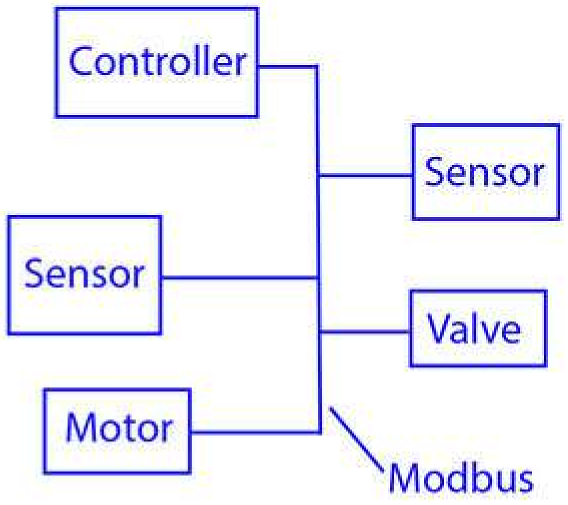
Modbus Overview

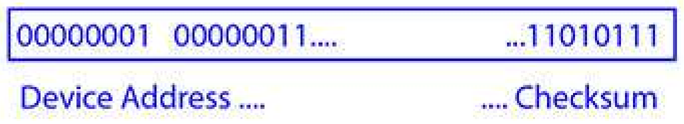
What is Modbus?

Modbus is an industrial protocol standard that was created by Modicon, now Schneider Electric, in the late 1970's for communication among programmable logic controllers (PLCs). Modbus remains the most widely available protocol for connecting industrial devices. The Modbus protocol specification is openly published and use of the protocol is royalty-free.



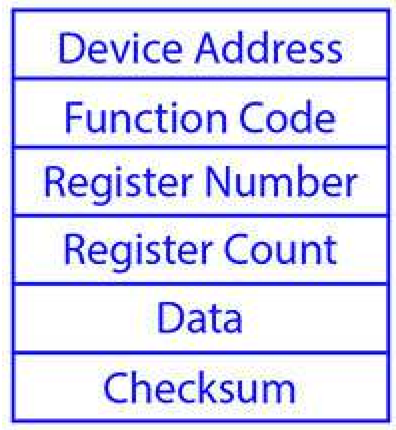
Modbus protocol is defined as a master/slave protocol, meaning a device operating as a master will poll one or more devices operating as a slave. This means a slave device cannot volunteer information; it must wait to be asked for it. The master will write data to a slave device's registers, and read data from a slave device's registers. A register address or register reference is always in the context of the slave's registers.

The most commonly used form of Modbus protocol is RTU over RS-485. Modbus RTU is a relatively simple serial protocol that can be transmitted via traditional UART technology. Data is transmitted in 8-bit bytes, one bit at a time, at baud rates ranging from 1200 bits per second (baud) to 115200 bits per second. The majority of Modbus RTU devices only support speeds up to 38400 bits per second.



A Modbus RTU network has one Master and one or more Slaves. Each slave has a unique 8-bit device address or unit number. Packets sent by the master include the address of the slave the message is intended for. The slave must respond only if its address is recognized, and must respond within a certain time period or the master will call it a "no response" error.

Each exchange of data consists of a request from the master, followed by a response from the slave. Each data packet, whether request or response, begins with the device address or slave address, followed by function code, followed by parameters defining what is being asked for or provided. The exact formats of the request and response are documented in detail in the Modbus protocol specification. The general outline of each request and response is illustrated below.



Modbus data is most often read and written as "registers" which are 16-bit pieces of data. Most often, the register is either a signed or unsigned 16-bit integer. If a 32-bit integer or floating point is required, these values are actually read as a pair of registers. The most commonly used register is called a Holding Register, and these can be read or written. The other possible type is Input Register, which is read-only.

The exceptions to registers being 16 bits are the coil and the discrete input, which are each 1 bit only. Coils can be read or written, while discrete inputs are read-only. Coils are usually associated with relay outputs.

