ANALYTICAL TECHNOLOGY, INC.

O & M Manual

MetriNet

Modbus RTU
Communications Manual

Home Office

Analytical Technology, Inc.
6 Iron Bridge Drive
Collegeville, PA 19426
Phone: 800-959-0299
610-917-0991
Fax: 610-917-0992
Email: sales@analyticaltechnology.com

European Office

ATI (UK) Limited
Unit 1 & 2 Gatehead Business Park
Delph New Road, Delph
Saddleworth OL3 5DE
Phone: +44 (0)1457-873-318
Fax: + 44 (0)1457-874-468
Email: sales@atiuk.com

Table of Contents

Table of Figures

Modbus Description

1.1 Modbus Technical Overview

Modbus protocol is a messaging structure, widely used to establish master-slave communication between intelligent devices. A message sent from a master to a slave contains a one-byte slave address, a one-byte command, data bytes (depending on command), and a two byte CRC. The protocol is independent of the underlying physical layer and is traditionally implemented using RS232, RS422, or RS485 over a variety of media (e.g. fiber, radio, cellular, etc.).

The protocol comes in 2 flavors – ASCII and RTU. The formats of messages are identical in both forms, except that the ASCII form transmits each byte of the message as two ASCII hexadecimal characters. Therefore, ASCII messages are twice as long as RTU messages. The main advantage of the RTU mode is that it achieves higher throughput, while the ASCII mode allows time intervals of up to 1 second to occur between characters without causing an error. As stated earlier, the transmitter uses the RTU form and does not support the ASCII form.
The basic structure of an RTU frame is shown below:

[ADDRESS][FUNCTION][DATA][CRC]

The address field of a message frame contains an eight-bit slave device address in the range of 0 ... 247 decimal. The individual slave devices are assigned addresses in the range of 1 ... 247, and address 0 is reserved as a broadcast address. A master addresses a slave by placing the slave address in the address field of the message. When the slave sends its response message, it places its own address in this address field of the response to let the master know which slave is responding. All slaves accept broadcast messages (address 0) as though they were addressed specifically to them, but do not transmit a response message.

The function code field of a message frame contains an eight-bit code in the range of 1 ... 255 decimal. When a query message is sent from the master, the function code field tells the slave device what kind of action to perform. Examples include reading the contents of a group of registers, writing to a single register, writing to a group of registers, and reading the exception status.

When the slave device responds to the master, it uses the function code field to indicate either a normal (error-free) response or that some kind of error occurred (called an exception response). For a normal response, the slave simply echoes the original function code. For an exception response, the slave returns a code that is equivalent to the original function code with its most significant bit set to logic 1.

The data field is constructed of one or more bytes and contains additional information, which the slave must use 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.

If no error occurs, the data field of a response from a slave to a master contains the data requested. If an error occurs, the field contains an exception code that the master application can use to determine the next action to be taken.

The data field can be nonexistent (of zero length) in certain kinds of messages. For example, in a request from a master device for a slave to respond with its communications event log (function code 0B hexadecimal), the slave does not require any additional information. The function code alone specifies the action.

Messages are terminated with a 16-bit CRC value that is computed from all of the bytes of the message. The two byte CRC is superior to just simple checksums because it can help reject more types of errors.

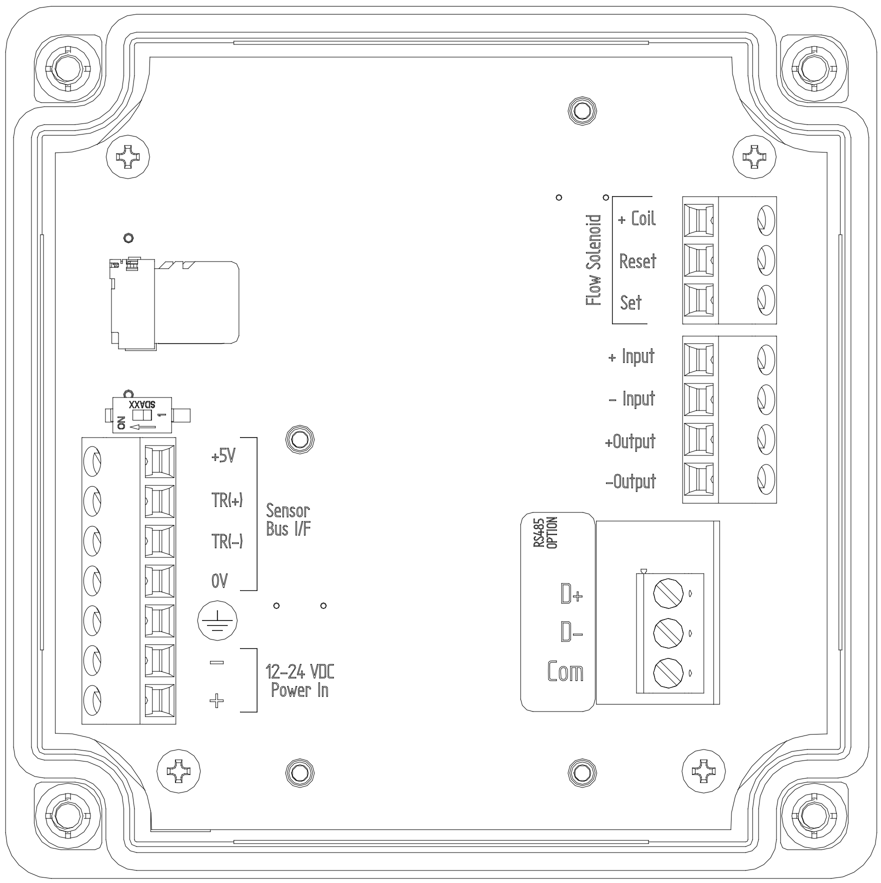


Figure 1 - Modbus Terminal Connections

1.2 Modbus Connection

Modbus wiring is done at a plug-in communication circuit board located in the outlined section of Figure 1 above. A call-out in that figure shows the location of RS-485 connections. The earth ground wire in the 485 cable should be terminated at “COM” on the Modbus card.

1.3 Registers and Coils

Modbus protocol was originally designed to transfer data to and from PLCs (Programmable Logic Controllers), which organize data into groups of registers and coils. PLC registers containing I/O information are called input registers and are numbered 30001 to 39999, while registers containing data or the results of calculations are known as holding registers and are numbered from 40001 to 49999. The term coils, on the other hand, refers to discrete (0 or 1) inputs and outputs. Traditionally, these are inputs from such things as switch closures and outputs to the coils of relays, which are under the control of the PLC.

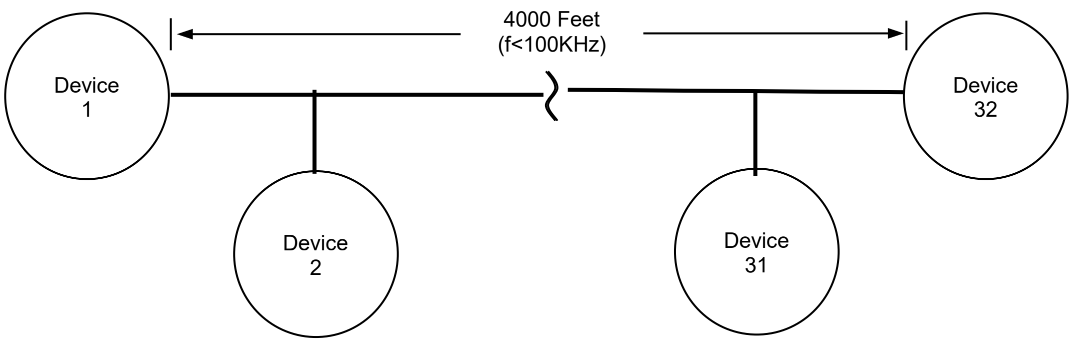
All registers are 16 bit values, which may be read or written to individually, or in blocks by using specific functions. Likewise for coils, which are one bit values. Since register functions transfer 16 bits and discrete (coil) functions transfer only one, it is usually more efficient to use register functions, which reduces the number of messages required to transfer data. For this reason, the MetriNet transmitter organizes all of its data into holding registers only, or more specifically, data is organized into the holding registers starting at 40001.

The protocol specifies which registers to access by the value of the function code embedded into the message. For example, to read one or more holding registers in a slave device, the master must use function 03 – "Read Holding Register". Similarly, the master must use function 04 – "Read Input Register" to read one or more of the input registers. The MetriNet only responds to request for reading holding registers (Function 3).

For more information on the protocol, please refer to the "Modicon Modbus Protocol Reference Guide" at http://www.modicon.com/techpubs/toc7.html or, "Modbus Protocol Specification", available for download at http://www.modbus-ida.org/specs.php. Deviations from this guide are noted in the appropriate section. More information regarding Modbus, in general, may be viewed at: http://www.modbus-ida.org/

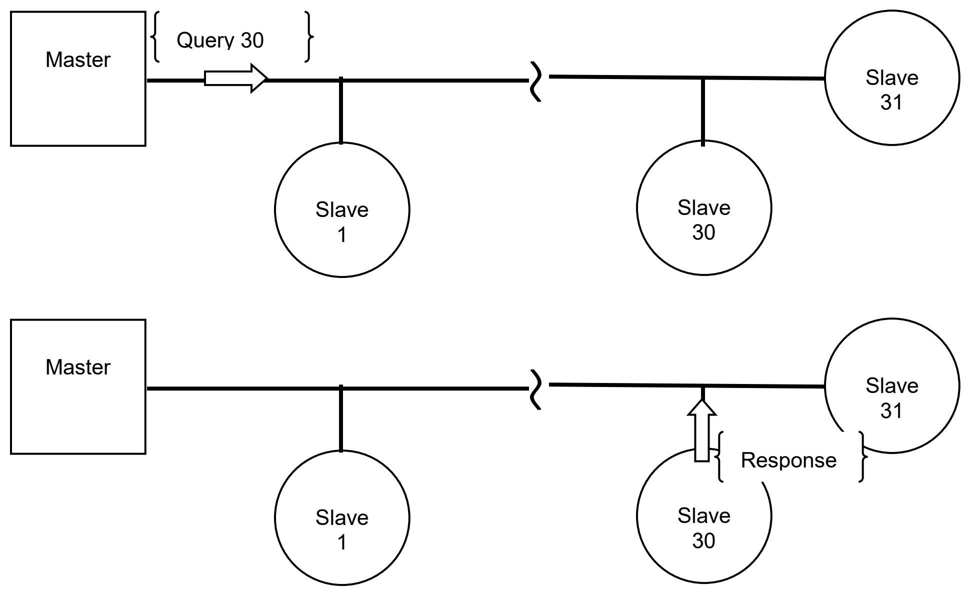
1.4 RS-485 Communication

Modbus data transmission is an RS-485 based communication protocol. The RS485 standard specifies a two-wire, half-duplex serial data bus for connecting up to 32 devices in parallel, at distances of up to 4000 feet at transmission rates at or below 100KHz. The RS485 standard allows the user to configure inexpensive local networks and multi-drop communications links using a twisted pair cable. A typical RS485 network can operate properly in the presence of reasonable ground differential voltages, withstand driver contentious situations, and provide reliable communications in electrically noisy environments with good common mode rejection.

