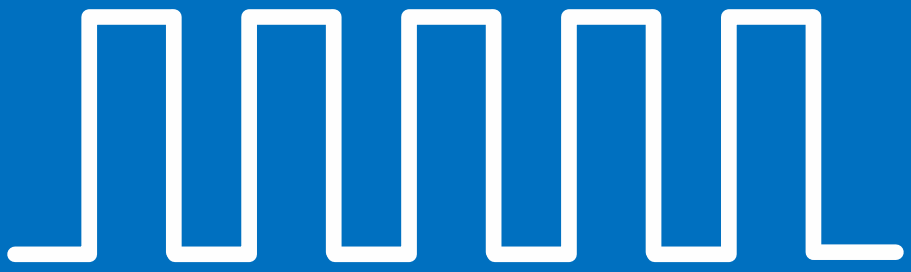
ANALYTICAL TECHNOLOGY, INC.

O & M Manual



MetriNet
Modbus-TCP
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-TCP Description

1.1 General

MetriNet Instruments are available with two forms of Modbus digital communication options: Modbus-RTU (RS485) and Modbus-TCP (Ethernet.) This manual applies only to instruments supplied with the Modbus-TCP communication option. The MetriNet utilizes a powerful built-in Lantronix XPORT Modbus bridge device to enable Modbus-TCP capability.

It is important to note that Modbus-TCP devices only communicate with other systems that are running the Modbus application protocol on Ethernet. You cannot plug a MetriNet into your office Ethernet network and expect to have the MetriNet talk to your computer, unless you have a Modbus protocol application running on that computer.

The discussion of standard Modbus and Ethernet are vast, and well beyond the ability to discuss here in great detail. The documentation for this option assumes working network knowledge by the user.

1.2 Modbus-TCP Communication

It should be understood that Modbus-TCP is simply an application layer protocol that is transferred over an Ethernet hardware link. The word "Ethernet" simply refers to the common physical cable, perhaps running to an office PC.

In the OSI model, “Ethernet” is the lower part of the model, the physical transfer method or the hardware. It says nothing about the way information is transferred, which is specified near the top of the OSI model. In the common office network, many different standard communication protocols are operating during normal office use, like IP, TCP, etc. None of these are designed to handle the industrial protocol formats, so that interface must be handled by a specific program that recognizes the format. Because of this, a Modbus-TCP device cannot be directly connected to your office network for transferring information unless an additional program exists to decode the Modbus frames.

Per the ***Modbus Messaging on TCP Implementation Guide V1.0b***, authored by Modbus.org, a 5-layer Internet model is used for Modbus-TCP instead of the familiar three-layer model for TCP/ASCII over serial line. This new standard encapsulates standard Modbus function code and data contents into a higher level TCP protocol.

The physical layer is not specifically mentioned in the guide, therefore, the wiring infrastructure generally just follows standard Ethernet wiring practices.

The Modbus "bus" structure in this case is an IP addressed direct link, not a bus at all, and slave addresses become IP addresses. No slave ID is used in the MetriNet, only an IP address. Also, the master/slave relationship in TCP/ASCII becomes more of a client/server relationship in Modbus-TCP – where multiple clients (masters) initiate requests to servers (slaves) for information. So, unlike the serial implementation of bussed RTU/ASCII, the TCP implementation becomes more "multi-master" in nature. However, even with all these variations, Modbus-TCP data and Modbus-RTU/ASCII data are in the same format.

1.3 Modbus-TCP Transmission Details

The data for the protocol is packed into a specific structure inside a standard TCP Packet. A user application program simply decodes the structure inside the received TCP or UDP packet. To create the Modbus-TCP frame, the standard Modbus data frame PDU (Protocol Data Unit) is appended by a new MBAP header (Modbus Application Protocol) to make a larger transmission frame called the ADU (Application Data Unit.) In order to send the new ADU over TCP, the registered port number 502 is used by Modbus.org.

MODBUS PDU

The Modbus protocol is a messaging structure, widely used to establish master-slave communication between intelligent devices. In serial format, 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 TCP format strips out the slave address and CRC and just uses the function code and data. So, the basic structure of the PDU frame section is simply:

|<- Modbus PDU ->|
[FUNCTION][DATA]

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.

MODBUS ADU

Modbus-TCP adds a header to the PDU to get the full ADU transmission frame –

|← MBAP HEADER ->| |← Modbus PDU ->|
[TRANS ID] [PROT ID] [LENGTH] [UNIT ID] [FUNCTION][DATA]
|← MODBUS-TCP ADU ->|

Where:
Transaction ID = 2 bytes set by client to uniquely identify the request
Protocol ID = 2 bytes set by client (always 00 00)
Length = 2 bytes identifying the number of bytes in the message
Unit ID = 1 byte set by client to ID a slave connected on a remote serial line

MODBUS REGISTERS AND COILS

The PDU data portion of the Modbus-TCP follows the exact same register format as the serial form of the Modbus-RTU protocol. 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 MetriNet data is aligned in the 16 or 32 bit values of the holding registers from 40001 to 49999, using only:

Function 3 – Read Holding Registers
Function 16 – Write Multiple Holding Registers

For more information, 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.

1.4 RJ45 Cable Connection

The cable used for Modbus-TCP communication should meet the CAT5 standard defined by the Electronic Industries Association and Telecommunications Industry Association. It is readily available in lengths up to 100 ft. (30 m) with plugs on each end.

To install an Ethernet cable in the MetriNet, pass the unterminated cable through the cable gland nearest the location of the RJ45 connector on the Modbus-TCP option board. Termination of Cat5/5e/6 cables is very easy and can be completed quickly with the commonly available RJ45 crimp tool.

Figure 1 - RJ45 Crimp Tool

While professional results are optimized with the use of the termination tool and custom cable lengths, patch cord style connection can be completed in some cases by passing the finished RJ45 connectors through the MetriNet cable gland. The connector will fit through the plastic part of the cable gland, and the rubber grommet can be slit (some RJ45s may be too big for this.) Once the connector is inside the enclosure, simply plug it into the jack provided on the Modbus-TCP communication board. Be sure to adjust the rubber insert in the cable gland so that the slit is on the bottom and then tighten the gland to seal around the wire.

1.5 Modbus-TCP Wiring Port

Once the Ethernet cable is properly terminated, it can be connected to the RJ45 port on the MetriNet. The MetriNet Modbus-TCP port is located in the lower right corner of the back panel of the instrument. Connection is made via RJ45 connector. The MAC ID for the interface is listed on the port instrument label.

Once the Ethernet cable is connected, apply proper 12-24VDC power to the MetriNet and make sure that **"Ethr"** is selected in the OPTIONS menu for **"^Host Comms."** The other end of the Ethernet cable can be connected to a PC so that the IP address can be configured.

Figure 2 - Modbus-TCP Interface Location

There are two LEDS on the face of the RJ45 Cable interface. To the left of the RJ45 (towards top of instrument) is the "link" LED –

Off	**No link**
Amber	**10 Mbps**
Green	**100 Mbps**

To the right of the RJ45 (near bottom of instrument) is the "activity" LED –

Off	**No activity**
Amber	**Half duplex**
Green	**Full duplex**

1.6 Setting PC to Locked IP Address (Disable DHCP)

The MetriNet is shipped with a default fixed IP address of 192.168.0.30. A new value can be set with a PC in one of two ways: Via an internal webpage on the MetriNet, or through the use of a special Lantronix software PC tool.

