

BB4-8422 Modbus TCP Client

Modbus TCP Client Error Counts

Dashboard / Protocol Status / Modbus Client

Error Counts
Read Data
Write Data

Show entries
Search:

Device number	Device name	IP address	Total messages	No response errors	Exception errors	Socket error
1	BB2-6010	192.168.1.87	60464	0	0	0
2	BBPro-V210	192.168.1.23	10648	0	7718	0
Device number	Device name	IP address	Total messages	No response errors	Exception errors	Socket error

Showing 1 to 2 of 2 entries
Previous **1** Next

✖ Clear Counts
↻ Refresh

- Click the Clear Counts to reset the counts to zero.
- Click the Refresh button to update the counts.

This page shows an error summary on a device by device basis. There will be one line per each device configured for access by the Modbus TCP Client. If there is a non-zero error count for a device, you can begin to track down where this error is occurring by viewing the Read and Write Data pages for the Modbus TCP Client.

Socket errors will be an error code reported by the system if nonzero. Refer to *Socket Errors* at the end of this document for a list of these errors.

Modbus TCP Client Read Data

Error Counts

Read Data

Write Data

Show 10 entries

Search:

Map	Error status	Time since update	Present value	Remote device	Remote register	Local object
1	No Error	0.58	0	1: BB2-6010 [1]	holding reg [1]	1 : Object 1
2	No Error	0.58	0	1: BB2-6010 [1]	holding reg [1]	2 : Object 2
3	No Error	0.58	0	1: BB2-6010 [1]	holding reg [2]	3 : Object 3
4	Exception: illegal addr	2.09	0	1: BB2-6010 [1]	holding reg [7]	4 : Object 4
5	Exception: illegal addr	2.09	0	1: BB2-6010 [1]	holding reg [8]	5 : Object 5
6	Exception: illegal addr	2.09	0	1: BB2-6010 [1]	holding reg [9]	6 : Object 6
Map	Error status	Time since update	Present value	Remote device	Remote register	Local object

Showing 1 to 6 of 6 entries

Previous 1 Next

This diagnostic page shows you the most recent data obtained by the respective read map. Most importantly, it shows error indications on a map by map basis.

Modbus TCP Client Write Data

Error Counts Read Data **Write Data**

Show 10 entries

Search:

Map	Error status	Time since update	Present value	Remote device	Remote register	Local object
1	No Error	2.32	0	1: BB2-6010 [1]	holding reg [3]	3 : Object 1
2	No Error	2.3	0	1: BB2-6010 [1]	holding reg [4]	1 : Object 2
3	No Error	2.3	0	1: BB2-6010 [1]	holding reg [4]	2 : Object 3
4	Exception: illegal addr	2.3	0	1: BB2-6010 [1]	holding reg [5]	1 : Object 4
5	Exception: illegal addr	2.3	0	1: BB2-6010 [1]	holding reg [6]	2 : Object 5
Map	Error status	Time since update	Present value	Remote device	Remote register	Local object

Showing 1 to 5 of 5 entries

Previous 1 Next

This diagnostic page shows you the most recent data written by the respective write map. Most importantly, it shows error indications on a map by map basis.

Configuration

Read map

Write map

Devices

Config File

Select Modbus TCP Client from the sidebar menu to access Modbus TCP Client Configuration. The Modbus TCP Client Configuration page will have the tabs illustrated above. Click on the tabs to navigate to the areas of interest which are described in detail at the links below.

You may edit individual maps or devices by first navigating to the tab/page listing those items. Then click the edit icon (pencil). This will take you to the respective "Edit" page listed below.

Modbus TCP Client Read Maps

Dashboard / Protocol Configuration / Modbus TCP Client (Ethernet)

Read map Write map Devices Config File

Show 10 entries

Search:

Map number	Read from Remote device	Remote register type	Remote register number	Remote register format	Save to Local object
1  	1: BB2-6010	holding reg	1	unsigned int [1]	1 : Object 1
2  	1: BB2-6010	holding reg	1	unsigned int [1]	2 : Object 2
3  	1: BB2-6010	holding reg	2	signed int [1]	3 : Object 3
4  	1: BB2-6010	holding reg	7	unsigned int [1]	4 : Object 4
5  	1: BB2-6010	holding reg	8	unsigned int [1]	5 : Object 5
6  	1: BB2-6010	holding reg	9	signed int [1]	6 : Object 6
Map number	Read from Remote device	Remote register type	Remote register number	Remote register format	Save to Local object

Showing 1 to 6 of 6 entries

Previous 1 Next

Add new map(s) before map number:







Copy maps from through: and insert before:

Modbus TCP Client Read Maps are where you configure this device to read Modbus registers from other Modbus TCP devices, and store that data in local data objects. This page shows a list of currently configured read maps.

When Modbus display setting "Number" is selected under User Settings, Modbus registers will be displayed as type (e.g. holding register) and register number.

Modbus Settings

Modbus register number display format: **number** ▼

Map number ↑↓	Read from Remote device ↑↓	Remote register type ↑↓	Remote register number ↑↓	Remote register format ↑↓	Save to Local object ↑↓
1  	1: BB2-6010	holding reg	1	unsigned int [1]	1 : Object 1
2  	1: BB2-6010	holding reg	1	unsigned int [1]	2 : Object 2
3  	1: BB2-6010	holding reg	2	signed int [1]	3 : Object 3

- Click on the map number or pencil icon to jump to the map editing page for this map.
- Click on the trash can icon to delete this map.

When Modbus display setting "Modicon" is selected under User Settings, Modbus registers will be displayed as a single value in Modicon format, e.g. 40001 for holding register 1.

Modbus Settings

Modbus register number display format: **modicon** ▼

Map number ↑↓	Read from Remote device ↑↓	Remote register ↑↓	Remote register format ↑↓	Save to Local object ↑↓
1  	1: BB2-6010	40001	unsigned int [1]	1 : Object 1
2  	1: BB2-6010	40001	unsigned int [1]	2 : Object 2
3  	1: BB2-6010	40002	signed int [1]	3 : Object 3

- Click on the map number or pencil icon to jump to the map editing page for this map.
- Click on the trash can icon to delete this map.

Add new map(s) before map number: [Insert map\(s\)](#)

To add or insert new maps, enter a number of maps to add, and select a starting point. Then click the Insert button.

Copy maps from through: and insert before: [Copy maps](#)

To copy one or more maps and insert a duplicate copy of them at another point in the list, select the range to copy, select the insert point, and click Copy.

[Set Source Info](#)

The Set Source Info button is a diagnostic tool. You do not want multiple read maps (from all protocols combined) trying to store data into the same local object. Doing so would result in erroneous data (data would jump back and forth showing the most recently read data from any source). If you only have one source of data, then this isn't an issue. But if there will be multiple sources of data, then use the Set Source Info button to "claim" objects for this protocol function (e.g. Modbus TCP Read).

If objects have already been claimed in another protocol and you try to claim the same objects, you will see one or more error messages displayed at the top of the screen. Go to the Data Objects -> Object Status page to see where objects have been claimed. The Clear Source Info button is also found on the Data Objects -> Object Status page. To clear the claims and try again, use that button. Clicking the Set Source Info button more than once will also result in error messages since objects have already been claimed once.

Modbus TCP Client Read Map Edit

Modbus TCP Client Read Maps are where you configure this device to query other Modbus TCP devices for data, and store that data in local data objects. This page is where you enter the various parameters to make that happen.

[Dashboard](#) / [Protocol Configuration](#) / [Modbus Client](#) / [Read Map #1](#)

Modbus TCP Read Map #1

Map Number – Used as a reference in the map list for ordering the maps. Polling is done in round robin fashion in the order of map number.

The appearance of the following line will vary depending on your user settings. You have the option of displaying Modbus registers as raw address (0-indexed), register number (1-indexed), or Modicon format (e.g. 40001 style).