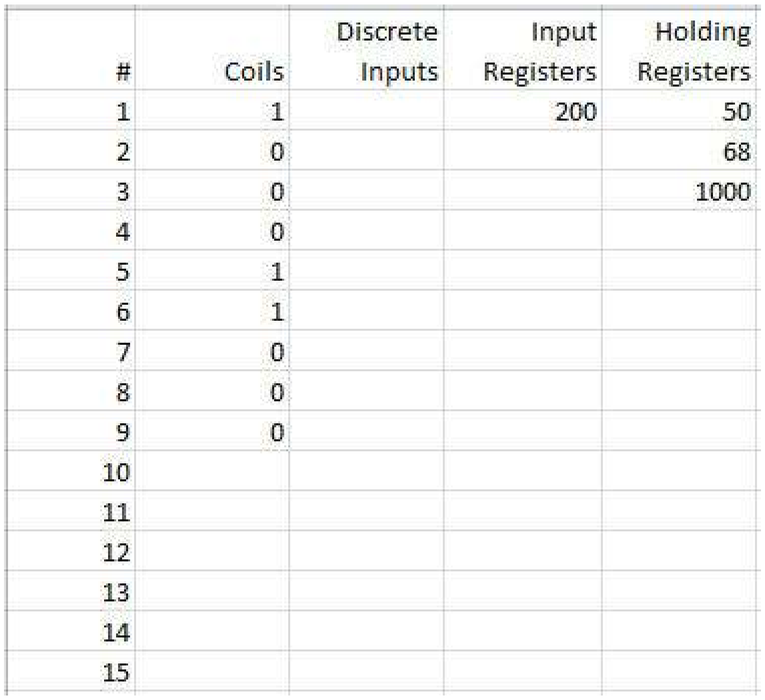
The type of register being addressed by a Modbus request is determined by the function code. The most common codes include 3 for "read holding registers", and may read 1 or more. Function code 6 is used to write a single holding register. Function code 16 is used to write one or more holding registers.

Visualizing Data in the Modbus Device

Modbus slave devices can be visualized as having an internal spread sheet filled with numbers. The Modbus master will ask a slave for its data value or number found in a given row and column, and the slave will respond by sending that piece of data back to the master. Of course, this process can be reversed with the Modbus master telling the slave what number to put into its data table at a given row and column.

The "columns" in a Modbus device's "spread sheet" are more formally known as register types. Register type may be a coil, a discrete input (aka status input), an input register, or a holding register.

The "rows" in a Modbus device's "spread sheet" are simply the register number. Most often, these start at 1 and count up sequentially. Some devices might not have a register 1, and their first register may be number 100 for example. If the register number does not exist in the slave device, it will send back an "oops" message properly known as an exception. The exception provides an error code that says "no such register" (exception code 2, illegal data address).



#	Coils	Discrete Inputs	Input Registers	Holding Registers
1	1		200	50
2	0			68
3	0			1000
4	0			
5	1			
6	1			
7	0			
8	0			
9	0			
10				
11				
12				
13				
14				
15				

What is Modbus TCP?

Modbus TCP encapsulates Modbus RTU request and response data packets in a TCP packet transmitted over standard Ethernet networks. The unit number is still included and its interpretation varies by application – the unit or slave address is not the primary means of addressing in TCP. The address of most importance here is the IP address, e.g. 192.168.1.100. The standard port for Modbus TCP is 502, but port number can often be reassigned if desired.

The checksum field normally found at the end of an RTU packet is omitted from the TCP packet. Checksum and error handling are handled by Ethernet in the case of Modbus TCP.

The TCP version of Modbus follows the OSI Network Reference Model. Modbus TCP defines the presentation and application layers in the OSI model.

Modbus TCP makes the definition of master and slave less obvious because Ethernet allows peer to peer communication. The definition of client and server are better known entities in Ethernet based networking. In this context, the slave becomes the server and the master becomes the client. There can be more than one client obtaining data from a server. In Modbus terms, this means there can be multiple masters as well as multiple slaves. Rather than defining master and slave on a physical device by device basis, it now becomes the system designer's responsibility to create logical associations between master and slave functionality.

What is Modbus ASCII?

Modbus ASCII is an older implementation that contains all of the elements of an RTU packet, but expressed entirely in printable ASCII characters. Modbus ASCII is considered deprecated, is rarely used any more, and is not included in the formal Modbus protocol specification.

Review of Modbus Register Types

The types of registers referenced in Modbus devices include the following:

- Coil (Discrete Output)
- Discrete Input (or Status Input)
- Input Register
- Holding Register

Whether a particular device includes all of these register types is up to the manufacturer. It is very common to find all I/O mapped to holding registers only. Coils are 1-bit registers, are used to control discrete outputs, and may be read or written. Discrete Inputs are 1-bit registers used as inputs, and may only be read. Input registers are 16-bit registers used for input, and may only be read. Holding registers are the most universal 16-bit register, may be read or written, and may be used for a variety of things including inputs, outputs, configuration data, or any requirement for "holding" data.

