RUTX10 Modbus

The information in this page is updated in accordance with firmware version **RUTX_R_00.07.04.5**.

□

Contents

Summary

Modbus is a serial communications protocol. Simple and robust, it has become a de facto standard communication protocol and is now a commonly available means of connecting industrial electronic devices.

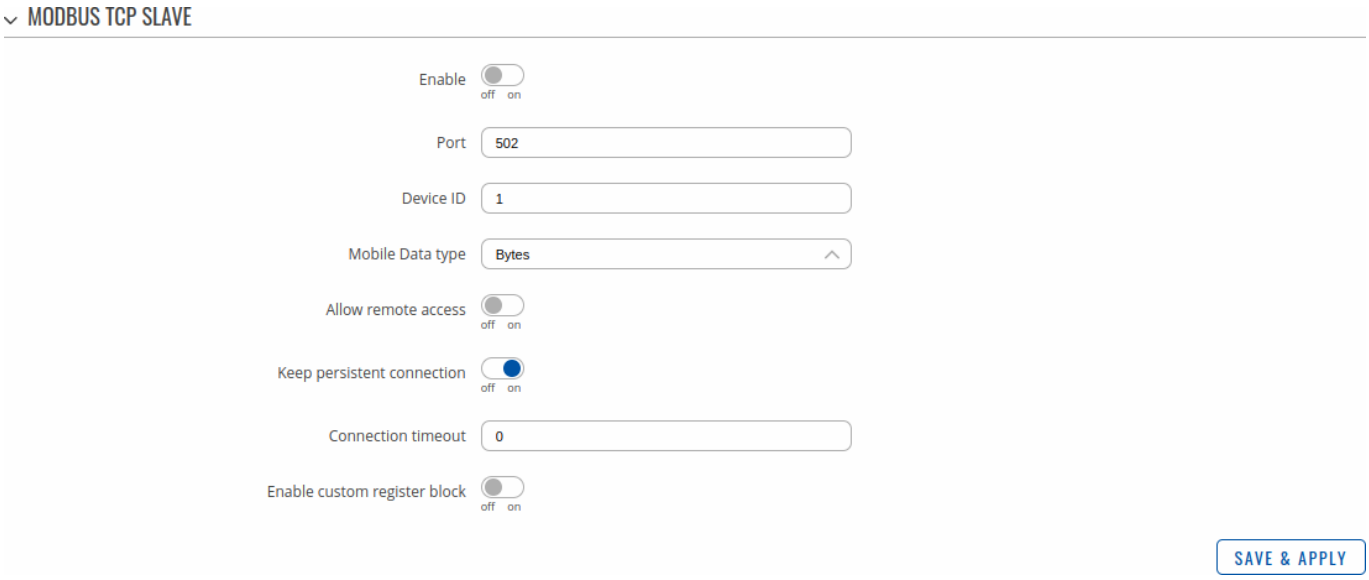
This manual page provides an overview of the Modbus functionality in RUTX10 devices.

If you're having trouble finding this page or some of the parameters described here on your device's WebUI, you should **turn on "Advanced WebUI" mode**. You can do that by clicking the "Basic" button under "Mode", which is located at the top-right corner of the WebUI.

Modbus TCP Slave

A **Modbus TCP Slave** listens for connections from a TCP master (client) and sends out a response or sets some system related parameter in accordance with the given query. This provides the user with the possibility to set or get system parameters.

The figure below is an example of the Modbus TCP window section and the table below provides information on the fields contained in that window:



Field	Value	Description
Enable	off | on; default: **off**	Turns Modbus TCP on or off.
Port	integer [0..65535]; default: **502**	TCP port used for Modbus communications.
Device ID	integer [0..255]; default: **1**	The device's Modbus slave ID. When set to 0, it will respond to requests addressed to any ID.
Mobile Data type	Bytes | Kilobytes | Megabytes; default: **Bytes**	Selects mobile data unit representation type.
Allow remote access	off | on; default: **off**	Allows remote Modbus connections by adding an exception to the device's firewall on the port specified in the field above.
Keep persistent connection	off | on; default: **onn**	Allows keep the connection open after responding a Modbus TCP master request.
Connection timeout	integer [0..60]; default: **0**	Sets TCP timeout in seconds after which the connection is forcefully closed.
Enable custom register block	off | on; default: **off**	Allows the usage of custom register block.

Register file path	path; default: **/tmp/regfile**	Path to file in which the custom register block will be stored. Files inside /tmp or /var are stored in RAM. They vanish after reboot, but do not degrade flash memory. Files elsewhere are stored in flash memory. They remain after reboot, but degrade flash memory (severely, if operations are frequent).
First register number	integer [1025..65536]; default: **1025**	First register in custom register block
Register count	integer [1..64512]; default: **128**	Register count in custom register block

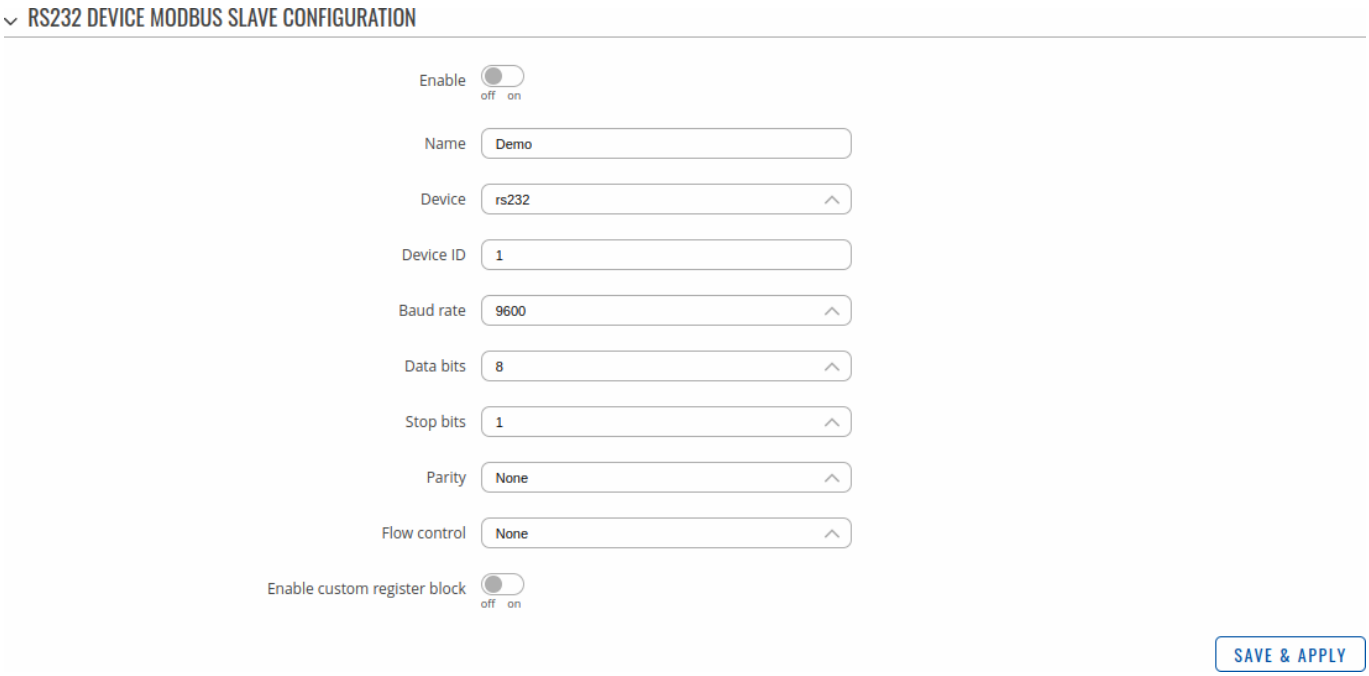
Modbus Serial Slave

A **Modbus Serial Slave** listens for connections from a serial master (client) and sends out a response or sets some system related parameter in accordance with the given query. This provides the user with the possibility to set or get system parameters.

Modbus Serial Slave Configuration

The **Modbus Serial Slave Configuration** section is used to configure serial slaves. By default, the list is empty. To add a new slave instance, enter the instance name, select serial interface and click the 'Add' button.

After clicking 'Add' you will be redirected to the newly added slave instance configuration page.



Field	Value	Description
Enable	off | on; default: **off**	Enables this Modbus Serial Slave instance configuration.

Name	string; default: **none**	Name of the serial slave instance. Used for management purposes only.
Device	USB RS232 interface; default: **USB RS232 interface**	Specifies which serial port will be used for serial communication.
Device ID	integer [0..255]; default: **1**	Specifies which serial port will be used for serial communication.
Baud rate	300 | 1200 | 2400 | 4800 | 9600 | 19200 | 38400 | 57600 | 115200; default: **9600**	Serial data transmission rate (in bits per second).
Data bits	5 | 6 | 7 |8; default: **8**	Number of data bits for each character.
Stop bits	1| 2; default: **1**	Stop bits sent at the end of every character allow the receiving signal hardware to detect the end of a character and to resynchronise with the character stream. Electronic devices usually use one stop bit. Two stop bits are required if slow electromechanical devices are used.
Parity	Even | Odd| Mark | Space | None; default: **None**	In serial transmission, parity is a method of detecting errors. An extra data bit is sent with each data character, arranged so that the number of 1 bits in each character, including the parity bit, is always odd or always even. If a byte is received with the wrong number of 1s, then it must have been corrupted. However, an even number of errors can pass the parity check. • **None** (**N**) - no parity method is used. • **Odd** (**O**) - the parity bit is set so that the number of "logical ones (1s)" has to be odd. • **Even** (**E**) - the parity bit is set so that the number of "logical ones (1s)" has to be even. • **Space** (**s**) - the parity bit will always be a binary 0. • **Mark** (**M**) - the parity bit will always be a binary 1.
Flow control	None | RTS/CTS | Xon/Xoff; default: **None**	In many circumstances a transmitter might be able to send data faster than the receiver is able to process it. To cope with this, serial lines often incorporate a "handshaking" method, usually distinguished between hardware and software handshaking. • **RTS/CTS** - hardware handshaking. RTS and CTS are turned OFF and ON from alternate ends to control data flow, for instance when a buffer is almost full. • **Xon/Xoff** - software handshaking. The Xon and Xoff characters are sent by the receiver to the sender to control when the sender will send data, i.e., these characters go in the opposite direction to the data being sent. The circuit starts in the "sending allowed" state. When the receiver's buffers approach capacity, the receiver sends the Xoff character to tell the sender to stop sending data. Later, after the receiver has emptied its buffers, it sends an Xon character to tell the sender to resume transmission.
Enable custom register block	off | on; default: **off**	Allows the usage of custom register block.