Before digging into IP address details, it is important to note that an existing network IP address cannot be reached unless both the client (your PC) and server (the slave, MetriNet) are on the same net/subnet/mask level. For example, assuming the Metrinet has an address at 192.168.0.30/255.255.255.0 and the client trying to make the connection is at 192.168.1.10, the MetriNet cannot be reached at the 192.168.1 level. When setting an IP with a PC, it is easiest to first lock the PC at a fixed IP address to ensure it can reach whatever IP is already present now, or desired, on the MetriNet. The vast majority of industrial applications will use some fixed IP address and not DHCP for automatic settings.

To lock your PC at a specific IP network address range -

1) Set laptop NIC port on a locked IP and subnet to stop DHCP (automatic IP) service. In IPV4 properties on your PC's Ethernet connection port, select "Use the following IP Address" and set the PC to fixed IP of -
IP Address = 192.168.0.10 (this PC network address, not the MetriNet)
Subnet Mask = 255.255.255.0

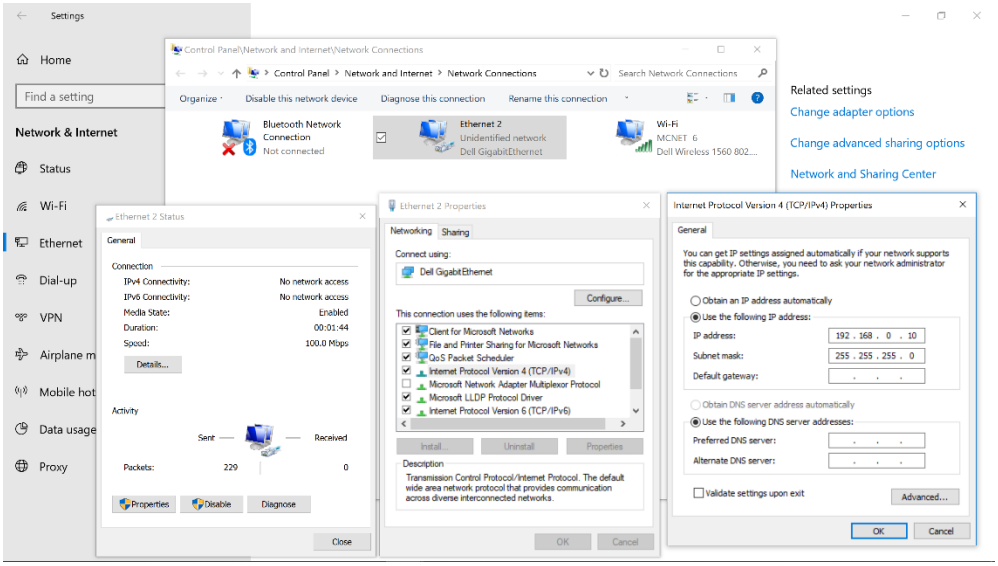


Figure 3 – Force PC to Specific Network Subnet

2) Once this IP/subnet is set, click OK to save and then close it all out. Turn PC off then back on to place it on the new IP address set in step 1. The PC will now be able to access any network device on 192.168.0.X (except for 0 and 10,) as the subnet mask 255.255.255 locks in the first three variables. The default value for the MetriNet is 192.168.0.30, so it can now be reached by this PC.

1.7 Setting MetriNet Address

Once the user's PC is on the proper net/subnet/mask level, the IP address for the MetriNet can be set in one of two ways: Internal port Web page or Lantronix Device-Installer software. The MetriNet port has a web page built into it, and it is very easy to quickly change the IP address from any Internet browser whne the port is already on the same net/subnet as the PC. The benefit of using the free Lantronix software is that it can also indicate if the MetriNet and PC nets/subnets don't match.

Option #1 – Internal Web Page

The MetriNet interface has an internal webpage, which can be accessed using any web browser, that enables the IP address to be changed very quickly. Once the PC is set to the same net/subnet as the MetriNet, simply type the IP address is the browser path window to get to the internal web page. See figure below for an example where the MetriNet is sitting at 192.168.0.30, and the PC is at 192.168.0.10. Using Firefox as the browser, the first page that comes up on connection is the password page.

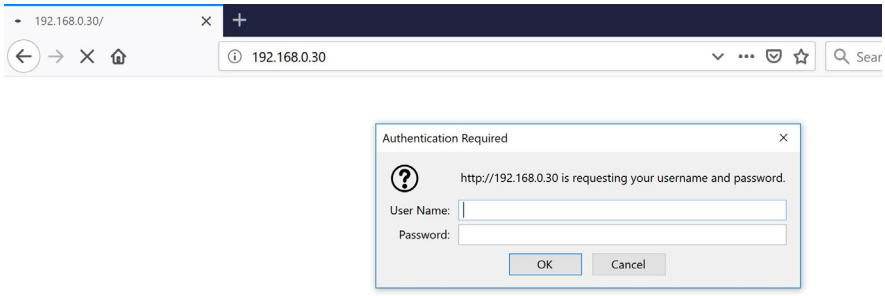


Figure 4 – Reaching Internal Lantronix Web Page with Firefox Browser.

At the password page, just click on OK and continue. Do not enter a password or name.

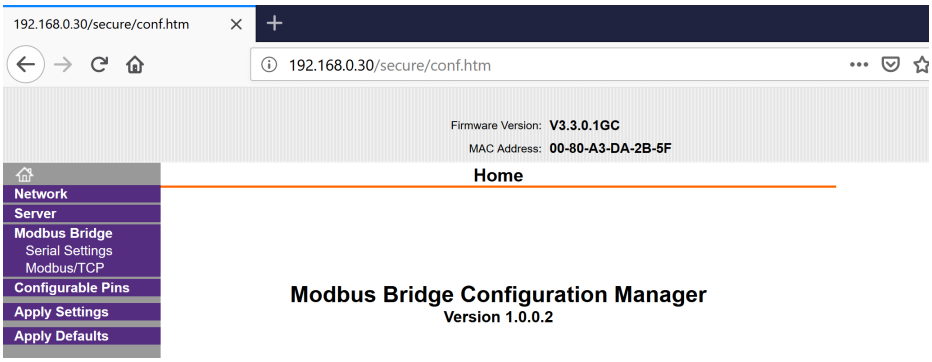


Figure 5 – Internal Modbus-TCP Manager Page

At the Manager page, click on Network.

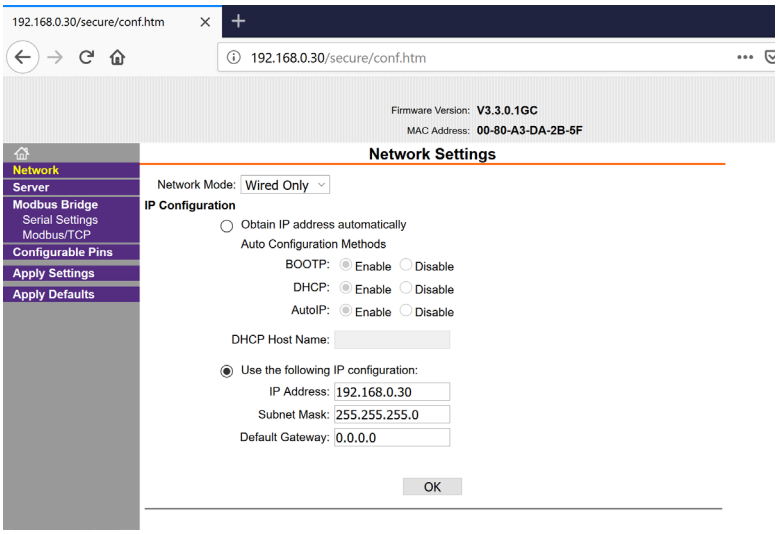


Figure 6 – Internal IP Address Page

On this Network page, enter the new IP address and mask desired. Then click on OK. Finally, click on Apply Settings and changes will be saved, along with a reset of the module. DO NOT CHANGE ANY OTHER SETTINGS.

