oModbusMaster
Reference Manual

Version 1.1

Index

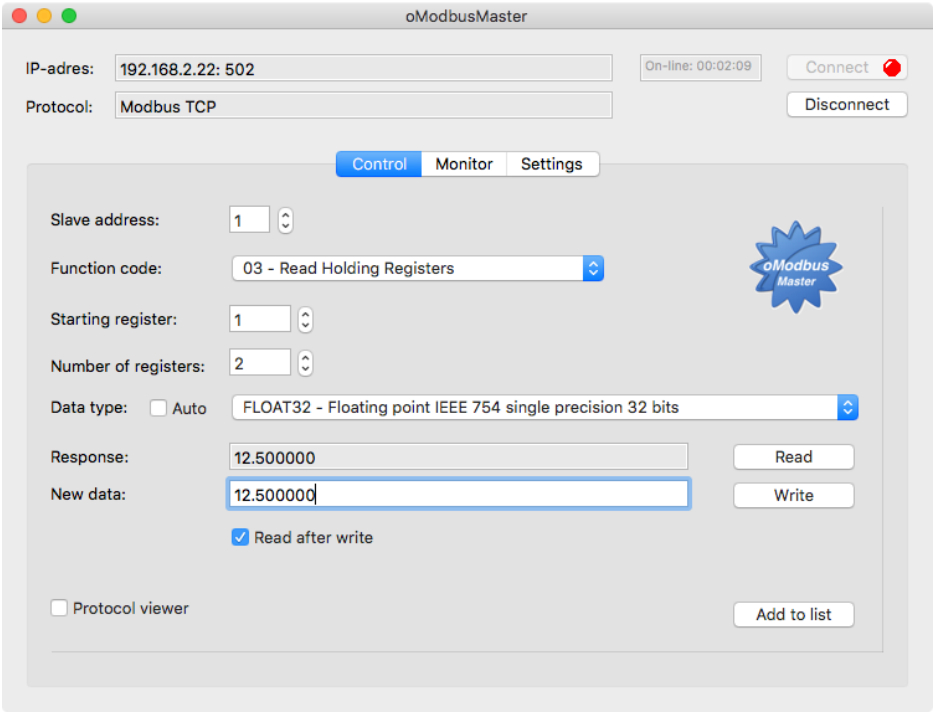
oModbusMaster

oModbusMaster is an easy to use program for communication with all kinds of Modbus devices, supporting **Modbus-TCP** and **Modbus-RTU over TCP**.

Control - Read and write data from and to a Modbus device
Monitor - Read and write data using a list of different Modbus items
Settings - Configure oModbusMaster

oModbusMaster Control

Control can be used to investigate Modbus devices for available items and their data types.



Read after write can be set to check whether the item was correctly written.

Function code 01 for Read used 05 for Write.
Function code 03 for Read used 06 for Write.

You can add an item directly to the monitor items list by using the Add to list button.

oModbusMaster can read and write Modbus items using te following **Function codes**:

01 – Read Coil Status
02 – Read Input Status
03 – Read Holding Registers
04 – Read Input Registers
05 – Force Single Coil
06 – Preset Single Register
15 – Force Multiple Coils
16 – Preset Multiple Registers
17 – Report Slave ID

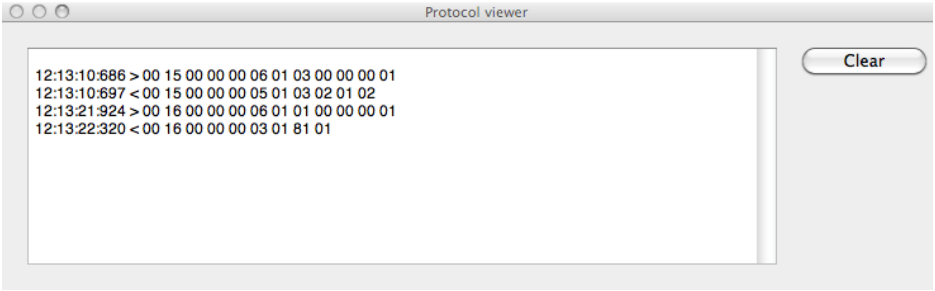
Note: not all Modbus devices support these function codes.

