 types, 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** - timestamp with length of 6 bytes (ss:mi:hh dd:mm:yy)
- The meaning of other parameters complies with the standard mode.

See the example of the configuration in the [next section](#).

Note on 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 the station in D2000 according to the 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 , the 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 , the 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 *ACm(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 *ACN 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 the station in D2000 must correspond to the configuration of FloBoss.
 - **Note 2:** It is possible to set time and date but it requires configuring extra 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

Honeywell UDC1700 controllers (probably generally UDC1xxx):

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 functions *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)

Note: Honeywell-made products (UDC 2xxx/3xxx, HC900, DPR, Trendview) use the common Modbus function codes 1-4 in spite of the UDC manuals referring only to *0x14/0x15 Read/write configuration reference data* register tables. For more detailed information, see this [post](#).















Literature

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

An example of communication

The attached ZIP contains the configuration of two lines and two stations with the MODBUS Client/**MODBUS Server** protocol that communicate via TCP (MODBUS Server listens on a TCP port 9999). Data written through one line is received through the other line.

Four I/O tags are configured on each line (Float, with address 3.0, Signed with addresses 4.0 and 4.1, and Bool with address 1.0). MODBUS Server I/O tags values are controlled by the system second (Sec) and use linear conversion (Float is divided by 1000, Signed with address 4.0 is multiplied by 10). The value of the Bool I/O tag is controlled by the eval tag *P.TrueFalse*, which changes the value True/False every second.

 Object name	Object description	Current value	 Time
 [B.MODBUSSRV_TCP]		STON	08:24:01.068 29-05-202
 [B.MODBUS_TCP]		STON	08:24:01.167 29-05-202
 [L.MODBUSSRV_TCP]	Modbus Server - TCP version	BTRUE	08:24:01.067 29-05-202
 [L.MODBUS_TCP]	Modbus Client - TCP version	BTRUE	08:24:01.067 29-05-202
 M.MODBUSSRV_TCP_Bool1_00		BTRUE	08:27:42.002 29-05-202
 M.MODBUSSRV_TCP_Float3.0		42	08:27:42.002 29-05-202
 M.MODBUSSRV_TCP_Signed4.0		42	08:27:42.002 29-05-202
 M.MODBUSSRV_TCP_Signed4.1		42	08:27:42.002 29-05-202
 M.MODBUS_TCP_Bool1_00		BTRUE	08:27:42.153 29-05-202
 M.MODBUS_TCP_Float3.0		0.04199999994	08:27:42.354 29-05-202
 M.MODBUS_TCP_Signed4.0		420	08:27:42.253 29-05-202
 M.MODBUS_TCP_Signed4.1		42	08:27:42.254 29-05-202



Modbus_test.zip

An example of communication - variable mode

The attached ZIP contains the configuration of the line, station, and I/O tags for communication with an Emerson chromatograph that implements MODBUS DANIEL®.

Thirteen I/O tags read 4-byte registers in Float32 format, two I/O tags read 2-byte registers in Int16 format. Byte order is big-endian.



Modbus_Client_V...riable_Mode.zip



Blog

You can read blogs about the Modbus protocol

- [Communication – Modbus protocol](#)
- [Communication - Modbus in practice](#)
- [Communication - HART, Modbus, and a Parrot](#)
- [D2000 and UniPi Neuron](#)
- [What load can Raspberry Pi handle?](#)

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.
- Ver. 1.5 - September 5th, 2022 - document update (support for 8-byte values in the extended format).
- Ver. 1.6 - September 7th, 2022 - document update (added parameter "Dummy Request Mode").
- Ver. 1.7 - February 2, 2024 - document update (added destination column support).



Related pages:

[Communication protocols](#)