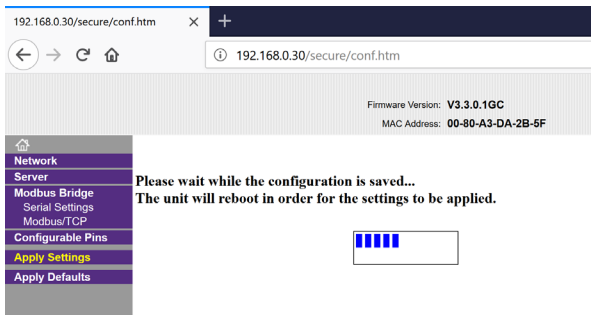


Figure 7 – Internal Modbus-TCP Reset Screen

Option #2 - Device Installer

The second option for configuring the IP address is to use the free program called "DeviceInstaller" from Lantronix - http://www.lantronix.com/products/deviceinstaller/

Once this program has been installed on a PC, plug in the MetriNet and connect an Ethernet cable from the MetriNet communications port to an Ethernet port of the PC running the program. Then, launch the DeviceInstaller program.

The DeviceInstaller program automatically searches for any connected Lantronix interfaces connected to the PC – even on other nets. The big benefit of using this program is that it can force an IP address change even if the MetriNet IP is not at the proper net/subnet level. It does this as it accesses the MAC ID of the interface. This can be a big benefit if the MetriNet IP address is unknown or at the wrong address. The only caveat is, once the MetriNet is detected, the "assigned" subnet must be the same as the current PC subnet.

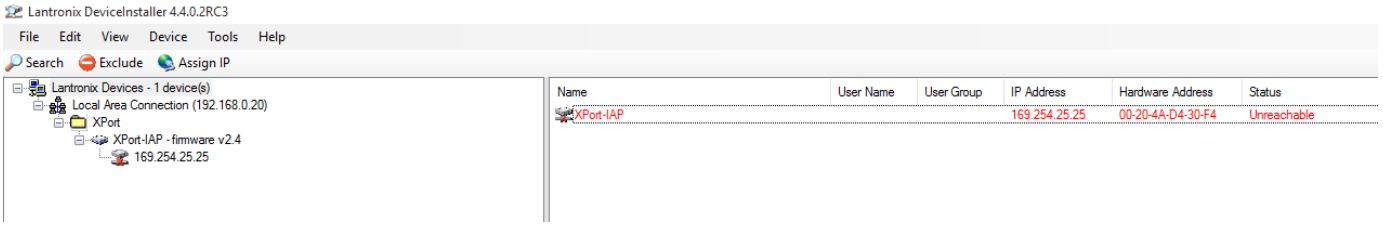


Figure 8 - DeviceInstaller Identifies MetriNet Interface Connection

In this example above, the MetriNet IP for the actual interface is found outside the PC's reach. The PC is at 192.168.0.20 and the MetriNet here is at 169.254.25.25. For this MetriNet port, we will force it to be at 192.168.0.21. To do this, simply click on the actual IP address of the part now, "**169.254.25.25**" above. A window will pop up indicating "**The configuration could not be retrieved from the device**", and just hit "**OK**." Now the following detail screen will be shown –

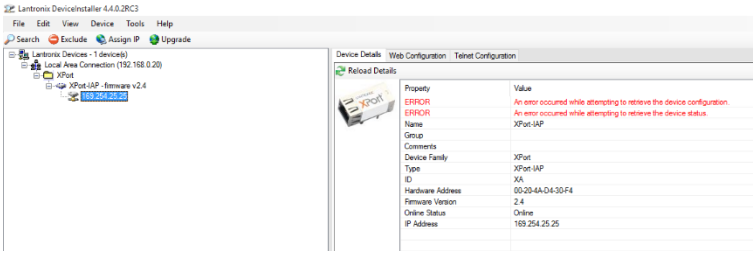


Figure 9 - DeviceInstaller Default Port Details

At this point, select "**Assign IP**" from the upper bar. A window will pop up asking if you want to set the IP automatically, or to set it manually. Choose "**Assign a Specific IP Address**." The following screen will appear with entry fields for IP, subnet, and gateway.

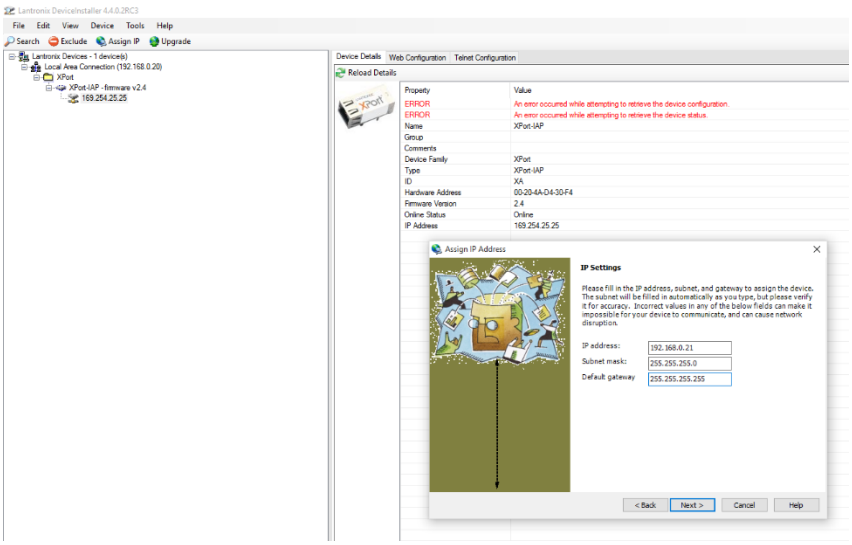


Figure 10 - DeviceInstaller Manual IP Settings

After the IP/subnet/gateway info is entered, hit "**Next**", then click on "**Assign**." Wait for "progress of task bar" to complete the operation. You will now get a pop-up screen again that shows "**The configuration could not be retrieved from the device**", just click OK. Then click "**Finish**" to clear window -

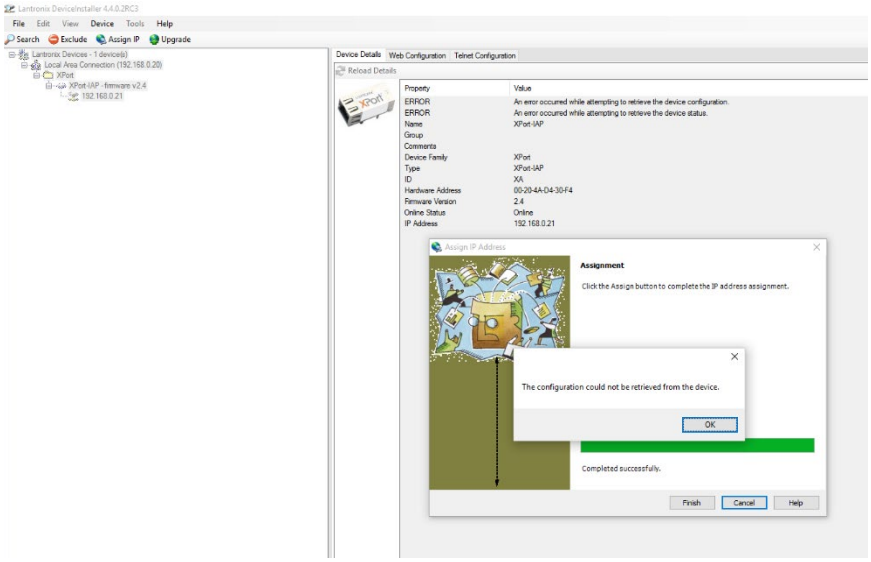


Figure 11 - DeviceInstaller Completed Port Assignment

The updated IP screen detail for the port will show the following new information