Register file path	path; default: **/tmp/regfile**	Path to file in which the custom register block will be stored. Files inside /tmp or /var are stored in RAM. They vanish after reboot, but do not degrade flash memory. Files elsewhere are stored in flash memory. They remain after reboot, but degrade flash memory (severely, if operations are frequent).
First register number	integer [1025..65536]; default: **1025**	First register in custom register block
Register count	integer [1..64512]; default: **128**	Path to file in which the custom register block will be stored. Files inside /tmp or /var are stored in RAM. They vanish after reboot, but do not degrade flash memory. Files elsewhere are stored in flash memory. They remain after reboot, but degrade flash memory (severely, if operations are frequent).

Modbus Registers

Get Parameters

Modbus parameters are held within **registers**. Each register contains 2 bytes of information. For simplification, the number of registers for storing numbers is 2 (4 bytes), while the number of registers for storing text information is 16 (32 bytes).

The register numbers and corresponding system values are described in the table below:

required value	register address	register number	number of registers	representation
System uptime	1	2	2	32 bit unsigned integer
System hostname	7	8	16	Text
Router serial number	39	40	16	Text
LAN MAC address	55	56	16	Text
Router name	71	72	16	Text
Current WAN IP address	139	140	2	8 bit unsigned integer
Digital non-isolated input	324	325	1	16 bit unsigned integer
Digital open collector output	325	326	1	16 bit unsigned integer
Unix timestamp	364	365	2	32 bit unsigned integer
Local ISO time	366	367	12	Text
UTC time	378	389	12	Text
LAN IP	394	395	2	8 bit unsigned integer

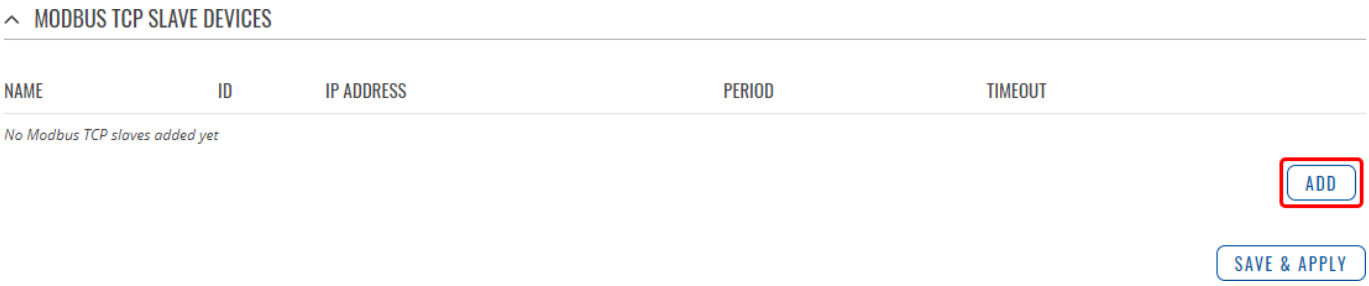
Set Parameters

The Modbus daemon can also set some device parameters.

value to set	register address	register number	register value	description
Hostname	7	8	Hostname (in decimal form)	Changes hostname
Device name	71	72	Device name (in decimal form)	Changes device name
Switch WiFi ON/OFF	203	204	1 | 0	Turns WiFi ON or OFF
Reboot	206	207	1	Reboots the router
Switch PIN 4 state	325	326	1|0	Toggles PIN 4 ON or OFF
Switch 2.4GHz WiFi ON/OFF	390	391	1 | 0	Turns 2.4GHz WiFi ON or OFF
Switch 5GHz WiFi ON/OFF	391	392	1 | 0	Turns 5GHz WiFi ON or OFF
Change LAN IP	394	395	IPv4 (in decimal form)	Changes device LAN IP

Modbus TCP Master

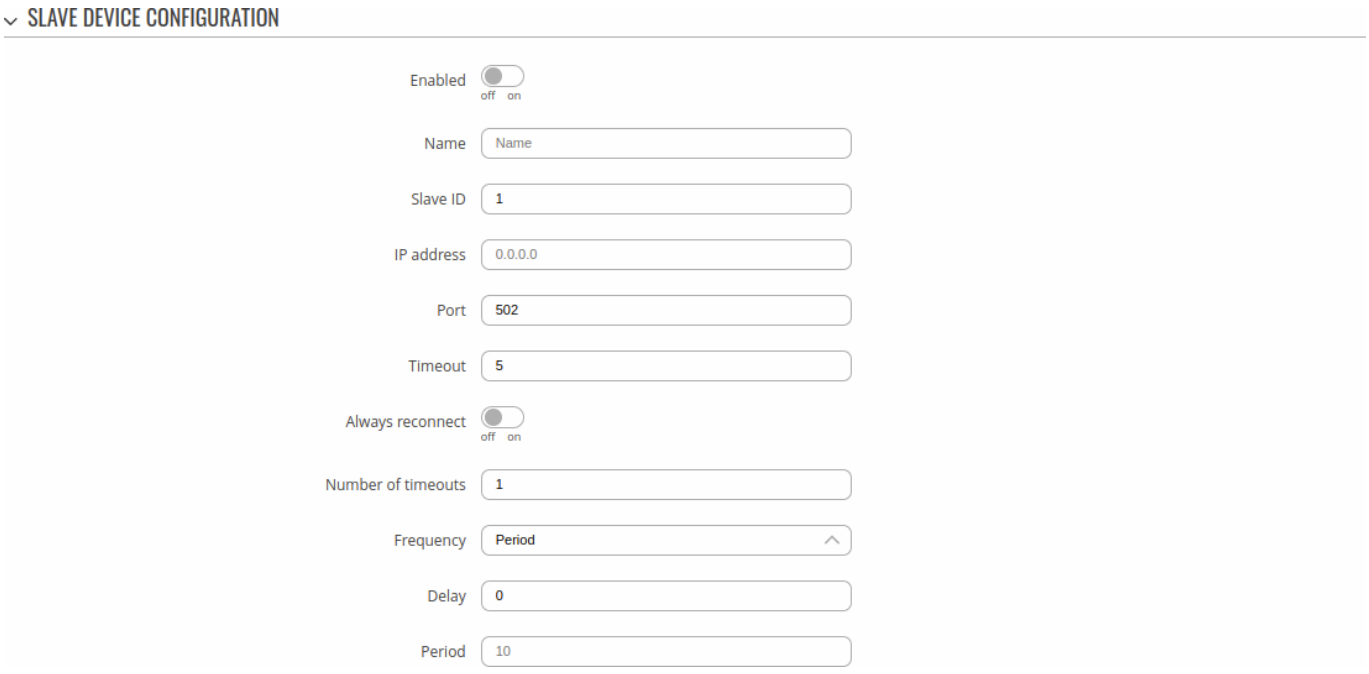
A **Modbus Master** device can request data from Modbus slaves. The Modbus TCP Master section is used to configure Modbus TCP slaves. By default, the slave list is empty. To add a new slave, click the 'Add' button



After clicking 'Add' you will be redirected to the newly added slave's configuration page.

Slave Device Configuration

The **Slave Device Configuration** section is used to configure the parameters of Modbus TCP slaves that the Master (this RUTX10 device) will be querying with requests. The figure below is an example of the Slave Device Configuration and the table below provides information on the fields contained in that section:



Field	Value	Description
Enabled	off | on; default: **off**	Turns communication with the slave device on or off.
Name	string; default: **none**	Slave device's name, used for easier management purposes.

Slave ID	integer [0..255]; default: **none**	Slave ID. Each slave in a network is assigned a unique identifier ranging from 1 to 255. When the master requests data from a slave, the first byte it sends is the Slave ID. When set to 0, the slave will respond to requests addressed to any ID.
IP address	ip4; default: **none**	Slave device's IP address.
Port	integer [0..65535]; default: **none**	Slave device's Modbus TCP port.
Period	integer [1..86400]; default: **60**	Interval at which requests are sent to the slave device.
Timeout	integer [1..30]; default: **5**	Maximum response wait time.
Always reconnect	off | on; default: **off**	Create new connection after every Modbus request.
Number of timeouts	integer [0..10]; default: **1**	Skip pending request and reset connection after number of request failures.
Frequency	Period | Schedule; default: **Period**	
Delay	integer [0..999]; default: **0**	Wait in milliseconds after connection initialization.
Period	integer [1..99999]; default: **none**	Interval in seconds for sending requests to this device

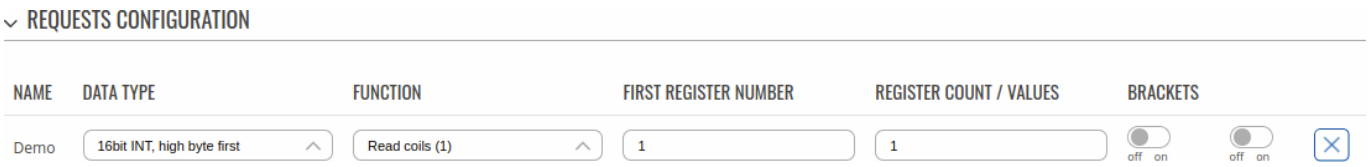
Requests Configuration

A Modbus **request** is a way of obtaining data from Modbus slaves. The master sends a request to a slave specifying the function code to be performed. The slave then sends the requested data back to the Modbus master.

