



Water Quality Monitors

MetriNet Modbus Communications



ATI-UM-04438-EN-05 (January 2026)

User Manual

CONTENTS

Modbus Description	3
Modbus Technical Overview	3
Modbus Connection.	4
Registers and Coils.	5
RS485 Communication	5
Cable Specification	7
RS485 Line Drivers/Receivers	7
120 Ohms Termination	8
Bias	8
Drops	8
Daisy Chaining	9
Shielding	9
Slave Connection Detail	10
MetriNet Modbus RTU Interface	10
MetriNet INFO - MB 40001 Start, Read-Write, (10 Registers)	12
M-Node MEASURE – MB 40011 Start, Read-Only, (88 Registers)	13
M-Node CONFIG – MB 41001 Start, Read-Write, (352 Registers)	18
M-Node CAL – MB 42001 Start (40 Registers)	20

MODBUS DESCRIPTION

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 (for example, fiber, radio, cellular and more).

The protocol comes in 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 one 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 [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.

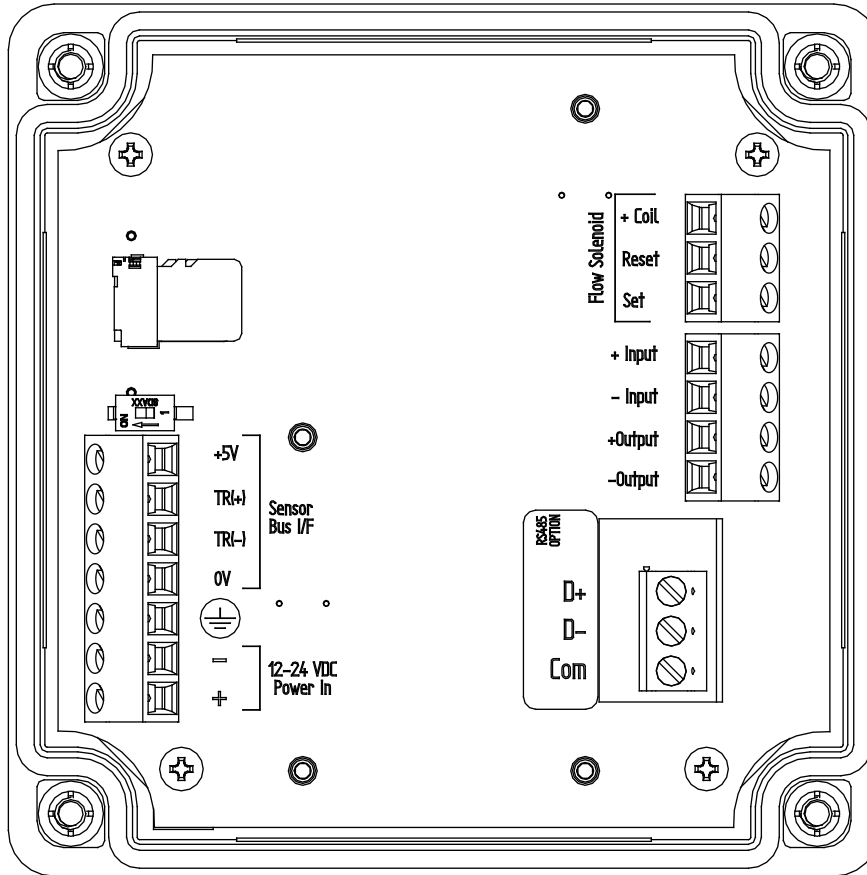


Figure 1: Modbus terminal connections

Modbus Connection

Modbus wiring is done at a plug-in communication circuit board located in the outlined section of [Figure 1](#). A callout in that figure shows the location of RS485 connections. The earth ground wire in the 485 cable should be terminated at “COM” on the Modbus card.

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 30001–39999, while registers containing data or the results of calculations are known as holding registers and are numbered 40001–49999. 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).

RS485 Communication

Modbus data transmission is an RS485 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 ft at transmission rates at or below 100 KHz. 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.