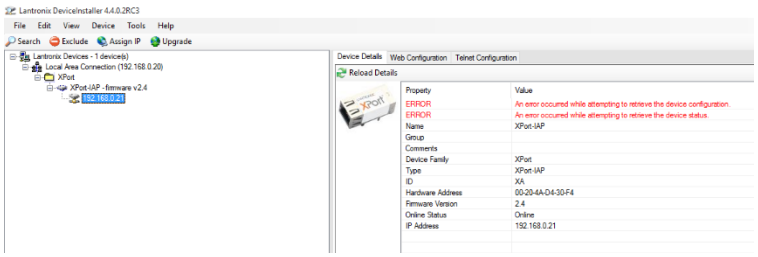


Figure 12 - DeviceInstaller Updated Port Details at New IP

You can now close this program, and the MetriNet is fixed on these new IP settings. You must cycle power to the MetriNet for this new IP to take effect.

1.8 Finding Lost IP Address

If the current IP address is lost for some reason and was not written on the MetriNet enclosure, there are two easy ways to figure it out. The first method was used in the prior example with DeviceInstaller.

In addition, a PC tool like “Wireshark” can be run on the port. At some point in the dialog right after the MetriNet is plugged in, a “**Gratuitous ARP**” will be broadcast from the MetriNet to announce its IP. You can see it in the figure below and it can be easily identified because the MAC ID for the interface is also in that line (00:80:A3:DA:2B:5F.) The MAC ID for the MetriNet is written on every Lantronix interface module.

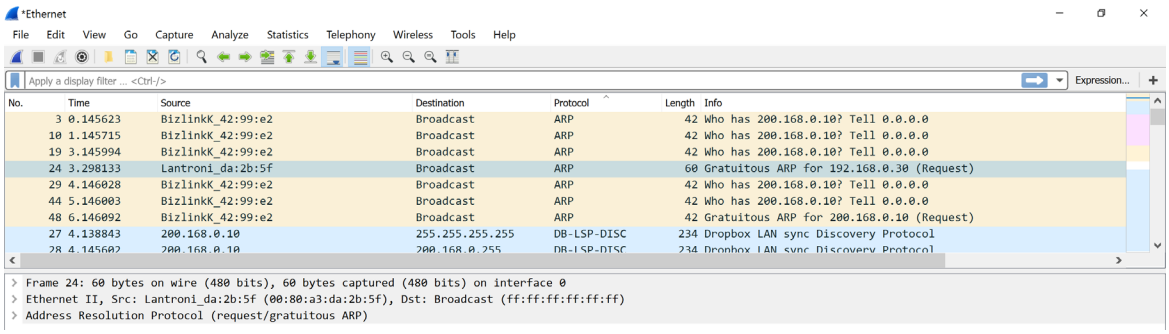


Figure 13 – Wireshark Finding Lost IP Via ARP Broadcast

1.9 Modbus-TCP Interface Operation

Once the cable is connected, the IP is set, Modbus-TCP data can be transferred across the network just like other Ethernet-based connection.

We recommend a simple master Modbus-TCP test program, by the name of **SimplyModbus-TCP**, 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/TCPmaster.htm

Here is a screenshot of the Simply Modbus-TCP 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.

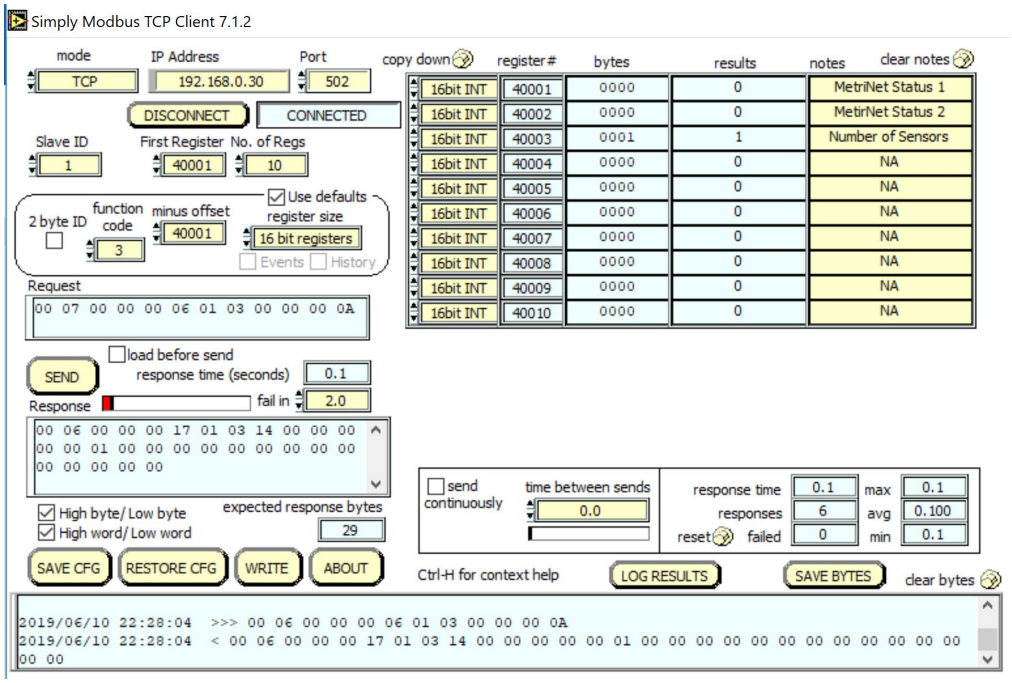


Figure 14 - SimplyModbus Client Scan of INFO block at New IP

1.10 MetriNet Modbus-TCP Data Structure

Once the TCP interface has been established at the proper IP address, Modbus-TCP operates just like Modbus RTU. 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 it is 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.

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. An example of this data block is shown in figure 14.

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	NA	
40005	UINT(16-bit)	MetriNet	NA	
40006	UINT(16-bit)	MetriNet	NA	
40007	UINT(16-bit)	MetriNet	NA	
40008	UINT(16-bit)	MetriNet	NA	
40009	UINT(16-bit)	MetriNet	NA	

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 OFF
	3 – 15	Undefined	

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 41096. 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.