Note: Modbus TCP Master uses *Register Number* instead of *Register Address* for pointing to a register. For example, to request the *Uptime* of a device, you must use **2** in the *First Register* field.

The Request Configuration list is empty by default. To add a new Request Configuration loon to the Add New Instance section. Enter a custom name into the 'Name' field and click the 'Add' button:

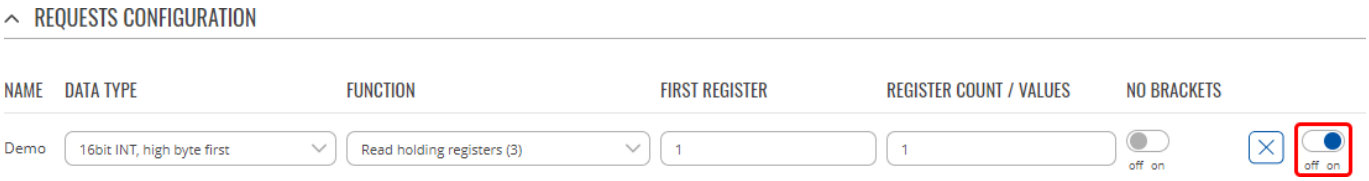
The new Request Configuration should become visible in the list:



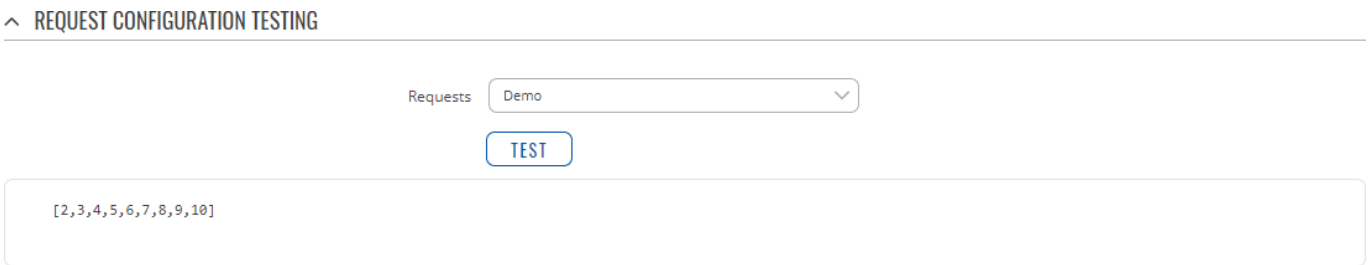
Field	Value	Description
Name	string; default: **Unnamed**	Name of this Request Configuration. Used for easier management purposes.

Data type	8bit INT | 8bit UINT | 16bit INT, high byte first | 16bit INT, low byte first | 16bit UINT, high byte first | 16bit UINT, low byte first | 32bit float (various Byte order) | 32bit INT (various Byte order) | 32bit UINT (various Byte order) | ASCII | Hex | Bool; default: **16bit INT, high byte first**	Defines how read data will be stored.
Function	Read coils (1) | Read input coils (2) | Read holding registers (3) | Read input registers (4) | Set single coil (5) | Set single coil register (6) | Set multiple coils (15) | Set multiple holding registers (16); default: **Read holding registers (3)**	Specifies the type of register being addressed by a Modbus request.
First Register	integer [0..65535]; default: **1**	First Modbus register from which data will be read.
Register Count / Values	integer [1..2000]; default: **1**	Number of Modbus registers that will be read during the request.
Remove Brackets	off | on; default: **off**	Removes the starting and ending brackets from the request (only for read requests).
off/on slider	off | on; default: **off**	Turns the request on or off.
Delete [X]	- (interactive button)	Deletes the request.

Additional note: by default the newly added Request Configurations are turned off. You can use the on/off slider to the right of the Request Configuration to turn it on:



After having configured a request, you should see a new 'Request Configuration Testing' section appear. It is used to check whether the configuration works correctly. Simply click the 'Test' button and a response should appear in the box below. A successful response to a test may look something like this:

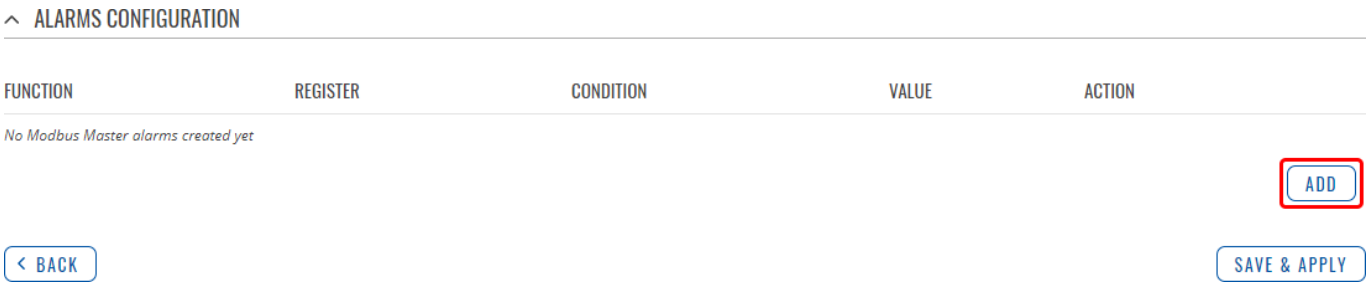


Alarms Configuration

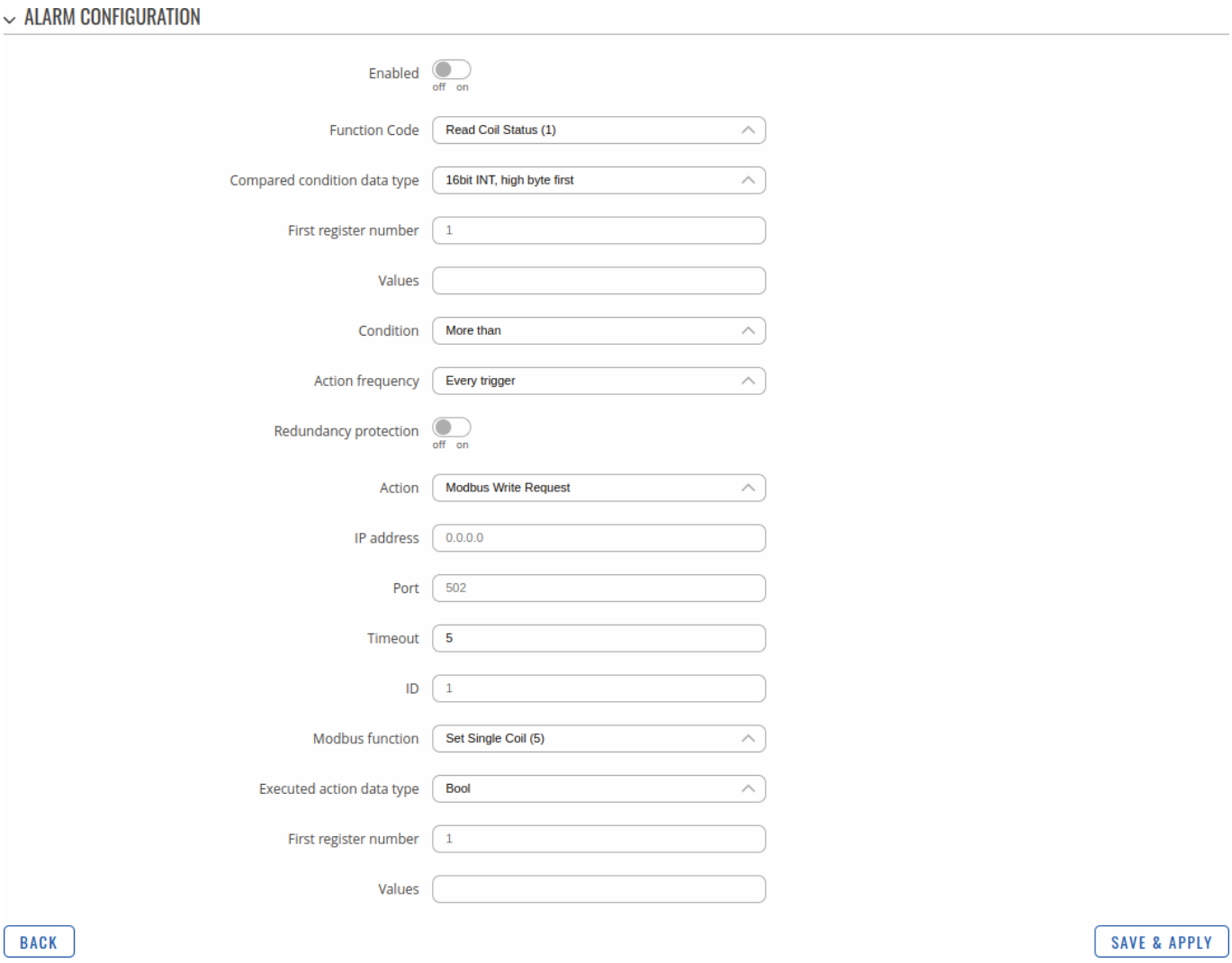
Alarms are a way of setting up automated actions when some Modbus values meet user-defined conditions. When the Modbus TCP Master (this RUTX10 device) requests some information from a

slave device it compares that data to with the parameters set in an Alarm Configuration. If the comparison meets the specified condition (more than, less than, equal to, not equal to), the Master performs a user-specified action, for example, a Modbus write request or switching the state of an output.

The figure below is an example of the Alarms Configuration list. To create a new Alarm, click the 'Add' button.



