

# RUT955 Modbus

[Main Page](#) > [RUT Routers](#) > [RUT955](#) > [RUT955 Manual](#) > [RUT955 WebUI](#) > [RUT955 Services section](#) > **RUT955 Modbus**

The information in this page is updated in accordance with firmware version [RUT9\\_R\\_00.07.04.5](#).

**Note:** [click here](#) for the old style WebUI (FW version RUT9XX\_R\_00.06.09.5 and earlier) user manual page.

□

## Contents

- [1 Summary](#)
- [2 Modbus TCP Slave](#)
- [3 Modbus Serial Slave](#)
  - [3.1 Modbus Serial Slave Configuration](#)
- [4 Modbus Registers](#)
  - [4.1 Get Parameters](#)
  - [4.2 Set Parameters](#)
- [5 Modbus TCP Master](#)
  - [5.1 Slave Device Configuration](#)
  - [5.2 Requests Configuration](#)
  - [5.3 Alarms Configuration](#)
- [6 Modbus Serial Master](#)
  - [6.1 Modbus Serial Device Configuration](#)
    - [6.1.1 RS Device Modbus Master Configuration](#)
  - [6.2 Modbus Slave Device Configuration](#)
    - [6.2.1 Slave Device Configuration](#)
      - [6.2.1.1 Requests Configuration](#)
      - [6.2.1.2 Modbus Master Alarms](#)
- [7 MQTT Gateway](#)
  - [7.1 Serial Gateway Configuration](#)
  - [7.2 Request messages](#)
  - [7.3 Response messages](#)
  - [7.4 Examples](#)
- [8 Modbus TCP over Serial Gateway](#)
  - [8.1 Modbus TCP over Serial Gateway Configuration](#)
  - [8.2 IP Filter](#)
- [9 See also](#)

## 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 RUT955 devices.

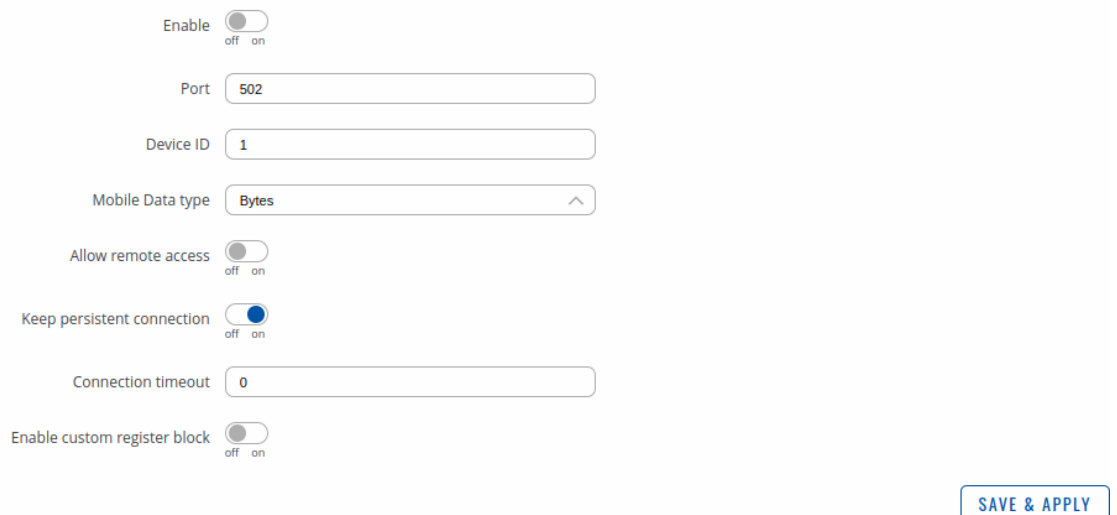
If you're having trouble finding this page or some of the parameters described here on your device's WebUI, you should **turn on "Advanced WebUI" mode**. You can do that by clicking the "Basic" button under "Mode", which is located at the top-right corner of the WebUI.