Control Solutions gateways will support all register types when the gateway is the master. Control Solutions gateways that connect a non-Modbus device to a Modbus network will in some cases use only holding registers to represent the non-Modbus device data.

Most Control Solutions I/O devices use holding registers for all types of inputs and outputs. In most cases, the same I/O is accessible as other register types as well, with the I/O status or value being mirrored in multiple registers.

Review of Modbus Function Codes

Modbus protocol defines several function codes for accessing Modbus registers. There are four different data blocks defined by Modbus, and the addresses or register numbers in each of those overlap. Therefore, a complete definition of where to find a piece of data requires both the address (or register number) and function code (or register type).

The function codes most commonly recognized by Modbus devices are indicated in the table below. This is only a subset of the codes available - several of the codes have special applications that most often do not apply.

Modbus Function Codes Recognized by Control Solutions Gateways:

Function Code	Register Type
1	Read Coil
2	Read Discrete Input
3	Read Holding Registers
4	Read Input Registers
5	Write Single Coil
6	Write Single Holding Register
15	Write Multiple Coils
16	Write Multiple Holding Registers

Review of Modbus Exception (error) Codes

When a Modbus slave recognizes a packet, but determines that there is an error in the request, it will return an exception code reply instead of a data reply. The exception reply consists of the slave address or unit number, a copy of the function code with the high bit set, and an exception code. If the function code was 3, for example, the function code in the exception reply will be 0x83. The exception codes will be one of the following:

1	Illegal Function	The function code received in the query is not recognized by the slave or is not allowed by the slave.
2	Illegal Data Address	The data address (register number) received in the query is not an allowed address for the slave, i.e., the register does not exist. If multiple registers were requested, at least one was not permitted.
3	Illegal Data Value	The value contained in the query's data field is not acceptable to the slave.
4	Slave Device Failure	An unrecoverable error occurred while the slave was attempting to perform the requested action
6	Slave Device Busy	The slave is engaged in processing a long-duration command. The master should try again later.
10	Gateway Path Unavailable	Specialized use in conjunction with gateways, usually means the gateway is misconfigured or overloaded
11	Gateway Target Device Failed to Respond	Specialized use in conjunction with gateways, indicates no response was received from the target device.

Where do I Start for Communicating with My Modbus Device?

Here are the first few things you need to find out:

(1) What is the physical connection?

Modbus RTU uses RS-485 or RS-232. Modbus TCP uses Ethernet. If you are looking for a Control Solutions gateway, you will need to pick the model that matches the electrical interface of the equipment you want to connect. If you are choosing a Control Solutions I/O device, pick one that matches your network.

(2) How are the registers mapped?

When using a gateway to interface a Modbus device to a non-Modbus network, you need to get documentation from the equipment manufacturer that describes the available registers and how to address them. Modbus protocol does not provide a means for registers to automatically identify themselves. Control Solutions cannot determine this information for you. You must consult the equipment manufacturer.

When using Control Solutions I/O devices (such as AddMe III) you will find this information in the online help files that came with the device, or on our web site.

(3) What are the communication parameters?

Modbus RTU requires that you know or define baud rate, character format (8 bits no parity, etc), and slave ID (aka slave address, unit number, unit ID). A mis-match in any of these will result in no communication.

Modbus TCP requires that you know or define IP addresses on the network. In some cases, you also need unit ID's. Control Solutions Modbus TCP devices may use the unit ID, or may ignore it, depending on the device and the application.

Modbus: When 40001 Really Means 1, or 0 Really Means 1

Documentation for Modbus is not well standardized. Actually there is a standard, but not well followed when it comes to documentation. You will have to do one or more of the following to decipher which register a manufacturer is really referring to:

a) Look for the register description, such as holding register, coil, etc. If the documentation says #1, and tells you they are holding registers, then you have holding register #1. You also have user friendly documentation.

b) Look at the numbers themselves. If you see the first register on the list having a number 40001, that really tells you register #1, and it is a holding register. This form of notation is often referred to as the old Modicon convention.

c) Look for a definition of function codes to be used. If you see a register #1, along with notation telling you to use function codes 3 and 16, that also tells you it is holding register #1.