After adding the Alarm you should be redirected to its configuration page which should look similar to this:



Field	Value	Description
Enabled	off | on; default: **off**	Turns the alarm on or off.

Function code	Read Coil Status (1) | Read Input Status (2) | Read Holding Registers (3) | Read Input Registers (4); default: **Read Coil Status (1)**	Modbus function used for this alarm's Modbus request. The Modbus TCP Master (this RUTX10 device) perform this request as often as specified in the 'Period' field in Slave Device Configuration.
Compared condition data type	8bit INT | 8bit UINT | 16bit INT, high byte first | 16bit INT, low byte first | 16bit UINT, high byte first | 16bit UINT, low byte first | 32bit float (various Byte order) | 32bit INT (various Byte order) | 32bit UINT (various Byte order) | ASCII | Hex | Bool; default: **16bit INT, high byte first**	Select data type that will be used for checking conditions.
First register number	integer [1..65536]; default: **none**	Number of the Modbus coil/input/holding-register/input-register to read from.
Values	various; default: **none**	The value against which the read data will be compared.
Condition	More than | Less than | Equal to | Not Equal to | Less or equal | More or equal; default: **More than**	When a value is obtained it will be compared against the value specified in the following field. The comparison will be made in accordance with the condition specified in this field.
Action frequency	Every trigger | First trigger; default: **Every trigger**	Describes how frequently the specified action will be taken.
Redundancy protection	off | on; default: **off**	Protection against executing a configured action too often.
Redundancy protection period	integer [1..86400]; default: **none**	Duration to activate redundancy protection for, measured in seconds. This field becomes visible only when 'Redundancy protection' is turned on.
Action	MODBUS Write Request| Trigger output | MQTT message; default: **MODBUS Write Request**	Action that will be taken if the condition is met. Possible actions: • **Modbus Request** - sends a Modbus Write request to a specified slave. • **Trigger output** - changes state of selected I/O output pin.
MODBUS Write Request: IP address	ip | host; default: **none**	Modbus slave's IP address.
MODBUS Write Request: Port	integer [0..65535]; default: **none**	Modbus slave's port.
MODBUS Write Request: Timeout	integer [1..30]; default: **5**	Maximum time to wait for a response.
MODBUS Write Request: ID	integer [1..255]; default: **none**	Modbus slave ID.

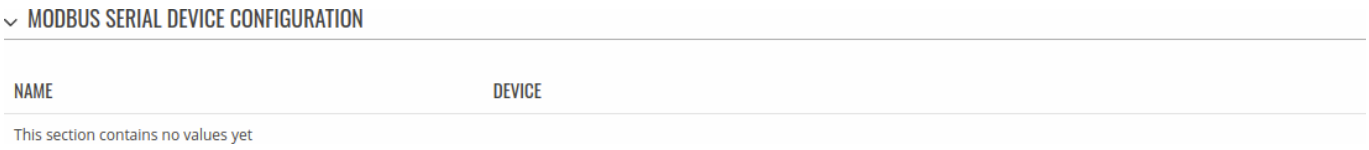
MODBUS Write Request: Modbus function	Set Single Coil (5) | Set Single Register (6) | Set Multiple Coils (15) | Set Multiple Registers (16); default: **Set Single Coil (5)**	A function code specifies the type of register being addressed by a Modbus request.
MODBUS Write Request: Executed action data type	8bit INT | 8bit UINT | 16bit INT, high byte first | 16bit INT, low byte first | 16bit UINT, high byte first | 16bit UINT, low byte first | 32bit float (various Byte order) | 32bit INT (various Byte order) | 32bit UNIT (various Byte order) | ASCII | Hex | Bool; default: **Bool**	Select data type that will be used for executing action.
MODBUS Write Request: First register number	integer [0..65535]; default: **none**	Begins reading from the register specified in this field.
MODBUS Write Request: Values	integer [0..65535]; default: **none**	Register/Coil values to be written (multiple values must be separated by space character).
Trigger output: Output	4PIN output; default: **4PIN output**	Selects which output will be triggered.
Trigger output: I/O Action	Turn On | Turn Off | Invert; default: **Turn On**	Selects the action performed on the output.
MQTT message: JSON format	string; default: **none**	Below this field you can find special codes that begin with the '%' sign. Each code represents a piece information related to the status of the device. Include these codes in the field for dynamic information reports.
MQTT message: Hostname	host | ip; default: **none**	Broker’s IP address or hostname.
MQTT message: Port	integer [0..65535]; default: **1883**	Broker's port number.
MQTT message: Keepalive	positive integer; default: **none**	The number of seconds after which the broker should send a PING message to the client if no other messages have been exchanged in that time
MQTT message: Topic	string; default: **none**	The name of the topic that the broker will subscribe to.
MQTT message: Client ID	positive integer; default: **none**	Client ID to send with the data. If empty, a random client ID will be generated
MQTT message: QoS	At most once (0) | At least once (1) | Exactly once (2); default: **At most once (0)**	A period of time (in seconds) which has to pass after a trigger event before this Action is executed.

Modbus Serial Master

The **Modbus Serial Master** page is used to configure the device as a Modbus RTU Master. Modbus RTU (remote terminal unit) is a serial communication protocol mainly used in communication via serial interfaces.

Modbus Serial Device Configuration

This section is used to create Modbus Serial Master's slave device instances. You may create a Serial Device instance for each supported serial interface.

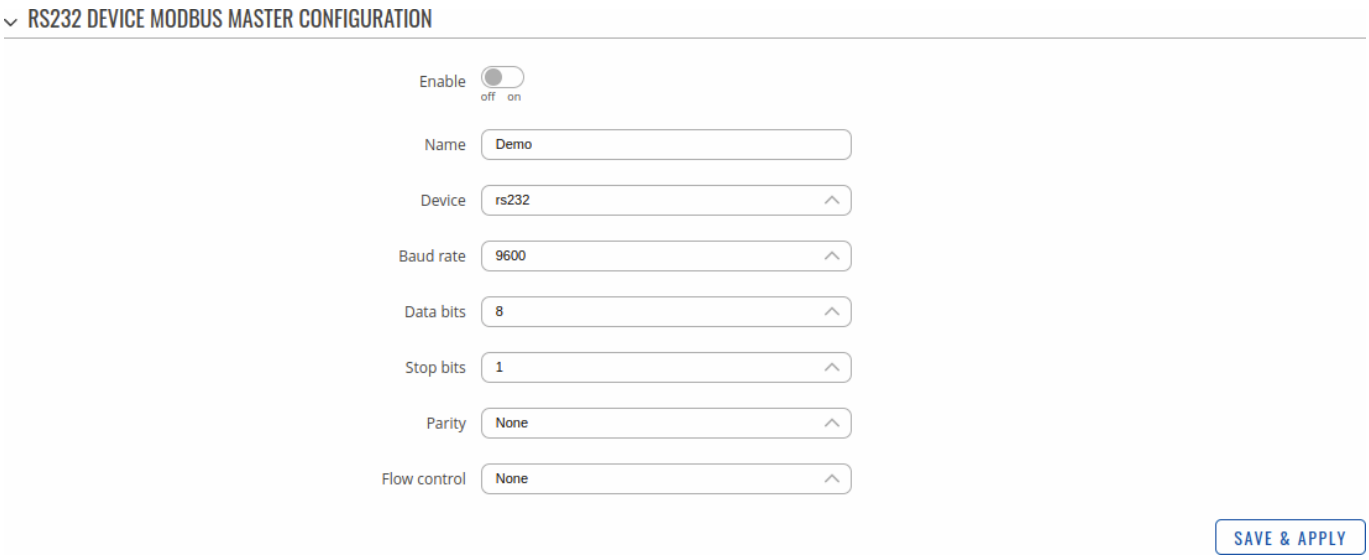


By default there are no instances created. To add a new serial device configuration, enter an instance name and click the 'Add' button.

After clicking 'Add' you will be redirected to the newly added device's configuration page.

RS Device Modbus Master Configuration

This section is used to configure the Modbus Serial Master's slave device interface settings.

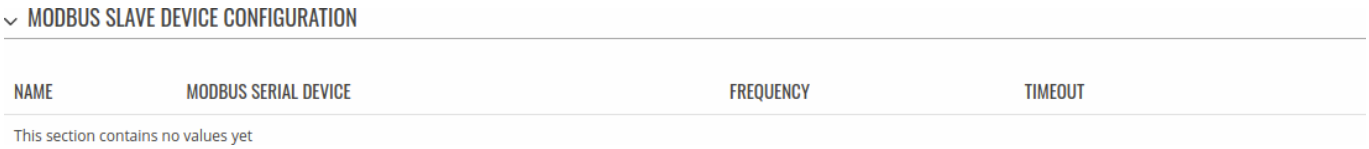


Field	Value	Description
Enable	off | on; default: **off**	Enables this Modbus Serial Device instance configuration.

