Modbus Master Connector

Devices compatible with Modbus protocol.

- Name: **Modbus.**
- Version: 1.11.0.12.
- Protocol: MODBUS RTU, ASCII and TCP.
- Interface: TCP/IP or Serial.
- Runtime: Multiplatform
- Configuration:
    - Devices / Protocols

Master Slave Nomenclature
Connector Configuration
Supported Operands
Channels Configuration
Nodes Configuration
Points Configuration
Device Identification (Function 43)
AccessType Configuration
Troubleshoot
Driver Revision History

Overview

Modbus driver implements communication with PLC and IO devices compatible with Modbus Open Standard protocol. It operates as a Client on TCP /IP or serial networks. The communication blocks are dynamically created according to the pooling cycle defined on the AccessType for each Device Point.

The following drivers are related with the Modbus Protocol:

- Modbus Master Protocol. (this driver)
- Modbus Slave Protocol
- Schneider Electric Control Expert Unity Pro PLCs. Relies on Modbus protocol, but it was its own Import Wizards.

Master Slave Nomenclature

The terminology Master and Slave refers to the original Modbus terms. The Modbus Organization announced the replacement of "master-slave" terminology with "client-server" in its communications protocol. The new nomenclature reflects the specific roles of the devices within the Modbus network. Here's the definitions:

1. **Modbus Master (Client):** In the context of the Modbus protocol, the "master" is the device that initiates communications. It's called a Client because it requests data from other devices. The master actively controls when and how data is requested and collected in the network.
2. **Modbus Slave (Server)**: Slave devices are referred to as Servers because they serve data to the master upon request. They passively wait for requests and respond by sending the requested data or performing actions based on the commands received. They do not initiate communication; they only respond to the master's requests.

In the product User Interface, as will in the documentation URLs, we opted to keep the original historical nomenclature, in order to ensure compatibility, as well to avoid potential confusion with the terms Server and Client which already have other meanings in the context of the software platform.

Connector Configuration

At Designer, solution configuration tool, add the connector at:

- Devices / Protocols

The following options are also available to this connector:

- Solution / TagImport Wizards
- UNS Tags / paste rows from clipboard

This document has only information about the device connection settings and its address syntax.

For further Information on its configuration, see:

- Devices Module
- Devices Module Reference

The configuration for this protocol can be created using The Tag Import CSV tools, or copy-paste directly to UNS Tags table.

Communication Driver Information

Driver name	Modbus
Assembly Name	T.ProtocolDriver.Modbus
Assembly Version	1.11.0.12
Available for Linux	True
Devices supported	Any PLC compatible with Modbus
Manufacturer	N/A (Not Applicable)
Protocol	MODBUS RTU, ASCII and TCP
Interface	TCP/IP and Serial
Protocol Options	Message Format (ASCII, RTU or RTU TCP), use block write command (0x0F or 0x10) or single write command (0x05 or 0x06) for single writings
Multi-threading	User configurable. Default is 5 threads to each network node
Max number of nodes	User defined
PC Requirements	Ethernet port

Supported Operands

Operand	Read	Write	Data Type	Address size	Equivalent Operand
0 – Coils	**Yes**	**Yes**	Bit	1 bit	N/A
1 – Input Status	**Yes**	-	Bit	1 bit	N/A
3 – Input Registers	**Yes**	-	Word	2 bytes	N/A
4 – Holding Registers	**Yes**	**Yes**	Word	2 bytes	N/A
%Q – Coils	**Yes**	**Yes**	Bit	1 bit	0
%M – Coils	**Yes**	**Yes**	Bit	1 bit	0
%I – Input Status	**Yes**	-	Bit	1 bit	1
%IW– Input Registers	**Yes**	-	Word	2 bytes	3
%MW – Holding Registers	**Yes**	**Yes**	Word	2 bytes	4
%QW – Holding Registers	**Yes**	**Yes**	Word	2 bytes	4
ID – Device Identification	**Yes**	-	Text	1 byte (Object Id)	N/A

All operands with a percentage symbol (%) mentioned above are used for specific cases. In general, Modbus only uses operands from 0 to 4.

Channels Configuration

Protocol Options

- **BlockSize:** Defines the maximum amount of items per group, the default value is **250**.

If the communication points are configured in sequence and the BlockSize equals to **250**, the driver can create the internal groups with **125** Registers or **2000** Coils.