The following variations all refer to exactly the same register:

Read as: size:

Read as: size:

Read as: size:

Click the Check to validate a Modicon number.

The options available on this line will vary depending on selections made. The following are a few examples.

Read as: size:

Read as: size:

Read as: size:

Read as: size:

Read as: size:

Register Type – Modbus register types available are listed in the following table. These labels are illustrated here as recognized in XML or CSV files, but are further annotated on the web page.

Label Modbus Register type

“none” No register defined

- “Coil” Coil
- “Disc” Discrete Input
- “Input” Input Register
- “Hold” Holding Register

Register Number or Address – Enter the number (starting at 1) or raw address (starting at 0) as applicable. Do NOT enter 40001 for holding register 1 if you have not selected Modicon as the display format in your User Settings.

Modicon Register – Enter numbers like 40001 for the first holding register if you have selected Modicon representation in your User Settings.

Register Format – Select the format of the data contained in the Modbus register(s). This is not used by the protocol, but is used by the gateway to interpret what the raw increments of 1 or 16 bits should mean. Select format from the following table.

Format Label	Format description
“None”	No format defined
“Bit”	Single bit, used ONLY for Register Type Coil or Disc
“Int”	Integer (size and whether signed are defined by labels below)
“Real”	Floating point (single or double precision)
“Char”	Character string with 2 ASCII characters per register
“Mod10”	Mod10 format, can be 2, 3, or 4-register, specific to Schneider Electric meters

Register Size – Register size refers to the number of consecutive input or holding registers that should be read for a value greater than 16 bits. A 16-bit value would have size of 1, a 32-bit value would have size of 2, and a 64-bit value would have size of 4. Single precision Real (32-bit IEEE 754 floating point) would be size 2, and double precision Real (64-bit IEEE 754 floating point) would be size 4. If format is Mod10, then valid sizes are 2, 3, or 4 – check manufacturer’s documentation if Mod10 is noted. Register “size” for a character string will be character count divided by 2 (plus 1 if string length is an odd number). Register Size is not used for Coil or Disc types.

Unsigned – Select signed or unsigned. Defaults to signed integer. Has no effect on Register Format other than Int.



Endian Selection – Used when Register Size is greater than 1 to indicate what order the registers should be interpreted in. Select "low" to indicate that the lowest numbered register contains the least significant portion of data. Select "high" to indicate that the lowest numbered register contains the most significant portion of data. Although Modbus protocol itself is not inherently

“Little Endian”, many devices operate that way due to Intel processors being inherently Little Endian. Modbus protocol does not stipulate what the register order should be when multiple registers are treated as a single data entity. Therefore, the user is required to pay attention to this.

From Unit

Device – Select a device from the TCP Device list that should be accessed for this read attempt on a TCP network.

Unit – Unit number to be included in the TCP request, will default to 1 if not given or is set to zero. Web page will force it to default to 1, but is optional for CSV or XML import.

After reading, apply (optional):

Bit Mask Hexadecimal

Mask – A bit mask given as an 8-digit hexadecimal value, if non-zero. The mask operation is skipped if mask value is zero, or register format is not Int (integer). When the data of interest is a single bit, or bit field less than the full register width, the Mask is used. When the Modbus register is read, its data is bit-wise ANDed with the Mask, then right justified so that the least significant mask bit becomes the least significant data bit. The result is then placed in the local data object selected by the read map.

Then apply (optional):

Scale Offset

Scale – Register content is multiplied by this value, if non-zero, before being placed into the local object. Scale is treated mathematically as 1 if omitted (set to zero).

Offset – This value is added to the register content (after being multiplied by scale) before being placed into the local object.

NOTE: The order of operation is as follows: (1) read Modbus register; (2) apply mask if applicable; (3) apply scale if non-zero; (4) apply offset. Result is then placed in local object.

Poll every seconds

Poll Time – Poll time in seconds (can be fractional). The sets the rate at which the remote Modbus register will be read. This poll time is not guaranteed to be met. Polling is done in

round-robin fashion. In a very busy system, more than this time may expire before the next poll. If less than this time has expired, then the system will wait this amount of time until polling again.

Saving result in named

Local Object – Local object number that the result of the Modbus read operation should be placed into.

Apply after read failure(s)

Default Value – Provides the default value that the local object should be set to in the event the FailCount is exceeded.

Fail Count – Optional, provides a count of read failures, if non-zero, that can occur before the local object will be set to the default value given in this map. If zero, the default value will never be applied. If 1, then the default value will be applied upon the first failure (probably not recommended), and so on. The count is reset by a successful read.

Read when: equals

Index Object – Optional, allows for selectively enabling this read operation. If an index object (local object number) is given, and its value matches the index value, then this read operation will take place. If an index object is given but the local object's value does not match the index value, then this read operation will be skipped.

Index Value – Optional, used in conjunction with Index Object (see note above).

To illustrate the use of Mask, consider the two following read maps. Note that they are reading the same register from the remote Modbus device. But the mask value is different.

Modbus Read Map #1

Read as: as: size:

After reading, apply (optional):

Bit Mask

Modbus Read Map #2

Read as: as: size:

After reading, apply (optional):

Bit Mask

The value derived from the first read map will be an 8-bit value taken from the low order byte of the 16-bit holding register. The value derived from the second read map will be an 8-bit value taken from the high order byte of the 16-bit holding register. This is how you "unpack" values from a packed Modbus register.

Another common requirement is to pick single bits out of a holding register. For a single bit, the Mask value will be a single bit like 00000001, 00000002, 00000004, 00000008, etc.

If the maps such as the two illustrated above are placed in sequential order in the read map list, they will result in only a single Modbus read request, and the data will be shared with all sequential maps requesting the same register from the same device.

Modbus TCP Client Write Maps

Read map Write map **Devices** Config File

Show 10 entries

Search:

Map number	Write from Local object	To Remote device	Remote register type	Remote register number	Remote register format
1  	3 : Object 3	1: BB2-6010	holding reg	3	signed int [1]
2  	1 : Object 1	1: BB2-6010	holding reg	4	unsigned int [1]
3  	2 : Object 2	1: BB2-6010	holding reg	4	unsigned int [1]
4  	1 : Object 1	1: BB2-6010	holding reg	5	signed int [1]
5  	2 : Object 2	1: BB2-6010	holding reg	6	signed int [1]
Map number	Write from Local object	To Remote device	Remote register type	Remote register number	Remote register format

Showing 1 to 5 of 5 entries

Previous 1 Next

Add new map(s) before map number:

Copy maps from through: and insert before:

Modbus TCP Client Write Maps are where you configure this device to write to Modbus registers in other Modbus TCP devices, taking data to be written from local data objects. This page shows a list of currently configured write maps.

When Modbus display setting "Number" is selected under User Settings, Modbus registers will be displayed as type (e.g. holding register) and register number.

Modbus Settings

Modbus register number display format: **number** ▼

Map number	Write from Local object	To Remote device	Remote register type	Remote register number	Remote register format
1  	3 : Object 3	1: BB2-6010	holding reg	3	signed int [1]
2  	1 : Object 1	1: BB2-6010	holding reg	4	unsigned int [1]
3  	2 : Object 2	1: BB2-6010	holding reg	4	unsigned int [1]

- Click on the map number or pencil icon to jump to the map editing page for this map.
- Click on the trash can icon to delete this map.

When Modbus display setting "Modicon" is selected under User Settings, Modbus registers will be displayed as a single value in Modicon format, e.g. 40001 for holding register 1.

Modbus Settings

Modbus register number display format: **modicon** ▼

Map number	Write from Local object	To Remote device	Remote register	Remote register format
1  	3 : Object 3	1: BB2-6010	40003	signed int [1]
2  	1 : Object 1	1: BB2-6010	40004	unsigned int [1]
3  	2 : Object 2	1: BB2-6010	40004	unsigned int [1]

- Click on the map number or pencil icon to jump to the map editing page for this map.
- Click on the trash can icon to delete this map.

