# 2 What You should know about Modbus

## 2.1 Some Background

The Modbus protocol family was originally developed by Schneider Automation Inc. as an industrial network for their Modicon programmable controllers.

Since then the Modbus protocol family has been established as vendor-neutral and open communication protocols, suitable for supervision and control of automation equipment.

## 2.2 Technical Information

Modbus is a master/slave protocol with half-duplex transmission.

One master and up to 247 slave devices can exist per network.

The protocol defines framing and message transfer as well as data and control functions.

The protocol does not define a physical network layer. Modbus works on different physical network layers. The ASCII and RTU protocol operate on RS-232, RS-422 and RS-485 physical networks. The Modbus/TCP protocol operates on all physical network layers supporting TCP/IP. This compromises 10BASE-T and 100BASE-T LANs as well as serial PPP and SLIP network layers.

**Note:**

To utilise the multi-drop feature of Modbus, you need a multi-point network like RS-485. In order to access a RS-485 network, you will need a protocol converter which automatically switches between sending and transmitting operation. However some industrial hardware platforms have an embedded RS-485 line driver and support enabling and disabling of the RS-485 transmitter via the RTS signal. FieldTalk supports this RTS driven RS-485 mode.

### 2.2.1 The Protocol Functions

Modbus defines a set of data and control functions to perform data transfer, slave diagnostic and PLC program download.

FieldTalk implements the most commonly used functions for data transfer as well as some diagnostic functions. The functions to perform PLC program download and other device specific functions are outside the scope of FieldTalk.

All Bit Access and 16 Bits Access Modbus Function Codes have been implemented. In addition the most frequently used Diagnostics Function Codes have been implemented. This rich function set enables a user to solve nearly every Modbus data transfer problem.

The following table lists the available Modbus Function Codes in this library:

| Function Code | Current Terminology | Classic Terminology |
|---|---|---|
| Bit Access | | |
| 1 | Read Coils | Read Coil Status |
| 2 | Read Discrete Inputs | Read Input Status |
| 5 | Write Single Coil | Force Single Coil |
| 15 (0F hex) | Write Multiple Coils | Force Multiple Coils |
| 16 Bits Access | | |
| 3 | Read Multiple Registers | Read Holding Registers |
| 4 | Read Input Registers | Read Input Registers |
| 6 | Write Single Register | Preset Single Register |
| 16 (10 Hex) | Write Multiple Registers | Preset Multiple Registers |
| 22 (16 hex) | Mask Write Register | Mask Write 4X Register |
| 23 (17 hex) | Read/Write Multiple Registers | Read/Write 4X Registers |
| Diagnostics | | |
| 7 | Read Exception Status | Read Exception Status |
| 8 subcode 00 | Diagnostics - Return Query Data | Diagnostics - Return Query Data |
| 8 subcode 01 | Diagnostics - Restart Communications Option | Diagnostics - Restart Communications Option |
| Vendor Specific | | |
| Advantech | Send/Receive ADAM 5000/6000 ASCII commands | |

### 2.2.2 How Slave Devices are identified

A slave device is identified with its unique address identifier. Valid address identifiers supported are 1 to 247. Some library functions also extend the slave ID to 255, please check the individual function's documentation.

Some Modbus functions support broadcasting. With functions supporting broadcasting, a master can send broadcasts to all slave devices of a network by using address identifier 0. Broadcasts are unconfirmed, there is no guarantee of message delivery. Therefore broadcasts should only be used for uncritical data like time synchronisation.

### 2.2.3 The Register Model and Data Tables

The Modbus data functions are based on a register model. A register is the smallest addressable entity with Modbus.

The register model is based on a series of tables which have distinguishing characteristics. The four tables are:

| Table | Classic Terminology | Modicon Register Table | Characteristics |
|---|---|---|---|
| Discrete outputs | Coils | 0:00000 | Single bit, alterable by an application program, read-write |
| Discrete inputs | Inputs | 1:00000 | Single bit, provided by an I/O system, read-only |
| Input registers | Input registers | 3:00000 | 16-bit quantity, provided by an I/O system, read-only |
| Output registers | Holding registers | 4:00000 | 16-bit quantity, alterable by an application program, read-write |

The Modbus protocol defines these areas very loose. The distinction between inputs and outputs and bit-addressable and register-addressable data items does not imply any slave specific behaviour. It is very common that slave devices implement all tables as overlapping memory area.

For each of those tables, the protocol allows a maximum of 65536 data items to be accessed. It is slave dependant, which data items are accessible by a master. Typically a slave implements only a small memory area, for example of 1024 bytes, to be accessed.

### 2.2.4 Data Encoding

Classic Modbus defines only two elementary data types. The discrete type and the register type. A discrete type represents a bit value and is typically used to address output coils and digital inputs of a PLC. A register type represents a 16-bit integer value. Some manufacturers offer a special protocol flavour with the option of a single register representing one 32-bit value.

All Modbus data function are based on the two elementary data types. These elementary data types are transferred in big-endian byte order.

Based on the elementary 16-bit register, any bulk information of any type can be exchanged as long as that information can be represented as a contiguous block of 16-bit registers. The protocol itself does not specify how 32-bit data and bulk data like strings is structured. Data representation depends on the slave's implementation and varies from device to device.