IMPORTANT: Register 1 is address 0. Read on…

d) Do the numbers in your documentation refer to the register number or address? Register #1 is address zero. If it is not clear whether your documentation refers to register or address, and you are not getting the expected result, try plus or minus one for register number. Most Control Solutions products refer to register numbers in configuration software or web pages. However, some manufacturers document their devices showing address, not register numbers. When you have addresses, you usually must add one when entering that register into configuration software from Control Solutions.

40001: Modicon Convention Notation for Modbus Registers

Modbus was originally developed by Gould-Modicon, which is presently Schneider Electric. The notation originally used by Modicon is still often used today, even though considered obsolete by present Modbus-IDA standards. The advantage in using the Modicon notation is that two pieces of information are included in a single number: (a) The register type; (b) The register number. A register number offset defines the type.

Many other equipment manufacturers still use this convention in their products. Depending on which combination of products you are using, you may have to translate between Modicon and current conventions.

The IoTServer and Babel Buster 4 support the ability to display Modbus registers in whatever format the user selects under User Settings. Options include address (0-indexed), number (1-indexed), and Modicon notation.

The types of registers referenced in Modbus devices, and supported by Babel Buster gateways, include the following:

- Coil (Discrete Output)
- Discrete Input (or Status Input)
- Input Register
- Holding Register

Valid address ranges as originally defined for Modbus were 0 to 9999 for each of the above register types. Valid ranges allowed in the current specification are 0 to 65,535. The address range originally supported by Babel Buster gateways was 0 to 9999. The extended range addressing was later added to all new Babel Buster products.

The address range applies to each type of register, and one needs to look at the function code in the Modbus message packet to determine what register type is being referenced. The Modicon convention uses the first digit of a register reference to identify the register type.

Register types and reference ranges recognized with Modicon notation are as follows:

- 0x = Coil = 00001-09999

- 1x = Discrete Input = 10001-19999
- 3x = Input Register = 30001-39999
- 4x = Holding Register = 40001-49999

On occasion, it is necessary to access more than 10,000 of a register type. Based on the original convention, there is another de facto standard that looks very similar. Additional register types and reference ranges recognized with Modicon notation are as follows:

- 0x = Coil = 000001-065535
- 1x = Discrete Input = 100001-165535
- 3x = Input Register = 300001-365535
- 4x = Holding Register = 400001-465535

When using the extended register referencing, it is mandatory that all register references be exactly six digits. This is the only way Babel Buster will know the difference between holding register 40001 and coil 40001. If coil 40001 is the target, it must appear as 040001.

Registers Are 16-bits - How Do I Read Floating Point or 32-bit Data?

Modbus protocol defines a holding register as 16 bits wide; however, there is a widely used de facto standard for reading and writing data wider than 16 bits. The most common are IEEE 754 floating point, and 32-bit integer. The convention may also be extended to double precision floating point and 64-bit integer data.

The wide data simply consists of two consecutive "registers" treated as a single wide register. Floating point in 32-bit IEEE 754 standard, and 32-bit integer data, are widely used. Although the convention of register pairs is widely recognized, agreement on whether the high order or low order register should come first is not standardized. For this reason, many devices, including all Control Solutions gateways, support a "swap" option. This means you simply check the "swapped" option if the other device treats wide data in the opposite order relative to Control Solutions default order. In some cases, the “swap” option is more explicitly identified as “high order data is in first register” or something to that effect.

Most Control Solutions Modbus products default to placing the high order register first, or in the lower numbered register. This is known as "big endian", and is consistent with Modbus protocol which is by definition big endian itself. The byte order for all 16-bit values is most significant byte first.

What Does Notation Like 40001:7 Mean?

This is a commonly used notation for referencing individual bits in a register. This particular example references (Modicon notation) register 40001, bit 7. Bits are generally numbered starting at bit 0, which is the least significant or right most bit in the field of 16 bits found in a Modbus register. If this style notation is used, you may see 40001:0 through 40001:15.

How Do I Read Individual Bits in a Register?

Documentation tends to be slightly different for every Modbus device. But if your device packs multiple bits into a single holding register, the documentation will note up to 16 different items found at the same register number or address. The bits may be identified with “Bn” or “Dn” or just “bit n”. Most of the time, the least significant bit will be called bit 0 and the most significant will be bit 15. It is possible you could find reference to bit 1 through bit 16, in which case just subtract one from the number to reference the table below.