Add new map(s) before map number:

To add or insert new maps, enter a number of maps to add, and select a starting point. Then click the Insert button.

Copy maps from through: and insert before:

To copy one or more maps and insert a duplicate copy of them at another point in the list, select the range to copy, select the insert point, and click Copy.

Modbus TCP Client Write Map Edit

Modbus TCP Client Write Maps are where you configure this device to write to Modbus registers in other Modbus TCP devices, taking data to be written from local data objects. This page is where you enter the various parameters to make that happen.

[Dashboard](#) / [Protocol Configuration](#) / [Modbus Client](#) / [Write Map #1](#)

Modbus TCP Write Map #1

Map Number – Used as a reference in the map list for ordering the maps. Polling is done in round robin fashion in the order of map number.

Take data from named

Source Object – Specifies the local object number that contains the data that should be sent by this write map.

Apply (optional):

Scale – Provides a scale factor if non-zero (has the effect of being 1 if zero). Data to be written is retrieved from the local object and then multiplied by this scale factor before being sent to the remote Modbus device. Applies to numeric values and numeric local objects only.

Offset – Provides an offset to work in conjunction with scale factor. This value is added to the value retrieved from the local object (after being multiplied by scale) before being sent to the remote Modbus device. Applies to numeric values and numeric local objects only.

Then apply (optional):

Bit Mask	00000000	Hexadecimal	Bit Fill	00000000	Hexadecimal
----------	----------	-------------	----------	----------	-------------

Mask – A bit mask given as a 8-digit hexadecimal value, if non-zero. The mask operation skipped if mask value is zero, or register format is not Int (integer). When the data of interest is a single bit, or bit field less than the full register width, the Mask is used. The process used in a read operation is reversed here. First, the mask is right justified so that the least significant “1” bit is in the least significant data position. That mask is then logically ANDed with the data found in the local object. The result is then left justified back into the position originally indicated by the mask. This value is now ready to be written to the Modbus register, pending any additional operation such as the Fill mask.

Fill – An additional bit mask given as a 4-digit or 8-digit hexadecimal value. This mask is logically ORed with the result of the Mask operation before the final result is written to the Modbus register. The Fill mask has the effect of making sure certain bits in the register are always set.

NOTE: The order of operation is as follows, operating on data retrieved from the local object: (1) apply scale if nonzero; (2) apply offset; (3) apply mask if applicable; (4) apply fill if applicable; (5) write to Modbus register.

IMPORTANT: In order for the gateway to accumulate all data for a "packed" register using mask (and optionally fill) into a single write request, it is required that all write maps to the same register be in sequential (contiguous) order. If the write maps are split up such that multiple writes to the same register occur, then the later write will erase the content written by the earlier write.

The appearance of the following line will vary depending on your user settings. You have the option of displaying Modbus registers as raw address (0-indexed), register number (1-indexed), or Modicon format (e.g. 40001 style).

The following variations all refer to exactly the same register:

Write holding req ▼ number: 3 as: signed ▼ int ▼ size: 1 ▼

Write holding req ▼ address: 2 as: signed ▼ int ▼ size: 1 ▼

Write Modicon register: 40003 as: signed ▼ int ▼ size: 1 ▼

Click the Check to validate a Modicon number.

The options available on this line will vary depending on selections made. The following are a few examples.

Write holding req ▼ number: 3 as: signed ▼ int ▼ size: 1 ▼

Write holding req ▼ number: 3 as: signed ▼ int ▼ size: 2 ▼

Write holding req ▼ number: 3 as: real ▼ size: 2 ▼

Write holding req ▼ number: 3 as: char ▼ size: 10 ▼

Write coil ▼ number: 3 as: bit ▼ size: 1 ▼

Register Type – Modbus register types available are listed in the following table. These labels are illustrated here as recognized in XML or CSV files, but are further annotated on the web page.

Label Modbus Register type

- “none” No register defined
- “Coil” Coil
- “Disc” Discrete Input
- “Input” Input Register
- “Hold” Holding Register

Register Number or Address – Enter the number (starting at 1) or raw address (starting at 0) as applicable. Do NOT enter 40001 for holding register 1 if you have not selected Modicon as the display format in your User Settings.

Modicon Register – Enter numbers like 40001 for the first holding register if you have selected Modicon representation in your User Settings.

Register Format – Select the format of the data contained in the Modbus register(s). This is not used by the protocol, but is used by the gateway to interpret what the raw increments of 1 or 16 bits should mean. Select format from the following table.

Format Label	Format description
“None”	No format defined
“Bit”	Single bit, used ONLY for Register Type Coil or Disc
“Int”	Integer (size and whether signed are defined by labels below)
“Real”	Floating point (single or double precision)
“Char”	Character string with 2 ASCII characters per register
“Mod10”	Mod10 format, can be 2, 3, or 4-register, specific to Schneider Electric meters

Register Size – Register size refers to the number of consecutive input or holding registers that should be written for a value greater than 16 bits. A 16-bit value would have size of 1, a 32-bit value would have size of 2, and a 64-bit value would have size of 4. Single precision Real (32-bit IEEE 754 floating point) would be size 2, and double precision Real (64-bit IEEE 754 floating point) would be size 4. If format is Mod10, then valid sizes are 2, 3, or 4 – check manufacturer’s documentation if Mod10 is noted. Register “size” for a character string will be character count divided by 2 (plus 1 if string length is an odd number). Register Size is not used for Coil or Disc types.

Unsigned – Select signed or unsigned. Defaults to signed integer. Has no effect on RegFormat other than Int.

with register first.

Endian Selection – Used when Register Size is greater than 1 to indicate what order the registers should be interpreted in. Select "low" to indicate that the lowest numbered register contains the least significant portion of data. Select "high" to indicate that the lowest numbered register contains the most significant portion of data. Although Modbus protocol itself is not inherently “Little Endian”, many devices operate that way due to Intel processors being inherently Little Endian. Modbus protocol does not stipulate what the register order should be when multiple registers are treated as a single data entity. Therefore, the user is required to pay attention to this.

To Unit

Device – Select a device from the TCP Device list that should be accessed for this write attempt on a TCP network.

Unit - Unit number to be included in the TCP request, will default to 1 if not given or is set to zero. Web page will force it to default to 1, but is optional for CSV or XML import.

Use function codes 5-6 for single writes (instead of default 15-16)

Function code 5-6 – Check this box to force single register writes to use Modbus function 5 to write a single coil, or function 6 to write a single holding register. Function codes will default to “write multiple” function codes 15 and 16 instead of 5 and 6 respectively if this box is not checked. This box only appears when the register count to be written is one.

Repeat periodically

Repeat periodically every seconds

Send Periodic – Uncheck to disable, check to enable periodic writing of the Modbus register at the poll rate given by Poll Time.

Poll Time – Poll time in seconds, can be fractional. This poll time is not guaranteed to be met. Polling is done in round-robin fashion. In a very busy system, more than this time may expire before the next poll. If less than this time has expired, then the system will wait this amount of time until polling again. The sets the rate at which the remote Modbus register will be written, provided “Send Periodic” has been enabled. This poll time will be disregarded if Send Periodic is not enabled.

Write when object changes

Write when object changes by seconds

Send On Delta – Uncheck to disable, or check to enable the “send on delta” feature where Modbus writes are made based on changes in the local object value (see delta below).

Delta – Specifies the margin by which the local object value should change before sending another Modbus write request to the remote Modbus device. Once the changed value has been sent, the new local value is retained for future comparison in determining subsequent additional change. The delta value is disregarded if Send On Delta is not enabled. Note that a delta of zero is treated as a special case: Any update to the local object by any process will result in a new Modbus write request.