The following **Data types** are available:

BOOL1 – Boolean 1 bit
BOOL8 – Boolean 8 bits
–
CHAR2 – Character ascii 2 bytes
CHAR4 – Character ascii 4 bytes
CHAR6 – Character ascii 6 bytes
CHAR8 – Character ascii 8 bytes
–
FLOAT32 – Floating point IEEE 754 single precision 32 bits
FLOAT32i – Floating point IEEE 754 single precision 32 bits inverse
–
HEX2 – Hexadecimal 2 bytes
HEX4 – Hexadecimal 4 bytes
HEX6 – Hexadecimal 6 bytes
HEX8 – Hexadecimal 8 bytes
–
INT16 – Signed integer 16 bits
INT32 – Signed integer 32 bits
INT32i – Signed integer 32 bits inverse
–
INTD16 – Signed integer 1 decimal 16 bits
INTD32 – Signed integer 1 decimal 32 bits
INTD32i – Signed integer 1 decimal 32 bits inverse
–
UINT16 – Unsigned integer 16 bits
UINT32 – Unsigned integer 32 bits
UINT32i – Unsigned integer 32 bits inverse
–
UINTD16 – Unsigned integer 1 decimal 16 bits
UINTD32 – Unsigned integer 1 decimal 32 bits
UINTD32i – Unsigned integer 1 decimal 32 bits inverse

The i character in the data type means that the two bytes high- and low-order are switched (inverse).

You can enable the **Protocol viewer** to get more information about the datastream.

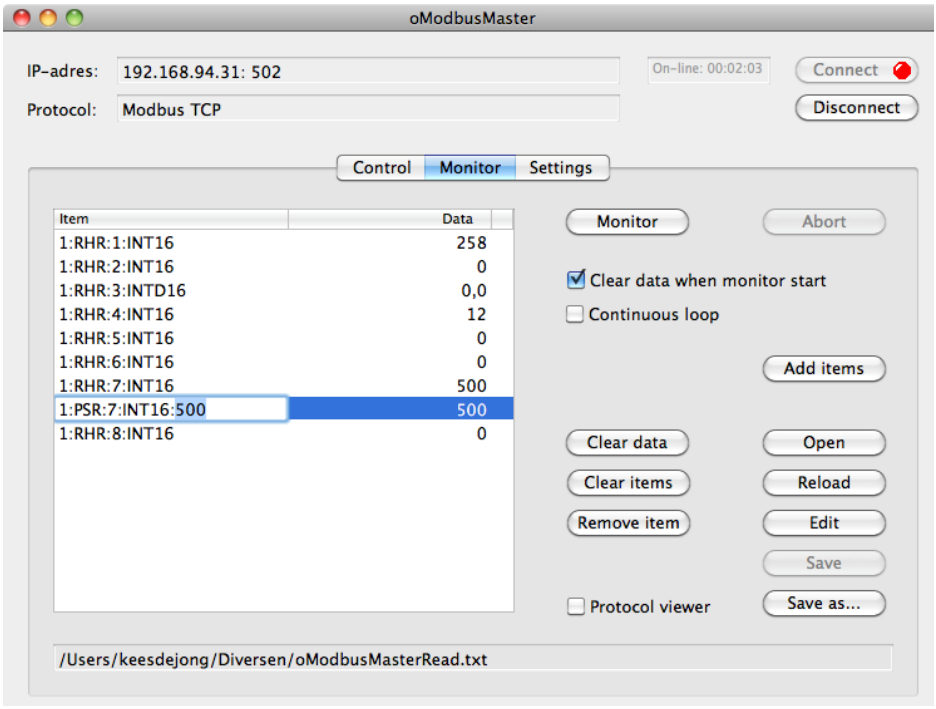


oModbusMaster Monitor

Monitor can be used to read and write a list of items used by the selected Modbus device.

oModbusMaster can Save, Save as... and Open a customized item list.

You can easily add, remove or modify items.
By dubble clicking an item, you can change it.

