Application Note

Getting Started with Modbus

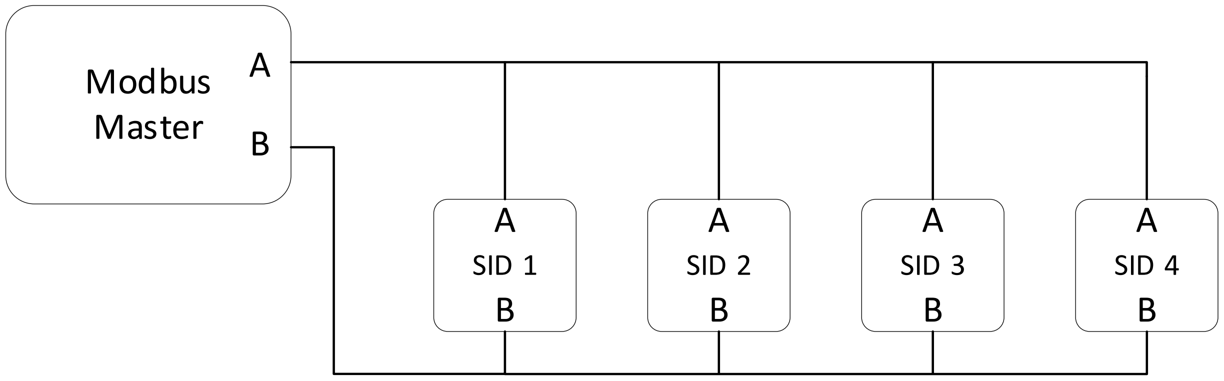
OVERVIEW

Modbus is a communication protocol used widely in industrial applications. Since its introduction in 1979, Modbus has become the standard mode of transferring data to PLCs. Due to its wide adoption, SignalFire's wireless telemetry system is natively built from the ground up to support Modbus. However, many technicians working on SignalFire equipment as well as other industrial machines may not be entirely familiar with this protocol or how it can be used to set up the instrumentation and controls they want. This guide – in conjunction with the System Overview app note – serves to introduce new users to getting their desired flow of data up and running.

HARDWARE

Modbus is the name of a communication protocol developed by Modicon (now Schneider Electric) in 1979. Modbus as a **protocol** is typically sent over an RS-485 serial **bus**, which offers it many advantages. 'Protocol' refers to the software while 'bus' refers to the hardware.

RS-485 is a two or four-wire serial multidrop bus. In a four-wire configuration, there's TxD+, TxD-, RxD+, and RxD-. In a two-wire configuration, TxD+ and RxD+ are jumped together, and TxD- and RxD- are jumped together, leaving a positive and negative line. Modbus typically uses this two-wire configuration, labeled 'A' (usually positive) and 'B' (usually negative). Multidrop means that multiple devices can all be daisy-chained in parallel together to communicate on the same line. Multiple devices can have all their 'A' terminals connected together, and all their 'B' terminals connected together to communicate with each other, making the wiring simple.



SERIAL COMMUNICATION

RS-485 is a means of serial communication called UART (Universal Asynchronous Receiver/Transmitter), transmitting and receiving data 1 bit at a time. The number of bits sent per second is called the baud rate. To properly time and frame messages, devices send either 7 or 8 bits of data, 1 or 2 "stop bits" that signals the end of the frame, and a “parity bit”. Parity is a simple and quick way to check that the frame was sent correctly. It adds up all the ‘1’ bits in the message, and checks if it’s an even or odd number. If the receiver is expecting an even number but gets an odd number, it can immediately tell something is wrong.

The default UART settings for most devices is a baud rate of 9,600 bits/s, 8 data bits, No parity bit, and 1 stop bit. You will often see this abbreviated as 9600-8N1. For any two devices to communicate over RS-485, they must have the same UART settings or it will cause a mismatch and data errors. Check the device’s manual to verify its RS-485 serial UART settings.

