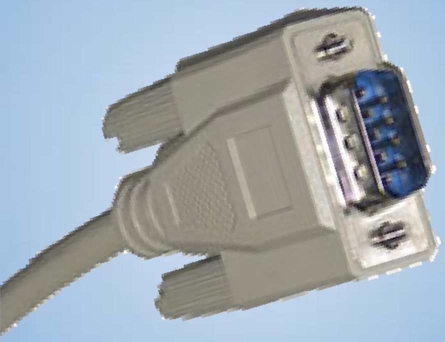
REAL TIME AUTOMATION



MODBUS RTU Unplugged

An Introduction to MODBUS RTU Networking

PREPARED BY REAL TIME AUTOMATION

WWW.RTAAUTOMATION.COM

MODBUS RTU Unplugged

An Introduction to MODBUS RTU Networking

Introduction

Modbus RTU is an open, serial (RS-232 or RS-485) protocol derived from the client/server architecture. It is a widely accepted protocol due to its ease of use and reliability. Modbus RTU is widely used within Building Management Systems (BMS) and Industrial Automation Systems (IAS). This wide acceptance is due in large part to MODBUS RTU's ease of use.

MODBUS RTU messages are simple 16-bit CRC (Cyclic-Redundant Checksum). The simplicity of these messages is to ensure reliability. Due to this simplicity, the basic 16-bit MODBUS RTU register structure can be used to pack in floating point, tables, ASCII text, queues, and other unrelated data. These drawbacks can be circumvented by using user-configurable units called ProtoCessor modules and FieldServers to make adjustments.

Rocco uses MODBUS RTU in his buildings.

Protocol of MODBUS

MODBUS is considered an application layer messaging protocol, providing client/server communication between devices connected together through buses or networks. On the OSI model, MODBUS is positioned at level 7. MODBUS is intended to be a request/reply protocol and delivers services specified by function codes. The function codes of MODBUS are elements of MODBUS's request/reply PDUs (Protocol Data Unit).

In order to build the MODBUS application data unit, the client must initiate a MODBUS transaction, the function that informs the server of which action to perform. The format of a request initiated by a client is established by the MODBUS application protocol. The function code field is coded into one byte. Only codes between 1 and 255 are considered valid, with 128-255 being reserved for exception responses. When the client sends a message to the server, the function code field informs the server what type of action to perform.

To define multiple actions, some functions have sub-function codes added to them. For instance, the client server is able to read the ON/OFF states of a group of discreet outputs or inputs. It can also read or write the data contents of a group of MODBUS registers. When the client receives the server response, the server uses the function code field to indicate either an error-free response or an exception response. The server echoes the request of the initial function code in the case of a normal response.

Drawbacks of MODBUS RTU - Reporting by Exception

There are some limitations to MODBUS RTU. For instance, the client/server architecture requires that in order for data to reach the client, the client node must poll the server node. Another limitation is that MODBUS RTU can be multi-dropped, meaning numerous server devices can operate on one MODBUS RTU network, so long as the physical layer is configured for this. To allow for the multiple slave network, an RS-485 device, or simple modems or radios, can be used. With this configuration, it is possible that numerous nodes are multi-dropped, even at lower baud rates. Seconds, or even minutes can pass before changes can be made in the field nodes. This is due to the slow speed and high quantity of MODBUS nodes involved.

There have been attempts to bypass these limitations, but as soon as "unsolicited message" or "report by exception" enhancements are made, the protocol can no longer be considered MODBUS RTU. If faster data transfer is an absolute requirement, or support of "report by exception" is a necessity, a different protocol should be sought, possibly EtherNet/IP or BACnet.

Data Object Properties

MODBUS RTU packets are only intended to send data; they do not have the capability to send parameters such as point name, resolution, units, etc. If the ability to send such parameters is needed, one should investigate BACnet, EtherNet/IP, or other modern protocols.

MODBUS RTU versus Other Protocols

Despite the limitations of MODBUS RTU, there are still many good reasons it is still a contender among other industrial automation protocols. For one, MODBUS RTU is much easier to implement than newer protocols, and it is a dominant force in the market place. MODBUS RTU also requires significantly less memory than most other solutions. To implement MODBUS RTU, you can fit the necessary 2Kb on a small 8-bit CPU or PIC processor, whereas with BACnet and EtherNet/IP, you may require 30-100Kb of memory.