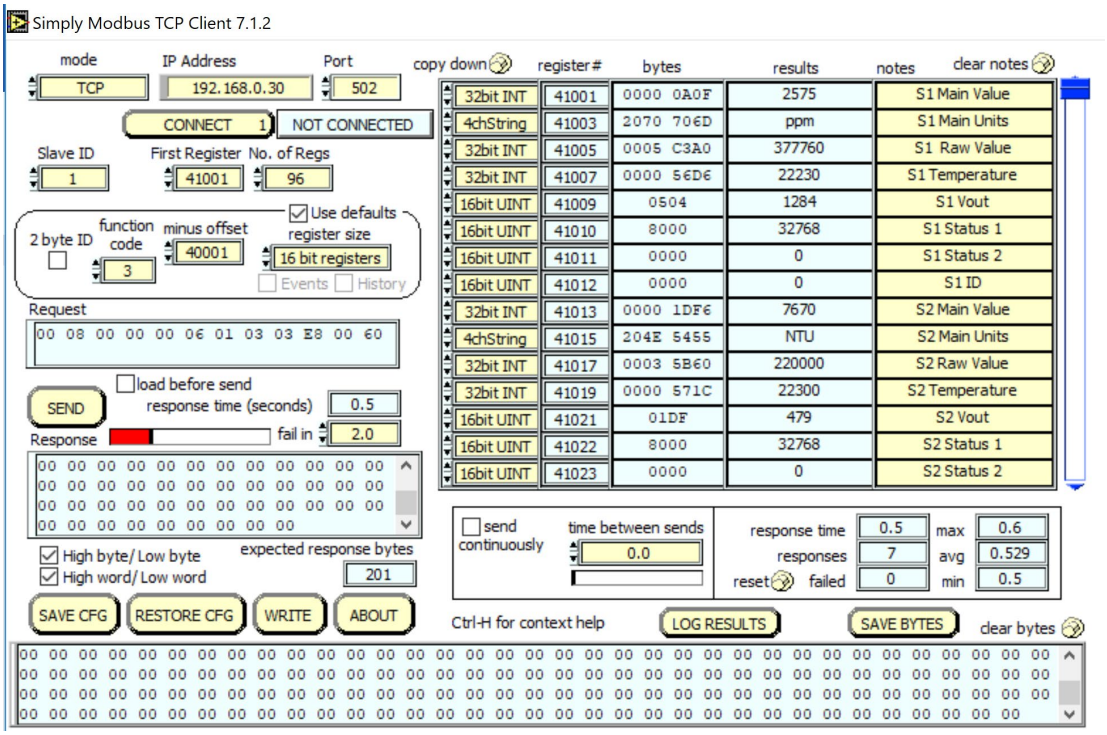


Figure 15 – Simply Modbus-TCP, S1/S2 Sensor Measure Data Block

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". Future feature, may be locked at current value on current sensors.

*[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.

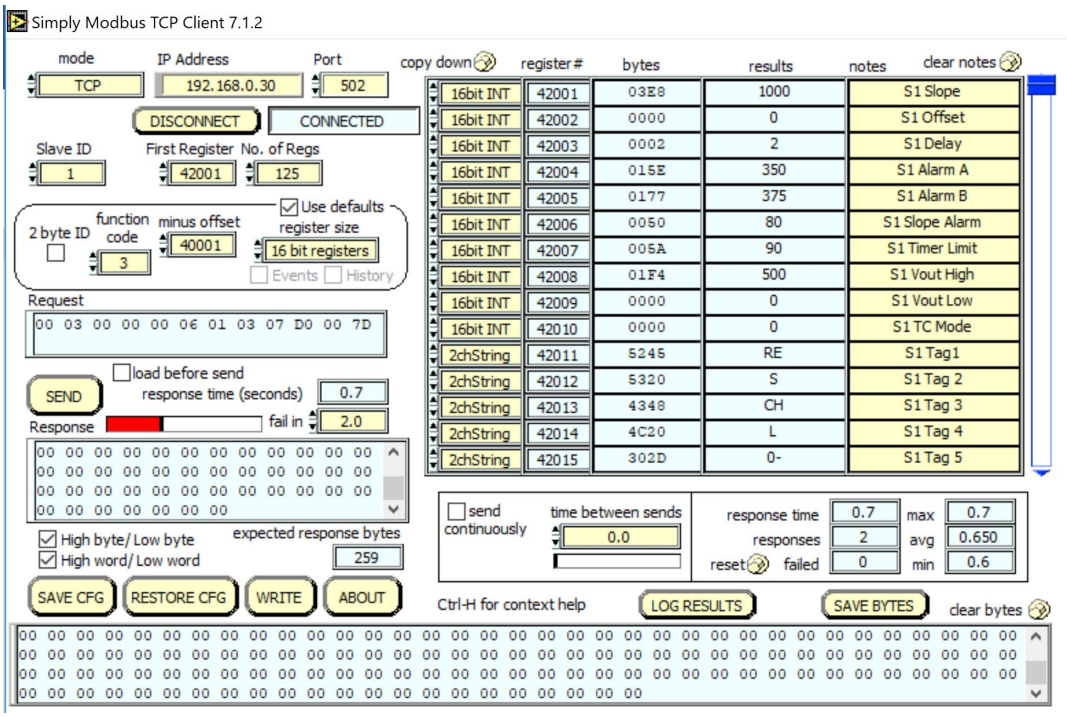


Figure 16 – Simply Modbus-TCP, S1 Sensor Info Data Block

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

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 function 16(10 hex) is 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 => 01 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 #1. In Modbus-TCP, slave address must always be 1, as each MetriNet requires a unique IP address. 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.

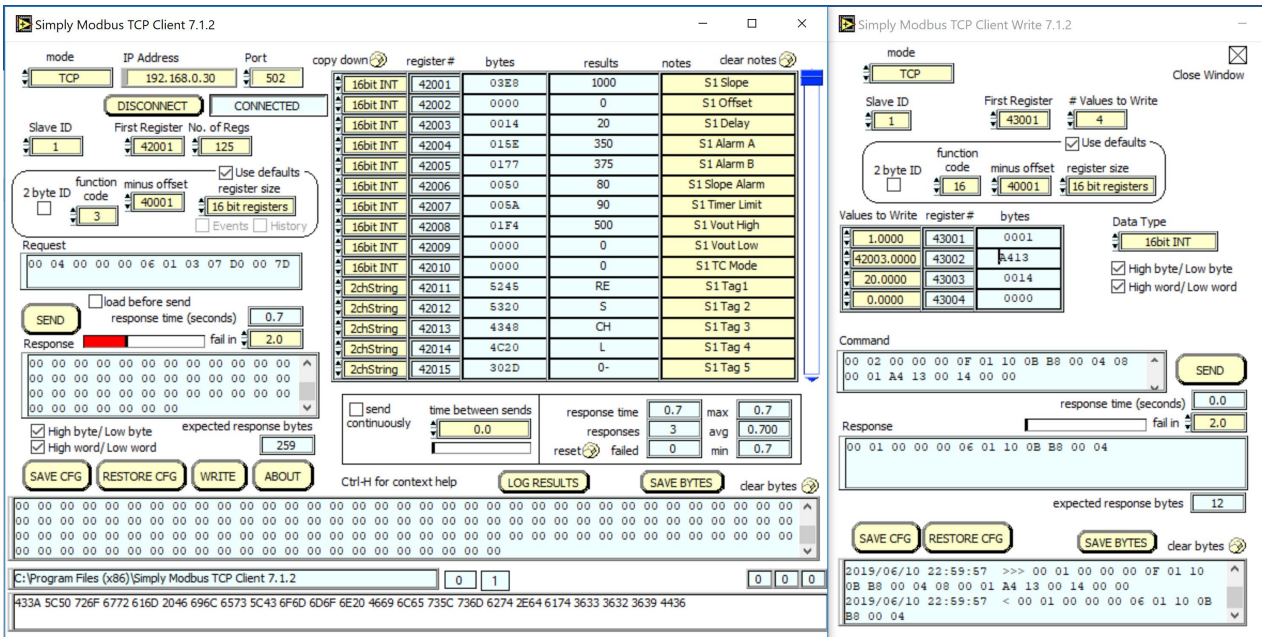


Figure 17 – Simply Modbus-TCP, Change S1 Delay Setting to 2.0

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 => 01 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 #1. 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 => 01 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 #1. 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-TCP 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 => 01 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 #1. 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.