MASTER-SLAVE

Modbus operates on a Master-Slave system. One Master device requests and collects information from multiple Slave devices, while Slave devices respond only to the Master device. In an industrial setting, the PLC/RTU/SCADA act as the Modbus Master while all the various instruments and sensors act as the Slaves. A Slave cannot have multiple Masters, or there will be conflicts.

To identify each other, every device on the network must have a unique ID between 1 and 255. Most Slaves have a default ID of 1. On the SignalFire network, nodes are sent out with a default Slave ID of 1, while Gateways are sent out with default Slave ID 247. ID’s 241 to 255 are reserved for Gateways, while nodes take up ID’s 1 through 240.

REGISTERS

Modbus devices store their data in registers, with their own addresses. Registers are divided into their own categories, based on their purpose (for example input vs. output), specified by a function code.

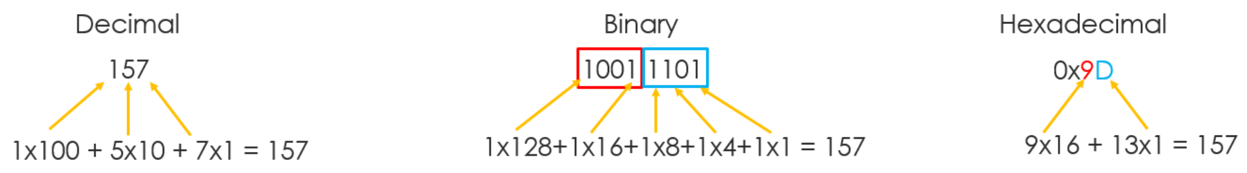
Register addresses can be defined in two ways: their register number, or their function code and address. The register number is a 5-digit combination of register address plus an offset based on the function code, while the register address is only 4 digits. For example, a register number of 43004 will translate to a function code 03 (holding register) and address of 3003, in other words there is an offset of 40001. Holding registers are always between register number 40,000 and 50,000. Most systems use the function code and register address, while others use the register number scheme. The table below specifies all the function codes, what they mean, and their range of register numbers.

Function Code	**Function**	**Register Number Range**
01	Read Discrete Output	00000-09999
02	Read Discrete Input	10000-19999
03	Read Holding Register	40000-49999
04	Read Analog Input	30000-39999
05	Write Single Discrete Output	00000-09999
06	Write Single Holding Register	40000-49999
15	Write Multiple Discrete Outputs	00000-09999
16	Write Multiple Holding Registers	40000-49999

BINARY

At times, it may be necessary to debug Modbus communications, which will involve dealing with binary. Humans read numbers in base 10, called the decimal system, where each digit can range from 0 to 9. Computers, being nothing more than millions of switches, operate in a base 2 system called binary, where each digit can be a 0 (off) or 1 (on). However, binary is fairly cumbersome and time consuming for people to read. For example, 93 would be represented in binary as 01011101.

We tend to write it in the more compact base 16 hexadecimal system, where each digit can range from 0 to 16, prefixed with a '0x' so the reader knows it's in hexadecimal. Digits greater than 9 are written as letters. Every digit in hexadecimal represents 4 digits in binary, and every two digits is a byte. The graphic below illustrates how the same number is written in decimal, binary, and hexadecimal.



Note to the reader, the calculator app on Windows has a binary function that you can activate by pressing Alt+3. It allows you to quickly convert between different formats, and can serve as a tool to enhance your understanding.

DATA TYPES

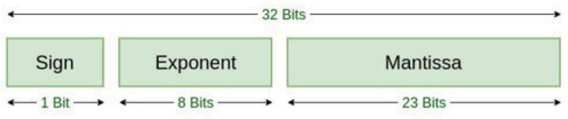
Registers come in multiple varieties depending on data type. The first type is Boolean, which is a 1-bit data type that indicates TRUE or FALSE. In Modbus, Boolean registers are also called coils, because they were originally used in Ladder Logic to control relay coils.