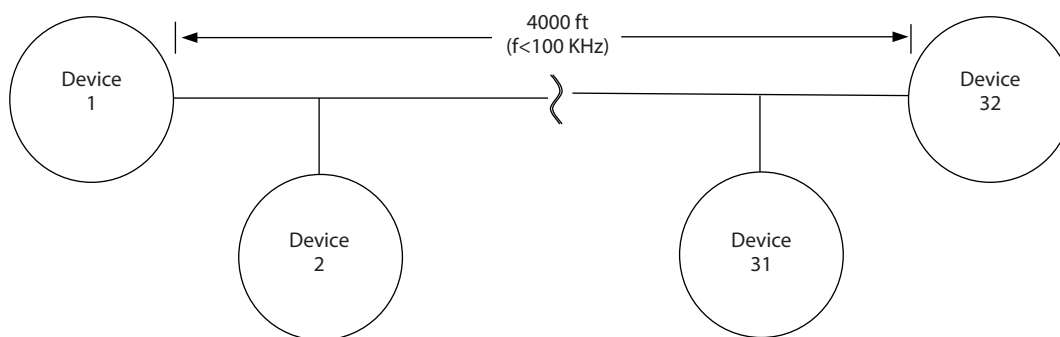


Figure 2: RS485 modbus data transmission

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 on page 4](#). Wiring connections for the communication bus are shown in that Figure.