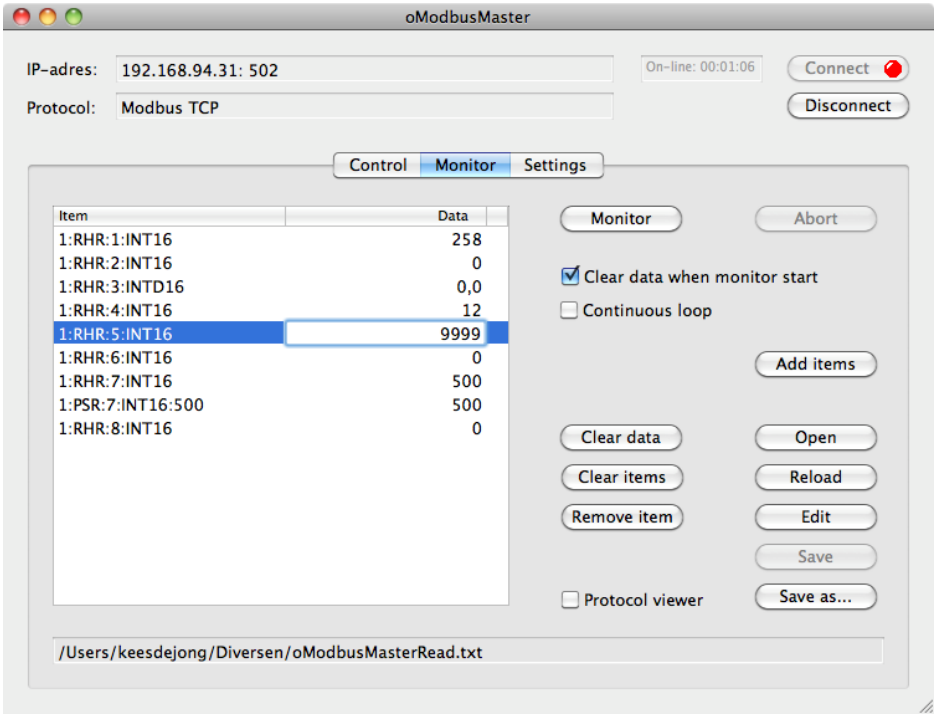


The following **Function descriptions** are used:

RCS – Read Coil Status
RIS – Read Input Status
RHR – Read Holding Registers
RIR– Read Input Registers
FSC – Force Single Coil
PSR – Preset Single Register

Items can be added to the list via the Control screen (for a single item).
By using the Add items button you can add multiple items.

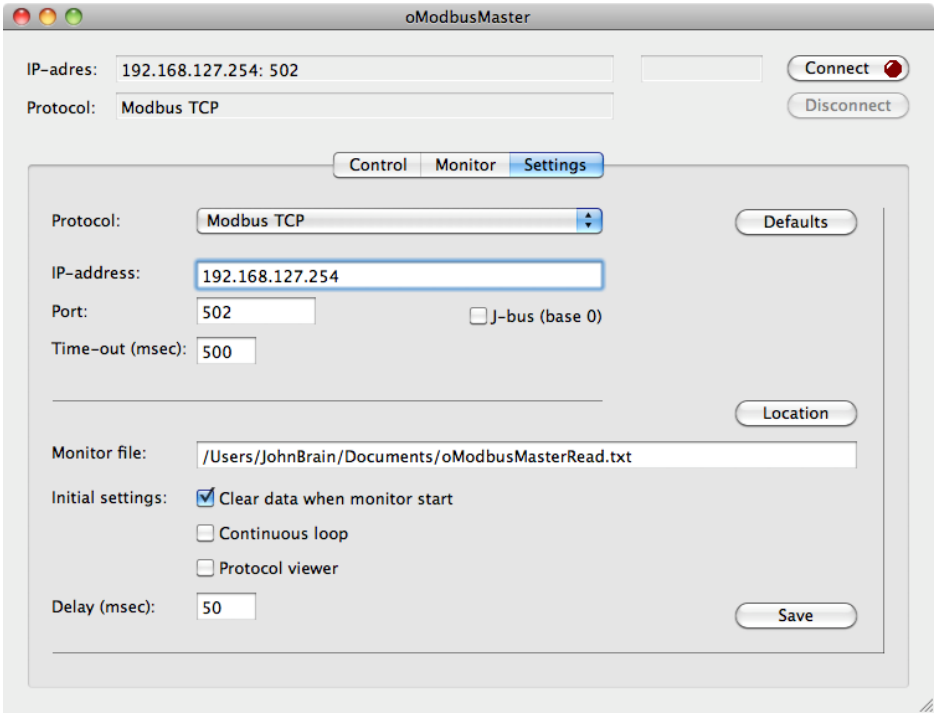
When data is read, you can modify the data by dubble clikking the datafield. When the data is changed, it is written directly to the Modbus device.



