

Tutorial 4 MODBUS Configuration

Contributed by Pavel
 Saturday, 09 January 2010
 Last Updated Friday, 15 January 2010

MODBUS Plugin Configuration and test

MODBUS Plugin Configuration and test Introduction MODBUS is an application layer messaging protocol, positioned at level 7 of the OSI model, that provides client/server communication between devices connected on different types of buses or networks. It is currently implemented using:

- TCP/IP over Ethernet at a reserved system port 502 on the TCP/IP stack
- Asynchronous serial transmission

MODBUS is a request/reply protocol and offers services specified by function codes. MODBUS function codes are elements of MODBUS request/reply PDUs. The MODBUS protocol defines a simple protocol data unit (PDU) independent of the underlying communication layers. The mapping of MODBUS protocol on specific buses or network can introduce some additional fields on the application data unit (ADU): The MODBUS application data unit is built by the client that initiates a MODBUS transaction. The function indicates to the server what kind of action to perform. The MODBUS application protocol establishes the format of a request initiated by a client. The function code field of a MODBUS data unit is coded in one byte. Valid codes are in the range of 1 ... 255 decimal (the range 128 – 255 is reserved and used for exception responses). When a message is sent from a Client to a Server device the function code field tells the server what kind of action to perform. Function code "0" is not valid. Sub-function codes are added to some function codes to define multiple actions. The data field of messages sent from a client to server devices contains additional information that the server uses to take the action defined by the function code. This can include items like discrete and register addresses, the quantity of items to be handled, and the count of actual data bytes in the field. The data field may be nonexistent (of zero length) in certain kinds of requests, in this case the server does not require any additional information. The function code alone specifies the action.

Error-Free Transaction: Erroneous Transaction (Client exists): The size of the MODBUS PDU is limited by the size constraint inherited from the first MODBUS implementation on Serial Line network (max. RS485 ADU = 256 bytes). Therefore: MODBUS PDU for serial line communication = 256 - Server address (1 byte) - CRC (2 bytes) = 253 bytes. Consequently: RS232 / RS485 ADU = 253 bytes + Server address (1 byte) + CRC (2 bytes) = 256 bytes. TCP MODBUS ADU = 253 bytes + MBAP (7 bytes) = 260 bytes. MODBUS Data model MODBUS bases its data model on a series of tables that have distinguishing characteristics. The four primary tables are: MODBUS Addressing model, The MODBUS application protocol defines precisely PDU addressing rules. In a MODBUS PDU each data is addressed from 0 to 65535. It also defines clearly a MODBUS data model composed of 4 blocks that comprises several elements numbered from 1 to n. In the MODBUS data Model each element within a data block is numbered from 1 to n.

MODBUS Addressing model: Modbus Functional codes: MODBUS Plugin settings In Designer, click Communication Properties: In this dialog, click Add Station (for adding a communication partner for FS2): Choose station type according your device. For testing with mod_RSsim.exe, choose TCP Master: For testing, the default values are OK. Otherwise set your favourite values for:

Error-free communication: Station Name: identification string for this communication partner
 Slave IP Address / Hostname: IP address or hostname of this communication partner
 Slave TCP Port: port number. MODBUS/TCP standard is 502 (IETF)
 Pause Between cycles [ms]: pause between performing all queries and starting a new cycle
 Error handling: Pause before retry [ms]: if slave does not respond, we retry query after this time
 Number of retries: after this number of unsuccessful retries we give up and mark the station as faulty and its channels as to have Bad quality
 Number of cycles before retrying communication to a failed device: we must time to time try if the slave is back again. Timeout = Pause Between cycles * this parameter
 Logging level: verbosity of log messages

Click OK and then Add Variable: You can modify: Channel Name: identification string for this channel
 FS2 Channel type: FS2 internal channel type, can be Boolean, String, Int32 or Double. In the case of Double, you can define a linear function for input value recalculation (see below)
 Station Name: Name of station which this channel belongs to
 MOD Device index: for serial MODBUS/RTU – identification of slave. For MODBUS/TCP has no sense and should be 0.
 MOD Register type: choice of MODBUS Data Model table to which the data belong. Can be: Holding Register 16-bit word read/write
 Input Register 16-bit word read only
 Coil Single bit read/write
 Input Single bit read only
 Device Failure Info Special internal FS2 type with information about this device failure
 MOD Data type: interpretation of data read from MODBUS. Can be: Bool Interprets data as boolean. Single bit types are native, for word types can be the bit offset (Index) adjusted (see below)
 Int Interprets data as 16-bit signed Integer
 UInt Interprets data as 16-bit unsigned Integer
 DInt Interprets data as 32-bit signed Integer (two consecutive 16-bit registers)
 DUInt Interprets data as 32-bit unsigned Integer (two consecutive 16-bit registers)
 Float Interprets data as IEEE single precision float (two consecutive 16-bit registers)
 String Interprets data as ASCII string, and translates it into UNICODE .NET string
 MOD Address Offset: Modbus register address, in the PDU addressing format eg starting from 0 to 65535. Depending on MOD Register type it means the register address see below
 MOD Register: Read only informative field with calculated register address. The IO ranges in MODICON PLC and other devices are: 00001 - 09999 Coils / Discrete Outputs
 10001 - 19999 Inputs (Discrete, Read only)
 30001 - 39999 Input registers/analog/counters (Read only)
 40001 - 49999 Holding registers / analog output
 MOD Data Length: Number of registers (coils) to be read.
 MOD Swap type: Due to Big/Little Endian problem, data have to be sometimes interpreted with swapped bytes and/or words: SwapNone: No swapping
 SwapBytes: Swap bytes

inside words only
SwapWords: Swap words only, not the bytes inside words
SwapAll: Swap both words and the bytes inside words
This strongly depends on the Slave device definitions.
MOD Read/Write: Direction of the channel:
ReadOnly: Data will be cyclically read from the device
WriteOnly: Data will be written to the device after a change
ReadAndWrite: Data will be both read and written
MOD Bit Index: Offset of a bit in word for Int->Bool conversion (0..15)
K constant: Anything->Double conversion: linear equation can be used $y=kx+d$
D constant: Anything->Double conversion: linear equation can be used $y=kx+d$
For testing one variable, the default values are OK. Otherwise you must set minimally the address offset to an other value.
Click OK, save the sample project. Now you can run the test MODBUS program mod_RSsim.exe. You can find it in the ...\\FreeSCADA2\3rdParty\NModbus directory. Set the protocol to MODBUS TCP/IP, and I/O view to Holding Registers. Now you can run values simulation (icon) or set register value by hand (doubleclick).
Start communication for example from Designer using Project->Variables Dialog. Press Autorefresh. You will see the same number in MODBUS and FS2. You can also experiment with various data types. That is all for now... Pavel.