Min Quiet Time – Time in seconds, can be fractional. This specifies the minimum amount of time that should elapse between sending of write requests for this write map. The minimum quiet time has the effect of throttling network traffic, especially where delta is a small value.

Enable Max. quiet time

Enable Max. quiet time of

Send Max Quiet – Uncheck to disable or check to enable the Max Quiet Time feature. If disabled, the Max Quiet Time value will be disregarded.

Max Quiet Time – “Max Quiet” time in seconds, can be fractional. If the Modbus register has not been written either as a result of poll timing or value changing by delta within this time period, then write request will be made anyway. This specifies the maximum amount of time that should expire without any write to the Modbus register for any reason.

Write when equals

Index Object – Optional, allows for selectively enabling this write operation. If an index object (local object number) is given, and its value matches the index value, then this write operation will take place. If an index object is given but the local object’s value does not match the index value, then this write operation will be skipped.

Index Value – Optional, used in conjunction with Index Object (see note above).

Modbus TCP Devices

Read map Write map **Devices** Config File

Show 10 entries

Search:

Device Number ↑↓	Name ↑↓	IP Address ↑↓
1  	BB2-6010	192.168.1.87
2  	BBPro-V210	192.168.1.23
Device Number	Name	IP Address

Showing 1 to 2 of 2 entries

Previous 1 Next

Add new device number 

The Modbus TCP Devices page is where you set up the list of Modbus TCP servers (slaves) that this IoTServer will read from and write to while acting as a Modbus TCP client (master).

Device Number ↑↓	Name ↑↓	IP Address ↑↓
1  	BB2-6010	192.168.1.87

- Click on the device number or pencil icon to jump to the device editing page for this device.
- Click on the trash can icon to delete this device.

Add new device number 

To add additional devices to the list, enter a device number to be added, and click Add Device. If the device number already exists, you will simply be editing that device. If the device number did not exist, you will create that device by editing it.

Modbus TCP Client Device Edit

The Modbus TCP Devices page is where you set up the list of Modbus TCP servers (slaves) that this IoTServer will read from and write to while acting as a Modbus TCP client (master). The full set of parameters required for each device is set up once here, and then referenced by device number in any number of read or write maps.

Dashboard / Protocol Configuration / Modbus Client / TCP Device #1

Modbus TCP Device #1

Number – Device number that will be reference in read and write maps.

Local Name: BB2-6010

Name – Arbitrary name for this device, used only as a reference in the web UI.

IP Address: 192.168.1.87

Port: 502

RemoteIP – The IP address in “a.b.c.d” form that the TCP client will attempt to connect to for read and write maps referencing this device number.

Port – The port number that should be opened at the remote IP address given. Port number will default to 502 if not given or set to zero.

Unit Number: 1

Unit – An optional unit number that will be used when connecting to this remote TCP device. If none is given or is set to zero, then unit 1 will be used.

Default Poll Period:  seconds

Poll Time – The default poll time that will be used if none is given explicitly in individual read and write maps.

Timeout:  seconds

Timeout – The amount of time that the client should wait before flagging the attempted read or write with a “no response” error.

Modbus TCP Client Config File

Dashboard / Protocol Configuration / Modbus TCP Client (Ethernet)

Read map Write map Devices **Config File**

All of your configuration information is stored in an internal database each time you click the Save button on any page where configuration entries may be made. To make configuration portable from one device to another, and for purposes of retaining a backup copy, the configuration information may be exported and imported as XML or CSV files. This page is where your configuration file management takes place.

It is important to note that the XML file saved within any one client/server function will contain the configuration information for only that function. Depending on overall system configuration, a complete backup may involve more than one XML or CSV file.

Configuration File Load/Save

Configuration file: 


 



XML Files: When an XML file has been selected, click the Load button to clear the configuration database and reload configuration from the given XML file.

Select an existing name to overwrite or enter a new file name, and then click Save to write the current configuration to the file in XML format.

You may type in a new name in the file name window for purposes of saving a new file. If you click the Refresh button, the file name will be restored to the name currently loaded into the client. The name could have been changed by selecting a file from the list below, or by typing in a new name. If the displayed name has not yet been used, then Refresh will restore the file name to what was most recently loaded.

Configuration File Load/Save



Configuration file: 


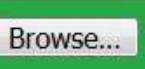


 

CSV Files: If a CSV file is selected, the Load and Save buttons will change into load/save CSV buttons. When a CSV file has been selected, click the Load button to clear the configuration database and reload configuration from the given CSV file.

Select an existing name to overwrite or enter a new file name, and then click Save to write the current configuration to the file in CSV format.

Configuration File Management

Available Configuration Files:  

  No files selected.  

The drop-down list will show a list of all configuration files currently found in the device's configuration folder. When you select an XML or CSV file from this list, the name will be copied to the Load/Save section of this page for pending load or save.

You may view the selected file by simply clicking View. You can delete the file by clicking Delete.

You may upload files to the IoTServer from your PC. Start by clicking Browse, and then use the browser's file dialog to locate the file on your PC. Once a file is selected on your PC, click the "Start upload" button to initiate the transfer.

You may also download files from the IoTServer to your PC. Click the Download button to transfer the selected file to your PC.

Configuration Error Logs

Configuration Error Logs:  

Any time an XML or CSV file is loaded, an error log file is generated. The error log file will be given the same name as the loaded file, but with ".err" as the suffix instead of ".xml" or ".csv". You may view the error log by selecting it from the list and clicking View.

Status is normally displayed in a message box at the top of the screen when the load or save operation is complete. But if you want to double check the status of the previous file operation, click Check Status.

 **Current Status: Running** 

The task (client or server) needs to be suspended while a file load operation is in progress to prevent acting on any partial configurations. This suspend/resume operation will normally happen automatically as part of the sequence invoked by the Load button when loading an XML file. The task must be explicitly suspended here for importing a CSV file. The Suspend button will become a Resume button when the task is suspended. Click Resume to continue operation. The current status is always displayed here.

Modbus TCP Client XML Files

Example XML File

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<configuration>

<modbus_client_devices>
<dev id="1" ipaddr="192.168.1.87" unit="1" name="BB2-6010"/>
</modbus_client_devices>

<modbus_client_read>
<map objnum="1" type="hold" addr="0" format="int" size="1" unsigned="1" mask="f" dev="1"
poll="2.0"/>
<map objnum="2" type="hold" addr="0" format="int" size="1" unsigned="1" mask="f0"
dev="1" poll="2.0"/>

```



```

<map objnum="3" type="hold" addr="1" format="int" size="1" dev="1" poll="2.0"/>
</modbus_client_read>

<modbus_client_write>
<map objnum="3" type="hold" addr="2" format="int" size="1" dev="1" poll="2.0"/>
<map objnum="1" type="hold" addr="3" mask="f" format="int" size="1" unsigned="1" dev="1"
poll="2.0"/>
<map objnum="2" type="hold" addr="3" mask="f0" fill="100" format="int" size="1"
unsigned="1" dev="1" poll="2.0"/>
<map objnum="1" type="hold" addr="4" format="int" size="1" dev="1" poll="2.0"/>
<map objnum="2" type="hold" addr="5" format="int" size="1" dev="1" poll="2.0"/>
</modbus_client_write>