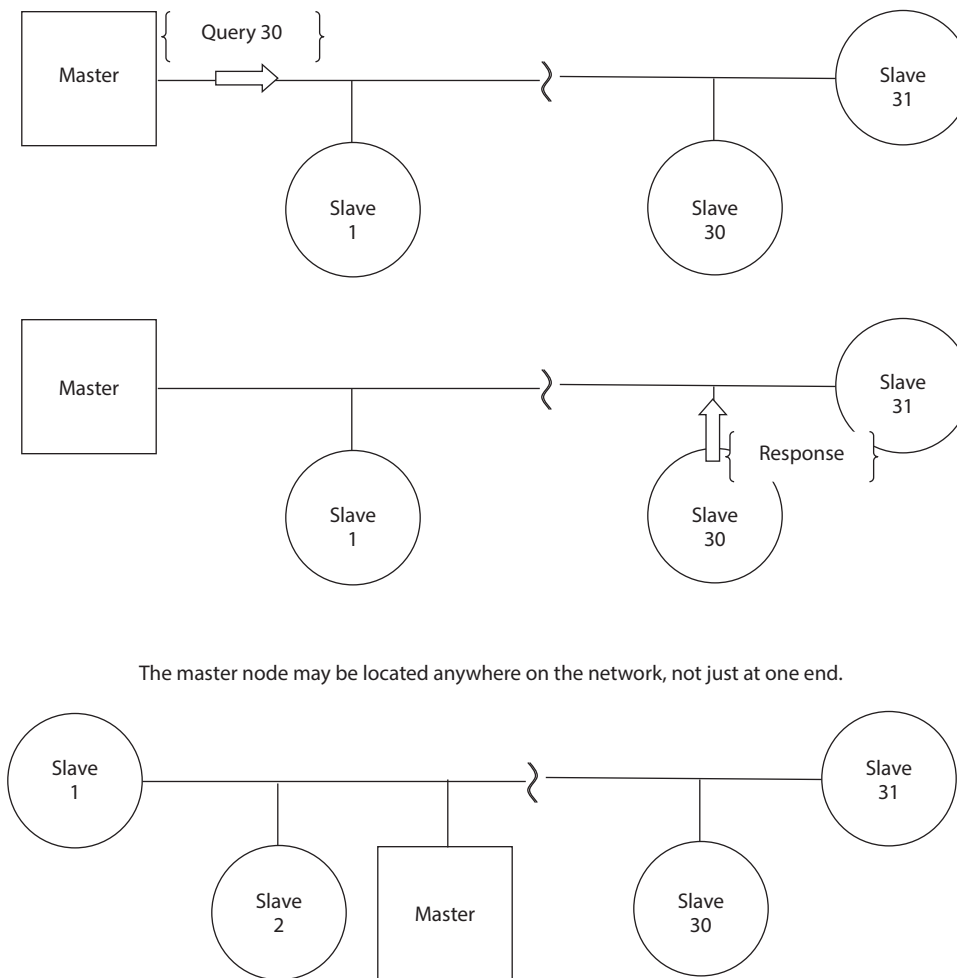


Figure 3: RS485 data exchange communication

Cable Specification

The bus is a cable composed of a twisted pair of wires with a characteristic impedance of 120 Ohms and a 120 Ohms termination resistor connecting the pair of wires at each end. Several manufacturers offer cables specifically designed for RS-485, 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.



Figure 4: Cable specifications

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–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 keeps 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 in [Figure 5](#).

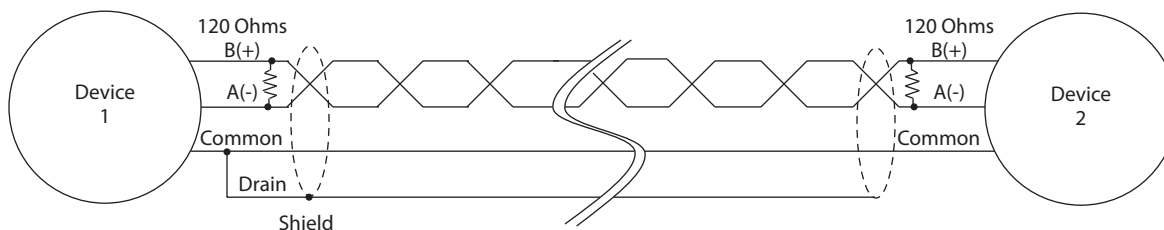


Figure 5: Twisted pair labels

RS485 Line Drivers/Receivers

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

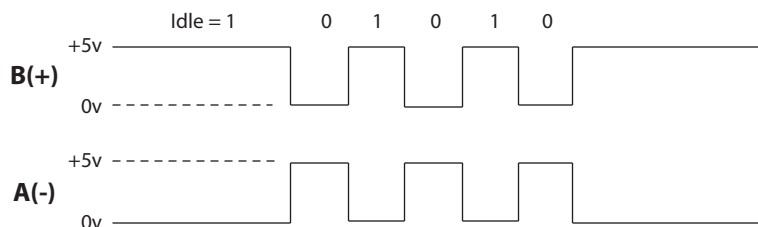


Figure 6: RS485 line drivers/receivers

Over a distance of 4000 ft, the 5V applied to either line may be dropped significantly. This usually doesn't present a problem since RS-485 receivers are specified to operate with a differential voltage of only 0.200V. In practice, however, the differential voltage should remain above 1.5V.

Logic State High (Idle or Marking State): $(B - A) \geq 200 \text{ mV}$

LogicState Low (SpacingState): $(A - B) \geq 200 \text{ mV}$

120 Ohms 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. **ONLY** set to “ON” position if the MetriNet is an End-of-Bus unit.

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 200 mV between the A and B lines.

NOTE: Bias resistors are not required for MetriNet transmitters, as the Modbus driver includes a “Failsafe” built-in Bias Design.

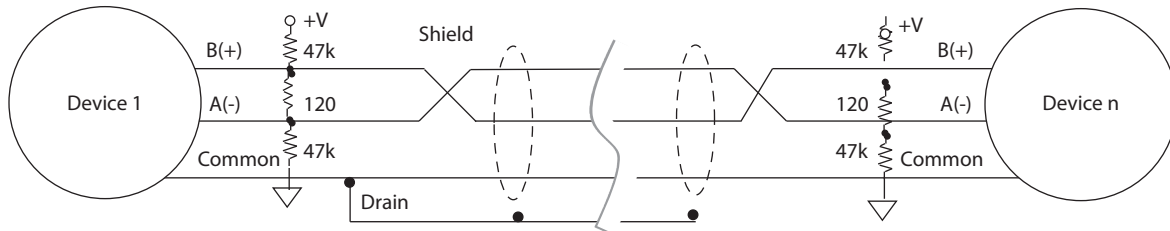


Figure 7: Bias

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 be 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).