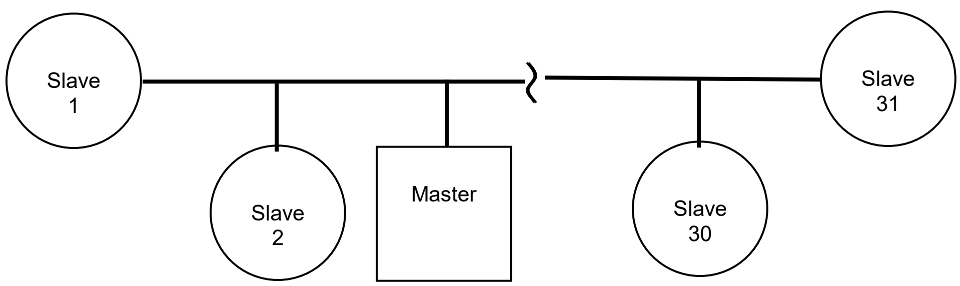


Half-duplex means outgoing messages share the same physical medium with incoming messages. Only one device may transmit at any given time. During any exchange of data communication, one device must act as master and one or more devices act as slaves. With no activity on the bus, the master device sends an addressed query to a slave and then gives up the bus. All slaves receive the message, but only the addressed slave responds.

MetriNet Analyzers use a plug-in Modbus circuit board shown in Figure 1 of this manual. Wiring connections for the communication bus are shown in that Figure.

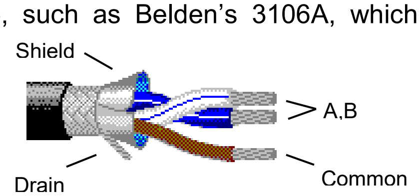


The master node may be located anywhere on the network, not just at one end.

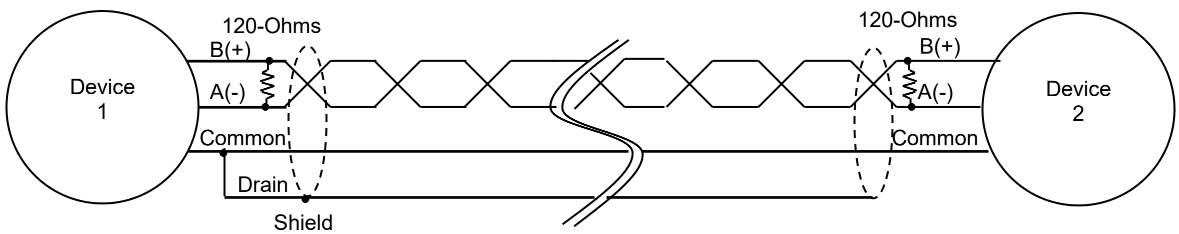


1.5 Cable Specification

The bus is a cable composed of a twisted pair of wires with a characteristic impedance of 120 ohms, and a 120-ohm termination resistor connecting the pair of wires at each end. Several manufacturers offer cables specifically designed for RS485, such as Belden's 3106A, which features one twisted pair, a separate signal common, a foil shield, and a drain wire in contact with the shield.

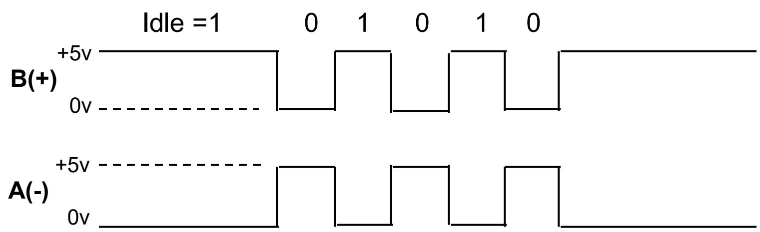


The twisted pair, labeled A and B (or – and +, respectively), form a differential transmission line capable of operating over a common mode voltage range from –7v to +12v. That is, the ground potential at each end of the network may differ by this amount. Connecting a signal common to each slave device will keep this potential to a minimum. The shield around the conductors provides protection from EMI (electromagnetic interference) and should be connected to common or ground at only one point to avoid circulating currents that might actually generate interference on the inner conductors. A schematic of the bus is shown below.



1.6 RS-485 Line Drivers/Receivers

The differential lines, A and B, may be operated at TTL levels of 0 and 5 volts. The RS485 line driver outputs the logic high state (marking, or idle state) by driving 5 volts on B, and 0 volts on A. Conversely, the driver outputs the logic low state (spacing) by driving 5 volts on A, and 0 volts on B.



Over a distance of 4000 feet, the 5 volts applied to either line may be dropped significantly. This usually doesn't present a problem since RS485 receivers are specified to operate with a differential voltage of only 0.200 volts. In practice, however, the differential voltage should remain above 1.5v.

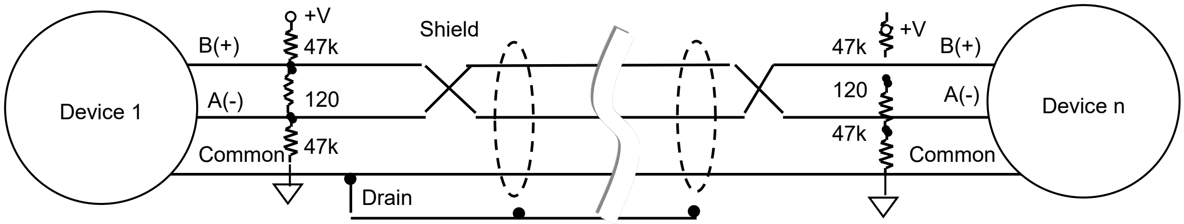
Logic State High	(Idle or Marking State):	(B – A) >= 200mV
LogicState Low	(SpacingState):	(A – B) >= 200mV

1.7 120 Ohm Termination

The two devices at the furthest end of the bus require termination resistors to cancel reflections. Intermediate devices do not. The MetriNet has a selectable termination resistor on the Modbus card just behind the terminal strip. **<u>ONLY</u>** set to "ON" position if the MetriNet is an End-of-Bus unit.

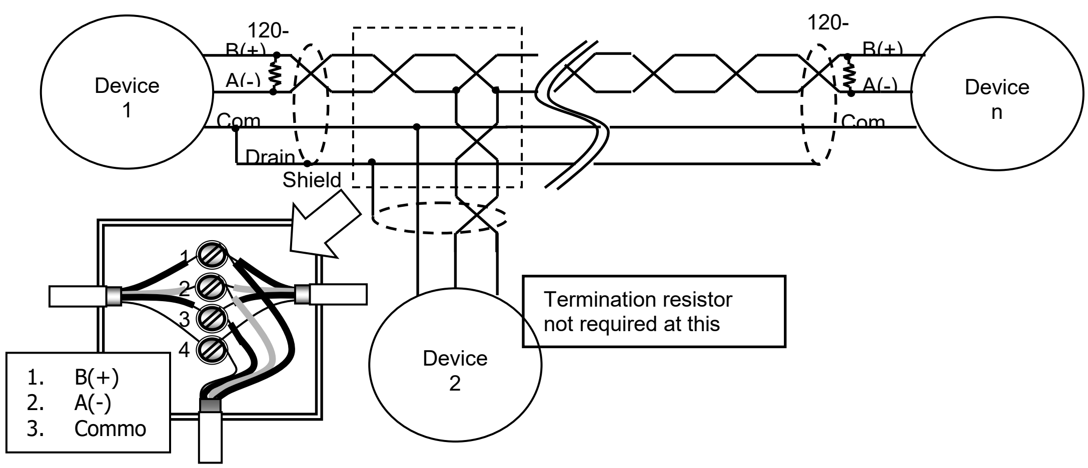
1.8 Bias

When there is no communication on the network, the A and B lines are floating. A small amount of noise could appear as the start of a message, which might interfere with the framing of valid messages. Biasing the transmission line keeps it in the idle state while it is not driven. The bias resistors maintain a differential of 200mV between the A and B lines. Note that bias resistors are not required for MetriNet transmitters, as the Modbus driver includes a "Failsafe" built-in Bias Design.



1.9 Drops

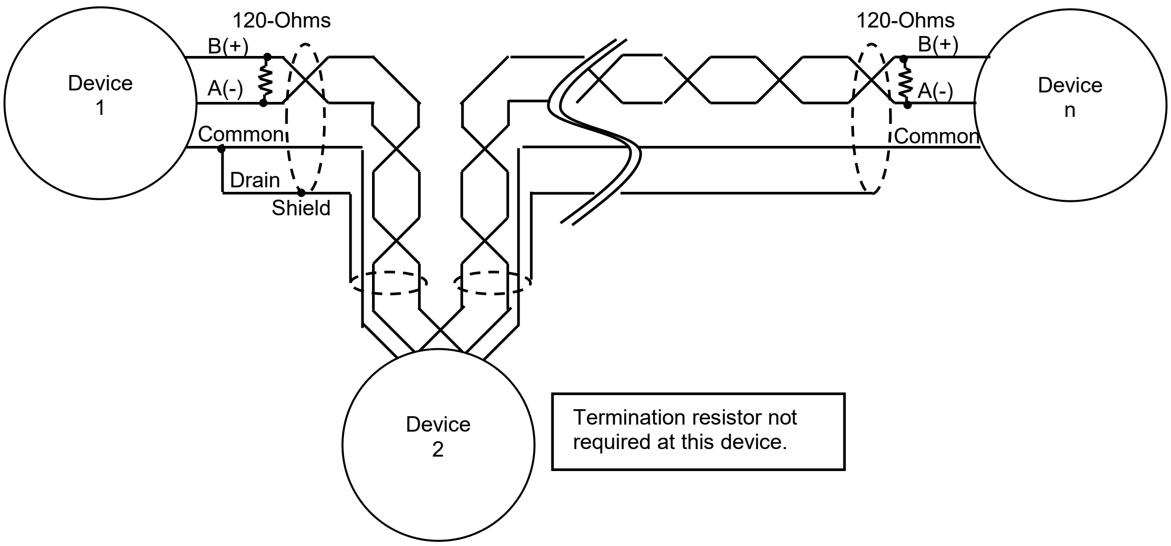
Often, a short length of cable is used at a junction box to form a branch, or "drop", from the bus to the device. These cables must very short as compared to the main trunk length of the bus so as to avoid signal reflections and require termination that would load the bus excessively. A rule of thumb is to not allow any single branch length to exceed 3% of the total trunk length. Again, only the devices at each end of bus require termination resistors, intermediate connections along the bus do not (bias resistors not shown for clarity).