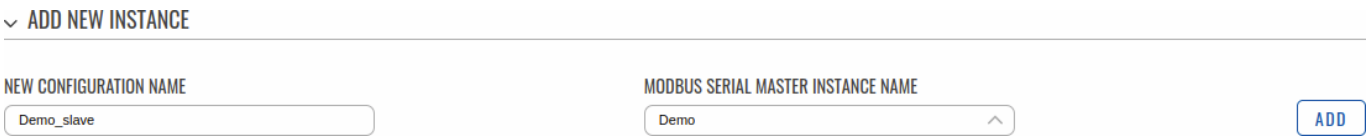
Name	string; default: **none**	Name of the serial device instance. Used for management purposes only.
Device	USB RS232 interface; default: **USB RS232 interface**	Specifies which serial port will be used for serial communication.
Baud rate	300 | 1200 | 2400 | 4800 | 9600 | 19200 | 38400 | 57600 | 115200; default: **9600**	Serial data transmission rate (in bits per second).
Data bits	5 | 6 | 7 |8; default: **8**	Number of data bits for each character.
Stop bits	1| 2; default: **1**	Stop bits sent at the end of every character allow the receiving signal hardware to detect the end of a character and to resynchronise with the character stream. Electronic devices usually use one stop bit. Two stop bits are required if slow electromechanical devices are used.
Parity	Even | Odd| Mark | Space | None; default: **None**	In serial transmission, parity is a method of detecting errors. An extra data bit is sent with each data character, arranged so that the number of 1 bits in each character, including the parity bit, is always odd or always even. If a byte is received with the wrong number of 1s, then it must have been corrupted. However, an even number of errors can pass the parity check. • **None** (**N**) - no parity method is used. • **Odd** (**O**) - the parity bit is set so that the number of "logical ones (1s)" has to be odd. • **Even** (**E**) - the parity bit is set so that the number of "logical ones (1s)" has to be even. • **Space** (**s**) - the parity bit will always be a binary 0. • **Mark** (**M**) - the parity bit will always be a binary 1.
Flow control	None | RTS/CTS | Xon/Xoff; default: **None**	In many circumstances a transmitter might be able to send data faster than the receiver is able to process it. To cope with this, serial lines often incorporate a "handshaking" method, usually distinguished between hardware and software handshaking. • **RTS/CTS** - hardware handshaking. RTS and CTS are turned OFF and ON from alternate ends to control data flow, for instance when a buffer is almost full. • **Xon/Xoff** - software handshaking. The Xon and Xoff characters are sent by the receiver to the sender to control when the sender will send data, i.e., these characters go in the opposite direction to the data being sent. The circuit starts in the "sending allowed" state. When the receiver's buffers approach capacity, the receiver sends the Xoff character to tell the sender to stop sending data. Later, after the receiver has emptied its buffers, it sends an Xon character to tell the sender to resume transmission.

Modbus Slave Device Configuration

This section is used to create slave instances that the Master (this RUTX10 device) will be querying with requests.



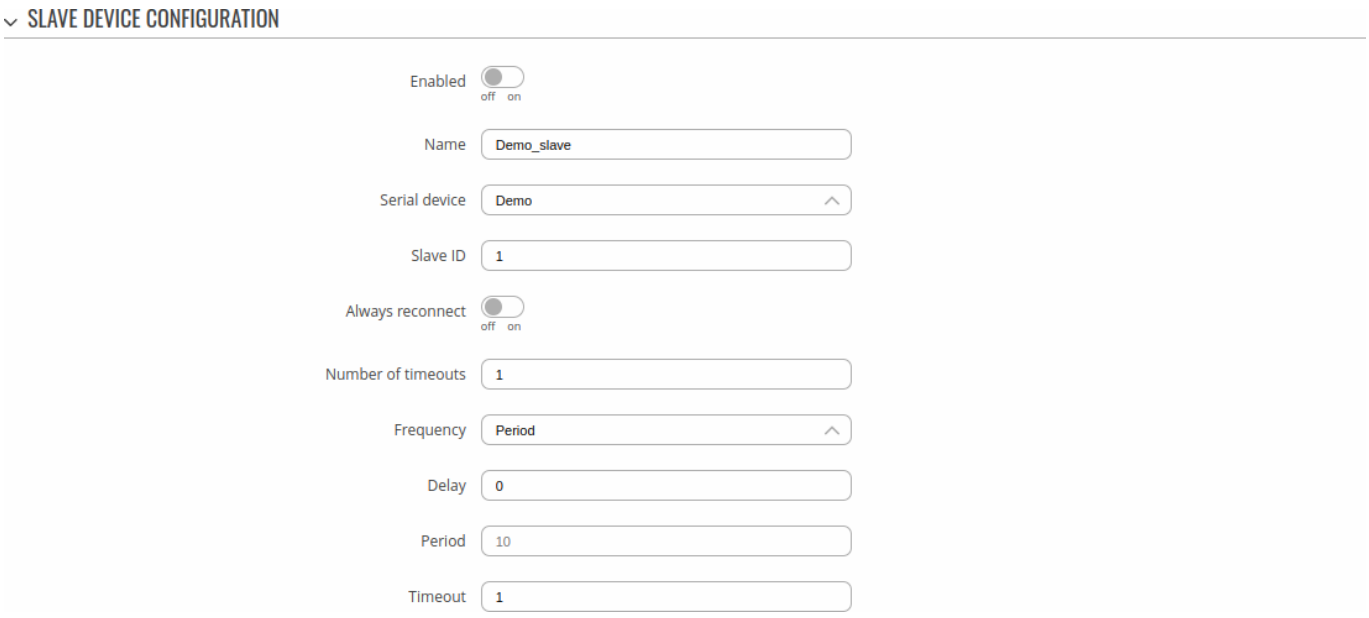
By default there are no instances created. To add a new slave configuration, enter an instance name, select a serial device instance and click the 'Add' button.



After clicking 'Add' you will be redirected to the newly added slave's configuration page.

Slave Device Configuration

The **Slave Device Configuration** section is used to configure the parameters of Modbus RTU slaves that the Master (this RUTX10 device) will be querying with requests. The figure below is an example of the Slave Device Configuration and the table below provides information on the fields contained in that section:



Field	Value	Description
Enabled	off | on; default: **off**	Turns communication with the slave device on or off.
Name	string; default: **none**	Slave device's name, used for easier management purposes.
Slave device	*slave device instance*; default: **none**	Specifies which serial device will be used on this slave.
Slave ID	integer [0..255]; default: **none**	Slave ID. Each slave in a network is assigned a unique identifier ranging from 1 to 255. When the master requests data from a slave, the first byte it sends is the Slave ID. When set to 0, the slave will respond to requests addressed to any ID.

Always reconnect	off | on; default: **off**	Create new connection after every Modbus request.
Number of timeouts	integer [0..10]; default: **1**	Skip pending request and reset connection after number of request failures.
Frequency	Period | Schedule; default: **Period**	
Delay	integer [0..999]; default: **0**	Wait in milliseconds after connection initialization.
Period	integer [1..99999]; default: **none**	Interval at which requests are sent to the slave device.
Timeout	integer [1..60]; default: **1**	Maximum response wait time.

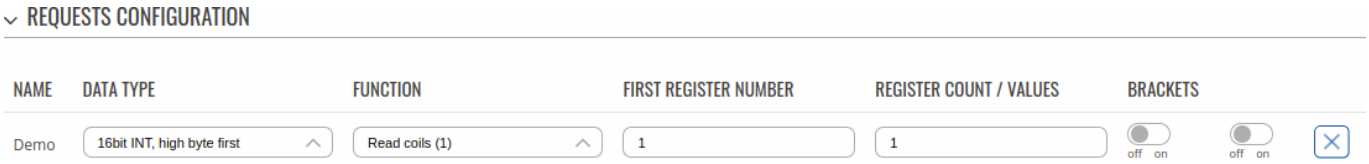
Requests Configuration

A Modbus **request** is a way of obtaining data from Modbus slaves. The master sends a request to a slave specifying the function code to be performed. The slave then sends the requested data back to the Modbus master.

Note: Modbus Serial Master uses *Register Number* instead of *Register Address* for pointing to a register. For example, to request the *Uptime* of a device, you must use **2** in the *First Register* field.

The Request Configuration list is empty by default. To add a new Request Configuration loon to the Add New Instance section. Enter a custom name into the 'Name' field and click the 'Add' button:

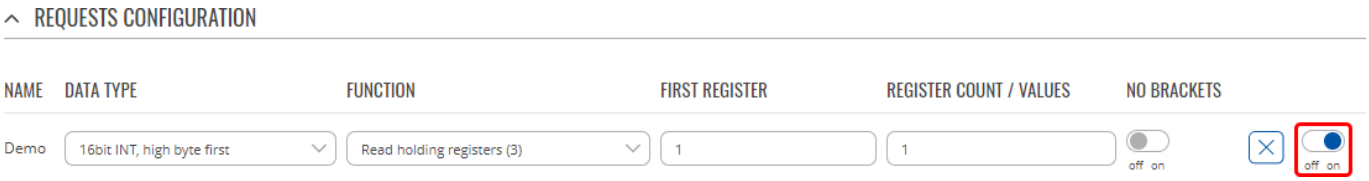
The new Request Configuration should become visible in the list:



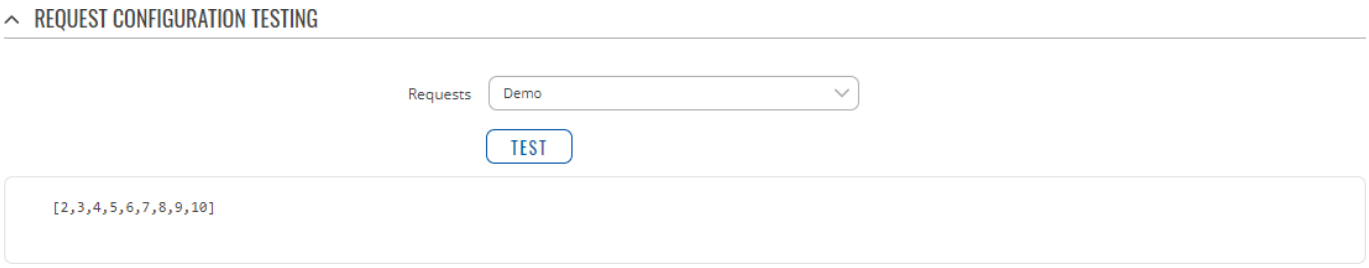
Field	Value	Description
Name	string; default: **Unnamed**	Name of this Request Configuration. Used for easier management purposes.
Data type	8bit INT | 8bit UINT | 16bit INT, high byte first | 16bit INT, low byte first | 16bit UINT, high byte first | 16bit UINT, low byte first | 32bit float (various Byte order) | 32bit INT (various Byte order) | 32bit UINT (various Byte order) | ASCII | Hex | Bool; default: **16bit INT, high byte first**	Defines how read data will be stored.