MODBUS RTU Address Requirements

Standard MODBUS RTU node addresses are 1 to 254, with 0 being reserved for broadcast messages and write only. However, the 0 address is rarely used due to the fact that there is no confirmation that the message was properly received at the server node. This doesn’t have much affect if your physical layer is RS-232 as only one node can be implemented anyway. RS-485 limits the number of nodes to 32, though some drivers will allow you to extend the amount.

Table 1 — RTU Memory Map

MODBUS RTU Date Type	Common Name	Starting Address
Modbus Coils	Bits, Binary Values, Flags	00001
Digital Inputs	Binary Inputs	10001
Analog Inputs	Binary Inputs	30001
Modbus Registers	Analog Values, Variables	40001

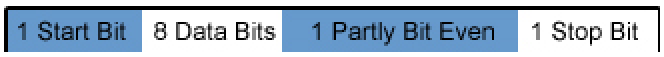
The difference between MODBUS RTU and MODBUS TCP

The most basic difference between MODBUS RTU and MODBUS TCP ,also known as MODBUS IP, MODBUS EtherNet, and MODBUS TCP/IP, is that MODBUS TCP uses a 6 byte header to allow routing.

Bit Structure in the Byte

The bit of least importance is sent and received first. All devices within the network must interpret each transmitted byte analogously in this manner. There are no methods for automated recognition of baud rates and the same baud rate must be utilized by the Server and all clients connected to the bus. MODBUS does not specify and certain baud rate, but typical baud rates are 9600 or 19200.

Figure 1 — A Transmitted Byte



A transmitted Byte is coded as an 8 Bit binary value, hexadecimal 0 - 9 and A - F. The least significant Bit is sent and received first.

The difference between MODBUS RTU and MODBUS/ASCII

There are two basic transmission modes found in serial MODBUS connections, ASCII and RTU. These transmission modes determine the way in which the MODBUS messages are coded. In ASCII format, the messages are readable, whereas in RTU, the messages are in binary coding and cannot be read while monitoring. The trade-off is that RTU messages are a smaller size, which allows for more data exchange in the same time span. One should be aware that all nodes within one MODBUS network must be of the same transmission mode, meaning MODBUS ASCII cannot communicate with MODBUS RTU and vice versa.

In MODBUS/ASCII, messages are encoded with hexadecimal value, represented with comprehensive ASCII characters. The characters used for this encoding are 0-9 and A-F. For every byte of information, two communication-bytes are used because every communication-byte can only define 4 bits in the hexadecimal system. MODBUS RTU, however, exchanges data in binary format where each byte of data is coded in one communication-byte.

The MODBUS messages on a serial connection are not broadcast in plain format. They are constructed in a way that allows receivers to easily detect the beginning and end of a message. The characters start and end a frame when in ASCII mode. To flag the start of a message, a colon is used, and each message is ended with a CR/LF combination.

MODBUS RTU uses a different method. In RTU, framing is constructed by measuring gaps of silence on the communication line. Before each message, there must be a minimum gap of 3.5 characters. To prepare for new messages, the receiver clears the buffer when a gap of 1.5 characters is detected. One of the main differences between MODBUS/ASCII and MODBUS RTU is that ASCII allows gaps between the bytes of a message with a maximum length of 1 second. With MODBUS RTU, continuous streams of messages must be sent.

Table 2 — Properties of MODBUS/ASCII and MODBUS RTU

Property	**MODBUS/ASCII**		**MODBUS RTU**	
Characters	ASCII (0-9, A-F)		Binary (0-255)	
Error Check	LRC (Longitudinal Redundancy Check)		CRC (Cyclic Redundancy Check)	
Frame Start	“:”		3.5 characters of silence	
Frame End	“CR/LF”		3.5 characters of silence	
Gaps In Message	1 second		1.5 x character length	
Start Bit	1		1	
Data Bits	7		8	
Parity	Even/odd	None	Even/odd	None
Stop Bits	1	2	1	2