- **Encoding**: Determines how the information will be packed into the message fields and decoded. The options are:
  - **RTU**: Remote Terminal Unit mode, where each 8–bit byte in a message contains two 4–bit hexadecimal characters
  - **ASCII**: The message is encoded in ASCII mode, where each 8–bit byte in a message is sent as two ASCII characters
  - **RTU TCP**: The default transmission mode when the message is carried on a MODBUS TCP/IP network. It contains information to allow the recipient to recognize message boundaries even if the message has been split into multiple packets

- **SingleWrite:** Indicates the driver behavior for the writings with only one item:
  - **Use block write** : The driver uses the 0x0F command for Coils, or the 0x10 (Function Code 16) command for Holding Registers

    - **Use single write:** The driver uses the 0x05 command for Coils, or the 0x06 (Function Code 6) command for Holding Registers

- **Offset address:** Indicates the driver is zero based address.

Settings

Serial and MultiSerial channels:

- Default configuration for ASCII mode :
    - **DataBits**: 7
    - **StopBits**: 1 if parity is used; 2 if no parity is used

- Default configuration for RTU mode :
    - **DataBits**: 8
    - **StopBits**: 1 if parity is used; 2 if no parity is used

Set the other fields according to your Serial or MultiSerial port configuration.

TCP/IP channels:

- **NodeConnections**: Defines the maximum number of parallel requests that will be sent to each node (asynchronous communication)

Nodes Configuration

Station Configuration

- **SlaveId**: Set this field with the address of the slave device in the Modbus Network. They can be addressed from 1 to 247 for serial nodes, or 0 to 255 for TCP/IP nodes. The address 0 is used for the broadcast.

Serial channels:

- Station syntax: ***<SlaveId>***

E.g.: ***1***

MultiSerial channels:

- Station syntax: ***<Com Port> ; <SlaveId>***

Where:

<Com Port>: the serial port number

- Ex: ***com1 ; 1***

TCP/IP channels:

- Station syntax: ***<IP address> ; <Port number> ; <SlaveId>***

Where :

- ***<IP address>***: IP address of the slave device in the modbus network
- ***<Port number>***: TCP port where the slave device is listening (default is **502**)

E.g.: ***192.168.1.101 ; 502 ; 1***

Online Changes

From FrameworX 10.1.6, an online configuration change to a Modbus Master node or its points rebuilds only the affected node. The other nodes on the channel keep communicating without missing a poll.

- Adding, removing or editing a point rebuilds the groups of the node holding the point.

- Moving a point to another node rebuilds both nodes.
- Adding a node starts it, and removing a node stops it.

These changes still restart the whole channel: any change to the channel or to the AccessType table, renaming a node or moving it to another channel, and server-mode, unsolicited or shared-connection channels. The restart runs on its own when you save the change with online configuration on. In 10.1.5, every online change to a node or point restarted the whole channel. See Runtime How-to.

Points Configuration

The syntax for the Modbus communication points is: ***<Operand><Address>***

Where:

- ***<Operand>***: indicates the memory area
- ***<Address>***: indicates the data address in the memory area, from **1** to **65535**

Valid values for *<Operand>*
0 – Coils
1 – Input Status
3 – Input Registers
4 – Holding Registers
%Q – Coils
%M– Coils
%I – Input Status
%IW– Input Registers
%MW – Holding Registers
%QW – Holding Registers
ID – Device Identification (see below)

E.g.: ***400001*** (Operand = **Holding Register**, Address = ***1***)

Configure ASCII Communication:

- To use the ASCII communication with Modbus, it is necessary to modify the column **Modifiers**. In **StringLength** you need to put an even value number.
- Create a new AccessType to use only with ASCII type points.

Bit Modifier

To read or write one bit of a holding register, set the Bit modifier on the Point and use DataType Word. The driver addresses the bit correctly only with DataType Word.

Changed in FrameworX 10.1.6

From FrameworX 10.1.6, the driver refuses these Points when it prepares the connection:

- A Point whose Bit modifier does not fit its DataType.
- A Point with an address above the 16-bit register range, 465537 to 499999, on a channel with **Offset address** cleared (the default).
- A Point at address 0 on a channel with **Offset address** cleared.

A refused Point is not added to the connection, and the Trace Window reports it by address. The other Points on the channel keep communicating.

In 10.1.5, the driver accepted these Points and returned wrong data. With DataType Byte, a bit index above 7 was ignored on read, and a write replaced the whole register with the single bit. With DataType Bit, the values did not follow the register. After the upgrade, check the Trace Window for refused Points and set bit Points to DataType Word.

Device Identification (Function 43)

Devices that implement MODBUS function 43 with MEI type 14 (Read Device Identification) publish a set of standard text objects describing themselves: vendor name, product code, revision and, optionally, model and application names. The connector reads these objects as read-only text Points, so a solution can identify equipment automatically instead of depending on vendor specific holding registers mapped by hand for each device model.