## 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:

### MODBUS TCP SLAVE



Field	Value	Description
Enable	off   on; default: <b>off</b>	Turns Modbus TCP on or off.
Port	integer [0..65535]; default: <b>502</b>	TCP port used for Modbus communications.
Device ID	integer [0..255]; default: <b>1</b>	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: <b>Bytes</b>	Selects mobile data unit representation type.
Allow remote access	off   on; default: <b>off</b>	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: <b>onn</b>	Allows keep the connection open after responding a Modbus TCP master request.
Connection timeout	integer [0..60]; default: <b>0</b>	Sets TCP timeout in seconds after which the connection is forcefully closed.
Enable custom register block	off   <b>on</b> ; default: <b>off</b>	Allows the usage of custom register block.

Register file path	path; default: <b>/tmp/regfile</b>	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: <b>1025</b>	First register in custom register block
Register count	integer [1..64512]; default: <b>128</b>	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.

#### ADD NEW INSTANCE

NEW CONFIGURATION NAME	DEVICE NAME	
<input type="text" value="Demo"/>	<input type="text" value="rs232"/>	<input type="button" value="ADD"/>

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

#### RS232 DEVICE MODBUS SLAVE CONFIGURATION

Enable

Name

Device

Device ID

Baud rate

Data bits

Stop bits

Parity

Flow control

Enable custom register block

Field	Value	Description
Enable	off   on; default: <b>off</b>	Enables this Modbus Serial Slave instance configuration.

Name	string; default: <b>none</b>	Name of the serial slave instance. Used for management purposes only.
Device	RS232   RS485   USB RS232 interface; default: <b>RS232</b>	Specifies which serial port will be used for serial communication.
Device ID	integer [0..255]; default: <b>1</b>	Specifies which serial port will be used for serial communication.
Baud rate	300   1200   2400   4800   9600   19200   38400   57600   115200   230400   460800   921600   1000000   3000000; default: <b>9600</b>	Serial data transmission rate (in bits per second).
Data bits	5   6   7   8; default: <b>8</b>	Number of data bits for each character.
Stop bits	1   2; default: <b>1</b>	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: <b>None</b>	<p>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.</p> <ul style="list-style-type: none"> <li>• <b>None (N)</b> - no parity method is used.</li> <li>• <b>Odd (O)</b> - the parity bit is set so that the number of "logical ones (1s)" has to be odd.</li> <li>• <b>Even (E)</b> - the parity bit is set so that the number of "logical ones (1s)" has to be even.</li> <li>• <b>Space (s)</b> - the parity bit will always be a binary 0.</li> <li>• <b>Mark (M)</b> - the parity bit will always be a binary 1.</li> </ul> <p>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.</p> <ul style="list-style-type: none"> <li>• <b>RTS/CTS</b> - 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.</li> <li>• <b>Xon/Xoff</b> - 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.</li> </ul>
Flow control	None   RTS/CTS   Xon/Xoff; default: <b>None</b>	
<b>RS485: Full Duplex</b>	off   on; default: <b>off</b>	Enables RS485 full duplex.

Enable custom register block	off   <b>on</b> ; default: <b>off</b>	Allows the usage of custom register block.
Register file path	path; default: <b>/tmp/regfile</b>	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: <b>1025</b>	First register in custom register block
Register count	integer [1..64512]; default: <b>128</b>	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
Mobile signal strength (RSSI in dBm)		3	4	2	32 bit integer
System temperature (in 0.1 °C)		5	6	2	32 bit integer
System hostname		7	8	16	Text
GSM operator name		23	24	16	Text
Router serial number		39	40	16	Text
LAN MAC address		55	56	16	Text
Router name		71	72	16	Text
Currently active SIM card slot		87	88	16	Text
Network registration info		103	104	16	Text
Network type		119	120	16	Text
Digital input (DIN1) state		135	136	2	32 bit integer
Digital galvanically isolated input (DIN2) state		137	138	2	32 bit integer
Current WAN IP address		139	140	2	8 bit unsigned integer
Analog input (PIN 9) value		141	142	2	32 bit unsigned integer
GPS latitude coordinate		143	144	2	32 bit float
GPS longitude coordinate		145	146	2	32 bit float
GPS fix time		147	148	16	Text (Unix timestamp×1000)
GPS date and time		163	164	16	Text (DDMMYYhhmmss)
GPS speed		179	180	2	32 bit float
GPS satellite count		181	182	2	32 bit unsigned integer
GPS accuracy		183	184	2	32 bit float
Mobile data received today (SIM1)		185	186	2	32 bit unsigned integer
Mobile data sent today (SIM1)		187	188	2	32 bit unsigned integer
Mobile data received this week (SIM1)		189	190	2	32 bit unsigned integer
Mobile data sent this week (SIM1)		191	192	2	32 bit unsigned integer
Mobile data received this month (SIM1)		193	194	2	32 bit unsigned integer
Mobile data sent this month (SIM1)		195	196	2	32 bit unsigned integer
Mobile data received last 24h (SIM1)		197	198	2	32 bit unsigned integer
Mobile data sent last 24h (SIM1)		199	200	2	32 bit unsigned integer
Galvanically isolated open collector output status		201	202	1	16 bit unsigned integer
Relay output status		202	203	1	16 bit unsigned integer
Active SIM card		205	206	1	16 bit unsigned integer
Mobile data received last week (SIM1)		292	293	2	32 bit unsigned integer