</configuration>

```

Modbus TCP Client <modbus_client_devices> Section

id="n" – Specifies the device number that will be referenced in read and write maps. The client uses the device number found in the read or write map to look up the Modbus TCP IP address in this table.

ipaddr="a.b.c.d" – IP address of remote Modbus TCP server (slave).

port="n" – Port number to query on remote Modbus TCP device, defaults to the standard Modbus TCP port 502 if not provided or is zero.

unit="n" – Unit number to include in Modbus TCP query, sometimes used to route queries to multiple RTU devices on the other side of a TCP gateway. Note: This unit number is only a default value; the unit number given in the read or write map (if any) will override this value.

rate="n" – The default poll rate for read or write maps used when no poll time is provided explicitly in the read or write map.

timeout="n" – Timeout for slave/server response. If no reply received in response to a query within this amount of time, the attempt is flagged as a “no response” error.

name="xxx" – A name given for reference purposes only, used on web pages and in XML files, but has no bearing on Modbus protocol activity.

Modbus TCP Client <modbus_client_read> Section

Map Number is implied by order in XML – Used as a reference in the map list for ordering the maps. Polling is done in round robin fashion in the order of map number.

type="xxx" – Specifies Modbus register type to read. Valid values in XML and are shown below:

XML value Modbus Register type

"none"	No register defined
"coil"	Coil
"disc"	Discrete Input
"input"	Input Register
"hold"	Holding Register

addr="n" – Specifies register address to read. Note that this is address starting from zero, not register number starting from one like most Control Solutions gateways. Valid address range is 0..65535. NOTE: The Web UI provides the means to let the user switch between display of addresses (0-indexed), register numbers (1-indexed), or Modicon notation (where number and type are combined into one value).

format="xxx" – Specifies the format in which the Modbus data should be interpreted. Valid formats for XML are shown below:

XML value	Format description
"none"	No format defined
"bit"	Single bit (coil, discrete only)
"int"	Integer (16-, 32-, or 64-bit)
"real"	Floating point (single or double)
"char"	ASCII character string
"mod10"	Schneider Electric Mod10 format

size="n" – Specifies the size of the object in number of Modbus registers. All Modbus registers are either a single bit or 16 bits. It is up to the application to interpret multiple registers as other data sizes, which the client does automatically when configured to read multiple registers. Only certain combinations of data formats and register counts are valid. Note that character strings are defined as a number of registers each holding two ASCII characters.

Valid sizes by register format are as follows:

Type	Number of registers
Bit	1
Integer	1, 2, 4 (for 16-, 32-, 64-bit)
Real	2, 4 (for single, double precision)
Character	1..63 (registers - 2 characters per register)
Mod10	2, 3, 4

unsigned="n" – Registers default to being treated as signed when integer. If this attribute value is non-zero, then it will be treated as unsigned.

lowfirst="n" – Specifies “little endian” when value is non-zero. Defaults to “big endian” if omitted or value is zero. In “little endian” mode, the least significant data is in the first or lowest numbered Modbus register. In “big endian” mode, the most significant data is in the first or lowest numbered Modbus register. This only applies to multiple-register non-character data entities (e.g. 32-bit integer). Character strings are always stored with the first character in the string located in the first or lowest numbered Modbus register.

dev="n" – Specifies the device to look up in the Modbus TCP device table to obtain the IP address, port, etc, of the Modbus server/slave to query. (Not included in Modbus RTU XML files.)

unit="n" – Specifies the unit number to include in the TCP query (same as slave for RTU).

scale="n" – Scale factor – data read from a Modbus device is multiplied by the scale factor, then offset is added, to produce a final result that is placed into the local data object. If “mask” is also provided, mask is applied before scale and offset.

offset="n" – Offset – added to data read from Modbus device (see scale).

poll="n" – Poll time in seconds determines how often this remote Modbus register will be read.

objnum="n" – Specifies the local object where the result of the Modbus read should be placed.

mask="n" – Optional, and if non-zero, then the data received from the remote Modbus device is bitwise AND-ed with this bit mask, and then shifted right so that the least significant bit in the resulting value is the least significant bit retained by the mask. Note that mask is only valid when the Modbus register format is integer.

default="n" – Used in conjunction with failCount, if successive read attempts fail, the local object will be set to this default value.

maxfail="n" – Optional, specifies the number of read failures (if non-zero) required before the local object will be set to the default value. If set to zero, then the local object will always retain the most recently received data value, if any, regardless of how old it is.

indexobj="n" – Optional, allows for selectively enabling this read operation. If an index object (local object number) is given, and its value matches the indexValue, then this read operation will take place. If an indexObject is given but the local object's value does not match the indexValue, then this read operation will be skipped.

indexval="n" – Optional, used in conjunction with indexObject (see note above).

Modbus TCP Client <modbus_client_write> Section

Map Number is implied by order in XML – Used as a reference in the map list for ordering the maps. Polling is done in round robin fashion in the order of map number.

type="xxx" – Specifies Modbus register type to read. Valid values in XML are shown below:

XML value Modbus Register type

"none"	No register defined
"coil"	Coil
"disc"	Discrete Input
"input"	Input Register
"hold"	Holding Register

fc56="n" – Specifies, if non-zero, that Modbus function codes 5 and 6 should be used instead of the default 15 and 16 for writing coils and holding registers (single versus multiple).

addr="n" – Specifies register address to read. Note that this is address starting from zero, not register number starting from one like most Control Solutions gateways. Valid address range is 0..65535. NOTE: The Web UI should provide the means to let the user switch between display of addresses (0-indexed), register numbers (1-indexed), or Modicon notation (where number and type are combined into one value).

format="xxx" – Specifies the format in which the Modbus data should be interpreted. Valid formats for XML are shown below:

XML value Format description

"none"	No format defined
--------	-------------------

- “bit” Single bit (coil, discrete only)
- “int” Integer (16-, 32-, or 64-bit)
- “real” Floating point (single or double)
- “char” ASCII character string
- “mod10” Schneider Electric Mod10 format

size=“n” – Specifies the size of the object in number of Modbus registers. All Modbus registers are either a single bit or 16 bits. It is up to the application to interpret multiple registers as other data sizes, which the client does automatically when configured to write multiple registers. Only certain combinations of data formats and register counts are valid. Note that character strings are defined as a number of registers each holding two ASCII characters.