Long branches requiring termination may be connected, however, a repeater must be used at a short distance from the connection. Star topologies should be avoided, since terminating each spoke will load the network excessively and reliable communications cannot be guaranteed.

1.10 Daisy Chaining

For devices not located at the ends of the bus, it may be possible to run the cable in and out of the device, a practice referred to as "daisy-chaining". Although this method eliminates the need for a separate drop wire, it will require more connections inside the transmitter housing and therefore consume more space.



1.11 Shielding

While it goes against conventional wisdom and can cause a problem with circulating currents, grounding a shielded cable at both ends can be very effective at keeping induced electrical noise away from the communications lines. In the alternative, ground one end of the shield and connect the other end to ground through a bi-directional transient protector (from a few volts to a few hundred volts depending on the situation).

Note that MetriNet transmitters are galvanically isolated from the RS-485 (Modbus) port. And utilize extensive spike/surge protection from lightening and other electrical sources.

1.12 Slave Connection Detail

The Modbus RTU connection settings will appear in the OPTIONS listing of the MetriNet software once the menu item "**^Host Comms**" has been set to "**Modb**." Once set, Modbus menu items for baud, parity, etc. will appear in OPTIONS listed menus. Note that those follow-up menus only appear if Modb is selected.

Note: The Unit Addr is the Modbus Address as well as the DataLog Unit ID.

Once set to Modb…

1- Press UP arrow. Next you will see the menu "**^Baud Rate**" come up. Default is 9600, select either 9600 or 19.2k.

2- Press UP arrow. Next you will see the menu "**^Parity**" come up. Default is none. Range of entry is none, odd or even.

3- Press up arrow **"^Stop Bits"** come up. Default is 2. Select either 1 of 2.

4- Press UP arrow **"^Wr Lock Enable"** comes up. Default is OFF. Select either OFF or ON.

1.13 MetriNet Modbus RTU Interface

The data map for the MetriNet is broken into logical blocks: Live measured data that changes constantly, information that only changes when a user adjustment is made, and finally an area dedicated to configuration and calibration via user entry. The MetriNet only recognizes Modbus functions 03 and 16, where 03 is used for all the INFO and MEASURE areas, and 16 is only allowed in the CONFIGURATION and CALIBRATION area.

Please refer to the **M-Node O&M Manual** for detailed information regarding the probe Modbus register and calibrations descriptions, values, settings and limits.

1-MB 40001-40009 = **SYSTEM INFO BLOCK**
(READ ONLY)
9 registers. MetriNet information.

2-MB 41001-41096 = **SENSOR MEASURE BLOCK**
(READ ONLY)
96 registers (12 x 8.) Measured sensor data. This area is normally read continuously, as it is actively changing.

3-MB 42001-42144 = **SENSOR INFO BLOCK**
(READ ONLY)
144 registers (18 x 8.) Sensor configuration settings. While it could be read at any time, this area is only updated after a power up, or after a "write" of new data has occurred.

4-MB 43001-43004 = **SYSTEM CONFIGURATION/CAL BLOCK**
(WRITE ONLY)
4 registers. This area is a unique "router" window that allows the user to write new data to a specific location.

So, during normal polling, the SENSOR MEASURE BLOCK would typically be read continuously, as its always changing. The INFO BLOCKs would likely only be read once at system start-up, and then after any CONFIGURATION or CALIBRATION change has been performed by user. This method greatly minimizes the bandwidth requirement for instrument by avoiding re-reading data which is not changing. The maximum polling rate by external master is 100 mS (old received frame stop-to-new frame start.)

If the Wr Lockcode is enabled in the MetriNet OPTIONS Menu, an un-lock write security code must be included as part of the write command in order to perform a write to any of these registers. See the unlock code section later in the manual on the CONFIGURATION/CAL portion of Modbus map.

We recommend a simple master Modbus RTU test program, by the name of SimplyModbus, for any pre-testing of Modbus slaves. This particular program is very easy to use and provides many same-page fields to enter all required communication parameters on one screen. In addition, this test program allows the user the flexibility to set different data types by combining various numbers of 16-bit registers into any desired field length.
http://www.simplymodbus.ca/RTUmaster.htm

Here is a screenshot of the Simply Modbus Master PC tool. Note that the MetriNet only responds to "holding" register requests (40000 block,) so only function code 03 is accepted. **Endian arrangement is set in the "high byte first" and "high word first" selections. Endian byte swapping must be correct to see data.**

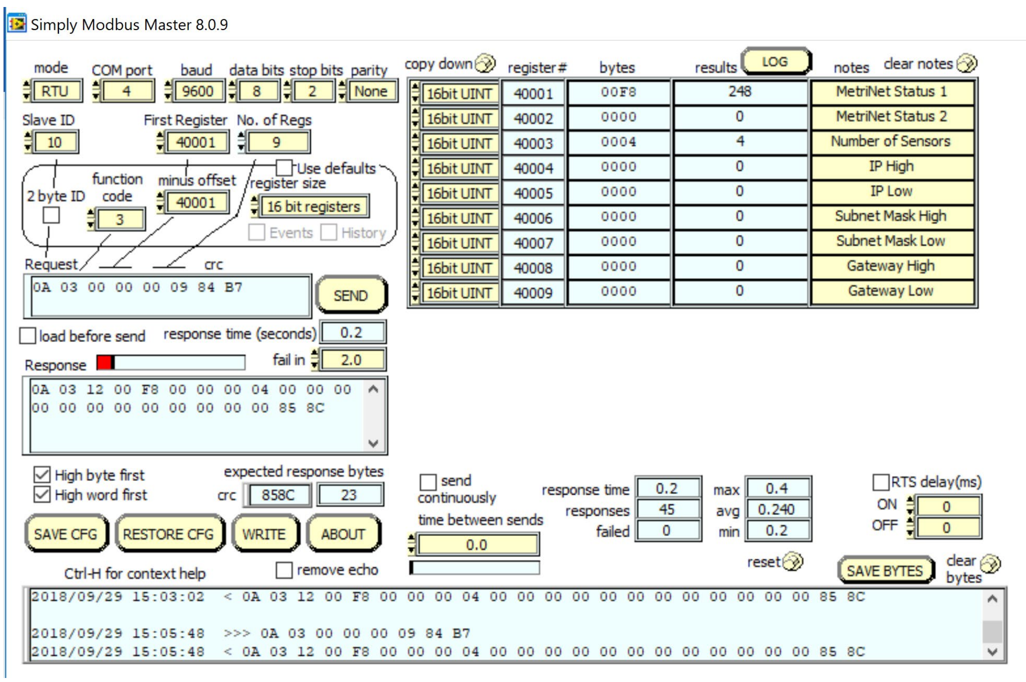


Figure 2 – SimplyModbus Setup Screen, Showing Setup and System Info Block

1-SYSTEM INFO - MB 40001 start, 9 registers (READ ONLY)

MetriNet information gets the first block in the overall map and starts at MB 40001.

Register	Data Type	Sensor	Description	Data Format
40001	UINT(16-bit)	MetriNet	Status 1	Binary
40002	UINT(16-bit)	MetriNet	Status 2	Binary
40003	UINT(16-bit)	MetriNet	Number of Sensors	1 to 8
40004	UINT(16-bit)	MetriNet	IP High	*192/168
40005	UINT(16-bit)	MetriNet	IP Low	*1/1
40006	UINT(16-bit)	MetriNet	Subnet Mask High	*255/255
40007	UINT(16-bit)	MetriNet	Subnet Mask Low	*255/0
40008	UINT(16-bit)	MetriNet	Gateway High	*255/255
40009	UINT(16-bit)	MetriNet	Gateway Low	*255/0

Register	Bit	Description
40001	0 (LSB)	Sensor 1 Comm Error
	1	Sensor 2 Comm Error
	2	Sensor 3 Comm Error
	3	Sensor 4 Comm Error
	4	Sensor 5 Comm Error
	5	Sensor 6 Comm Error
	6	Sensor 7 Comm Error
	7	Sensor 8 Comm Error
	8	Undefined
	9	Undefined
	10	Undefined
	11	Undefined
	12	Undefined
	13	Undefined
	14	Undefined
	15	Undefined

Register	Bit	Description	
40002	0	Solenoid Output	0 = Valve Closed, 1 = Valve Open
	1	Opto Input	0 = Input OFF, 1 = Input ON
	2	Opto Output	0 = Output OFF, 1 = Output ON
	3 – 15	Undefined	

**Only used for Ethernet options.*

2- SENSOR MEASURE – MB 41001 start, 96 registers (READ ONLY)

This live measurement block is "read-only" data via Modbus Function 03 - Read Holding Registers, and can be accessed from 41001 to 40096. To read all sensor data at once for 8 sensors, call for a 96 register read starting at MB41001. Otherwise, only poll the register range corresponding to the total number of sensors connected.
NOTE – First 4 values for each sensor are 32-bit signed integers, last 4 are 16-bit.

Sensor #1 (12 registers)

Register	Data Type	Description	Data Format
41001-41002	DINT(32-bit)	Main Value	1000 = 1.000
41003-41004	DINT(32-bit)	Main Units	ASCII (i.e. ppm)
41005-41006	DINT(32-bit)	Raw Sensor Value	1000 = 1.000
41007-41008	DINT(32-bit)	Temperature (F/C)	25000= 25.000C
41009	UINT(16-bit)	Output Value (VDC)	2500 = 2.500 VDC
41010	UINT(16-bit)	Status 1	Binary
41011	UINT(16-bit)	Status 2	Binary
41012	UINT(16-bit)	*Sensor ID	ASCII, (i.e. H0)

Bit	Reg 41010	Reg 41011
0 (LSB)	ALARM_A	EE_INIT_FAIL
1	ALARM_B	MAIN_UNITS_HI
2	ALARM_C	MAIN_UNITS_LO
3	ALARM_D	MAIN_INPUT_ERR
4	ALARM_E	TC_UNITS_HI
5	ENTRY_OUT_OF_RANGE	TC_UNITS_LO
6	ENTRY_ACCEPTED	TC_INPUT_ERR
7	ENTRY_FAIL	CAL_MAIN_SLOPE_HI
8	MAIN_CAL_PASS	CAL_MAIN_SLOPE_LO
9	MAIN_CAL_FAIL	CAL_MAIN_ZERO_HI
10	TC_CAL_PASS	CAL_MAIN_OFFSET_HI
11	TC_CAL_FAIL	MAIN_UNSTABLE
12	TC_F	CAL_TC_OFFSET_HI
13	SENSOR_LOCK	TC_UNSTABLE
14	NU	NU
15	NU	NU

Sensor ID is a unique two-byte ASCII code that identifies that sensor base model number. Q32H0 model would show here as "H0." Future feature, may not be currently available on all sensors.