Mobile data sent last week (SIM1)	294	295	2	32 bit unsigned integer
Mobile data received last month (SIM1)	296	297	2	32 bit unsigned integer
Mobile data sent last month (SIM1)	298	299	2	32 bit unsigned integer
Mobile data received today (SIM2)	300	301	2	32 bit unsigned integer
Mobile data sent today (SIM2)	302	303	2	32 bit unsigned integer
Mobile data received this week (SIM2)	304	305	2	32 bit unsigned integer
Mobile data sent this week (SIM2)	306	307	2	32 bit unsigned integer
Mobile data received this month (SIM2)	308	309	2	32 bit unsigned integer
Mobile data sent this month (SIM2)	310	311	2	32 bit unsigned integer
Mobile data received last 24h (SIM2)	312	313	2	32 bit unsigned integer
Mobile data sent last 24h (SIM2)	314	315	2	32 bit unsigned integer
Mobile data received last week (SIM2)	316	317	2	32 bit unsigned integer
Mobile data sent last week (SIM2)	318	319	2	32 bit unsigned integer
Mobile data received last month(SIM2)	320	321	2	32 bit unsigned integer
Mobile data sent last month (SIM2)	322	323	2	32 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
Modem ID	328	329	8	Text
ACL activity	345	346	1	16 bit unsigned integer
ACL status	346	347	2	32 bit float
IMSI	348	349	16	Text
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
Add SMS	397	398	90	Text

## 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
Galvanically isolated open collector output status	201	202	1   0	Toggles DOUT1 ON or OFF
Relay output status	202	203	1   0	Toggles DOUT2 ON or OFF
Switch WiFi ON/OFF	203	204	1   0	Turns WiFi ON or OFF
Switch mobile data connection (ON/OFF*)	204	205	1   0	Turns mobile data connection ON or OFF
Switch SIM card	205	206	1   2   0	Changes the active SIM card slot <ul style="list-style-type: none"> <li>• 1 - switch to SIM1</li> <li>• 2 - switch to SIM2</li> <li>• 0 - switch from the the SIM card opposite of the one currently in use (SIM1 → SIM2 or SIM2 → SIM1)</li> </ul>
Reboot	206	207	1	Reboots the router
Change APN	207	208	APN code	Changes APN. The number of input registers may vary depending on the length of the APN, but the very first byte of the set APN command denotes the number of the SIM card for which to set the APN. This byte should be set to: <ul style="list-style-type: none"> <li>• 1 - to set APN for SIM1</li> <li>• 2 - to set APN for SIM2</li> </ul>
Switch PIN 4 state	325	326	1 0	Toggles PIN 4 ON or OFF
Switch ACL activity	345	346	1 0	Turns ACL activity ON or OFF
Switch 2.4GHz WiFi ON/OFF	390	391	1   0	Turns 2.4GHz WiFi ON or OFF
Change LAN IP	394	395	IPv4 (in decimal form)	Changes device LAN IP
Send SMS	396	397	1 0	Sends an SMS with content defined in Add SMS (397) register
Add SMS	397	398	Message (in decimal form)	Define SMS content which will be sent using Send SMS (396) register. The register array is split into two parts that represent the recipient's "phone number" (first 10 registers) and the "SMS message contents" (remaining 80 registers).