Valid sizes by register format are as follows:

Type	Number of registers
Bit	1
Integer	1, 2, 4 (for 16-, 32-, 64-bit)
Real	2, 4 (for single, double precision)
Character	1..63 (registers - 2 characters per register)
Mod10	2, 3, 4

unsigned=“n” – Registers default to being treated as signed when integer. If this attribute value is non-zero, then it will be treated as unsigned.

lowfirst=“n” – Specifies “little endian” when value is non-zero. Defaults to “big endian” if omitted or value is zero. In “little endian” mode, the least significant data is in the first or lowest numbered Modbus register. In “big endian” mode, the most significant data is in the first or lowest numbered Modbus register. This only applies to multiple-register non-character data entities (e.g. 32-bit integer). Character strings are always stored with the first character in the string located in the first or lowest numbered Modbus register.

dev=“n” – Specifies the device to look up in the Modbus TCP device table to obtain the IP address, port, etc, of the Modbus server/slave to query. (Not included in Modbus RTU XML files.)

unit=“n” – Specifies the unit number to include in the TCP query (same as slave for RTU).

scale="n" – Scale factor – data from the local object is first multiplied by the scale factor, then offset is added, to produce the register content written to the remote Modbus device. If “mask” and/or “fill” are also provided, mask and fill are applied after scale and offset (opposite order compared to read operation).

offset="n" – Offset – added to data to be written to Modbus device (see scale).

poll="n" – Poll time in seconds specifies how often this Modbus register will be written, if periodic polling is enabled. Writing only when the local object changes is an option, in which case the poll time is disregarded.

objnum="n" – Specifies the local object that is the source of data to be written to the remote Modbus device.

mask="n" – Optional, if non-zero, then the data to be written to Modbus is shifted into the position marked by bits set in the mask, and the data is then bitwise AND-ed with the mask. If fill is also provided, then fill is applied after mask, and the result is written to the remote Modbus device. Note that mask is only valid when the Modbus register format is integer.

fill="n" – Optional, and if non-zero, provides a collection of bits that should always be set in the data written to Modbus. The fill is bitwise OR-ed with the data after scale, offset, and mask have been applied. Note that fill is only valid when the Modbus register format is integer.

maxquiet="n" – Specifies the maximum amount of time that may elapse with no transmission of new data to the Modbus slave. This effectively provides a fallback to periodic sending if set to “send on delta” but there is never any change. The Max Quiet Time function is disabled if this attribute is omitted in XML,

minquiet="n" – Specifies the minimum amount of time that must elapse between transmissions of new data to the Modbus slave. This effectively throttles network traffic when “send on delta” is enabled and the value is changing rapidly. A time of zero effectively disables this feature.

delta="n" – Specifies the amount by which the object value must change before the new value will be transmitted to the Modbus slave. If this attribute is omitted in XML, then the “send on delta” function is disabled. The delta value may be zero, in which case any change is transmitted.

indexobj="n" – Optional, allows for selectively enabling this write operation. If an index object (local object number) is given, and its value matches the indexValue, then this write operation will take place. If an indexObject is given but the local object’s value does not match the indexValue, then this write operation will be skipped.

indexval="n" – Optional, used in conjunction with indexObject (see note above).

Modbus TCP Client CSV Files

A CSV file may be imported to configure various aspects of the IoTServer (or Babel Buster gateway). A single CSV file may contain multiple sections. When a file including an “Objects” section is imported by the Data Engine, local objects will be configured. When a file including one or more “Modbus” sections is imported by an instance of the Modbus Engine, Modbus gateway functionality will be configured. The same Modbus file may be imported by a Modbus Client or Modbus Server, and either RTU or TCP, and only those sections of interest to that Modbus function will be imported. The CSV file may also contain one or more SNMP sections, and so forth.

A section begins when the word “Begin” appears in the first column of a line. All lines up to and including a line that begins with the word “End” will be taken to be part of that section.

The line immediately following the “Begin” line must be a header line. A Header line is one which labels the columns of data that will follow the Header line.

All lines following the Header line are data lines that are expected to contain the same number of columns as the Header line, and whose contents are defined by the labels found in each column of the Header line.

Labels in the section Begin and End lines, and labels in the Header line are NOT case sensitive and will be interpreted equally whether upper case, lower case, or some combination of both (for readability).

Labels may NOT contain embedded spaces. A label is terminated by a comma, line-end, or space. Labels may not be encapsulated in quote characters; however, data content in data lines may be encapsulated in quote characters and may contain embedded spaces or blanks if quoted.

Some labels in the Header line may be considered optional. The minimum required columns are indicated in the definition of each data section.

Columns in the Header line do not have to follow any particular order. They may be rearranged to the user’s liking. The only restriction is that data in subsequent data lines must match up with the labels placed in the Header line. Data lines may contain fewer columns than the Header line, but may not contain more. Data columns that the user wishes to deliberately omit, but omit between included columns, should be indicated by place holder commas (which will simply appear as blank cells in a spread sheet program).

A Begin line will contain three columns:

Column 1: BEGIN

Column 2: Function as noted below

Column 3: Sub-function as noted in definition of the Function.

Functions may be any of the following (with this listed expanded from time to time):

- LOCALDATA
- MODBUS
- SNMP

Sub-functions:

MODBUS

- DEVICES
- READMAPS
- WRITEMAPS
- SERVERMAPS
- SERVERREMAPS

NOTE: A Modbus CSV file may contain both client and server sections as the respective functionality will select sections relevant to its purpose. It should also be noted that the same application, `csiModbusEngine`, can function as both client and server (master and slave).

MODBUS Client CSV Example

The following illustrates a valid Modbus TCP Client CSV configuration file. The DEVICES, READMAPS, and WRITEMAPS sections of a CSV file will be processed when the Modbus Engine is operating as a Client. Server sections (see later section) will only be processed by an instance of a Modbus Server, and the server will skip over the Client sections illustrated here.

A valid Modbus RTU Client (Master) CSV configuration file would look largely the same, except there is no DEVICES section in a Modbus RTU client. The other minor variation is that “Device,Unit” in the read/write maps are replaced with “Slave”.