When **Continuous loop** is selected and Monitor is in progress, changing items or data is not alowed.

oModbusMaster Settings

Use Settings to configure oModbusMaster.



You can select **Modbus TCP** or **Modbus RTU over TCP**.

Time-out (msec) is used for each Modbus protocol package.

When **J-bus (base 0)** is selected, all register- and coil numbers are decremented by one.

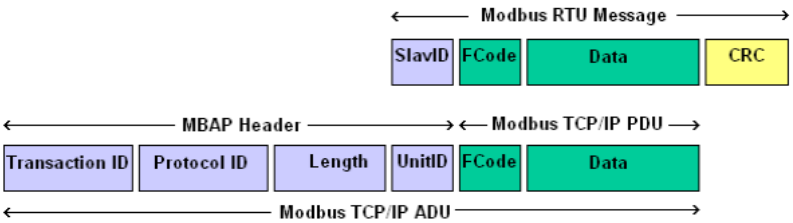
The **Monitor file** is used to store your monitor item list.

The **Delay (msec)** is used as time between each item read during monitoring.

oModbusMaster Modbus information

This overview shows the two protocols used by oModbusMaster, **Modbus TCP** and **Modbus RTU over TCP**.

Modbus RTU over TCP is the same as Modbus RTU (RS485), but for communication you can use a standard ethernet to serial converter (e.g. Moxa).



Modbus TCP Frame Format		
Name	**Length**	**Function**
Transaction ID	2 bytes	*For synchronization between messages of server & Client*
Protocol ID	2 bytes	*Zero for Modbus TCP*
Length	2 bytes	*Number of remaining bytes in this frame*
Unit ID	1 byte	*Unit address (sometimes optional)*
FCode	1 byte	*Function code*
Data	n bytes	*Data as response or commands*

Modbus RTU Frame Format		
Name	**Length**	**Function**
Start	3.5c idle	*Minimal 3-1/2 character times of silence*
Slave ID	8 bits	*Slave address*
FCode	8 bits	*Function code*
Data	n bytes	*Data as response or commands*
CRC	16 bits	*Error checks*
End	3.5c idle	*Minimal 3-1/2 character times of silence between frames*

Read Coil Status (01) - RCS

Request

Byte 0: FCode = 01

Byte 1-2: Starting coil

Byte 3-4: Number of coils

Response

Byte 0: FCode = 01

Byte 1: Byte count of response (B=(bit count+7)/8)

Byte 2-(B+1): Bit values (least significant bit is first coil!)

Exceptions

Byte 0: FCode = 81 (hex)

Byte 1: Exception code = 01 or 02

Example

Read 1 coil at reference 0 (00001 in Modicon 984) resulting in value 1

01 00 00 00 01 => 01 01 01

Read Input Status (02) - RIS

Request

Byte 0: FCode = 02

Byte 1-2: Starting input

Byte 3-4: Number of inputs

Response

Byte 0: FCode = 02

Byte 1: Byte count of response (B = (bit count + 7) / 8)

Byte 2 - (B + 1): Bit values (least significant bit is first coil!)

Exceptions

Byte 0: FCode = 82 (hex)

Byte 1: Exception code = 01 or 02

Example

Read 1 discrete input at reference 0 (10001 in Modicon 984) resulting in value 1

02 00 00 00 01 => 02 01 01

Read Holding Registers (03) - RHR

Request

Byte 0: FCode = 03

Byte 1-2: Starting register

Byte 3-4: Number of registers

Response

Byte 0: FCode = 03

Byte 1: Byte count of response (B = 2 x number of regsiters)

Byte 2 - (B + 1): Register values

Exceptions

Byte 0: FCode = 83 (hex)

Byte 1: Exception code = 01 or 02

Example

Read 1 register at reference 0 (40001 in Modicon 984) resulting in value 1234 hex

03 00 00 00 01 => 03 02 12 34

Read Input Registers (04) - RIR

Request

Byte 0: FCode = 04

Byte 1-2: Starting register

Byte 3-4: Number of registers

Response

Byte 0: FCode = 04

Byte 1: Byte count of response (B = 2 x number of registers)

Byte 2 - (B + 1): Register values

Exceptions

Byte 0: FCode = 84 (hex)