Sensor #2-#8 (12 total registers each)

Snsr 2 Reg	Snsr 3 Reg	Snsr 4 Reg	Snsr 5 Reg	Snsr 6 Reg	Snsr 7 Reg	Snsr 8 Reg	Data Type	Sensor Data	Data Format
41013	41025	41037	41049	41061	41073	41085	DINT	Main Value	1000=1.000
41015	41027	41039	41051	41063	41075	41087	DINT	Main Units	ASCII (ie _ppm)
41017	41029	41041	41053	41065	41077	41089	DINT	Raw Value	32000=32.000
41019	41031	41043	41055	41067	41079	41091	DINT	Temperature	25000=25.000
41021	41033	41045	41057	41069	41081	41093	UINT	Output Value	2500=2.5000
41022	41034	41046	41058	41070	41082	41094	UINT	Status 1	Binary
41023	41035	41047	41059	41071	41083	41095	UINT	Status 2	Binary
41024	41036	41048	41060	41072	41084	41096	UINT	ID	ASCII

Note: If there is a comm error (loss of communications with a sensor), the error bit corresponding to the sensor will be set in register 41001. The data in the sensor registers will hold the last valid read from the sensor.

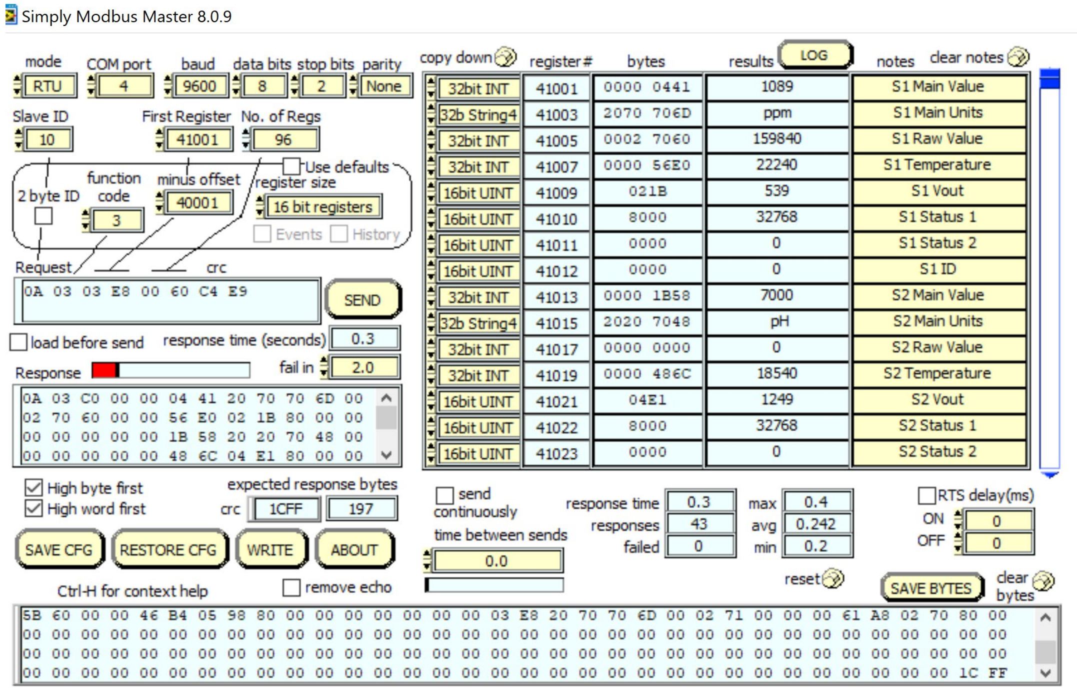


Figure 3 – SimplyModbus, Showing Sensor Measure Data Block

3- SENSOR INFO – MB 42001 start, 144 registers (READ ONLY)

These are data registers in the sensor that only change if the user sets in a new value. Therefore, these registers are read at power up from the sensors and kept in the MetriNet Modbus registers so the host may read them at any time. If a “communication timeout” error occurs, or a write command is received from the host, the MetriNet will read these registers from the sensors and update the data held in the MetriNet Modbus registers. This block would be read-only though Modbus Function 03 - Read Holding Registers.

18 Modbus Registers Per Sensor, all UINT(16)

Snsr 1 Reg	Snsr 2 Reg	Snsr 3 Reg	Snsr 4 Reg	Snsr 5 Reg	Snsr 6 Reg	Snsr 7 Reg	Snsr 8 Reg	Sensor Data	Data Format
42001	42019	42037	42055	42073	42091	42109	42127	[4]Slope	100=100%
42002	42020	42038	42056	42074	42092	42110	42128	[1,4]Offset	(sensor dependent)
42003	42021	42039	42057	42075	42093	42111	42129	Delay	10=1.0min
42004	42022	42040	42058	42076	42094	42112	42130	[1]Alarm A	(sensor dependent)
42005	42023	42041	42059	42077	42095	42113	42131	[1]Alarm B	(sensor dependent)
42006	42024	42042	42060	42078	42096	42114	42132	Slp Alarm	80=80%
42007	42025	42043	42061	42079	42097	42115	42133	Tmr Limit	90=90 days
42008	42026	42044	42062	42080	42098	42116	42134	[1,2]VoutHI	(sensor dependent)
42009	42027	42045	42063	42081	42099	42117	42135	[1,2]VoutLO	(sensor dependent)
42010	42028	42046	42064	42082	42100	42118	42136	TcMode	0 = F, 1 = C
42011	42029	42047	42065	42083	42101	42119	42137	[3]Tag1	0x70,0x48=”p”,”H”
42012	42030	42048	42066	42084	42102	42120	42138	[3]Tag2	…
42013	42031	42049	42067	42085	42103	42121	42139	[3]Tag3	…
42014	42032	42050	42068	42086	42104	42122	42140	[3]Tag4	…
42015	42033	42051	42069	42087	42105	42123	42141	[3]Tag5	…
42016	42034	42052	42070	42088	42106	42124	42142	[3]Tag6	…
42017	42035	42053	42071	42089	42107	42125	42143	[3]Tag7	…
42018	42036	42054	42072	42090	42108	42126	42144	[3]Tag8	…

[1] *Sensor dependent variable. The formatting of these variables are based on the specific data value from that sensor. See the M-Node sensor manual for details.*

[2] *There are no analog voltage outputs of the bussed MetriNet system. However, the scaled 0-2.5V value from the sensor can be used to simplify the creation of the scale value for other purposes.*

[3] *The Tag values are compressed ASCII characters stored in the sensor, and together they create a 16 character string for unique sensor identification. The user may change these to whatever they desire. For a Tag entry of 0x70 0x48 (hex 70, 48,) you would store the characters “pH”*

[4] *The Slope and Offset values above in **RED** are read-only, and may not be written to by the user. These values will update on accepted calibration.*

Note that this data area is contiguous, so sensor #2 slope register setting is located right after the last Tag register setting for sensor #1. The table above also shows the specific register location across all 8 sensors.

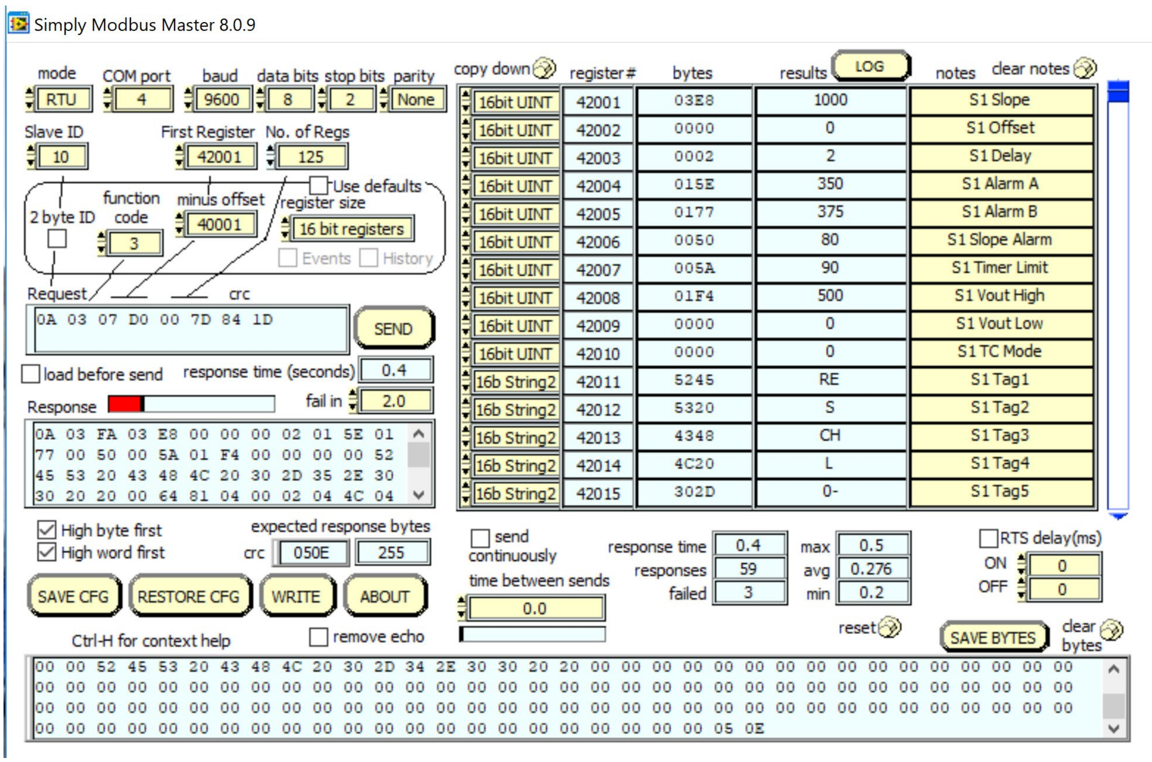


Figure 4 – SimplyModbus, Showing Sensor Info Data Block

4- SYSTEM CONFIGURATION/CAL – MB 43001 start, 4 registers (WRITE ONLY)

CONFIGURATION OF SETTINGS

This map section is somewhat unique, as it is special read-write window. User entered data changes are all made here through the same secure 4-register window. There is nothing to be "read" here, as these are the data entry functions for more complex configuration changes and calibrations. The result of calibrations must be determined by the result-flags from that specific sensor.

For optimum security, these areas are tied to the appropriate system functions using a router scheme where either the sensor or the main system settings are specified as part of the 4-register data write. The window registers for writing data are always at 43001-43004. See below. All 4 elements are part of the full write commend. Only MB functions 16(10 hex) or 6(6 hex) are allowed here. Data which may be written to is highlighted in **GREEN** in the SENSOR INFO table of section 3 above. Data marked in **RED** may not be written, as that data results from a calculated function.

An optional unlock code can be included if the "Wr Lock" feature is enabled on the MetriNet. This is a write protection lock-out, so when enabled on the MetriNet, a serial calibration or configuration change request must include the proper unlock code for every write-command, or the command will be ignored.