Function	Read coils (1) | Read input coils (2) | Read holding registers (3) | Read input registers (4) | Set single coil (5) | Set single coil register (6) | Set multiple coils (15) | Set multiple holding registers (16); default: **Read holding registers (3)**	Specifies the type of register being addressed by a Modbus request.
First Register	integer [0..65535]; default: **1**	First Modbus register from which data will be read.
Register Count / Values	integer [1..2000]; default: **1**	Number of Modbus registers that will be read during the request.
Remove Brackets	off | on; default: **off**	Removes the starting and ending brackets from the request (only for read requests).
off/on slider	off | on; default: **off**	Turns the request on or off.
Delete [X]	- (interactive button)	Deletes the request.

Additional note: by default the newly added Request Configurations are turned off. You can use the on/off slider to the right of the Request Configuration to turn it on:



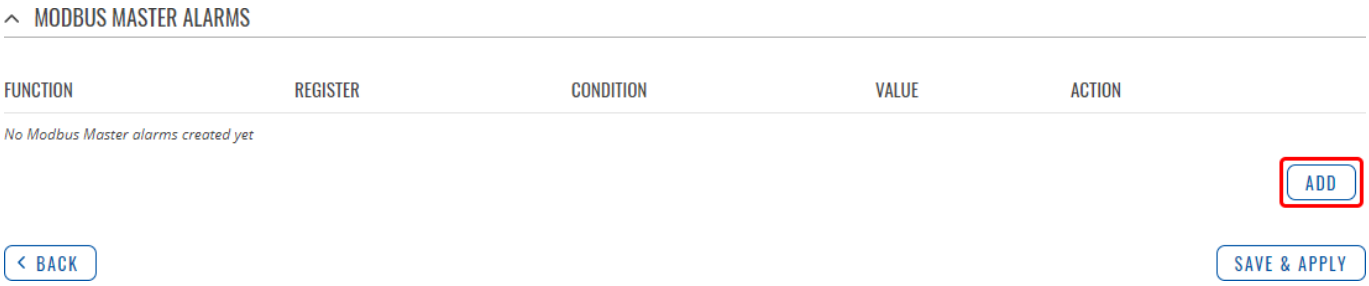
After having configured a request, you should see a new 'Request Configuration Testing' section appear. It is used to check whether the configuration works correctly. Simply click the 'Test' button and a response should appear in the box below. A successful response to a test may look something like this:



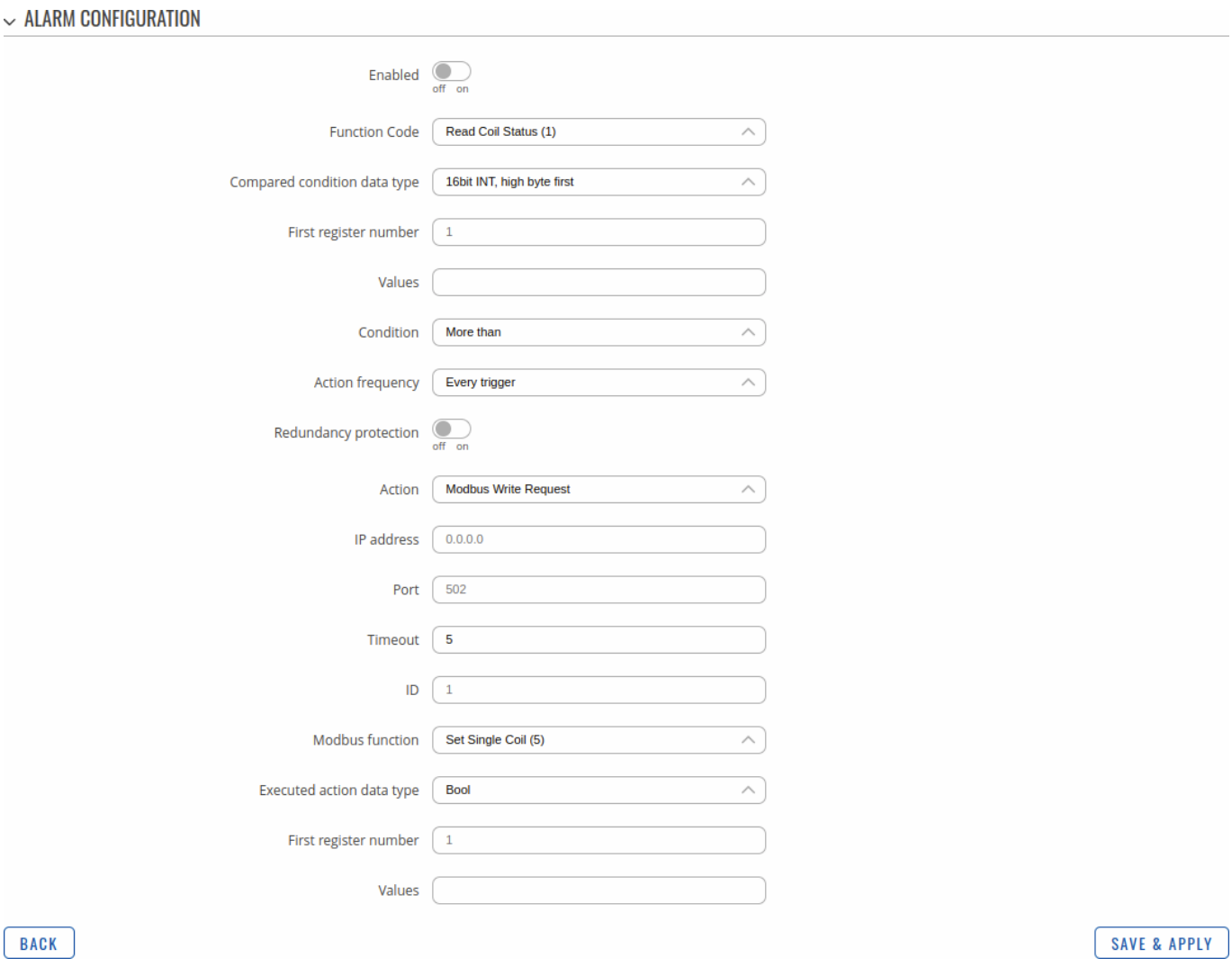
Modbus Master Alarms

Alarms are a way of setting up automated actions when some Modbus values meet user-defined conditions. When the Modbus Serial Master (this RUTX10 device) requests some information from a slave device it compares that data to with the parameters set in an Alarm Configuration. If the comparison meets the specified condition (more than, less than, equal to, not equal to), the Master performs a user-specified action, for example, a Modbus write request or switching the state of an output.

The figure below is an example of the Modbus Master Alarms list. To create a new Alarm, click the 'Add' button.



After this you should be redirected to that Alarm's configuration page which should look similar to this:



Field	Value	Description
Enabled	off | on; default: **off**	Turns the alarm on or off.
Function code	Read Coil Status (1) | Read Input Status (2) | Read Holding Registers (3) | Read Input Registers (4); default: **Read Coil Status (1)**	Modbus function used for this alarm's Modbus request. The Modbus TCP Master (this RUTX10 device) perform this request as often as specified in the 'Period' field in Slave Device Configuration.

Compared condition data type	8bit INT | 8bit UINT | 16bit INT, high byte first | 16bit INT, low byte first | 16bit UINT, high byte first | 16bit UINT, low byte first | 32bit float (various Byte order) | 32bit INT (various Byte order) | 32bit UINT (various Byte order) | ASCII | Hex | Bool; default: **16bit INT, high byte first**	Select data type that will be used for checking conditions.
First register number	integer [1..65536]; default: **none**	Number of the Modbus coil/input/holding-register/input-register to read from.
Values	various; default: **none**	The value against which the read data will be compared.
Condition	More than | Less than | Equal to | Not Equal to | Less or equal | More or equal; default: **More than**	When a value is obtained it will be compared against the value specified in the following field. The comparison will be made in accordance with the condition specified in this field.
Action frequency	Every trigger | First trigger; default: **Every trigger**	Describes how frequently the specified action will be taken.
Redundancy protection	off | on; default: **off**	Protection against executing a configured action too often.
Redundancy protection period	integer [1..86400]; default: **none**	Duration to activate redundancy protection for, measured in seconds. This field becomes visible only when 'Redundancy protection' is turned on.
Action	MODBUS Write Request| Trigger output; default: **MODBUS Write Request**	Action that will be taken if the condition is met. Possible actions: • **Modbus Request** - sends a Modbus Write request to a specified slave. • **Trigger output** - changes state of selected I/O output pin.
MODBUS Write Request: Timeout	integer [1..30]; default: **5**	Maximum time to wait for a response.
MODBUS Write Request: ID	integer [1..255]; default: **none**	Modbus slave ID.
MODBUS Write Request: Modbus function	Read Single Coil (5) | Set Single Register (6) | Set Multiple Coils (15) | Set Multiple Registers (16); default: **Set Single Coil (5)**	A function code specifies the type of register being addressed by a Modbus request.