Byte 1: Exception code = 01 or 02

Example

Read 1 input register at reference 0 (30001 in Modicon 984) resulting in value 1234 hex

04 00 00 00 01 => 04 02 12 34

Force Single Coil (05) - FSC

Request

Byte 0: FCode = 05

Byte 1-2: Coil

Byte 3: = FF to turn coil ON, =00 to turn coil OFF

Byte 4: = 00

Response

Byte 0: FCode = 05

Byte 1-2: Coil

Byte 3: = FF to turn coil ON, =00 to turn coil OFF (echoed)

Byte 4: = 00

Exceptions

Byte 0: FCode = 85 (hex)

Byte 1: Exception code = 01 or 02

Example

Write 1 coil at reference 0 (00001 in Modicon 984) to the value 1

05 00 00 FF 00 => 05 00 00 FF 00

Preset Single Register (06) - PSR

Request

Byte 0: FCode = 06

Byte 1-2: Register

Byte 3-4: Register value

Response

Byte 0: FCode = 06

Byte 1-2: Register

Byte 3-4: Register value

Exceptions

Byte 0: FCode = 86 (hex)

Byte 1: Exception code = 01 or 02

Example

Write 1 register at reference 0 (40001 in Modicon 984) of value 1234 hex

06 00 00 12 34 => 06 00 00 12 34

Force Multiple Coils (15)

Request

Byte 0: FCode = 0F (hex)

Byte 1-2: Starting coil

Byte 3-4: Number of coils

Byte 5: Byte count (B = (bit count + 7) / 8)

Byte 6 - (B + 5): Data to be written (least significant bit = first coil)

Response

Byte 0: FCode = 0F (hex)

Byte 1-2: Starting coil

Byte 3-4: Bit count

Exceptions

Byte 0: FCode = 8F (hex)

Byte 1: Exception code = 01 or 02

Example

Write 3 coils at reference 0 (00001 in Modicon 984) to values 0,0,1

0F 00 00 00 03 01 04 => 0F 00 00 00 03

Preset Multiple Registers (16)

Request

Byte 0: FCode = 10 (hex)

Byte 1-2: Starting register

Byte 3-4: Number of registers

Byte 5: Byte count (B = 2 x number of regsiters)

Byte 6 - (B + 5): Register values

Response

Byte 0: FCode = 10 (hex)

Byte 1-2: Starting registers

Byte 3-4: Number of registers

Exceptions

Byte 0: FCode = 90 (hex)

Byte 1: Exception code = 01 or 02

Example

Write 1 register at reference 0 (40001 in Modicon 984) of value 1234 hex

10 00 00 00 01 02 12 34 => 10 00 00 00 01

Report Slave ID (17) - optional

Request

Byte 0: FCode = 11 (hex)

Response

Byte 0: FCode = 11 (hex)

Byte 1-n: Device specific data

Exception codes

All exceptions are signaled by adding 0x80 to the function code of the request, and following this byte by a single reason byte for example as follows:

03 12 34 00 01 => 83 02

Request read 1 register at index 0x1234 response exception type 2 - 'illegal data address'.

Here are some of the most common exception codes:

01 ILLEGAL FUNCTION

The function code received in the query is not an allowable action for the slave. This may be because the function code is only applicable to newer controllers, and was not implemented in the unit selected. It could also indicate that the slave is in the wrong state to process a request of this type, for example because it is unconfigured and is being asked to return register values.

02 ILLEGAL DATA ADDRESS

The data address received in the query is not an allowable address for the slave. More specifically, the combination of reference number and transfer length is invalid. For a controller with 100 registers, a request with offset 96 and length 4 would succeed, a request with offset 96 and length 5 will generate exception 02.

03 ILLEGAL DATA VALUE

A value contained in the query data field is not an allowable value for the slave. This indicates a fault in the structure of the remainder of a complex request, such as that the implied length is incorrect. It specifically does NOT mean that a data item submitted for storage in a register has a value outside the expectation of the application program, since the MODBUS protocol is unaware of the significance of any particular value of any particular register.

04 ILLEGAL RESPONSE LENGTH

Indicates that the request as framed would generate a response whose size exceeds the available MODBUS data size. Used only by functions generating a multi-part response, such as functions 20 and 21 (not used by oModbusMaster).