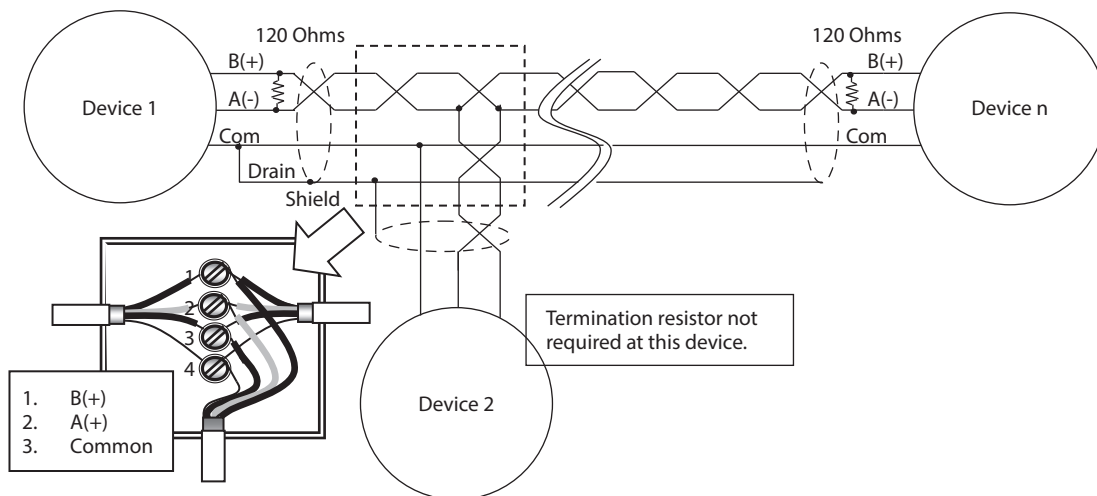


Figure 8: Drops

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 loads the network excessively and reliable communications cannot be guaranteed.

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 requires more connections inside the transmitter housing and therefore consume more space.

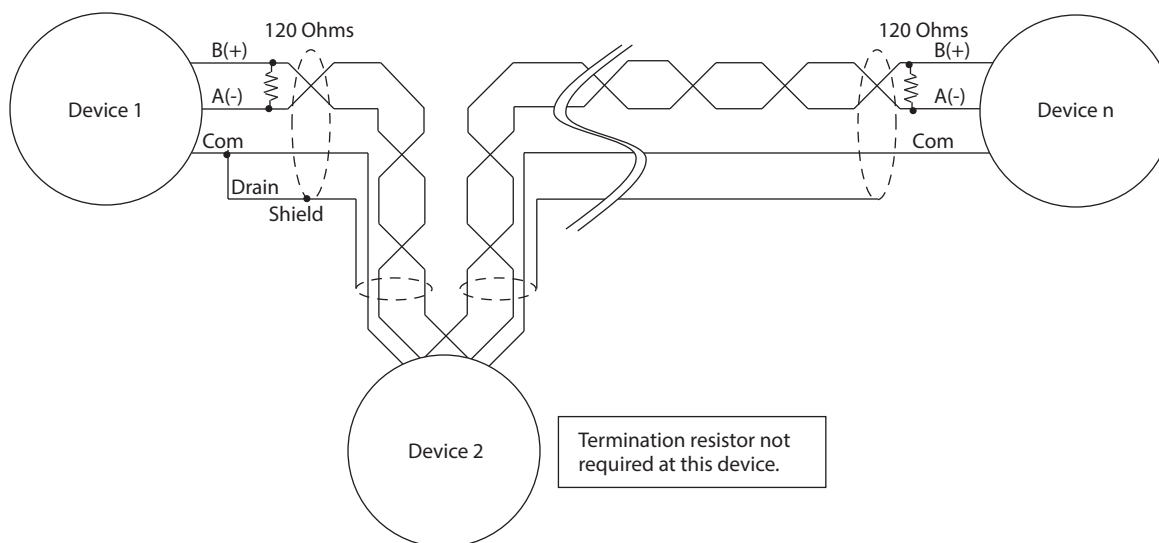


Figure 9: Daisy chaining

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 bidirectional transient protector (from a few volts to a few hundred volts depending on the situation).