Example 1 -

CONFIGURATION (4 Registers)

Register	Data Type	Sensor/system	Description	Data Format
43001	UINT(16)	1	1-8 Sensor, 9 is system	1=1
43002	UINT(16)	42003	Specific Modbus Register	42003=42003
43003	UINT(16)	20	Data Value	Specific to Reg
43004	UINT(16)	0	Optional Unlock Code	User defined

Raw Hex Byte MB SEND => 0A 10 0B B8 00 04 08 00 01 A4 13 00 14 00 00 88 34

The above example would attempt to write a value of 20 to location 42003 of sensor #1 – at MetriNet slave address #10. Looking above at the map of the sensor info data, this means that the user is trying to update the DELAY setting of sensor #1 to 2.0 minute. No lock is required here, so that register simply contains 0. Once this is written, the user can either check the specific flags for that sensor to see the results, or simply re-read the SENSOR INFO block of data to see that the value has been updated.

This data is sent through SimplyModbus as shown in figure 5 below. Note that the value for S1 Delay register 42003 on the left window has changed to 20.

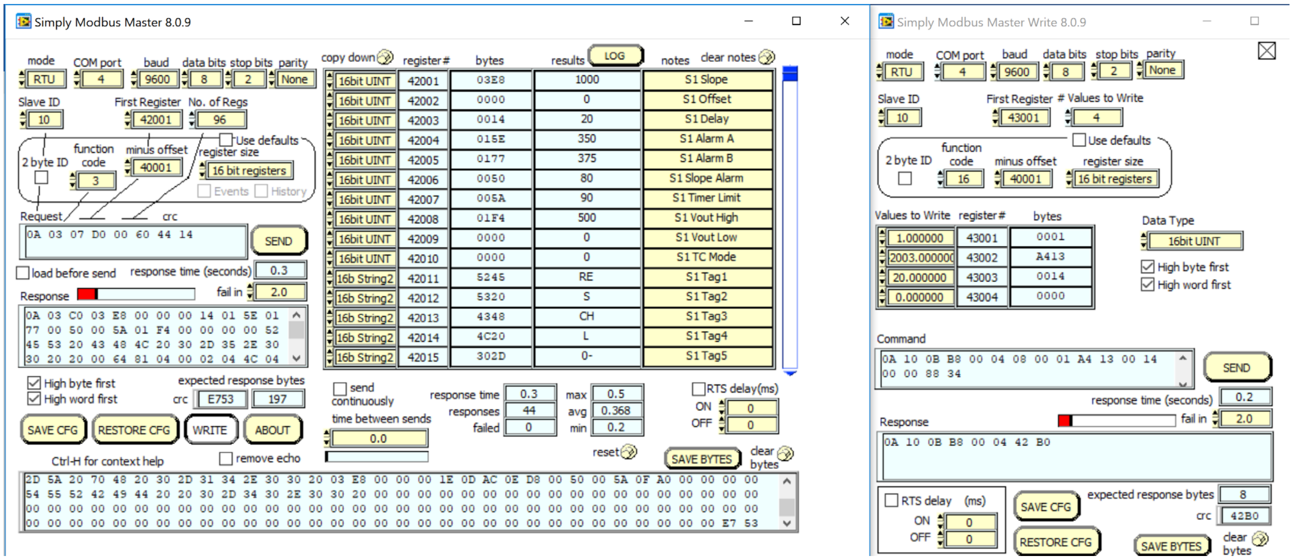


Figure 5 – SimplyModbus, Change S1 Delay Setting to 2.0 Via MB Func 16.

Example 2 -

CONFIGURATION (4 Registers)

Register	Data Type	Sensor/system	Description	Data Format
43001	UINT(16)	1	1-8 Sensor, 9 is system	1=1
43002	UINT(16)	42010	Specific Modbus Register	42010=42010
43003	UINT(16)	1	Data Value	TcMode=Fahrenheit
43004	UINT(16)	0	Optional Unlock Code	User defined

Raw Hex Byte MB SEND => 0A 10 0B B8 00 04 08 00 01 A4 1A 00 01 00 00 45 F1

This example illustrates an entry to the TcMode setting of sensor #1 temperature, which will alter the temperature display to Fahrenheit degrees by writing a value of 1 to Modbus register 42010. See the sensor register map on the previous page for sensor Modbus register numbers. The MetriNet slave address is #10. No lock is required here, so that register simply contains 0. This is a good command to use to verify communications as the MetriNet displays the temperature for the selected sensor on the lower display, so you can immediately see that the command is working correctly. Once this is written, the user can either check the specific flags for that sensor to see the results, or simply re-read the SENSOR INFO block of data to see that the value has been updated.

Example 3 -

CONFIGURATION (4 Registers)

Register	Data Type	Sensor/system	Description	Data Format
43001	UINT(16)	5	1-8 Sensor, 9 is system	5
43002	UINT(16)	42077	Specific Modbus Register	42077=42077
43003	UINT(16)	5000	Data Value	5000
43004	UINT(16)	123	Optional Unlock Code	User Defined

Raw Hex Byte MB SEND => 0A 10 0B B8 00 04 08 00 05 A4 5D 13 88 00 7B 21 73

So, the above example would attempt to write a value of 5000 to location 42077 of sensor #5 – at MetriNet slave address #10. A lock code is included this time, as the lock option is set on the MetriNet. Looking above at the map of the sensor info data, this means that the user is trying to update the ALARM B setting of sensor #5 to 5.000. Once this is written, the user can either check the specific flags for that sensor to see the results, or simply re-read the SENSOR INFO block of data to see that the value has been updated.

Note how the window register used in every example is always the same, 43001- 43004. All data writes occur through this 4 register window.

CALIBRATIONS

In addition to sending updated value to several registers of the system, sensor calibrations can be made from the serial interface if absolutely required, but the user must fully understand the weaknesses of this "blind" approach. Values can be sent directly to the sensor to force a specific calibration point on a user entered value.

NOTE: While calibration over the Modbus serial interface is possible, it is not recommended. ATI always recommends removal, cleaning, and inspection of all sensors prior to calibration at LCD/keypad. This avoids the possibility of calibration on unknown solutions or standards, and also avoids the possibility of erroneous calibrations resulting from residual foulants or undetected sensor damage. Sensors can be calibrated quickly and easily via the user interface on the MetriNet transmitter.

Calibrations are done the same way as they are for configuration, using the same 4-register "window" structure used in the CONFIGURATION cases. However, the user must use unique register values for each of the calibration function calls. There are 5 total calibration registers, but not all of them apply to every sensor. Consult M-Node manual for details on how calibrations work and which registers are utilized for each version.

Universal Calibration Window Registers (unique to sensor type, not all apply. See M-Node manual.)

43006 **Calibrate Sensor Temperature Element**
43007 **Calibrate Sensor Span**
43008 **Calibrate Sensor Zero**
43009 **Calibrate Sensor Offset**
43010 **Sensor Reset Defaults**

Example 1 -

CALIBRATION (4 Registers)

Register	Data Type	Sensor/system	Description	Data Format
43001	UINT(16)	2	1-8 Sensor, 9 is system	2
43002	UINT(16)	**43006**	Specific Modbus Register	43006
43003	UINT(16)	240	Data Value	240
43004	UINT(16)	0	Optional Unlock Code	0

Raw Hex Byte MB SEND => 0A 10 0B B8 00 04 08 00 02 A7 FE 00 F0 00 00 57 27

So, the above example would attempt to write a value of 240 to calibration register location 43006 (Cal Temperature) of sensor #2 _– at MetriNet slave address #10_. This means that the user is trying to calibrate the temperature element of sensor #2 to 24.0C. No lock code is included in this case, as lock option is OFF in the MetriNet. Once this is written, the user can either check the specific flags for that sensor to see the results, or simply re-read the SENSOR MEASURE block of data to see that the value has been updated.

Note below that once the data above is written to the proper cal/config window, the S2 Temperature value at 41019 has been updated to 2400 (24.0 C,) so calibration was successful.

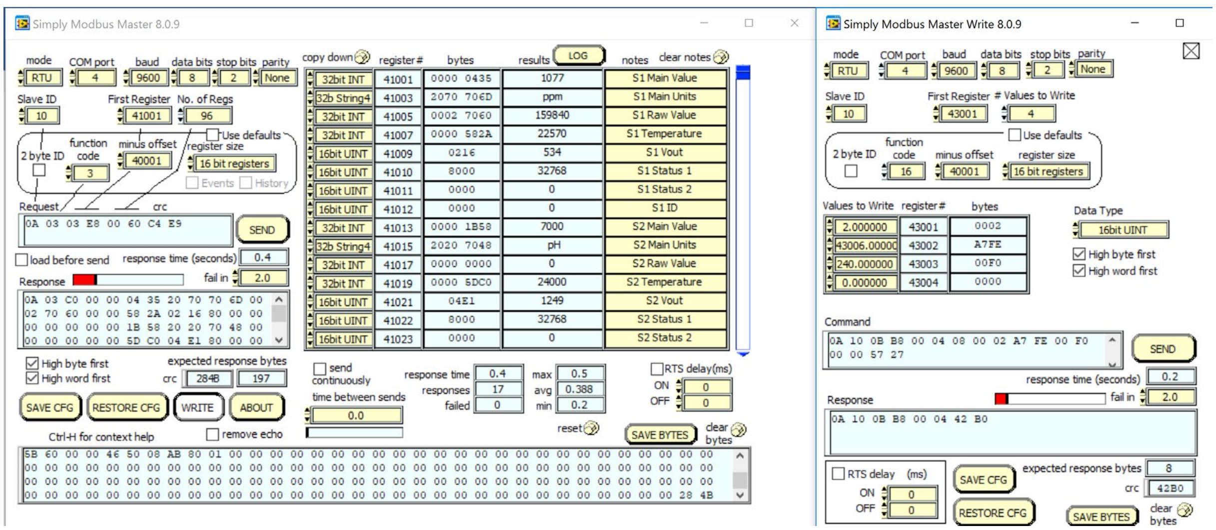


Figure 6 – SimplyModbus, Calibrate S2 Temperature to 24.0C Via MB Func 16.

1.14 Metrinet OPC-UA Advanced Example

To aid in the set-up of PLCs and computers, a brief example is built here around a mainstream industrial OPC tool called “KEPServerEX.” This OPC UA tool allows complete visualization of the entire multi-sensor map, and also allows the user to set data types for all the objects in the Modbus map. This tool represents a very comprehensive utility for a real industrial application, as it can move the collected data out from the PC in a number of ways. The example assumes that the user is already familiar with this tool or similar OPC tools, but even if they are not, it points out some common set-up issues that more adanced can aid in sorting out connection problems to the MetriNet. This tool is very similar to many PLC Modbus OPC configuration tools on the market today.