Addressing

Use the **ID** operand. The address is the MODBUS **Object Id** itself, 0-based, written with the same 5-digit padding as any other Modbus address. The object ids below are defined by the MODBUS specification and are the same on every device that implements the function.

Address	Object Id	Object	Support
ID00000	0x00	VendorName	Mandatory
ID00001	0x01	ProductCode	Mandatory
ID00002	0x02	MajorMinorRevision	Mandatory
ID00003	0x03	VendorUrl	Optional
ID00004	0x04	ProductName	Optional
ID00005	0x05	ModelName	Optional
ID00006	0x06	UserApplicationName	Optional
ID00007 to ID00127	0x07 to 0x7F	Reserved by the MODBUS specification	-
ID00128 to ID00255	0x80 to 0xFF	Vendor private objects, meaning defined by the manufacturer	Optional

E.g.: ***ID00001*** reads the device product code.

Configuring the Points

- Identification Points are **read-only**. Write commands to them are rejected.
- Bind them to **Text tags**. The values are text; a numeric tag displays 0 with no error.
- The **Offset address** option in the channel Protocol Options does not apply to these Points. Object ids are always 0-based, whether or not that option is enabled for register addressing.
- The length of each value comes from the device itself, so the **StringLength** modifier used for ASCII register Points is not needed here.
- All identification Points of a node are served by a **single request per poll**, whatever their object ids, so adding more of them costs no extra communication.
- Identification data is static. Assign these Points to a slow AccessType rather than polling them at process data rate.

When identification data is not available

Condition	Result
The device does not implement function 43	All identification Points read empty with Bad quality. A warning is written to the Trace naming the exception returned by the device. Communication is not affected: the exchange completes normally and the channel stays connected.
The device implements function 43 but does not publish one of the objects	That Point reads empty with Bad quality. Every other identification Point keeps its value and Good quality.
The device is unreachable	Normal communication failure handling applies, the same as for any register Point.



Function 43 is optional in the MODBUS specification. A device that does not implement it answers with an exception, which the connector reports without disturbing the rest of the channel: register Points on the same node keep reading normally.

If a device splits its identification data across several responses, this version reads the first response only. Objects that the device would return in a following response read empty with Bad quality. The three mandatory objects (VendorName, ProductCode, MajorMinorRevision) are always in the first response.

AccessType Configuration

To perform block writes (as done with function 16), you should use **WriteTrigger-based writing** so that all addresses are sent in the same block (i.e., within the same network packet). To do this, create a separate **Devices / AccessType** with the **"WriteEvent"** column disabled, and add a **tag** in the **WriteTrigger** column to trigger the message.

Troubleshoot

These codes come from the slave device in a Modbus exception response. Connection, transport and driver level errors are reported separately as negative codes: see Devices Monitor Reference for the full negative code table.

Modbus protocol exception codes returned by the slave device		
Error	**Name**	**Description**
1	ILLEGAL FUNCTION	The function code received in the query is not allowable.
2	ILLEGAL DATA ADDRESS	The data address received in the query is not allowable.
3	ILLEGAL DATA VALUE	A value contained in the query data field is not allowable.
4	SLAVE DEVICE FAILURE	Error while attempting to perform the requested action.
5	ACKNOWLEDGE	Request accepted, but a long duration of time will be required.
6	SLAVE DEVICE BUSY	The slave is engaged in a long–duration program command.
7	NEGATIVE ACKNOWLEDGE	Cannot perform the program function received in the query.
8	MEMORY PARITY ERROR	Parity error in the extended memory.

Driver Revision History

Modbus Revision History	
Version	**Notes**
1.11.0.12	Writing a text (ASCII) point now sends exactly two data bytes per register, and the byte count matches the number of registers written. Before, the request carried extra zero bytes, and devices that check the request rejected the write with exception 03 (Illegal Data Value).
1.11.0.11	A write to a point with a bit selection first reads the register. When the device refuses that read, the write now reports the device's exception code instead of an error with code 0.
1.11.0.10	The node Test in Designer now tells a device that answered with an exception apart from a failed connection, and names the answer. In RTU mode, an exception reply from the device is no longer read as a byte count.
1.11.0.9	Points whose bit selection does not fit the point's data type, or whose register address is out of range, are now rejected at start with Bad quality and a Trace message; before, such a point read 0 silently and a write could erase the whole register; online changes to a node or its points now restart only that node instead of the whole channel.
1.11.0.1	Initial release on new documentation standards.

In this section...

In this section...