NOTE: MetriNet transmitters are galvanically isolated from the RS485 (Modbus) port and use extensive spike/surge protection from lightening and other electrical sources.

Slave Connection Detail

The Modbus RTU connection settings 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 and more appear in *OPTIONS* listed menus.

NOTE: 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. The menu *^Baud Rate* displays. Default is 9600. Select **9600** or **19.2k**.
2. Press **UP** arrow. The menu *^Parity* displays. Default is none. Range of entry is none, odd or even.
3. Press **UP** arrow. *^Stop Bits* displays. Default is 1. Select **1** or **2**.
4. Press **UP** arrow. *^Wr Lock Enable* displays. Default is OFF. Select **OFF** or **ON**.

MetriNet Modbus RTU Interface

There are really two basic modes of operation for the external Modbus RTU interface to the MetriNet. One mode is purely reading critical data on a continual basis, and the other is completely different as it involves reading/writing sensor configuration registers or calibration registers. The MetriNet only implements *Modbus* functions 03 and 06.

The maximum polling rate by external master is 100 mS (old received frame stop-to-new frame start).

The overall map consists of the following address ranges in the holding register area:

1. MB 40001–40098 = 10 registers. MetriNet *INFO* block.
88 registers (11 × 8.) M-Node *MEASURE* block. This area read continuously.
2. MB 41001–41352 = 352 registers (44 × 8.) M-Node *CONFIG* block.
While it could be read at any time, this area is only updated after a power up, or after a “write” has occurred.
3. MB 42001–42040 = 40 registers (5 × 8.) M-Node *CAL* block.
This is a “write” area only, that sends data to a specific sensor cal function.

If the *Wr Lockcode* is enabled in the MetriNet *OPTIONS* menu, an unlock write security code must be written first in order to perform a write to any of these registers. Write a value of 1224 to MB42500. The unit is locked after the next Modbus command is received even if it is not a write command. Therefore, the next command sent after an unlock command must be the write to register command for the write to occur successfully.

All writes are 1 Modbus register length only. No multiple writes required.

<http://www.simplymodbus.ca/RTUmaster.htm>

MetriNet INFO - MB 40001 Start, Read-Write, (10 Registers)

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)	MetriNet	Status 1	Binary
40002	UINT(16)	MetriNet	Status 2	Binary
40003	UINT(16)	MetriNet	Number of Sensors	1–8
40004	UINT(16)	MetriNet	Reserved 1	
40005	UINT(16)	MetriNet	Reserved 2	
40006	UINT(16)	MetriNet	Reserved 3	
40007	UINT(16)	MetriNet	Reserved 4	
40008	UINT(16)	MetriNet	Reserved 5	
40009	UINT(16)	MetriNet	Reserved 6	
40010	UINT(16)	MetriNet	Reserved 7	

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
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

M-Node MEASURE – MB 40011 Start, Read-Only, (88 Registers)

Here is an 8-chlorine sensor map example:

This block would be read only though *Modbus* function 03 - Read Holding Registers, starting anywhere from 40001–40098. If you wanted it all, you would just call for a 98 register read starting at MB40001. If you only wanted the Sensor data, read 88 registers starting at 40011.

NOTE: If there is a comm error (loss of communications with a sensor), the error bit corresponding to the sensor is set in register 40001. The data in the sensor registers holds the last valid read from the sensor.