The next type is a 16-bit integer, either signed or unsigned. This type is held in a single register, and is simply a whole number represented in binary. In a signed integer the first bit is a 0 or 1 to indicate whether the number is positive or negative, while unsigned registers can only be positive.

The next type is a 32-bit floating point. Floating point is a format, specified by IEEE standard 754, that allows computers to represent values with greater precision than integers with several decimal places. Because Modbus only supports 16-bit registers, floating point values are held in two registers, an "upper word" and "lower word". Each word is made of two 8-bit bytes. The bytes don't have to be in any particular order. They're usually set to ABCD order, but can be CDAB (word swapped), BADC (byte swapped), or DCBA (byte and word swapped). Check your device's manual carefully to know what order its floating points are in.

It's important to keep in mind that unlike integers, floating points are not a direct conversion; they are interpreted. The graphic below illustrates how reading a floating point directly in hexadecimal can give a nonsensical value. 0x3E2F does not actually convert to 62.47, it's just meant to show how interpreting values gives computers greater flexibility in representing numbers. The Sign-Exponent-Mantissa scheme shown below that is what machines actually use.

0x3E = 62 → Easy!   0x3E2F = 15,159 → What?   0x3E2F = 62.47 → Magic!



Single Precision
IEEE 754 Floating-Point Standard

COMMANDS/POLLING

Modbus Masters request the data through commands, or what's called polling. Modbus polls follow a very simple format. The Modbus Master specifies a Slave ID, function code, starting register address, and number of consecutive registers to read from that starting address. Non-consecutive addresses or from other Slave ID's must be requested with multiple commands. The commands are sent in hexadecimal, broken down piece by piece as shown below. Here, the Modbus Master is requesting to read 10 holding registers starting at address 3042 from Slave ID 86.

0x56030BE2000A6A38

Slave ID = 86 (0x56)
Function Code = 03 (0x03)
Start Address = 3042 (0x0BE2)
Address Count = 10 (0x000A)

The last two bytes, 0x6A38 in the above example, are a CRC (Cyclic Redundancy Check). The CRC is a method of ensuring the command was properly sent without any errors. The Master and Slave both perform a mathematical operation on the command and response, and if the command is uncorrupted, they should both get the same value. It would be like sending a screenshot of an email while replying so everyone knows the reply is being sent to the right email.

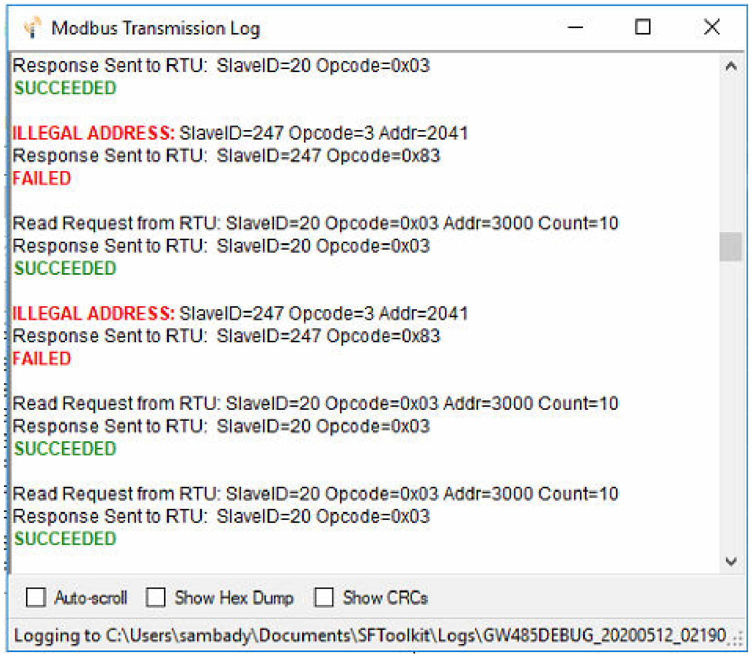
If there are any issues, the Modbus Master or Slave may reply with an error. The error response indicates which function code caused the problem appended with an '8' (so a bad function code 0x03 command will receive a function code 0x83) and an exception code that specifies what the issue was. The table below indicates what each exception code means.