## 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

NAME	ID	IP ADDRESS	PERIOD	TIMEOUT
------	----	------------	--------	---------

No Modbus TCP slaves added yet

ADD

SAVE & APPLY

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 RUT955 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:

### SLAVE DEVICE CONFIGURATION

Enabled

Name

Slave ID

IP address

Port

Timeout

Always reconnect

Number of timeouts

Frequency

Delay

Period

Field	Value	Description
Enabled	off   on; default: <b>off</b>	Turns communication with the slave device on or off.
Name	string; default: <b>none</b>	Slave device's name, used for easier management purposes.
Slave ID	integer [0..255]; default: <b>none</b>	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: <b>none</b>	Slave device's IP address.
Port	integer [0..65535]; default: <b>none</b>	Slave device's Modbus TCP port.
Period	integer [1..86400]; default: <b>60</b>	Interval at which requests are sent to the slave device.

Timeout	integer [1..30]; default: <b>5</b>	Maximum response wait time.
Always reconnect	off   on; default: <b>off</b>	Create new connection after every Modbus request.
Number of timeouts	integer [0..10]; default: <b>1</b>	Skip pending request and reset connection after number of request failures.
Frequency	Period   Schedule; default: <b>Period</b>	
Delay	integer [0..999]; default: <b>0</b>	Wait in milliseconds after connection initialization.
Period	integer [1..99999]; default: <b>none</b>	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:

^ ADD NEW INSTANCE

NAME

Demo

ADD

The new Request Configuration should become visible in the list:

v REQUESTS CONFIGURATION

NAME	DATA TYPE	FUNCTION	FIRST REGISTER NUMBER	REGISTER COUNT / VALUES	BRACKETS
Demo	16bit INT, high byte first	Read coils (1)	1	1	off on off on

Field	Value	Description
Name	string; default: <b>Unnamed</b>	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: <b>16bit INT, high byte first</b>	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: <b>Read holding registers (3)</b>	Specifies the type of register being addressed by a Modbus request.
First Register	integer [0..65535]; default: <b>1</b>	First Modbus register from which data will be read.
Register Count / Values	integer [1..2000]; default: <b>1</b>	Number of Modbus registers that will be read during the request.
Remove Brackets	off   on; default: <b>off</b>	Removes the starting and ending brackets from the request (only for read requests).
off/on slider	off   on; default: <b>off</b>	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:

#### ^ REQUESTS CONFIGURATION

NAME	DATA TYPE	FUNCTION	FIRST REGISTER	REGISTER COUNT / VALUES	NO BRACKETS
Demo	16bit INT, high byte first	Read holding registers (3)	1	1	<input type="checkbox"/> off <input checked="" type="checkbox"/> 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:

#### ^ REQUEST CONFIGURATION TESTING

Requests

[2,3,4,5,6,7,8,9,10]

## 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 RUT955 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.

## ^ ALARMS CONFIGURATION

FUNCTION	REGISTER	CONDITION	VALUE	ACTION
----------	----------	-----------	-------	--------

No Modbus Master alarms created yet

ADD

< BACK

SAVE & APPLY

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

## ^ ALARM CONFIGURATION

Enabled

Function Code

Compared condition data type

First register number

Values

Condition

Action frequency

Redundancy protection

Action

IP address

Port

Timeout

ID

Modbus function

Executed action data type

First register number

Values

BACK

SAVE & APPLY

Field	Value	Description
Enabled	off   on; default: <b>off</b>	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: <b>Read Coil Status (1)</b>	Modbus function used for this alarm's Modbus request. The Modbus TCP Master (this RUT955 device) perform this request as often as specified in the 'Period' field in <a href="#">Slave Device Configuration</a> .