Sensor #1 (11 Registers)

Register	Data Type	Sensor	Description	Data Format
40011 40012	Long(32)	Sensor 1	Main Value	1000 = 1.000
40013 40014	Long(32)	Sensor 1	Main Units	ASCII (that is ppm)
40015 40016	Long(32)	Sensor 1	Raw Sensor Value	1000 = 1.000
40017 40018	Long(32)	Sensor 1	Temperature (°F or °C)	25400 = 25.4
40019	UINT(16)	Sensor 1	DAC Output Value	0–4095
40020	UINT(16)	Sensor 1	Status 1	Binary
40021	UINT(16)	Sensor 1	Status 2	Binary

Register	Bit	Description
40020	0 (LSB)	ALARM_A
	1	ALARM_B
	2	ALARM_C
	3	ALARM_D
	4	ALARM_E
	5	ENTRY_OUT_OF_RANGE
	6	ENTRY_ACCEPTED
	7	ENTRY_FAIL
	8	MAIN_CAL_PASS
	9	MAIN_CAL_FAIL
	10	TC_CAL_PASS
	11	TC_CAL_FAIL
	12	TC_F
	13	SENSOR_LOCK
	14	MetriNet
	15	DISPLAY_FORMAT 00 = 0000 01 = 000.0 10 = 00.00 11 = 0.000
40021	0	EE_INIT_FAIL
	1	MAIN_UNITS_HI
	2	MAIN_UNITS_LO
	3	MAIN_INPUT_ERR
	4	TC_UNITS_HI
	5	TC_UNITS_LO
	6	TC_INPUT_ERR
	7	CAL_MAIN_SLOPE_HI
	8	CAL_MAIN_SLOPE_LO
	9	CAL_MAIN_ZERO_HI
	10	CAL_MAIN_OFFSET_HI
	11	MAIN_UNSTABLE
	12	CAL_TC_OFFSET_HI
	13	TC_UNSTABLE
	14	Undefined
	15	Undefined

Sensor #2 (11 Registers)

Register	Data Type	Sensor	Description	Data Format
40022 40023	Long(32)	Sensor 2	Main Value	1000 = 1.000
40024 40025	Long(32)	Sensor 2	Main Units	ASCII (that is ppm)
40026 40027	Long(32)	Sensor 2	Raw Sensor Value	1000 = 1.000
40028 40029	Long(32)	Sensor 2	Temperature (°F or °C)	25400 = 25.4
40030	UINT(16)	Sensor 2	DAC Output Value	0–4095
40031	UINT(16)	Sensor 2	Status 1	Binary
40032	UINT(16)	Sensor 2	Status 2	Binary

Sensor #3 (11 Registers)

Register	Data Type	Sensor	Description	Data Format
40033 40034	Long(32)	Sensor 3	Main Value	1000 = 1.000
40035 40036	Long(32)	Sensor 3	Main Units	ASCII (that is ppm)
40037 40038	Long(32)	Sensor 3	Raw Sensor Value	1000 = 1.000
40039 40040	Long(32)	Sensor 3	Temperature (°F or °C)	25400 = 25.4
40041	UINT(16)	Sensor 3	DAC Output Value	0–4095
40042	UINT(16)	Sensor 3	Status 1	Binary
40043	UINT(16)	Sensor 3	Status 2	Binary

Sensor #4 (11 Registers)

Register	Data Type	Sensor	Description	Data Format
40044 40045	Long(32)	Sensor 4	Main Value	1000 = 1.000
40046 40047	Long(32)	Sensor 4	Main Units	ASCII (that is ppm)
40048 40049	Long(32)	Sensor 4	Raw Sensor Value	1000 = 1.000
40050 40051	Long(32)	Sensor 4	Temperature (°F or °C)	25400 = 25.4
40052	UINT(16)	Sensor 4	DAC Output Value	0–4095
40053	UINT(16)	Sensor 4	Status 1	Binary
40054	UINT(16)	Sensor 4	Status 2	Binary

Sensor #5 (11 Registers)