KEPServerEx
KEPServerEX is an OPC UA based tool that provides numerous hardware and protocol driver tools for connecting a wide number of industrial devices. By running as a background PC live server application, the tool allows the user to group vastly different types networks together on one PC and collect data in a common format. In addition, KEPServerEX provides a wide range of advanced features that allow logging, diagnostics, and movement of data to local servers or even Internet servers online in IIoT applications.

In the following application, the project is built around a KEPServerEx 6.5 suite that includes only a Modbus RTU serial driver. Many other drivers can be added in parallel.
https://www.kepware.com/en-us/products/kepserverex/suites/modbus-suite/

As shown is section 1.13, there are 4 distinct areas in the MetriNet Modbus memory map. Three of these areas are contiguous read-only data areas, and the fourth is a special window area used for writing new data to the sensors. There are numerous ways to break this map up in networking OPC tools, but an example is shown here that creates separate “devices” in an OPC server that all have different contiguous memory areas. Although the “devices” are really all just data blocks from the same slave #10 MetriNet/Q52, we break the map up this way so that only the live sensor data area is continuously polled for updates by the OPC tool. The other sections may be manually polled as needed, as they only change when new data values are written. This limits network bandwidth to focus on only the data which is continuously changing. This is only one of many ways to break this application down, and KepServer provides tremendous flexibility for other options.

To begin to build this application, the entire Q52 access is built into one “channel” called “USB RS485 Bus,” and each “Device” is then set to represent a different section of the 4 areas of the Modbus memory map.

Once the channel configuration has been established, the memory map sections are set up as separate devices to collect data for display in the same way as the Modbus map structure. There are many ways to accomplish this, but this method is chosen in the example because the “Q52 Sensor Data,” which is live, can be polled continuously. The static data for “Sensor Info” and “System Info” can be polled as required – as it doesn’t change unless a change is made by the user. See figure 7 below.

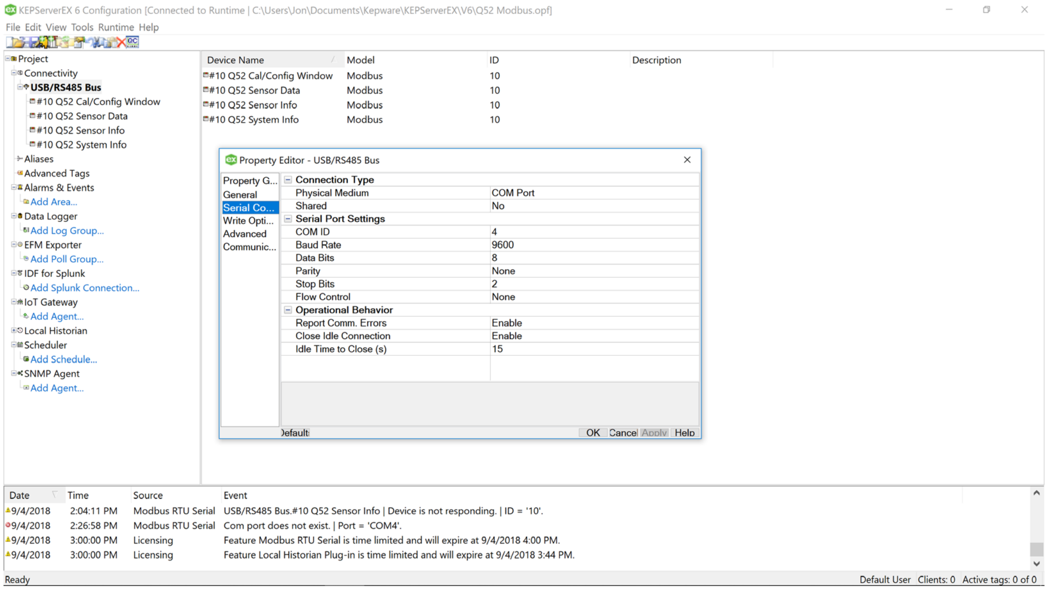


Figure 7 – KEPServer Channel Set-up For One RS485 Serial Port.

Setting up the sensors is then a simple matter of plugging in the register locations for all variables in all the map areas/devices. We use S1, S2, and S3 in the tag names as an easy way to identify the individual sensors, but note that the units of "ppm", "pH", and "NTU" also appear in the data blocks and show the user which sensor is located at that register block. By setting the KEPServer application up this way, the user can visualize all the sensor live measurements in the section of data at "Sensor Data," and then jump over to view the more static values in the other screens as needed.

Going through each device section of the application, you can see the register values are set to the contiguous memory areas referenced in the operating manual map shown earlier in each case. <u>Note that in this tool "address" actually translates to "register" due to a specific setting in the tool. The "address" and "register" names in Modbus mean two different things, and a Modbus register number of 40001 is actually at Modbus address 40000 – so these two terms are off by one count.</u> In this KEPServer tool, a special setting called "Zero Based Addressing" allows the register value to appear as the address value in our example – so it matches the manual memory map above. Note also that Modbus Byte Order feature must be enabled and properly configured to read the high-low byte order correctly. See figure 12 below for byte order and zero-byte addressing setting. The final device settings are all shown below in figure 8, 9, 10, 11.

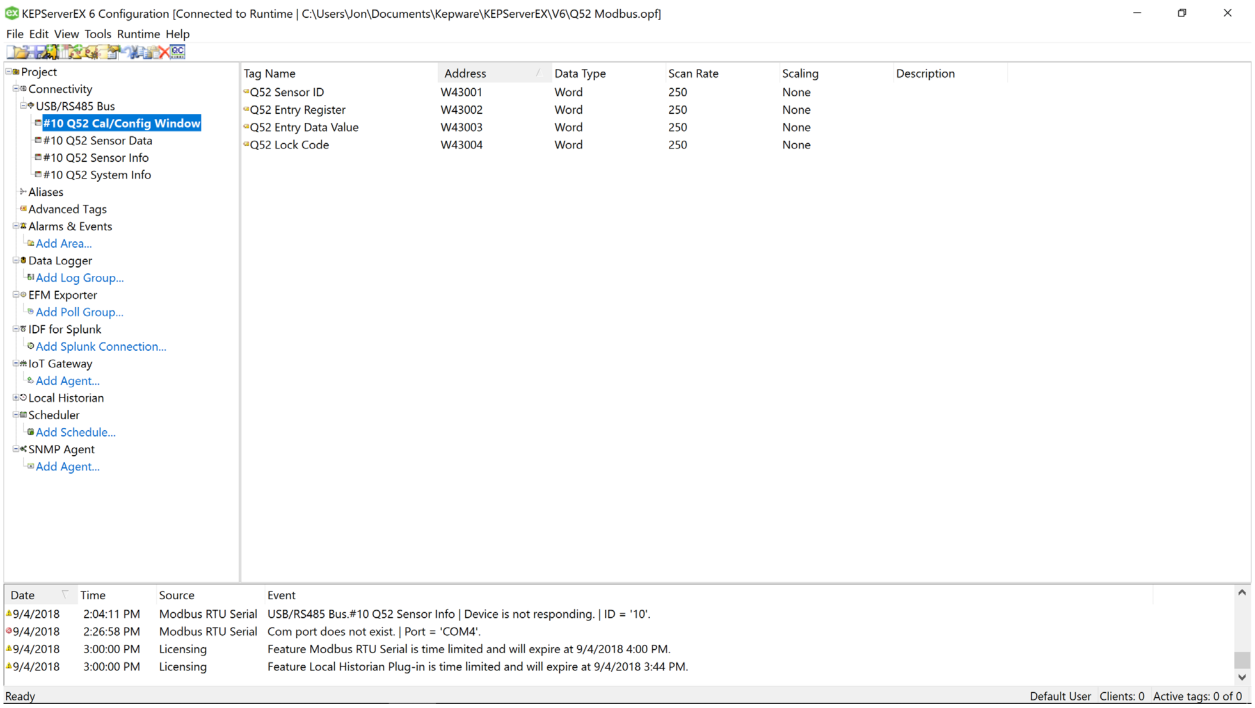


Figure 8 – KEPServer Cal/Config Window Device Registers.