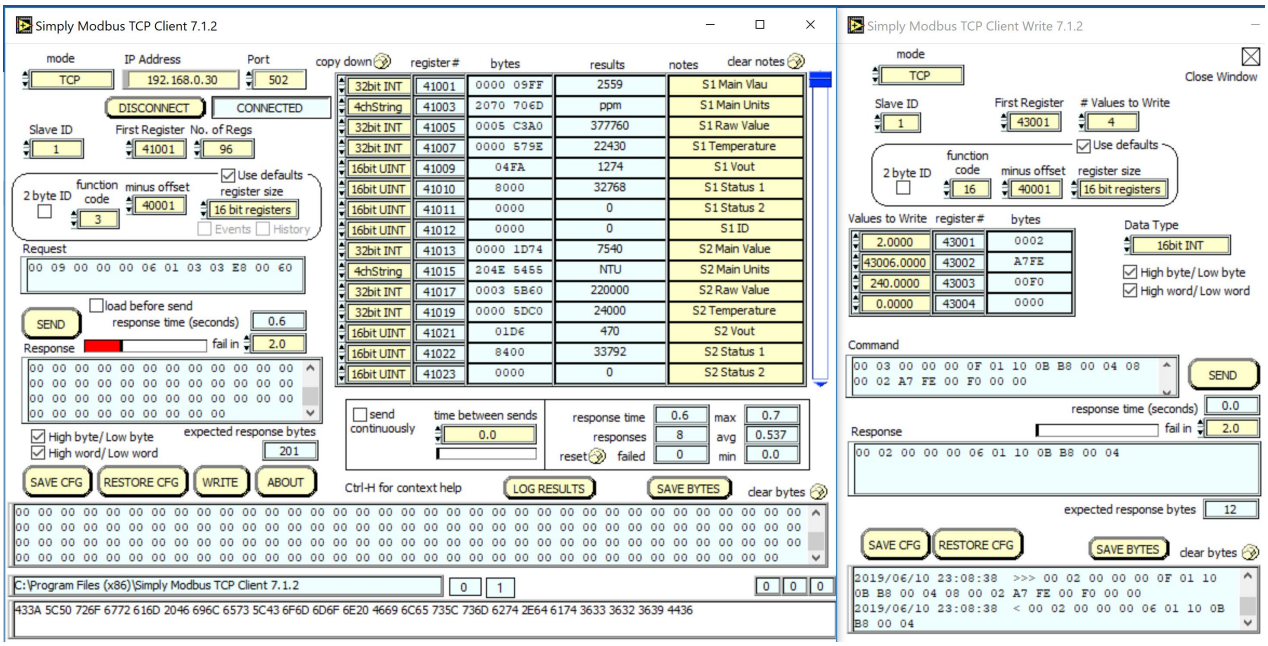


Figure 18 – Simply Modbus-TCP, Calibrate S2 Temperature to 24.0C

1.11 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-TCP 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 1.10, 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 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 “MetriNet Modbus-TCP,” 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 below.

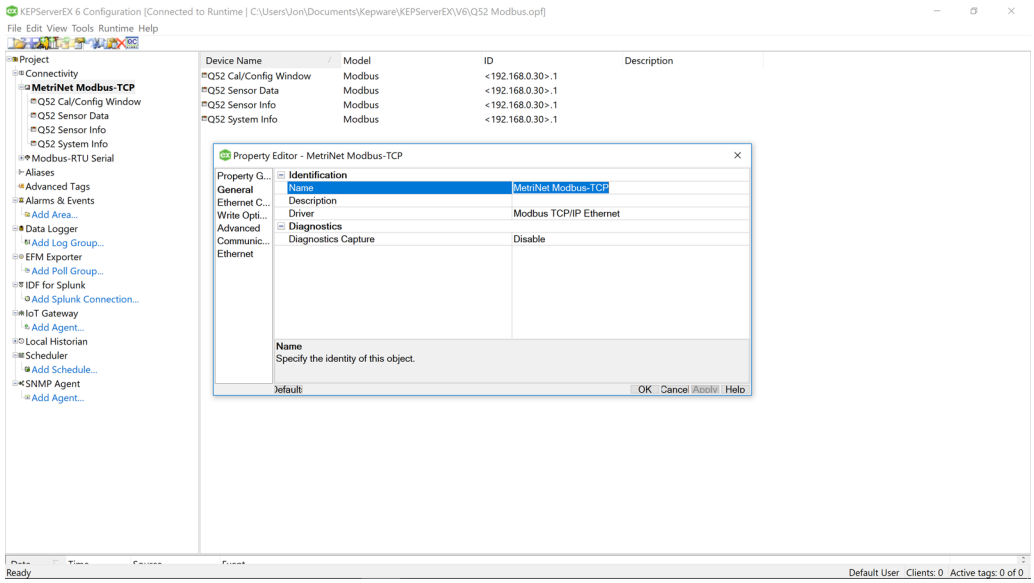


Figure 19 – KEPServer Channel Set-up For One Ethernet Port.

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”, “uS”, 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. 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. 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.

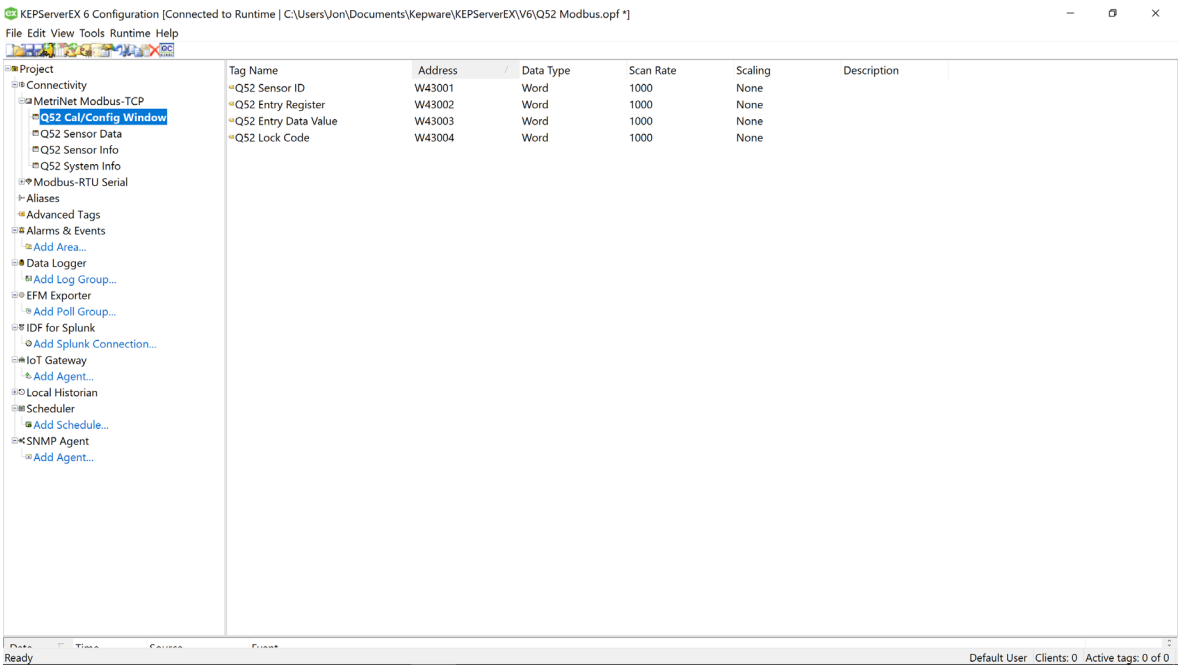


Figure 20 – KEPServer Cal/Config Window Device Registers.