It is very common to transfer 32-bit float values and 32-bit integer values as pairs of two consecutive 16-bit registers in little-endian word order. However some manufacturers like Daniel and Enron implement an enhanced flavour of Modbus which supports 32-bit wide register transfers. FielTalk supports Daniel/Enron 32-bit wide register transfers.

The FieldTalk Modbus Master Library defines functions for the most common tasks like:

- Reading and Writing bit values
- Reading and Writing 16-bit integers

- Reading and Writing 32-bit integers as two consective registers
- Reading and Writing 32-bit floats as two consective registers
- Reading and Writing 32-bit integers using Daniel/Enron single register transfers
- Reading and Writing 32-bit floats using Daniel/Enron single register transfers
- Configuring the word order and representation for 32-bit values

### 2.2.5 Register and Discrete Numbering Scheme

Modicon PLC registers and discretes are addressed by a memory type and a register number or a discrete number, e.g. 4:00001 would be the first reference of the output registers.

The type offset which selects the Modicon register table must not be passed to the FieldTalk functions. The register table is selected by choosing the corresponding function call as the following table illustrates.

| Master Function Call | Modicon Register Table |
|---|---|
| readCoils(), writeCoil(), forceMultipleCoils() | 0:00000 |
| readInputDiscretes | 1:00000 |
| readInputRegisters() | 3:00000 |
| writeMultipleRegisters(), readMultipleRegisters(), writeSingleRegister(), maskWriteRegister(), readWriteRegisters() | 4:00000 |

Modbus registers are numbered starting from 1. This is different to the conventional programming logic where the first reference is addressed by 0.

Modbus discretes are numbered starting from 1 which addresses the most significant bit in a 16-bit word. This is very different to the conventional programming logic where the first reference is addressed by 0 and the least significant bit is bit 0.

The following table shows the correlation between Discrete Numbers and Bit Numbers:

| Modbus Discrete Number | Bit Number | Modbus Discrete Number | Bit Number |
|---|---|---|---|
| 1 | 15 (hex 0x8000) | 9 | 7 (hex 0x0080) |
| 2 | 14 (hex 0x4000) | 10 | 6 (hex 0x0040) |
| 3 | 13 (hex 0x2000) | 11 | 5 (hex 0x0020) |
| 4 | 12 (hex 0x1000) | 12 | 4 (hex 0x0010) |
| 5 | 11 (hex 0x0800) | 13 | 3 (hex 0x0008) |
| 6 | 10 (hex 0x0400) | 14 | 2 (hex 0x0004) |
| 7 | 9 (hex 0x0200) | 15 | 1 (hex 0x0002) |
| 8 | 8 (hex 0x0100) | 16 | 0 (hex 0x0001) |

When exchanging register number and discrete number parameters with FieldTalk functions and methdos you have to use the Modbus register and discrete numbering scheme. (Internally the functions will deduct 1 from the start register value before transmitting the value to the slave device.)

### 2.2.6 The ASCII Protocol

The ASCII protocol uses an hexadecimal ASCII encoding of data and a 8 bit checksum. The message frames are delimited with a ':' character at the beginning and a carriage return/linefeed sequence at the end.

The ASCII messaging is less efficient and less secure than the RTU messaging and therefore it should only be used to talk to devices which don't support RTU. Another application of the ASCII protocol are communication networks where the RTU messaging is not applicable because characters cannot be transmitted as a continuos stream to the slave device.

The ASCII messaging is state-less. There is no need to open or close connections to a particular slave device or special error recovery procedures.

A transmission failure is indicated by not receiving a reply from the slave. In case of a transmission failure, a master simply repeats the message. A slave which detects a transmission failure will discard the message without sending a reply to the master.

### 2.2.7 The RTU Protocol

The RTU protocol uses binary encoding of data and a 16 bit CRC check for detection of transmission errors. The message frames are delimited by a silent interval of at least 3.5 character transmission times before and after the transmission of the message.

When using RTU protocol it is very important that messages are sent as continuous character stream without gaps. If there is a gap of more than 3.5 character times while receiving the message, a slave device will interpret this as end of frame and discard the bytes received.

The RTU messaging is state-less. There is no need to open or close connections to a particular slave device or special error recovery procedures.

A transmission failure is indicated by not receiving a reply from the slave. In case of a transmission failure, a master simply repeats the message. A slave which detects a transmission failure will discard the message without sending a reply to the master.

### 2.2.8 The MODBUS/TCP Protocol

MODBUS/TCP is a TCP/IP based variant of the Modbus RTU protocol. It covers the use of Modbus messaging in an 'Intranet' or 'Internet' environment.

The MODBUS/TCP protocol uses binary encoding of data and TCP/IP's error detection mechanism for detection of transmission errors.

In contrast to the ASCII and RTU protocols MODBUS/TCP is a connection oriented protocol. It allows concurrent connections to the same slave as well as concurrent connections to multiple slave devices.

In case of a TCP/IP time-out or a protocol failure, a master shall close and re-open the connection and then repeat the message.