```

Begin,Modbus,Devices
Number,RemoteIP,Port,PollTime,Timeout,Name
1,192.168.1.135,502,2,2,SPX
End
Begin,Modbus,ReadMaps
Device,RegType,RegAddr,RegFormat,RegSize,DestObj,PollTime
1,Hold,0,Int,1,1,5
1,Hold,1,Int,1,2,5
1,Hold,2,Int,1,3,5
End
Begin,Modbus,WriteMaps
SourceObj,Device,RegType,RegAddr,RegFormat,RegSize,PollTime
4,1,Hold,3,Int,1,10
5,1,Hold,4,Int,1,10
End
    
```


MODBUS (TCP client) DEVICES Section

The DEVICES section is only processed by an instance of the Modbus Engine that is running as a TCP client. An RTU client will not reference TCP devices. RTU devices are referenced simply by slave address included in the read and write maps. RTU port settings are not represented as CSV since there is only a single instance which may be set via the web UI.

Number – Device number that will be reference in read and write maps.

RemoteIP – The IP address in “a.b.c.d” form that the TCP client will attempt to connect to for read and write maps referencing this device number.

Port – The port number that should be opened at the remote IP address given. Port number will default to 502 if not given or set to zero.

Unit – An optional unit number that will be used when connecting to this remote TCP device. If none is given or is set to zero, then unit 1 will be used.

PollTime – The default poll time that will be used if none is given explicitly in individual read and write maps.

Timeout – The amount of time that the client should wait before flagging the attempted read or write with a “no response” error.

Name – Arbitrary name for this device, used only as a reference in the web UI.

MODBUS (client/master) READMAPS Section

Device – (REQUIRED if TCP) – Device number from the TCP Device list that should be accessed for this read attempt on a TCP network.

Unit – (TCP only) – Unit number to be included in the TCP request, will default to 1 if not given or is set to zero.

RegType – Modbus register type from following table, will default to “HOLD” if omitted. The labels must be entered exactly as depicted in the table.

Label	Modbus Register type
-------	----------------------

“none”	No register defined
--------	---------------------

“Coil”	Coil
--------	------

“Disc”	Discrete Input
--------	----------------

“Input” Input Register

“Hold” Holding Register

RegAddr – (REQUIRED if MODICON not used) – Raw 0-indexed address of the register to be read. IMPORTANT: If manufacturer’s documentation indicates register 40001, DO NOT enter 40001 for RegAddr. This number is short-hand for holding register 1, and its address is zero. Therefore, if you see 40001, select “Hold” for RegType, and enter 0 for RegAddr.

Modicon – (In lieu of RegType, RegAddr) – If one wishes to use Modicon notation, i.e., enter 40001 when the manufacturer’s documentation says 40001, then OMIT RegType AND RegAddr, and use the Modicon label instead. Both standard and extended Modicon are recognized. However, you cannot use both Modicon and RegType/RegAddr in the same section. When Modicon is used, the RegType and RegAddr columns will be generated internally based on the Modicon number given. Modicon is only available for Import. On export, RegType and RegAddr will be used (Modicon notation is not recognized by the Modbus protocol standard even though widely used as a defacto stanard).

RegFormat – Format of the data contained in the Modbus register(s), not used by the protocol, but used by the gateway to interpret what the raw increments of 1 or 16 bits should mean. Select format from the following table.

Format Label	Format description
“None”	No format defined
“Bit”	Single bit, used ONLY for RegType Coil or Disc
“Int”	Integer (size and whether signed are defined by labels below)
“Real”	Floating point (single or double precision)
“Char”	Character string with 2 ASCII characters per register
“Mod10”	Mod10 format, can be 2, 3, or 4-register, specific to Schneider Electric meters

RegSize – Register size refers to the number of consecutive input or holding registers should be read for a value greater than 16 bits. A 16-bit value would have size of 1, a 32-bit value would have size of 2, and a 64-bit value would have size of 4. Single precision Real (32-bit IEEE 754 floating point) would be size 2, and double precision Real (64-bit IEEE 754 floating point) would be size 4. If format is Mod10, then valid sizes are 2, 3, or 4 – check manufacturer’s documentation if Mod10 is noted. Register “size” for a character string will be character count divided by 2 (plus 1 if string length is an odd number). RegSize is not used for Coil or Disc types.

Unsigned – Indicate “Y” if unsigned, or “N” if signed. Defaults to signed integer. Has no effect on RegFormat other than Int.

LittleEnd – Used when RegSize is greater than 1 to indicate what order the registers should be interpreted in. Enter “Y” to indicate that the lowest numbered register contains the least significant portion of data. Enter “N” or omit to indicate that the lowest numbered register contains the most significant portion of data. Although Modbus protocol itself is not inherently “Little Endian”, many devices operate that way due to Intel processors being inherently Little Endian. Modbus protocol does not stipulate what the register order should be when multiple registers are treated as a single data entity. Therefore, the user is required to pay attention to this.

Mask – A bit mask given as a 4-digit or 8-digit hexadecimal value, if non-zero (Mask operation skipped if mask value is zero, or register format is not Int). When the data of interest is a single bit, or bit field less than the full register width, the Mask is used. When the Modbus register is read, its data is bit-wise ANDed with the Mask, then right justified so that the least significant mask bit becomes the least significant data bit. The result is then placed in the local data object selected by the read map.

Scale – Register content is multiplied by this value, if non-zero, before being placed into the local object.

Offset – This value is added to the register content (after being multiplied by scale) before being placed into the local object.

NOTE: The order of operation is as follows: (1) read Modbus register; (2) apply mask if applicable; (3) apply scale if non-zero; (4) apply offset. Result is then placed in local object.

DestObj – (REQUIRED) – Local object number that the result of the Modbus read operation should be placed into.

PollTime – Poll time in seconds (can be fractional). The sets the rate at which the remote Modbus register will be read. This poll time is not guaranteed to be met. Polling is done in round-robin fashion. In a very busy system, more than this time may expire before the next poll. If less than this time has expired, then the system will wait this amount of time until polling again.

DefValue – Provides the default value that the local object should be set to in the event the FailCount is exceeded.

FailCount – Optional, provides a count of read failures, if non-zero, that can occur before the local object will be set to the default value given in this map. If zero, the default value will never be applied. If 1, then the default value will be applied upon the first failure (probably not recommended), and so on. The count is reset by a successful read.

IndexObj – Optional, allows for selectively enabling this read operation. If an index object (local object number) is given, and its value matches the IndexVal value, then this read operation

will take place. If an IndexObj is given but the local object's value does not match the IndexVal, then this read operation will be skipped.

IndexVal – Optional, used in conjunction with IndexObj (see note above).

MODBUS (client/master) WRITEMAPS Section

SourceObj – Specifies the local object number that contains the data that should be sent by this write map.

Scale – Provides a scale factor if non-zero (has the effect of being 1 if zero). Data to be written is retrieved from the local object and then multiplied by this scale factor before being sent to the remote Modbus device. Applies to numeric values and numeric local objects only.

Offset – Provides an offset to work in conjunction with scale factor. This value is added to the value retrieved from the local object (after being multiplied by scale) before being sent to the remote Modbus device. Applies to numeric values and numeric local objects only.

Mask – A bit mask given as a 4-digit or 8-digit hexadecimal value, if non-zero (Mask operation skipped if mask value is zero, or register format is not Int). When the data of interest is a single bit, or bit field less than the full register width, the Mask is used. The process used in a read operation is reversed here. First, the mask is right justified so that the least significant “1” bit is in the least significant data position. That mask is then logically ANDed with the data found in the local object. The result is then left justified back into the position originally indicated by the mask. This value is now ready to be written to the Modbus register, pending any additional operation such as the Fill mask.

Fill – An additional bit mask given as a 4-digit or 8-digit hexadecimal value. This mask is logically ORed with the result of the Mask operation before the final result is written to the Modbus register. The Fill mask has the effect of making sure certain bits in the register are always set.

NOTE: The order of operation is as follows, operating on data retrieved from the local object: (1) apply scale if nonzero; (2) apply offset; (3) apply mask if applicable; (4) apply fill if applicable; (5) write to Modbus register.

Device – (REQUIRED if TCP) – Device number from the TCP Device list that should be accessed for this write attempt on a TCP network.

Unit – (TCP only) – Unit number to be included in the TCP request, will default to 1 if not given or is set to zero.

RegType – Modbus register type from following table, will default to “HOLD” if omitted. The labels must be entered exactly as depicted in the table.

Label Modbus Register type

- “none” No register defined
- “Coil” Coil
- “Disc” Discrete Input
- “Input” Input Register
- “Hold” Holding Register

RegAddr – (REQUIRED if MODICON not used) – Raw 0-indexed address of the register to be read. IMPORTANT: If manufacturer’s documentation indicates register 40001, DO NOT enter 40001 for RegAddr. This number is short-hand for holding register 1, and its address is zero. Therefore, if you see 40001, select “Hold” for RegType, and enter 0 for RegAddr.