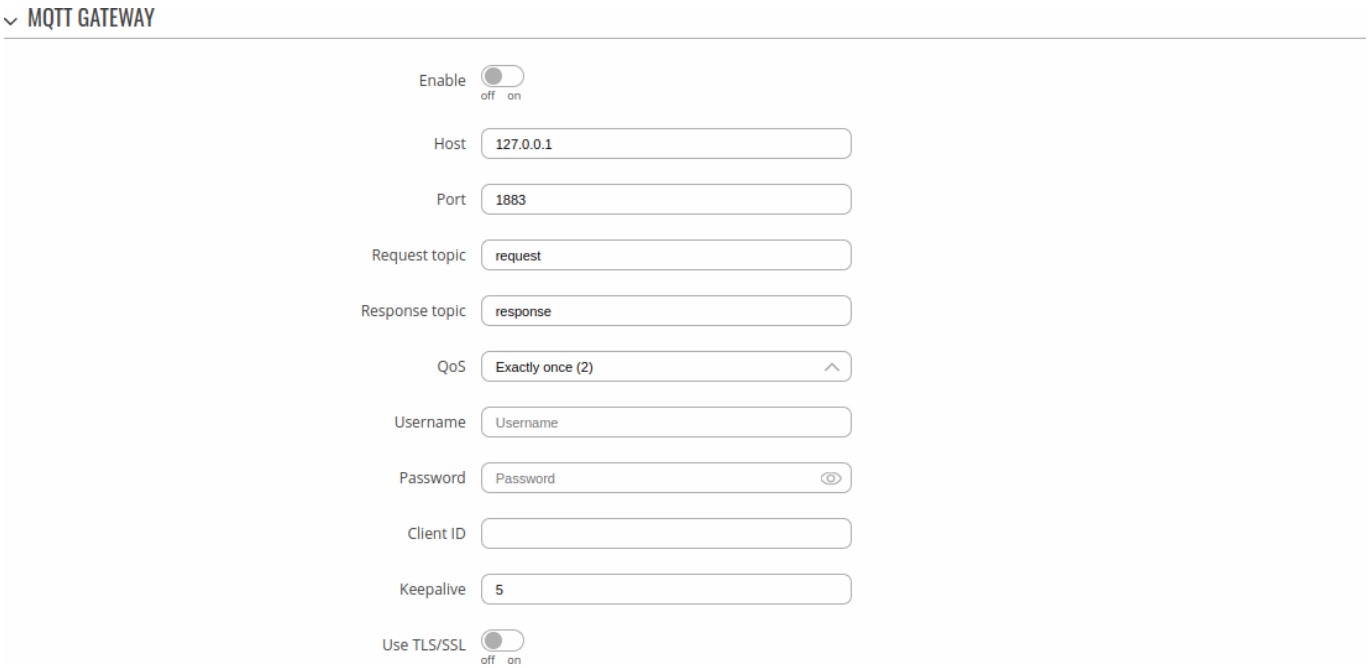
MODBUS Write Request: Executed action data type	8bit INT | 8bit UINT | 16bit INT, high byte first | 16bit INT, low byte first | 16bit UINT, high byte first | 16bit UINT, low byte first | 32bit float (various Byte order) | 32bit INT (various Byte order) | 32bit UNIT (various Byte order) | ASCII | Hex | Bool; default: **Bool**	Select data type that will be used for executing action.
MODBUS Write Request: First register number	integer [0..65535]; default: **none**	Begins reading from the register specified in this field.
MODBUS Write Request: Values	integer [0..65535]; default: **none**	Register/Coil values to be written (multiple values must be separated by space character).
Trigger output: Output	4PIN output; default: **4PIN output**	Selects which output will be triggered.
Trigger output: I/O Action	Turn On | Turn Off | Invert; default: **Turn On**	Selects the action performed on the output.

MQTT Gateway

The **MQTT Gateway** function is used to transfer Modbus data (send requests, receive responses) over MQTT. When it is enabled, the device (this RUTX10) subscribes to a REQUEST topic and publishes on a RESPONSE topic on a specified MQTT broker. It translates received MQTT message payload to a Modbus request and relays it to the specified Modbus TCP slave.

When the MQTT Gateway receives a response from the slave, it translates it to an MQTT message and publishes it on the RESPONSE topic.

Below is an example of the MQTT Gateway page. Refer to the table for information on MQTT Gateway configuration fields.



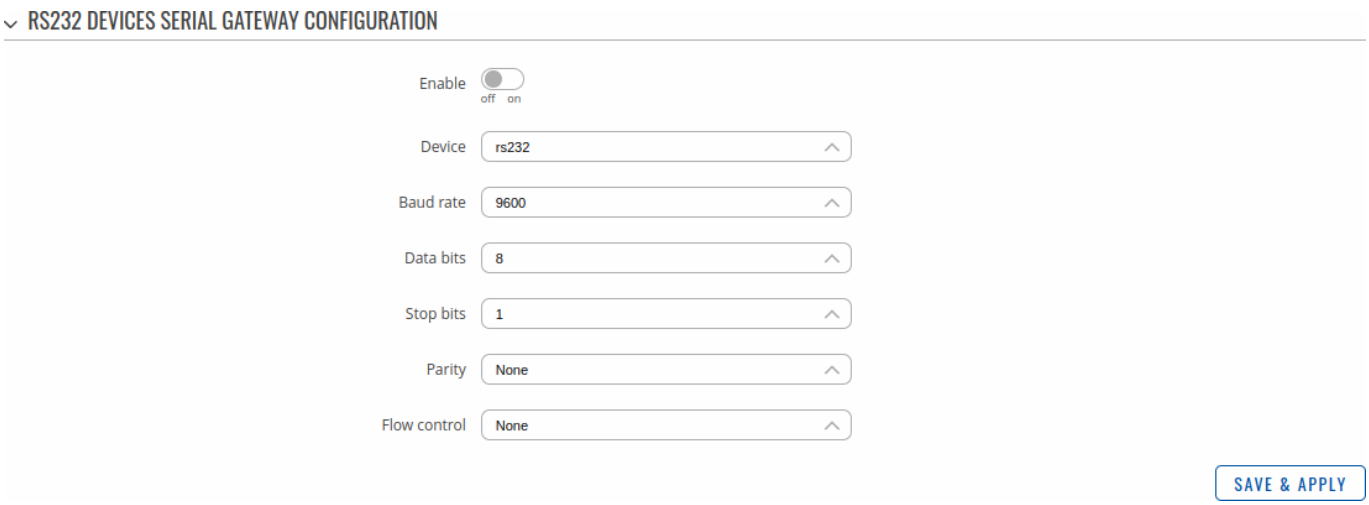
Field	Value	Description
Enable	off | on; default: **off**	Turns MQTT gateway on or off.
Host	ip | host; default: **127.0.0.1**	IP address or hostname of an MQTT broker.
Port	integer [0..65535]; default: **1883**	Port number of the MQTT broker.
Request topic	alphanumeric string; default: **request**	MQTT topic for sending requests.
Response topic	alphanumeric string; default: **response**	MQTT topic for subscribing to responses.
QoS	At most once (0) | At least once (1) | Exactly once (2); default: **Exactly once (2)**	Specifies quality of service.
Username	string; default: **none**	Username for authentication to the MQTT broker.
Password	string; default: **none**	Password for authentication to the MQTT broker.
Client ID	integer; default: **none**	Specifies client ID for MQTT broker.
Keepalive	integer; default: **5**	Keepalive message to MQTT broker (seconds)
Use TLS/SSL	off | on; default: **off**	Turns TLS support on or off
TLS type	cert | psk; default: **cert**	Selects the type of TLS encryption
TLS insecure	off | on; default: **off**	Disables TLS security
Certificate files from device	off | on; default: **off**	Choose this option if you want to use certificate files generated on device.
CA file	string; default: **none**	Upload/select certificate authority file.
Certificates file	string; default: **none**	Upload/select certificate file.
Key file	string; default: **none**	Upload/select certificate key file.
PSK	string; default: **none**	Specifies the pre-shared key.
Identity	string; default: **none**	Specifies identity.

Serial Gateway Configuration

Serial Gateway Configuration section displays Serial gateway instances currently existing on the router.

By default the list is empty. To create a new gateway instance, enter the ID of serial device, select serial interface and click the 'Add' button.

After this you should be redirected to instance's configuration page which should look similar to this:



Field	Value	Description
Enable	off | on; default: **off**	Enables this Serial Gateway instance configuration.
Name	string; default: **none**	Name of the gateway instance. Used for management purposes only.
Device	USB RS232 interface; default: **USB RS232 interface**	Specifies which serial port will be used for serial communication.
Baud rate	300 | 1200 | 2400 | 4800 | 9600 | 19200 | 38400 | 57600 | 115200; default: **9600**	Serial data transmission rate (in bits per second).
Data bits	5 | 6 | 7 |8; default: **8**	Number of data bits for each character.
Stop bits	1| 2; default: **1**	Stop bits sent at the end of every character allow the receiving signal hardware to detect the end of a character and to resynchronise with the character stream. Electronic devices usually use one stop bit. Two stop bits are required if slow electromechanical devices are used.

Parity	Even | Odd| Mark | Space | None; default: **None**	In serial transmission, parity is a method of detecting errors. An extra data bit is sent with each data character, arranged so that the number of 1 bits in each character, including the parity bit, is always odd or always even. If a byte is received with the wrong number of 1s, then it must have been corrupted. However, an even number of errors can pass the parity check. • **None** (**N**) - no parity method is used. • **Odd** (**O**) - the parity bit is set so that the number of "logical ones (1s)" has to be odd. • **Even** (**E**) - the parity bit is set so that the number of "logical ones (1s)" has to be even. • **Space** (**s**) - the parity bit will always be a binary 0. • **Mark** (**M**) - the parity bit will always be a binary 1.
Flow control	None | RTS/CTS | Xon/Xoff; default: **None**	In many circumstances a transmitter might be able to send data faster than the receiver is able to process it. To cope with this, serial lines often incorporate a "handshaking" method, usually distinguished between hardware and software handshaking. • **RTS/CTS** - hardware handshaking. RTS and CTS are turned OFF and ON from alternate ends to control data flow, for instance when a buffer is almost full. • **Xon/Xoff** - software handshaking. The Xon and Xoff characters are sent by the receiver to the sender to control when the sender will send data, i.e., these characters go in the opposite direction to the data being sent. The circuit starts in the "sending allowed" state. When the receiver's buffers approach capacity, the receiver sends the Xoff character to tell the sender to stop sending data. Later, after the receiver has emptied its buffers, it sends an Xon character to tell the sender to resume transmission.

Request messages

Note: MQTT Gateway uses *Register Number* instead of *Register Address* for pointing to a register. For example, to request the *Uptime* of a device, you must use **2** in the *Register Number* field.