Exception Code	**Name**
01	Illegal Function
02	Illegal Data Address
03	Illegal Data Value
04	Slave Device Failure
05	Acknowledge
06	Slave Device Busy
07	Negative Acknowledge

08	Memory Parity Error
10	Gateway Path Unavailable
11	Gateway Target Device Failed to Respond

ADDRESSING SIGNALFIRE

When Modbus was created in 1979, Slave instruments were wired directly to the Master. The Master would address those Slave ID's directly, with a function code and register, to get the values it wanted. With the SignalFire network, while the instruments are no longer directly wired to the Master, the Gateway simulates direct connection. Whenever a node checks in to the Gateway, the Gateway caches its registers. A Modbus Master can then poll that node's registers as if it were hard wired to it. In this way, a SignalFire network can be dropped into an existing system with no change to PLC programming.



```
Response Sent to RTU:  SlaveID=20 Opcode=0x03
SUCCEEDED

ILLEGAL ADDRESS: SlaveID=247 Opcode=3 Addr=2041
Response Sent to RTU:  SlaveID=247 Opcode=0x83
FAILED

Read Request from RTU: SlaveID=20 Opcode=0x03 Addr=3000 Count=10
Response Sent to RTU:  SlaveID=20 Opcode=0x03
SUCCEEDED

ILLEGAL ADDRESS: SlaveID=247 Opcode=3 Addr=2041
Response Sent to RTU:  SlaveID=247 Opcode=0x83
FAILED

Read Request from RTU: SlaveID=20 Opcode=0x03 Addr=3000 Count=10
Response Sent to RTU:  SlaveID=20 Opcode=0x03
SUCCEEDED

Read Request from RTU: SlaveID=20 Opcode=0x03 Addr=3000 Count=10
Response Sent to RTU:  SlaveID=20 Opcode=0x03
SUCCEEDED
```

Logging to C:\Users\sambady\Documents\SFToolkit\Logs\GW485DEBUG_20200512_02190

As an example, if a Pressure Scout is checking in to the Gateway on Slave ID 20 a Modbus Master can send a function code 03 command to Slave ID 20 for 1 register starting at address 3000 to get the pressure as a 16-bit unsigned integer. A register address table for each device can be found in the ToolKit or in the respective device's manual.

The Gateway also allows for Modbus register remapping, so that non-consecutive registers or registers from multiple slave ID's can be consolidated into one group and read with one command. See the Gateway manual for more details.

MODBUS ETHERNET

The version of Modbus described so far is known as Modbus RTU, which is for an RTU/PLC/SCADA wired directly to a Slave over RS-485. Other possible connections are Modbus TCP, Modbus ASCII, and Modbus RTU Over TCP. It's important to not get Modbus TCP confused with Modbus RTU Over TCP. While they all send commands of the same format, they package them differently and are thus incompatible with each other. It’s also important to not get Modbus TCP confused with Ethernet/IP, which is a separate interface, and will need its own converter switch.

While the SignalFire Gateway comes standard with Modbus RTU, it can be ordered with an Ethernet Interface Module which converts between Modbus RTU and Modbus TCP. The system can be assigned an IP address and connected to a switch, where it can then be accessed by a Modbus Master over the network.

Instead of specifying a COM port, the Modbus Master will have to specify an IP address, and connect to port 502. Port 502 is for Modbus communication, while port 10002 is for ToolKit connection, and port 80 is for the web browser configuration page. See the Ethernet Interface Module manual for more details.