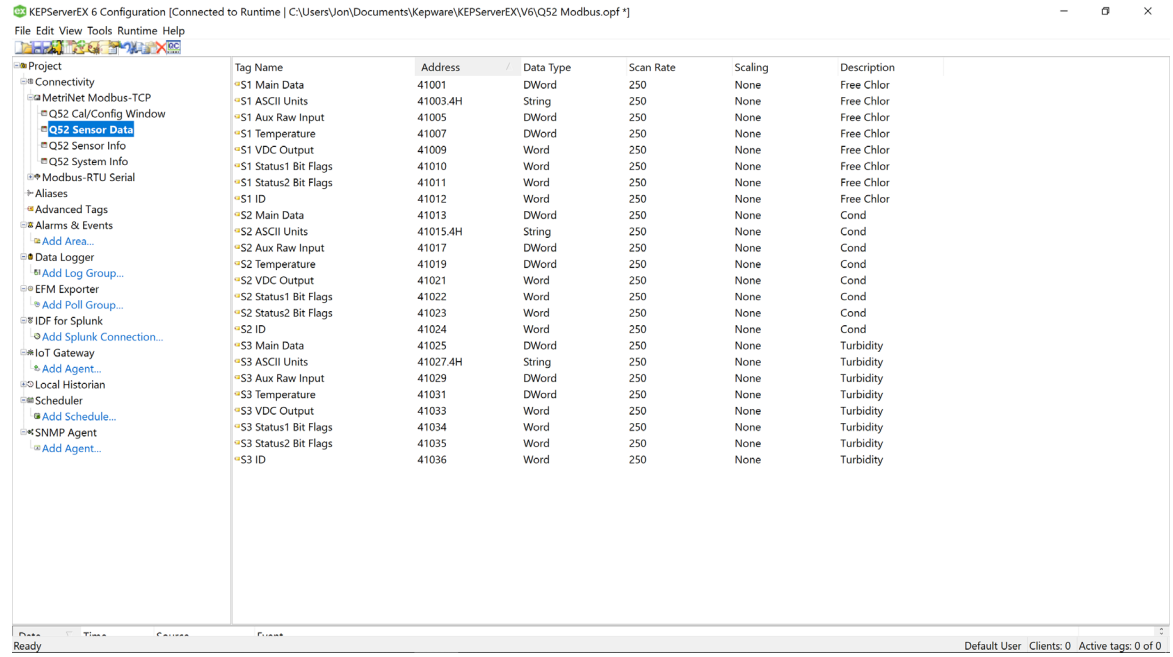


Figure 21 – KEPServer Sensor Data Device Registers

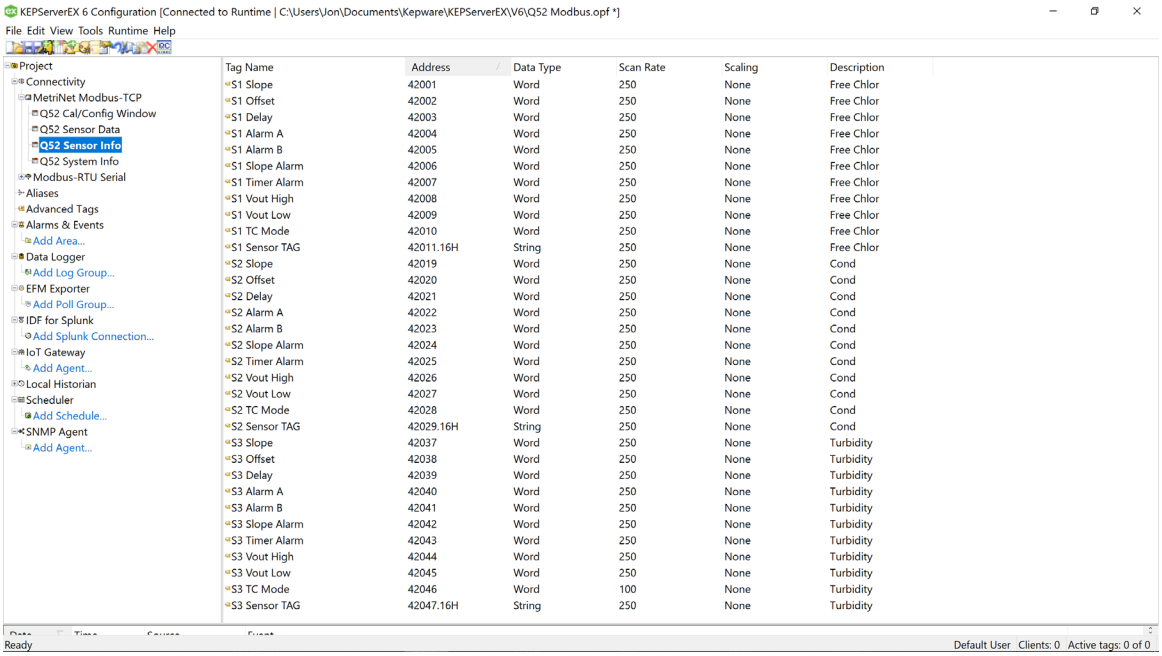


Figure 22 – KEPServer Sensor Info Device Registers.

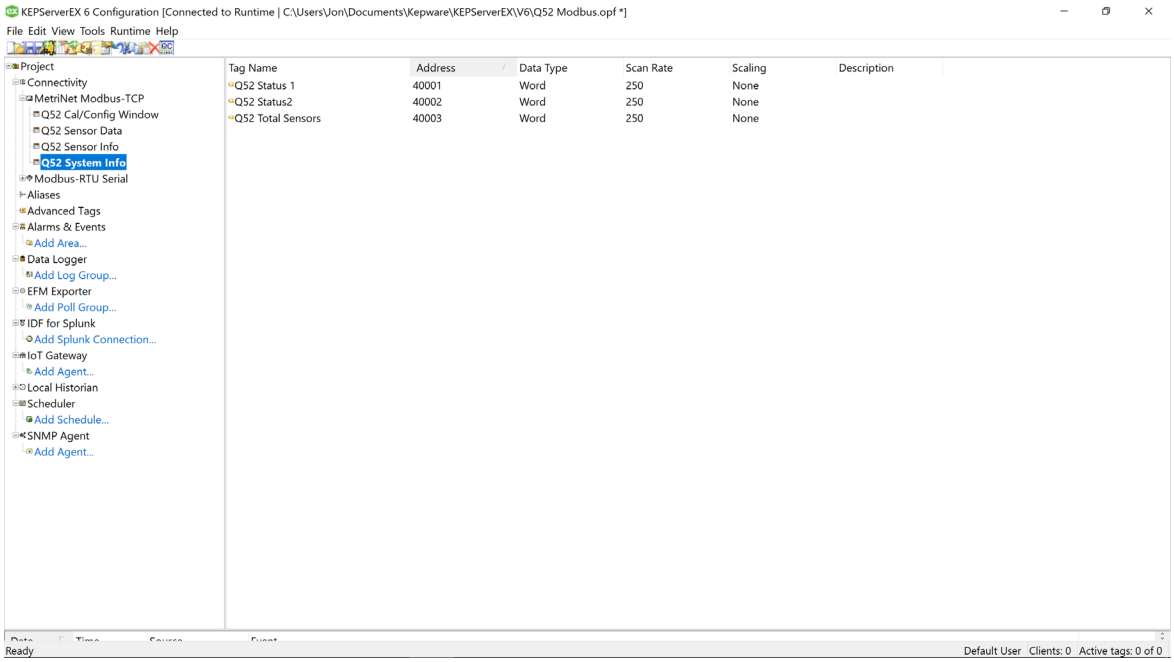


Figure 23 – KEPServer System Info Device Registers.

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.