You cannot read just one bit from a holding register. There is no way to do that - Modbus protocol simply does not provide that function. You must read all 16 bits, and then test the individual bit you are interested in for true or false (1 or 0). Babel Buster gateways provide an automatic way of doing that by including a "mask" in each register map or rule. Each time the register is read, the mask will be logically AND-ed with the data from the register, and the result will be right justified to yield a 1 or 0 based on whether the selected bit was 1 or 0. Babel Buster gateways provide optimization when successive read maps or rules are selecting different bits from the same register. The Modbus register will be read from the slave once, and the 16-bit (or 32-bit) data will be shared with successive maps or rules, with each map or rule selecting its bit of interest.

The bit mask shown in the expanded form of the Babel Buster RTU read map is a 4 digit hexadecimal (16 bit) value or 8 digit hexadecimal (32 bit) value used to mask out one or more bits in a register. The selected bits will be right justified, so a single bit regardless of where positioned in the source register will be stored locally as 0 or 1. The hex bit mask values would be as follows:

- B0/D0/bit 0 mask = 0001
- B1/D1/bit 1 mask = 0002
- B2/D2/bit 2 mask = 0004
- B3/D3/bit 3 mask = 0008
- B4/D4/bit 4 mask = 0010
- B5/D5/bit 5 mask = 0020
- B6/D6/bit 6 mask = 0040
- B7/D7/bit 7 mask = 0080
- B8/D8/bit 8 mask = 0100
- B9/D9/bit 9 mask = 0200
- B10/D10/bit 10 mask = 0400
- B11/D11/bit 11 mask = 0800
- B12/D12/bit 12 mask = 1000
- B13/D13/bit 13 mask = 2000
- B14/D14/bit 14 mask = 4000
- B15/D15/bit 15 mask = 8000

Note: The IoTServer or Babel Buster 4 supports both 16-bit and 32-bit masking, and mask values are always displayed as 32-bit masks even if applied only to a 16-bit register. In the case of a 16-bit register, the four leading zeros are simply disregarded.

Some Modbus devices also back two 8-bit values into a single 16-bit register. The two values will typically be documented as “high byte” and “low byte” or simply have “H” and “L” indicated. If you run into this scenario, the masking for bytes is as follows:

- High byte mask = FF00
- Low byte mask = 00FF

When the mask value in a Babel Buster gateway is more than just one bit, the mask is still logically AND-ed with the data from the Modbus slave, and the entire resulting value is right justified to produce an integer value of less than the original bit width of the original register.

There have been a few instances of documenting packed bits in a 32-bit register. Although Modbus protocol is strictly 16-bit registers, some implementations force you to read pairs of registers. If your device documents 32 packed bits, then you would insert 0000 in front of each mask above, and the remainder of the list would be as follows:

- B16/D16/bit 16 mask = 00010000
- B17/D17/bit 17 mask = 00020000
- B18/D18/bit 18 mask = 00040000
- B19/D19/bit 19 mask = 00080000
- B20/D20/bit 20 mask = 00100000
- B21/D21/bit 21 mask = 00200000
- B22/D22/bit 22 mask = 00400000
- B23/D23/bit 23 mask = 00800000
- B24/D24/bit 24 mask = 01000000
- B25/D25/bit 25 mask = 02000000
- B26/D26/bit 26 mask = 04000000
- B27/D27/bit 27 mask = 08000000
- B28/D28/bit 28 mask = 10000000
- B29/D29/bit 29 mask = 20000000
- B30/D30/bit 30 mask = 40000000
- B31/D31/bit 31 mask = 80000000

Can I Put 2 Gateways on the Same Modbus Network?

You can not have more than one Master on a Modbus RTU (RS-485) network. Therefore, if the gateway is to be configured as the Master, you can only have 1 gateway. You cannot use multiple gateways to read more points from the same Modbus slave device.

Multiple gateways configured as slaves can reside on the same Modbus RS-485 network.

If you are using RS-232 devices, you can have only two devices total, regardless of how they are configured. RS-232 is not multi-drop.

How Many Devices Can I Have on a Modbus Network?

Logically you can address over 250 devices; however, the RS-485 transceivers are not capable of physically driving that many devices. Modbus protocol states that the limit is 32 devices, and most RS-485 transceivers will agree with this. Only if all devices on the network have low load transceivers can you have more than 32 devices.

Where Can I Get a Copy of the Modbus Protocol Specification?

You can get a copy of the Modbus protocol specification by visiting www.modbus.org. There are three documents of primary interest: (1) The application layer protocol which defines the function codes and addressing; (2) The definition of use over a serial line (RTU); (3) The definition of use over Ethernet (TCP). You will be asked to accept terms of use, but there is no cost for these specifications.