Register	Data Type	Sensor	Description	Data Format
40055 40056	Long(32)	Sensor 5	Main Value	1000 = 1.000
40057 40058	Long(32)	Sensor 5	Main Units	ASCII (that is ppm)
40059 40060	Long(32)	Sensor 5	Raw Sensor Value	1000 = 1.000
40061 40062	Long(32)	Sensor 5	Temperature (°F or °C)	25400 = 25.4
40063	UINT(16)	Sensor 5	DAC Output Value	0–4095
40064	UINT(16)	Sensor 5	Status 1	Binary
40065	UINT(16)	Sensor 5	Status 2	Binary

Sensor #6 (11 Registers)

Register	Data Type	Sensor	Description	Data Format
40066 40067	Long(32)	Sensor 6	Main Value	1000 = 1.000
40068 40069	Long(32)	Sensor 6	Main Units	ASCII (that is ppm)
40070 40071	Long(32)	Sensor 6	Raw Sensor Value	1000 = 1.000
40072 40073	Long(32)	Sensor 6	Temperature (°F or °C)	25400 = 25.4
40074	UINT(16)	Sensor 6	DAC Output Value	0–4095
40075	UINT(16)	Sensor 6	Status 1	Binary
40076	UINT(16)	Sensor 6	Status 2	Binary

Sensor #7 (11 registers)

Register	Data Type	Sensor	Description	Data Format
40077 40078	Long(32)	Sensor 7	Main Value	1000 = 1.000
40079 40080	Long(32)	Sensor 7	Main Units	ASCII (that is ppm)
40081 40082	Long(32)	Sensor 7	Raw Sensor Value	1000 = 1.000
40083 40084	Long(32)	Sensor 7	Temperature (°F or °C)	25400 = 25.4
40085	UINT(16)	Sensor 7	DAC Output Value	0–4095
40086	UINT(16)	Sensor 7	Status 1	Binary
40087	UINT(16)	Sensor 7	Status 2	Binary

Sensor #8 (11 Registers)

Register	Data Type	Sensor	Description	Data Format
40088 40089	Long(32)	Sensor 8	Main Value	1000 = 1.000
40090 40091	Long(32)	Sensor 8	Main Units	ASCII (that is ppm)
40092 40093	Long(32)	Sensor 8	Raw Sensor Value	1000 = 1.000
40094 40095	Long(32)	Sensor 8	Temperature (°F or °C)	25400 = 25.4
40096	UINT(16)	Sensor 8	DAC Output Value	0–4095
40097	UINT(16)	Sensor 8	Status 1	Binary
40098	UINT(16)	Sensor 8	Status 2	Binary

M-Node CONFIG – MB 41001 Start, Read-Write, (352 Registers)

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 comm timeout error occurs, or a write command is received from the host, the MetriNet reads these registers from the sensors and update the data held in the MetriNet Modbus registers.

This block would be read only through *Modbus* function 03 - Read Holding Registers, or written through *Modbus* function 06 – Write Single Registers. The top of the block starts at MetriNet RAM MB41001 and extends down to MB41352.

Sensor #1 (44 Registers)

Register	Data Type	Sensor	Description	Data Format
41001	UINT(16)	Sensor 1	Address	1–247
41002	UINT(16)	Sensor 1	Lock code	Not used
41003	UINT(16)	Sensor 1	Delay	
41004	UINT(16)	Sensor 1	Alarm A	
41005	UINT(16)	Sensor 1	Alarm B	
41006	UINT(16)	Sensor 1	Slope Alarm	
40007	UINT(16)	Sensor 1	Timer Cal Limit	
40008	UINT(16)	Sensor 1	Reserved 1	
40009	UINT(16)	Sensor 1	VoutHI	
40010	UINT(16)	Sensor 1	VoutLO	
40011	UINT(16)	Sensor 1	TcComp	
40012	UINT(16)	Sensor 1	TcRefTemp	
40013	UINT(16)	Sensor 1	TcCalOffset	
40014	UINT(16)	Sensor 1	TcMode	0 = °F, 1 = °C
40015	UINT(16)	Sensor 1	Sensitivity	
40016	UINT(16)	Sensor 1	reserved2	
40017	UINT(16)	Sensor 1	Tag1	
40018	UINT(16)	Sensor 1	Tag2	
40019	UINT(16)	Sensor 1	Tag3	
40020	UINT(16)	Sensor 1	Tag4	
40021	UINT(16)	Sensor 1	Tag5	
40022	UINT(16)	Sensor 1	Tag6	
40023	UINT(16)	Sensor 1	Tag7	
40024	UINT(16)	Sensor 1	Tag8	
40025	UINT(16)	Sensor 1	Power Mode	
40026	UINT(16)	Sensor 1	Baud	
40027	UINT(16)	Sensor 1	ParMode	
40028	UINT(16)	Sensor 1	reserved6	
40029	UINT(16)	Sensor 1	reserved7	
40030	UINT(16)	Sensor 1	reserved8	
40031	UINT(16)	Sensor 1	reserved9	
40032	UINT(16)	Sensor 1	reserved10	