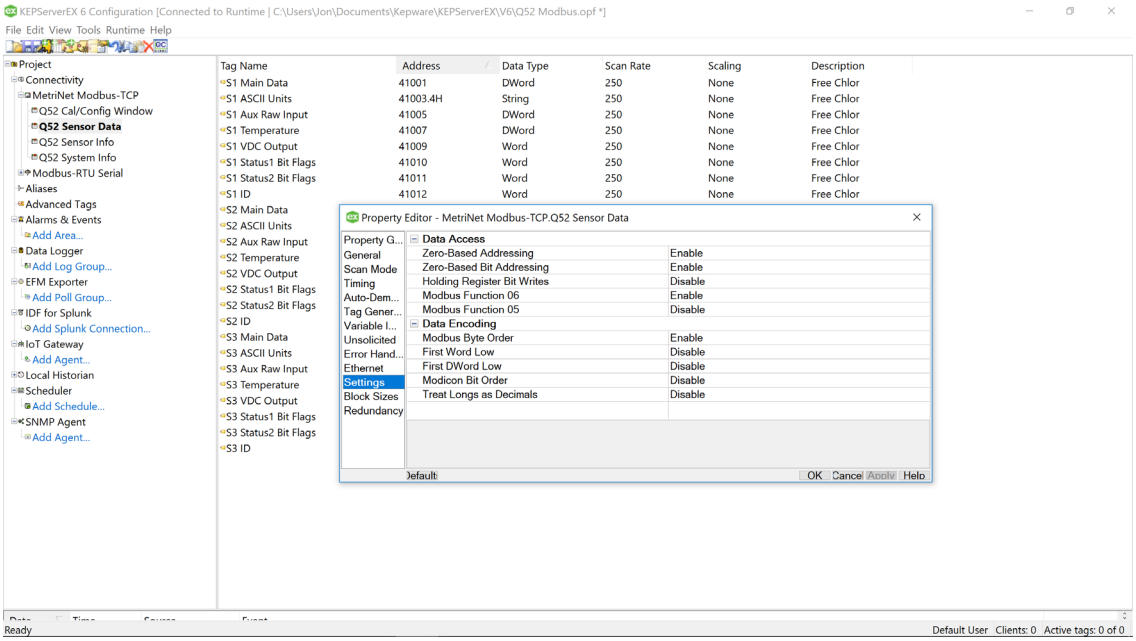


Figure 24 – KEPServer Zero-Based-Addressing and Modbus-Byte-Order Features.

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 the figures 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.

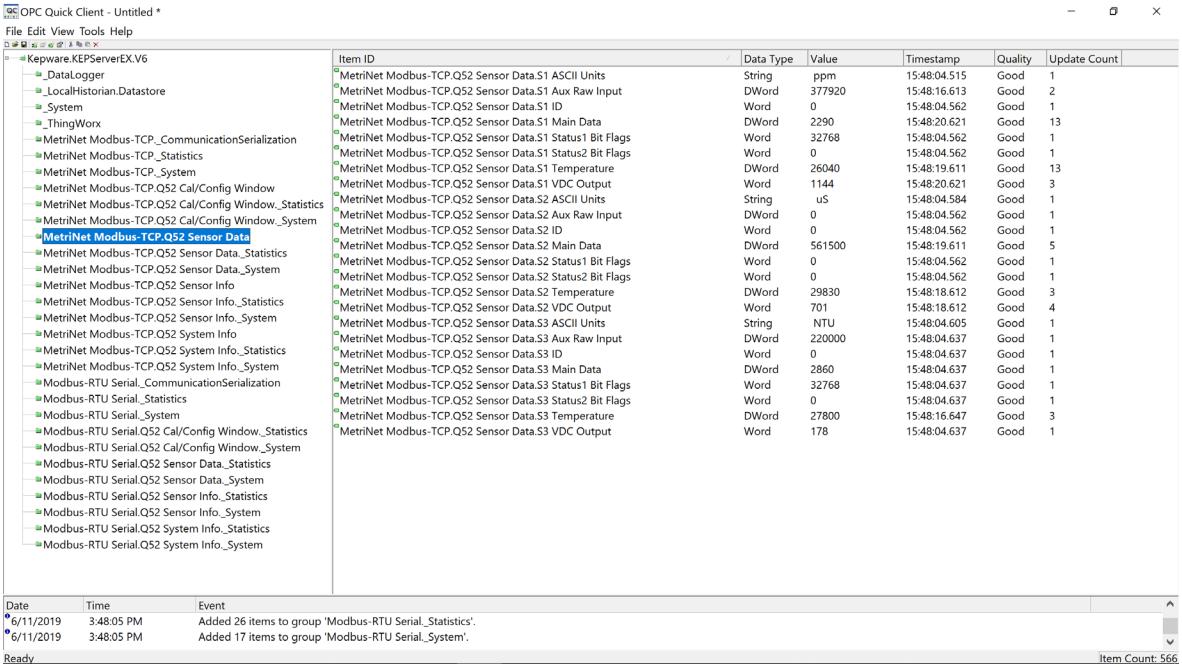


Figure 25 – KEPServer Real-Time Measurement Data (All) for Example System.

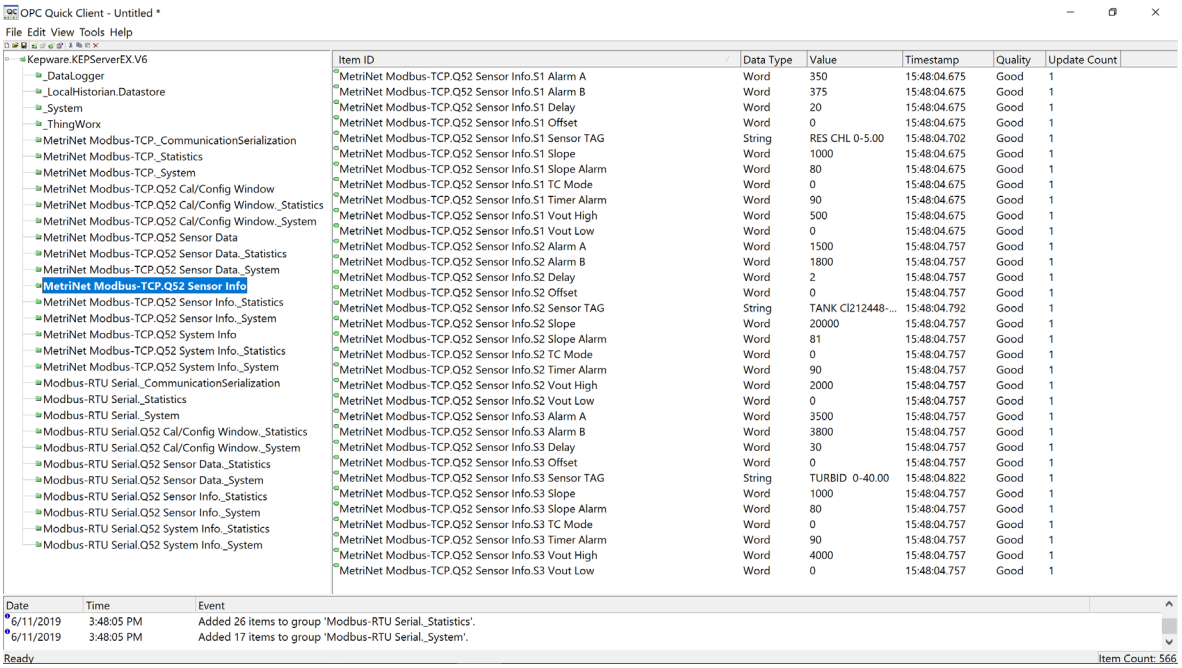


Figure 26 – KEPServer Sensor Info Settings (All) for Example System.

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