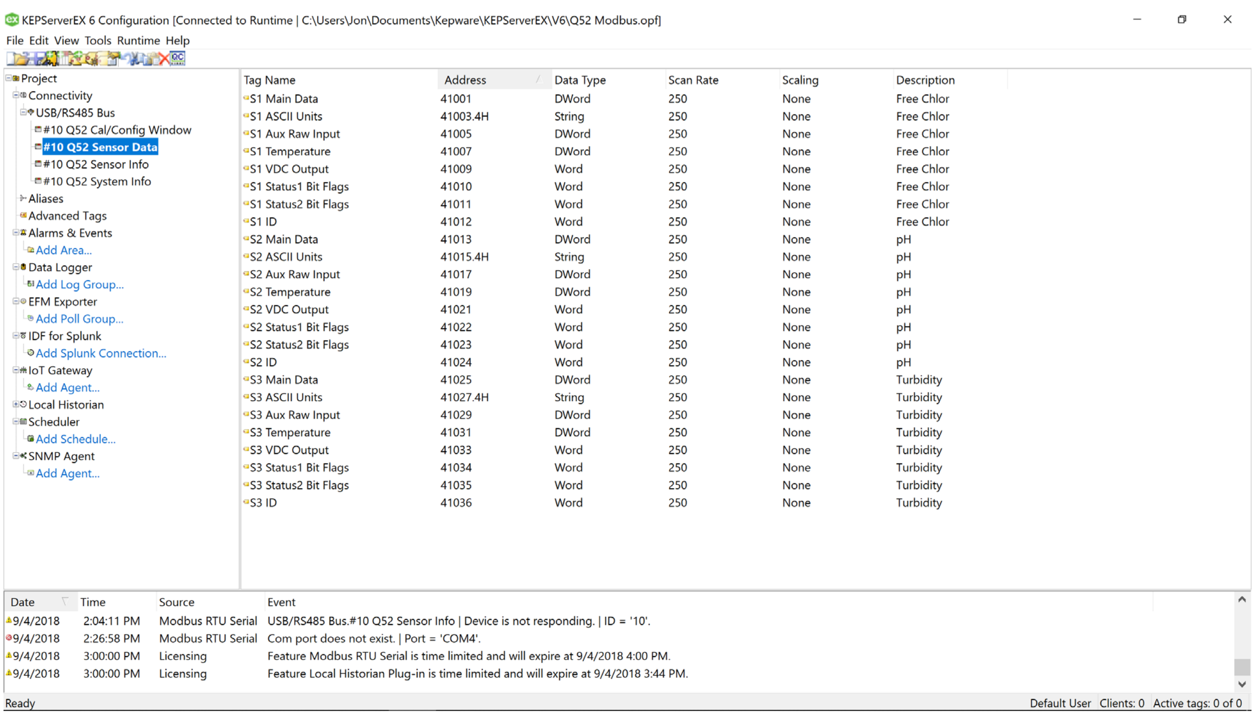
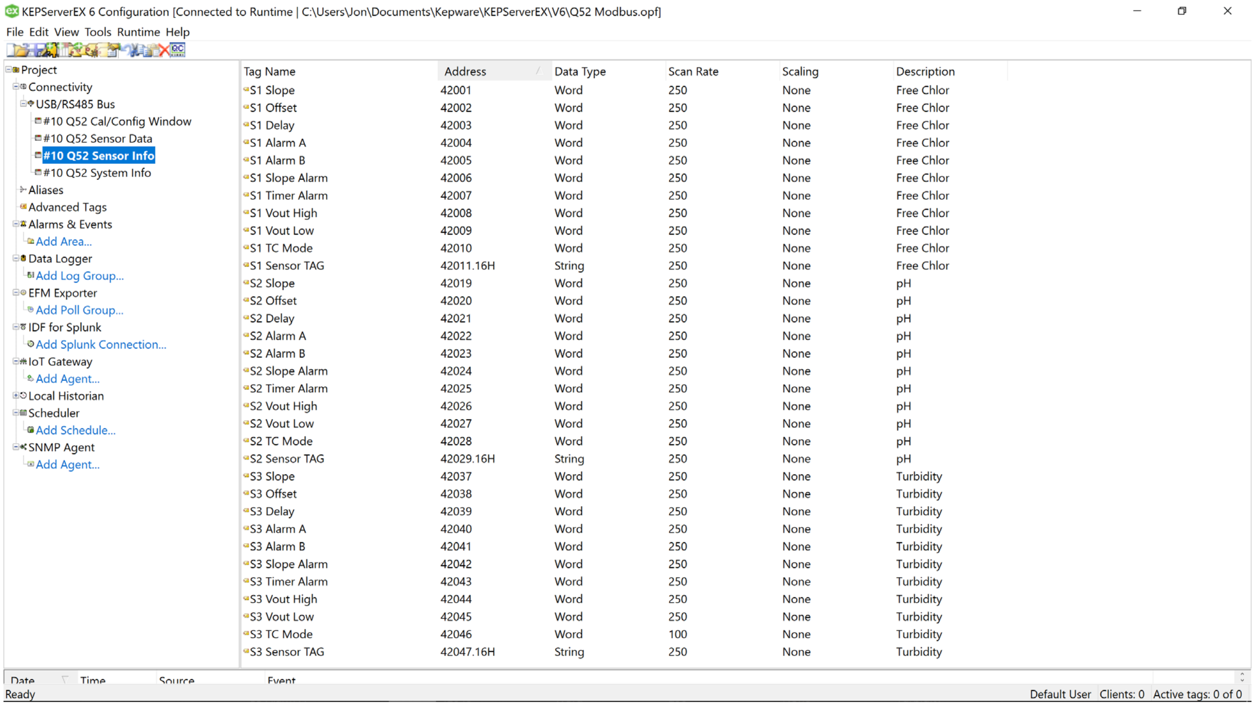
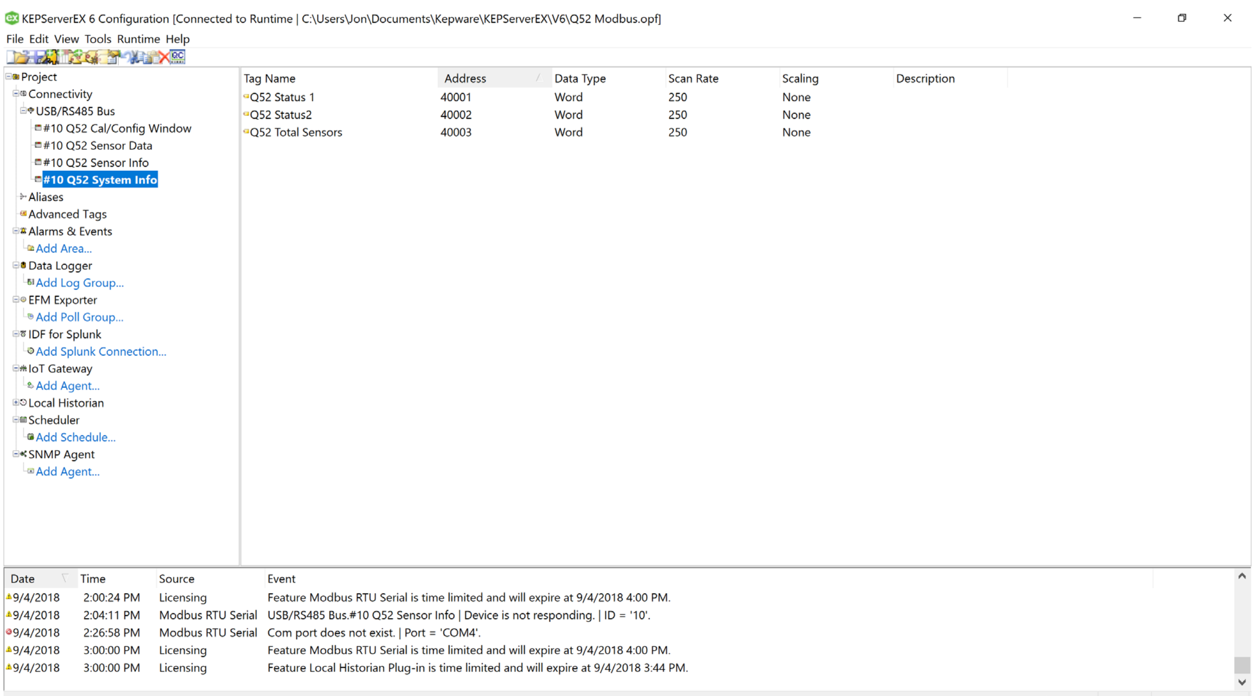


Figure 9 – KEPServer Sensor Data Device Registers



Tag Name	Address	Data Type	Scan Rate	Scaling	Description
S1 Slope	42001	Word	250	None	Free Chlor
S1 Offset	42002	Word	250	None	Free Chlor
S1 Delay	42003	Word	250	None	Free Chlor
S1 Alarm A	42004	Word	250	None	Free Chlor
S1 Alarm B	42005	Word	250	None	Free Chlor
S1 Slope Alarm	42006	Word	250	None	Free Chlor
S1 Timer Alarm	42007	Word	250	None	Free Chlor
S1 Vout High	42008	Word	250	None	Free Chlor
S1 Vout Low	42009	Word	250	None	Free Chlor
S1 TC Mode	42010	Word	250	None	Free Chlor
S1 Sensor TAG	42011.16H	String	250	None	Free Chlor
S2 Slope	42019	Word	250	None	pH
S2 Offset	42020	Word	250	None	pH
S2 Delay	42021	Word	250	None	pH
S2 Alarm A	42022	Word	250	None	pH
S2 Alarm B	42023	Word	250	None	pH
S2 Slope Alarm	42024	Word	250	None	pH
S2 Timer Alarm	42025	Word	250	None	pH
S2 Vout High	42026	Word	250	None	pH
S2 Vout Low	42027	Word	250	None	pH
S2 TC Mode	42028	Word	250	None	pH
S2 Sensor TAG	42029.16H	String	250	None	pH
S3 Slope	42037	Word	250	None	Turbidity
S3 Offset	42038	Word	250	None	Turbidity
S3 Delay	42039	Word	250	None	Turbidity
S3 Alarm A	42040	Word	250	None	Turbidity
S3 Alarm B	42041	Word	250	None	Turbidity
S3 Slope Alarm	42042	Word	250	None	Turbidity
S3 Timer Alarm	42043	Word	250	None	Turbidity
S3 Vout High	42044	Word	250	None	Turbidity
S3 Vout Low	42045	Word	250	None	Turbidity
S3 TC Mode	42046	Word	100	None	Turbidity
S3 Sensor TAG	42047.16H	String	250	None	Turbidity

Figure 10 – KEPServer Sensor Info Device Registers.



Tag Name	Address	Data Type	Scan Rate	Scaling	Description
Q52 Status 1	40001	Word	250	None	
Q52 Status2	40002	Word	250	None	
Q52 Total Sensors	40003	Word	250	None	

Date	Time	Source	Event
9/4/2018	2:00:24 PM	Licensing	Feature Modbus RTU Serial is time limited and will expire at 9/4/2018 4:00 PM.
9/4/2018	2:04:11 PM	Modbus RTU Serial	USB/RS485 Bus.#10 Q52 Sensor Info | Device is not responding. | ID = '10'.
9/4/2018	2:26:58 PM	Modbus RTU Serial	Com port does not exist. | Port = 'COM4'.
9/4/2018	3:00:00 PM	Licensing	Feature Modbus RTU Serial is time limited and will expire at 9/4/2018 4:00 PM.
9/4/2018	3:00:00 PM	Licensing	Feature Local Historian Plug-in is time limited and will expire at 9/4/2018 3:44 PM.

Figure 11 – KEPServer System Info Device Registers.

In addition to setting up all data sections/devices, you must ensure that the Modbus byte order is the proper Endian format, and is swapped if necessary. In addition, enabling "Zero-byte Addressing" in this tool allows the registers to be displayed as we have done here (due to address vs register +1 issue.) See figure 12 below.

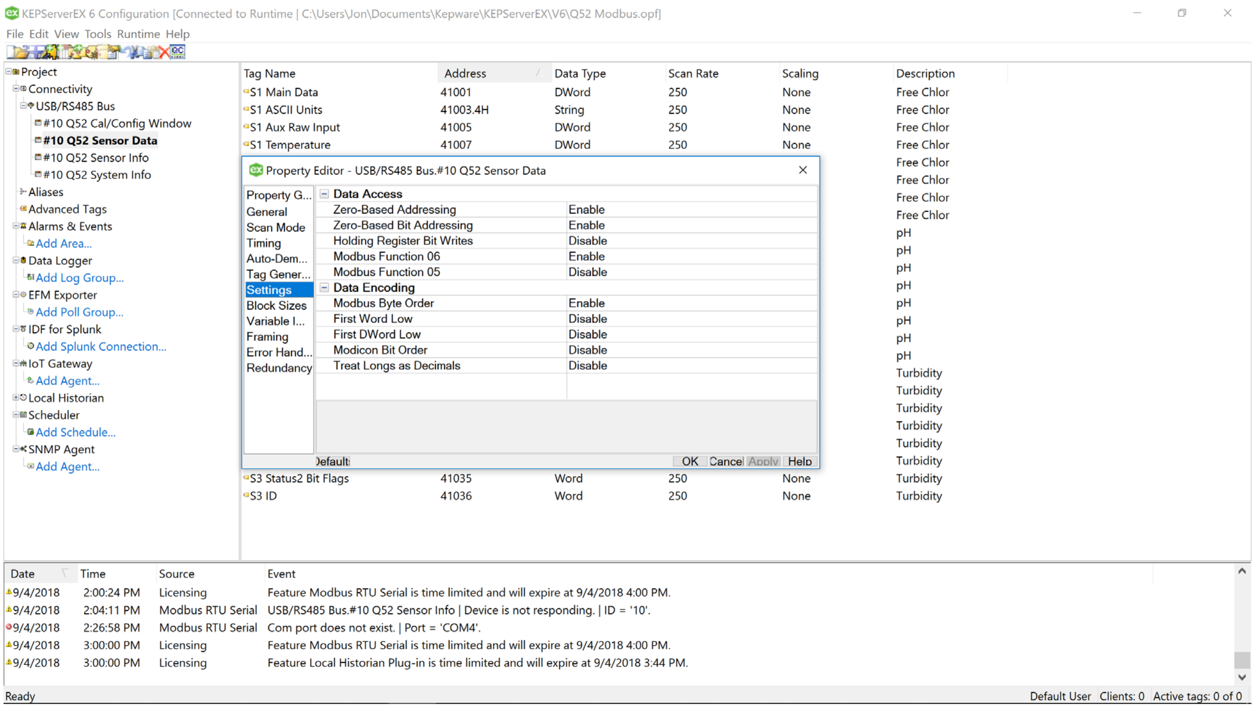


Figure 12 – KEPServer Zero-Based-Addressing and Modbus-Byte-Order Features.