Modbus request data sent in the MQTT payload should be generated in accordance with the following format:

`0 <COOKIE> <IP_TYPE> <IP> <PORT> <TIMEOUT> <SLAVE_ID> <MODBUS_FUNCTION> <REGISTER_NUMBER> <REGISTER_COUNT/VALUE>`

Explanation:

- **0** - must be 0, which signifies a textual format (currently the only one implemented).
- **Cookie** - a 64-bit unsigned integer in range [0..2^{64}]). A cookie is used in order to distinguish which response belongs to which request, each request and the corresponding response contain a matching cookie: a 64-bit unsigned integer.
- **IP type** - host IP address type. Possible values:
  - **0** - IPv4 address;
  - **1** - IPv6 address;
  - **2** - hostname that will be resolved to an IP address.
- **IP** - IP address of a Modbus TCP slave. IPv6 must be presented in full form (e.g., *2001:0db8:0000:0000:0000:8a2e:0370:7334*).

- **Port** - port number of the Modbus TCP slave.
- **Timeout** - timeout for Modbus TCP connection, in seconds. Range [1..999].
- **Slave ID** - Modbus TCP slave ID. Range [1..255].
- **Modbus function** - Modbus task type that will be executed. Possible values are:
  - **3** - read holding registers;
  - **6** - write to a single holding register;
  - **16** - write to multiple holding registers.
- **Register number** - number (not address) of the first register (in range [1..65536]) from which the registers will be read/written to.
- **Register count/value** - this value depends on the Modbus function:
  - **3** - register count (in range [1..125]); must not exceed the boundary (first register number + register count <= 65537);
  - **6** - register value (in range [0..65535]);
  - **16** - register count (in range [1..123]); must not exceed the boundary (first register number + register count <= 65537); and register values separated with commas, without spaces (e.g., *1,2,3,654,21,789*); there must be exactly as many values as specified (with register count); each value must be in the range of [0..65535].

Response messages

A special response message can take one of the following forms:

```
<COOKIE> OK                                  - for functions 6 and 16
<COOKIE> OK <VALUE> <VALUE> <VALUE>...   - for function 3, where <VALUE>
<VALUE> <VALUE>... are read register values
<COOKIE> ERROR: ...                          - for failures, where ... is the
error description
```

Examples

Below are a few **examples** of controlling/monitoring the internal Modbus TCP Slave on RUTX10.

Reboot the device

- Request:

```
0 65432 0 192.168.1.1 502 5 1 6 206 1
```

- Response:

```
65432 OK
```

Retrieve uptime

- Request:

  ```
  0 65432 0 192.168.1.1 502 5 1 3 2 2
  ```

- Response:

  ```
  65432 OK 0 5590
  ```

If you're using Eclipse Mosquitto (MQTT implementation used on RUTX10), Publish/Subscribe commands may look something like this:

Retrieve uptime

- Request:

  ```
  mosquitto_pub -h 192.168.1.1 -p 1883 -t request -m "0 65432 0
  192.168.1.1 502 5 1 3 2 2"
  ```

- Response:

  ```
  mosquitto_sub -h 192.168.1.1 -p 1883 -t response
  65432 OK 0 5590
  ```

Modbus TCP over Serial Gateway

The **Modbus TCP over Serial gateway** serial type allows redirecting TCP data coming to a specified port to an RTU specified by the Slave ID. The Slave ID can be specified by the user or be obtained directly from the Modbus header.

Modbus TCP over Serial Gateway Configuration

Modbus TCP over Serial Gateway Configuration section displays gateway instances currently existing on the router.

By default the list is empty. To create a new gateway instance, enter the name of instance, select serial interface and click the 'Add' button.

After this you should be redirected to instance's configuration page which should look similar to this:

RS232 DEVICE MODBUS TCP OVER SERIAL CONFIGURATION

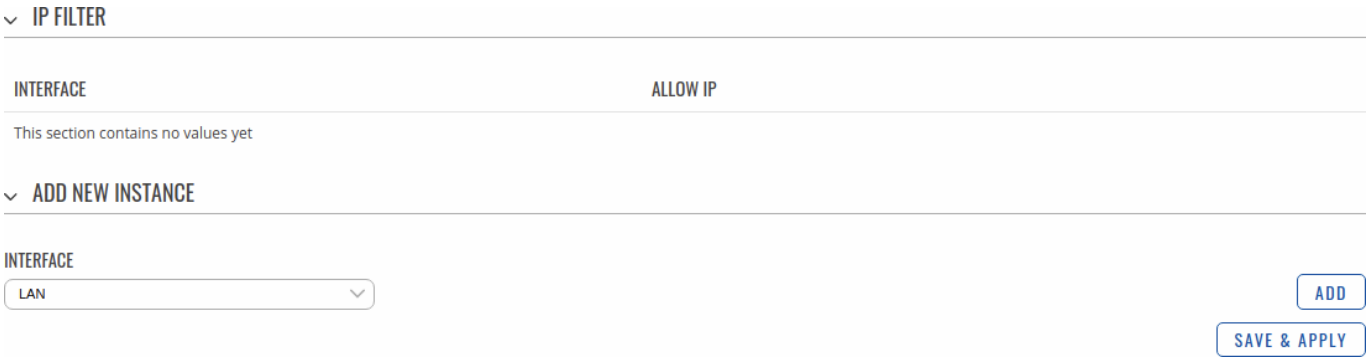
Field	Value	Description
Enable	off | on; default: **off**	Enables this Modbus TCP over Serial Gateway instance configuration.
Name	string; default: **none**	Name of the gateway instance. Used for management purposes only.
Device	USB RS232 interface; default: **USB RS232 interface**	Specifies which serial port will be used for serial communication.
Baud rate	300 | 1200 | 2400 | 4800 | 9600 | 19200 | 38400 | 57600 | 115200; default: **9600**	Serial data transmission rate (in bits per second).
Data bits	5 | 6 | 7 |8; default: **8**	Number of data bits for each character.
Stop bits	1| 2; default: **1**	Stop bits sent at the end of every character allow the receiving signal hardware to detect the end of a character and to resynchronise with the character stream. Electronic devices usually use one stop bit. Two stop bits are required if slow electromechanical devices are used.

Parity	Even | Odd| Mark | Space | None; default: **None**	In serial transmission, parity is a method of detecting errors. An extra data bit is sent with each data character, arranged so that the number of 1 bits in each character, including the parity bit, is always odd or always even. If a byte is received with the wrong number of 1s, then it must have been corrupted. However, an even number of errors can pass the parity check. • **None** (**N**) - no parity method is used. • **Odd** (**O**) - the parity bit is set so that the number of "logical ones (1s)" has to be odd. • **Even** (**E**) - the parity bit is set so that the number of "logical ones (1s)" has to be even. • **Space** (**s**) - the parity bit will always be a binary 0. • **Mark** (**M**) - the parity bit will always be a binary 1.
Flow control	None| RTS/CTS | Xon/Xoff; default: **None**	In many circumstances a transmitter might be able to send data faster than the receiver is able to process it. To cope with this, serial lines often incorporate a "handshaking" method, usually distinguished between hardware and software handshaking. • **RTS/CTS** - hardware handshaking. RTS and CTS are turned OFF and ON from alternate ends to control data flow, for instance when a buffer is almost full. • **Xon/Xoff** - software handshaking. The Xon and Xoff characters are sent by the receiver to the sender to control when the sender will send data, i.e., these characters go in the opposite direction to the data being sent. The circuit starts in the "sending allowed" state. When the receiver's buffers approach capacity, the receiver sends the Xoff character to tell the sender to stop sending data. Later, after the receiver has emptied its buffers, it sends an Xon character to tell the sender to resume transmission.
Listening IP	ip; default: **none**	IP address to listen for incoming connections. (*0.0.0.0*) value may be used to listen for incoming connections on any interface or IP address.
Port	integer [0..65535]; default: **none**	Port number to listen for incoming connections.
Slave ID configuration type	User defined | Obtained from TCP; default: **User defined**	Specifies whether slave IDs are user defined or automatically obtained from TCP.
Slave ID	integer; default: **none**	Specifies the slave ID of range of permitted slave IDs. The way this field is named and its function depends on the value of the *Slave ID configuration* field. A range of IDs can be specified by placing a hyphen (-) between two integer numbers. For example, if you permit slave IDs in the range of 10 to 20, you would specify it as: *10-20* You can also specify multiple values that are not connected in a range using commas (,). For example, to specify 6, 50 and 100 as permitted slave IDs, you would have to use: *6,50,100*
Permitted slave IDs	range of integers; default: **1-247**	Read *Slave ID* field description.
CRC verification	off | on; default: **off**	Checks if sent serial message is not disturbed.

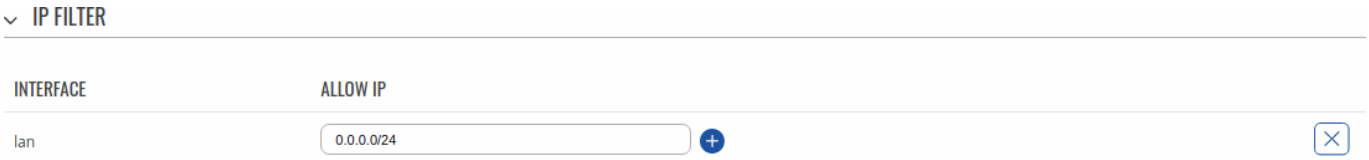
Echo	off | on; default: **off**	Turns RS232 echo on or off. RS232 echo is a loopback test usually used to check whether the RS232 cable is working properly.

IP Filter

The **IP Filter** section is used for configuring which network is allowed to communicate with the device. You may add a new instance by selecting the Interface and pressing Add.



Then enter the IP address and save.



See also

- **Monitoring via Modbus** - detailed examples on how to use Modbus TCP