Modicon – (In lieu of RegType, RegAddr) – If one wishes to use Modicon notation, i.e., enter 40001 when the manufacturer’s documentation says 40001, then OMIT RegType AND RegAddr, and use the Modicon label instead. Both standard and extended Modicon are recognized. However, you cannot use both Modicon and RegType/RegAddr in the same section. When Modicon is used, the RegType and RegAddr columns will be generated internally based on the Modicon number given. Modicon is only available for Import. On export, RegType and RegAddr will be used (Modicon notation is not recognized by the Modbus protocol standard even though widely used as a defacto stanard).

RegFormat – Format of the data contained in the Modbus register(s), not used by the protocol, but used by the gateway to interpret what the raw increments of 1 or 16 bits should mean. Select format from the following table.

Format Label	Format description
“None”	No format defined
“Bit”	Single bit, used ONLY for RegType Coil or Disc
“Int”	Integer (size and whether signed are defined by labels below)
“Real”	Floating point (single or double precision)
“Char”	Character string with 2 ASCII characters per register
“Mod10”	Mod10 format, can be 2, 3, or 4-register, specific to Schneider Electric meters

RegSize – Register size refers to the number of consecutive input or holding registers should be written for a value greater than 16 bits. A 16-bit value would have size of 1, a 32-bit value would

have size of 2, and a 64-bit value would have size of 4. Single precision Real (32-bit IEEE 754 floating point) would be size 2, and double precision Real (64-bit IEEE 754 floating point) would be size 4. If format is Mod10, then valid sizes are 2, 3, or 4 – check manufacturer’s documentation if Mod10 is noted. Register “size” for a character string will be character count divided by 2 (plus 1 if string length is an odd number). RegSize is not used for Coil or Disc types.

UseFC56 – Enter “Y” to force single register writes to use Modbus function 5 to write a single coil, or function 6 to write a single holding register. Function codes will default to “write multiple” function codes 15 and 16 instead of 5 and 6 respectively if “N” is entered or this column is omitted.

Unsigned – Indicate “Y” if unsigned, or “N” if signed. Defaults to signed integer. Has no effect on RegFormat other than Int.

LittleEnd – Used when RegSize is greater than 1 to indicate what order the registers should be interpreted in. Enter “Y” to indicate that the lowest numbered register contains the least significant portion of data. Enter “N” or omit to indicate that the lowest numbered register contains the most significant portion of data. Although Modbus protocol itself is not inherently “Little Endian”, many devices operate that way due to Intel processors being inherently Little Endian. Modbus protocol does not stipulate what the register order should be when multiple registers are treated as a single data entity. Therefore, the user is required to pay attention to this.

SendPeriodic – Set to “N” to disable, or “Y” to enable periodic writing of the Modbus register at the poll rate given by PollTime.

PollTime – Poll time in seconds, can be fractional. This poll time is not guaranteed to be met. Polling is done in round-robin fashion. In a very busy system, more than this time may expire before the next poll. If less than this time has expired, then the system will wait this amount of time until polling again. The sets the rate at which the remote Modbus register will be written, provided “SendPeriodic” has been enabled. This poll time will be disregarded if SendPeriodic is not enabled.

SendMaxQuiet – Set to “N” to disable or “Y” to enable the MaxQuietTime feature. If disabled, the MaxQuietTime will be disregarded.

MaxQuietTime – “Max Quiet” time in seconds, can be fractional. If the Modbus register has not been written either as a result of poll timing or value changing by delta within this time period, then write request will be made anyway. This specifies the maximum amount of time that should expire without any write to the Modbus register for any reason.

SendOnDelta – Set to “N” to disable, or “Y” to enable the “send on delta” feature where Modbus writes are made based on changes in the local object value (see delta below).

Delta – Specifies the margin by which the local object value should change before sending another Modbus write request to the remote Modbus device. Once the changed value has been

sent, the new local value is retained for future comparison in determining subsequent additional change. The delta value is disregarded if SendOnDelta is not enabled. Note that a delta of zero is treated as a special case: Any update to the local object by any process will result in a new Modbus write request.

MinQuietTime – Time in seconds, can be fractional. This specifies the minimum amount of time that should elapse between sending of write requests for this write map. The minimum quiet time has the effect of throttling network traffic.

IndexObj – Optional, allows for selectively enabling this write operation. If an index object (local object number) is given, and its value matches the IndexVal value, then this write operation will take place. If an IndexObj is given but the local object’s value does not match the IndexVal, then this write operation will be skipped.

IndexVal – Optional, used in conjunction with IndexObj (see note above).

Socket Errors

The following is the full list of standard operating system error codes. Only some of these are applicable to sockets.

Symbolic	Numeric	Description
EPERM	1	Operation not permitted
ENOENT	2	No such file or directory
ESRCH	3	No such process
EINTR	4	Interrupted system call
EIO	5	I/O error
ENXIO	6	No such device or address
E2BIG	7	Argument list too long
ENOEXEC	8	Exec format error
EBADF	9	Bad file number
ECHILD	10	No child processes
EAGAIN	11	Try again

BB4-8422 Modbus TCP Client

ENOMEM	12	Out of memory
EACCES	13	Permission denied
EFAULT	14	Bad address
ENOTBLK	15	Block device required
EBUSY	16	Device or resource busy
EEXIST	17	File exists
EXDEV	18	Cross-device link
ENODEV	19	No such device
ENOTDIR	20	Not a directory
EISDIR	21	Is a directory
EINVAL	22	Invalid argument
ENFILE	23	File table overflow
EMFILE	24	Too many open files
ENOTTY	25	Not a typewriter
ETXTBSY	26	Text file busy
EFBIG	27	File too large
ENOSPC	28	No space left on device
ESPIPE	29	Illegal seek
EROFS	30	Read-only file system
EMLINK	31	Too many links
EPIPE	32	Broken pipe
EDOM	33	Math argument out of domain of func
ERANGE	34	Math result not representable
EDEADLK	35	Resource deadlock would occur

BB4-8422 Modbus TCP Client

ENAMETOOLONG	36	File name too long
ENOLCK	37	No record locks available
ENOSYS	38	Invalid system call number
ENOTEMPTY	39	Directory not empty
ELOOP	40	Too many symbolic links encountered
EWouldBLOCK	EAGAIN (11)	Operation would block
ENOMSG	42	No message of desired type
EIDRM	43	Identifier removed
ECHRNG	44	Channel number out of range
EL2NSYNC	45	Level 2 not synchronized
EL3HLT	46	Level 3 halted
EL3RST	47	Level 3 reset
ELNRNG	48	Link number out of range
EUNATCH	49	Protocol driver not attached
ENOCSI	50	No CSI structure available
EL2HLT	51	Level 2 halted
EBADE	52	Invalid exchange
EBADR	53	Invalid request descriptor
EXFULL	54	Exchange full
ENOANO	55	No anode
EBADRQC	56	Invalid request code
EBADSLT	57	Invalid slot
EDEADLOCK	EDEADLK (35)	Resource deadlock would occur
EBFONT	59	Bad font file format

BB4-8422 Modbus TCP Client

ENOSTR	60	Device not a stream
ENODATA	61	No data available
ETIME	62	Timer expired
ENOSR	63	Out of streams resources
ENONET	64	Machine is not on the network
ENOPKG	65	Package not installed
EREMOTE	66	Object is remote
ENOLINK	67	Link has been severed
EADV	68	Advertise error
ESRMNT	69	Srmount error
ECOMM	70	Communication error on send
EPROTO	71	Protocol error
EMULTIHOP	72	Multihop attempted
EDOTDOT	73	RFS specific error
EBADMSG	74	Not a data message
E_OVERFLOW	75	Value too large for defined data type
ENOTUNIQU	76	Name not unique on network
EBADFD	77	File descriptor in bad state
EREMCHG	78	Remote address changed
ELIBACC	79	Can not access a needed shared library
ELIBBAD	80	Accessing a corrupted shared library
ELIBSCN	81	.lib section in a.out corrupted
ELIBMAX	82	Attempting to link in too many shared libraries
ELIBEXEC	83	Cannot exec a shared library directly

BB4-8422 Modbus TCP Client

EILSEQ	84	Illegal byte sequence
ERESTART	85	Interrupted system call should be restarted
ESTRPIPE	86	Streams pipe error
EUSERS	87	Too many users
ENOTSOCK	88	Socket operation on non-socket
EDESTADDRREQ	89	Destination address required
EMSGSIZE	90	Message too long
EPROTOTYPE	91	Protocol wrong type for socket
ENOPROTOOPT	92	Protocol not available
EPROTONOSUPPORT	93	Protocol not supported
ESOCKTNOSUPPORT	94	Socket type not supported
EOPNOTSUPP	95	Operation not supported on transport endpoint
EPFNOSUPPORT	96	Protocol family not supported
EAFNOSUPPORT	97	Address family not supported by protocol
EADDRINUSE	98	Address already in use
EADDRNOTAVAIL	99	Cannot assign requested address
ENETDOWN	100	Network is down
ENETUNREACH	101	Network is unreachable
ENETRESET	102	Network dropped connection because of reset
ECONNABORTED	103	Software caused connection abort
ECONNRESET	104	Connection reset by peer
ENOBUFS	105	No buffer space available
EISCONN	106	Transport endpoint is already connected
ENOTCONN	107	Transport endpoint is not connected

BB4-8422 Modbus TCP Client

ESHUTDOWN	108	Cannot send after transport endpoint shutdown
ETOOMANYREFS	109	Too many references: cannot splice
ETIMEDOUT	110	Connection timed out
ECONNREFUSED	111	Connection refused
EHOSTDOWN	112	Host is down
EHOSTUNREACH	113	No route to host
EALREADY	114	Operation already in progress
EINPROGRESS	115	Operation now in progress
ESTALE	116	Stale file handle
EUCLEAN	117	Structure needs cleaning
ENOTNAM	118	Not a XENIX named type file
ENAVAIL	119	No XENIX semaphores available
EISNAM	120	Is a named type file
EREMOTEIO	121	Remote I/O error
EDQUOT	122	Quota exceeded
ENOMEDIUM	123	No medium found
EMEDIUMTYPE	124	Wrong medium type
ECANCELED	125	Operation Canceled
ENOKEY	126	Required key not available
EKEYEXPIRED	127	Key has expired
EKEYREVOKED	128	Key has been revoked
EKEYREJECTED	129	Key was rejected by service
EOWNERDEAD	130	Owner died
ENOTRECOVERABLE	131	State not recoverable

BB4-8422 Modbus TCP Client

ERFKILL	132	Operation not possible due to RF-kill
EHWPOISON	133	Memory page has hardware error