Register	Data Type	Sensor	Description	Data Format
40033	UINT(16)	Sensor 1	CalPt1_value	
40034	UINT(16)	Sensor 1	CalPt1_counts	
40035	UINT(16)	Sensor 1	CalPt1_temp	
40036	UINT(16)	Sensor 1	Offset	
40037	UINT(16)	Sensor 1	Offset Display	
40038	UINT(16)	Sensor 1	CalPt1_day	
40039	UINT(16)	Sensor 1	CalPt2_value	
40040	UINT(16)	Sensor 1	CalPt2_counts	
40041	UINT(16)	Sensor 1	CalPt2_temp	
40042	UINT(16)	Sensor 1	Slope	
40043	UINT(16)	Sensor 1	Slope Display	
40044	UINT(16)	Sensor 1	CalPt2_day	

Similarly, the remaining sensors all load in next:

Sensor #2

41045–41088

Sensor #3

41089–41132

Sensor #4

41133–41176

Sensor #5

41177–41220

Sensor #6

412201–41264

Sensor #7

41265–41308

Sensor #8

41309–41352

M-Node CAL – MB 42001 Start (40 Registers)

NOTE: While calibration over the Modbus serial interface is possible, it is not recommended. Badger Meter always recommends cleaning and inspection of all sensors prior to calibration. 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.

This section is somewhat unique, as it's really "write-only." There is nothing to be "read" here, as these are the data entry functions for more complex calibrations. So, a register-read would result in an "INVALID ADDRESS" return error. The result of calibrations must be determined by the result-flags from that specific sensor.

These areas are tied to actual internal sensor functions as opposed to simple registers. They are still mapped in the overall scheme, so it is easy to sort out which sensor the user is attempting to calibrate.

Like the *MEASURE* data above, all 8 sensors are blocked into one contiguous space.

The top of the block starts at MetriNet Modbus Register 42001 and extends down to register 42040. The *Modbus* function 06 is the only usable function here.

Int	CalTC;	// MB 0, TC Calibration function
Int	CalMainSpan;	// MB 1, Main input span calibration
Int	CalMainZero;	// MB 2, Main input zero calibration
Int	CalMainOffset;	// MB 3, Main input offset calibration
Int	UserDefault;	// MB 4, Sensor user default set

Sensor #1 (5 Registers)

Register	Data Type	Sensor	Description	Data Format
42001	UINT(16)	Sensor 1	Cal Tc	
42002	UINT(16)	Sensor 1	CalMainSpan	
42003	UINT(16)	Sensor 1	CalMainZero	
42004	UINT(16)	Sensor 1	CalMainOffset	
42005	UINT(16)	Sensor 1	UserDefault	

Similarly, the remaining sensors all load in next:

Sensor #2

42006–42010

Sensor #3

42011–42015

Sensor #4

42016–42020

Sensor #5

42021–42025

Sensor #6

42026–42030

Sensor #7

42031–42035

Sensor #8

42036–42040

Trademarks appearing in this document are the property of their respective entities. Due to continuous research, product improvements and enhancements, Badger Meter reserves the right to change product or system specifications without notice, except to the extent an outstanding contractual obligation exists. © 2026 Badger Meter, Inc. All rights reserved.