IMPORTANT – Ensure Modbus byte order and register vs. address terminology is correct for your network master/tool when setting up the Q52.

Now that all data has been set-up in the tool, the server can be launched to run the current object and begin collecting data (Launch OPC quick Client.) The data can then be easily viewed in each section of the map as shown in figures 13 and 14 below, and logging can be enabled if desired. Although not shown below, System Info would be displayed in the same way.

The power of this tool is seen in the structured collection and visualization of the Modbus data, and the ability to very easily manage the different types of data collected, such as INT, DINT, STRINGS, etc. Although limited in overview in the example presentation, "tags" on data become a very powerful way to label and manage complex information. Finally, This OPC sever can manage the offload of the collected data via newer protocols like MQTT to other servers or sites, including cloud sites such as ThingWorx.

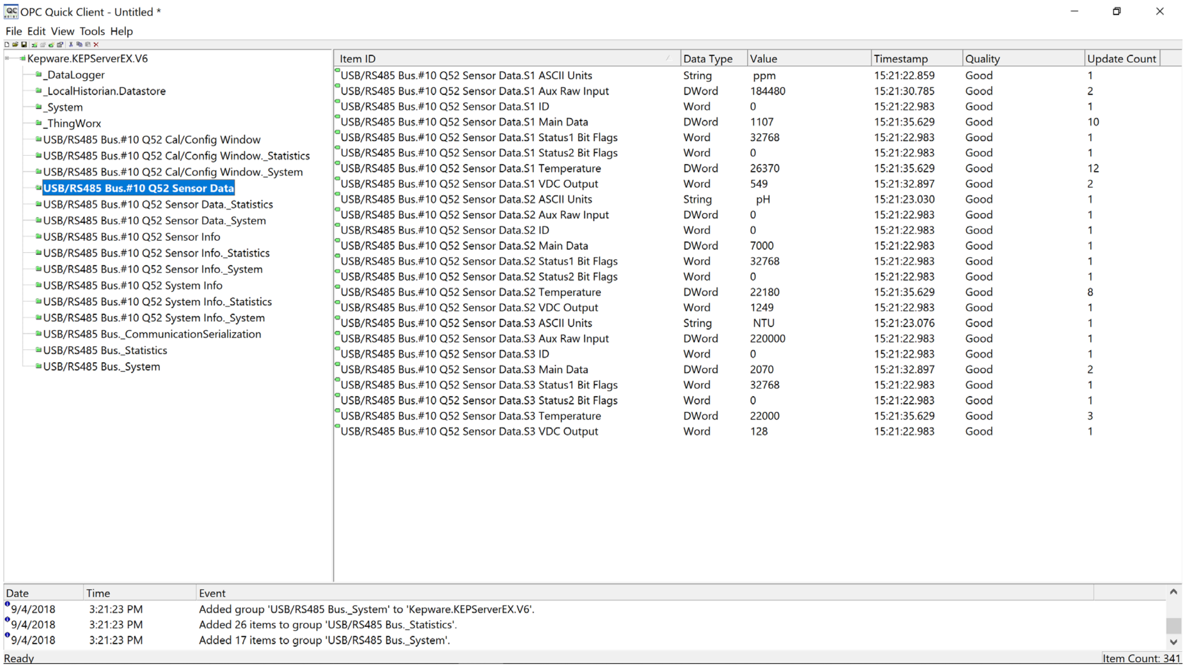


Figure 13 – KEPServer Real-Time Measurement Data (All) for Example System.

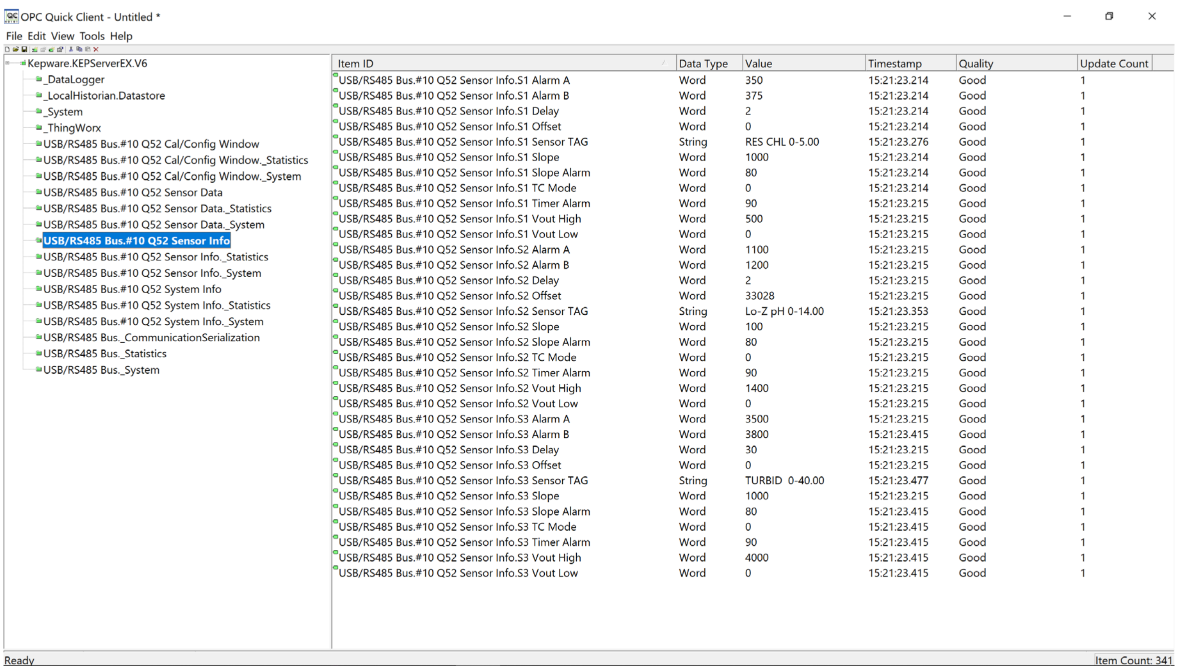


Figure 14 – KEPServer Sensor Info Settings (All) for Example System.

PRODUCT WARRANTY

Analytical Technology, Inc. (Manufacturer) warrants to the Customer that if any part(s) of the Manufacturer's equipment proves to be defective in materials or workmanship within the earlier of 18 months of the date of shipment or 12 months of the date of start-up, such defective parts will be repaired or replaced free of charge. Inspection and repairs to products thought to be defective within the warranty period will be completed at the Manufacturer's facilities in Collegeville, PA. Products on which warranty repairs are required shall be shipped freight prepaid to the Manufacturer. The product(s) will be returned freight prepaid and allowed if it is determined by the manufacturer that the part(s) failed due to defective materials or workmanship.

This warranty does not cover consumable items, batteries, or wear items subject to periodic replacement including lamps and fuses.

Gas sensors carry a 12 months from date of shipment warranty and are subject to inspection for evidence of misuse, abuse, alteration, improper storage, or extended exposure to excessive gas concentrations. Should inspection indicate that sensors have failed due to any of the above, the warranty shall not apply.

The Manufacturer assumes no liability for consequential damages of any kind, and the buyer by acceptance of this equipment will assume all liability for the consequences of its use or misuse by the Customer, his employees, or others. A defect within the meaning of this warranty is any part of any piece of a Manufacturer's product which shall, when such part is capable of being renewed, repaired, or replaced, operate to condemn such piece of equipment.

This warranty is in lieu of all other warranties (including without limiting the generality of the foregoing warranties of merchantability and fitness for a particular purpose), guarantees, obligations or liabilities expressed or implied by the Manufacturer or its representatives and by statute or rule of law.

This warranty is void if the Manufacturer's product(s) has been subject to misuse or abuse, or has not been operated or stored in accordance with instructions, or if the serial number has been removed.

Analytical Technology, Inc. makes no other warranty expressed or implied except as stated above.

WATER QUALITY MONITORS

Dissolved Oxygen
Free Chlorine
Combined Chlorine
Total Chlorine
Residual Chlorine Dioxide
Potassium Permanganate
Dissolved Ozone
pH/ORP
Conductivity
Hydrogen Peroxide
Peracetic Acid
Dissolved Sulfide
Residual Sulfite
Fluoride
Dissolved Ammonia
Turbidity
Suspended Solids
Sludge Blanket Level
MetriNet Distribution Monitor

GAS DETECTION PRODUCTS

NH_3	Ammonia
CO	Carbon Monoxide
H_2	Hydrogen
NO	Nitric Oxide
O_2	Oxygen
CO	Cl2 Phosgene
Br_2	Bromine
Cl_2	Chlorine
ClO_2	Chlorine Dioxide
F_2	Fluorine
I_2	Iodine
H_X	Acid Gases
C_2H_4O	Ethylene Oxide
C_2H_6O	Alcohol
O_3	Ozone
CH_4	Methane (Combustible Gas)
H_2O_2	Hydrogen Peroxide
HCl	Hydrogen Chloride
HCN	Hydrogen Cyanide
HF	Hydrogen Fluoride
H_2S	Hydrogen Sulfide
NO_2	Nitrogen Dioxide
NO_x	Oxides of Nitrogen
SO_2	Sulfur Dioxide
H_2Se	Hydrogen Selenide
B_2H_6	Diborane
GeH_4	Germane
AsH_3	Arsine
PH_3	Phosphine
SiH_4	Silane
HCHO	Formaldehyde
$C_2H_4O_3$	Peracetic Acid
DMA	Dimethylamine