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: <b>16bit INT, high byte first</b>	Select data type that will be used for checking conditions.
First register number	integer [1..65536]; default: <b>none</b>	Number of the Modbus coil/input/holding-register/input-register to read from.
Values	various; default: <b>none</b>	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: <b>More than</b>	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: <b>Every trigger</b>	Describes how frequently the specified action will be taken.
Redundancy protection	off   <b>on</b> ; default: <b>off</b>	Protection against executing a configured action too often.
Redundancy protection period	integer [1..86400]; default: <b>none</b>	Duration to activate redundancy protection for, measured in seconds. This field becomes visible only when 'Redundancy protection' is turned on.
Action	SMS   MODBUS Write Request  Trigger output   MQTT message; default: <b>MODBUS Write Request</b>	Action that will be taken if the condition is met. Possible actions: <ul style="list-style-type: none"> <li>• <b>SMS</b> - sends and SMS message to a specified recipient(s).</li> <li>• <b>Modbus Request</b> - sends a Modbus Write request to a specified slave.</li> <li>• <b>Trigger output</b> - changes state of selected I/O output pin.</li> </ul>
SMS: Message	string; default: <b>none</b>	SMS message text.
SMS: Phone number	phone number; default: <b>none</b>	Recipient's phone number.
MODBUS Write Request: IP address	ip   host; default: <b>none</b>	Modbus slave's IP address.
MODBUS Write Request: Port	integer [0..65535]; default: <b>none</b>	Modbus slave's port.
MODBUS Write Request: Timeout	integer [1..30]; default: <b>5</b>	Maximum time to wait for a response.
MODBUS Write Request: ID	integer [1..255]; default: <b>none</b>	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: <b>Set Single Coil (5)</b>	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: <b>Bool</b>	Select data type that will be used for executing action.
MODBUS Write Request: First register number	integer [0..65535]; default: <b>none</b>	Begins reading from the register specified in this field.
MODBUS Write Request: Values	integer [0..65535]; default: <b>none</b>	Register/Coil values to be written (multiple values must be separated by space character).
Trigger output: Output	Open collector output   Relay output   4PIN output; default: <b>Open collector output</b>	Selects which output will be triggered.
Trigger output: I/O Action	Turn On   Turn Off   Invert; default: <b>Turn On</b>	Selects the action performed on the output.
MQTT message: JSON format	string; default: <b>none</b>	Below this field you can find special codes that begin with the '%' sign. Each code represents a piece of information related to the status of the device. Include these codes in the field for dynamic information reports.
MQTT message: Hostname	host   ip; default: <b>none</b>	Broker's IP address or hostname.
MQTT message: Port	integer [0..65535]; default: <b>1883</b>	Broker's port number.
MQTT message: Keepalive	positive integer; default: <b>none</b>	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: <b>none</b>	The name of the topic that the broker will subscribe to.
MQTT message: Client ID	positive integer; default: <b>none</b>	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: <b>At most once (0)</b>	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.

### MODBUS SERIAL DEVICE CONFIGURATION

NAME	DEVICE
This section contains no values yet	

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

### ADD NEW INSTANCE

NEW CONFIGURATION NAME	DEVICE NAME	
<input type="text" value="Demo"/>	<input type="text" value="rs232"/>	<input type="button" value="ADD"/>

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.

### RS232 DEVICE MODBUS MASTER CONFIGURATION

Enable

Name

Device

Baud rate

Data bits

Stop bits

Parity

Flow control

Field	Value	Description
Enable	off   on; default: <b>off</b>	Enables this Modbus Serial Device instance configuration.

Name	string; default: <b>none</b>	Name of the serial device instance. Used for management purposes only.
Device	RS232   RS485   USB RS232 interface; default: <b>RS232</b>	Specifies which serial port will be used for serial communication.
Baud rate	300   1200   2400   4800   9600   19200   38400   57600   115200   230400   460800   921600   1000000   3000000; default: <b>9600</b>	Serial data transmission rate (in bits per second).
Data bits	5   6   7   8; default: <b>8</b>	Number of data bits for each character.
Stop bits	1   2; default: <b>1</b>	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: <b>None</b>	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. <ul style="list-style-type: none"> <li>• <b>None (N)</b> - no parity method is used.</li> <li>• <b>Odd (O)</b> - the parity bit is set so that the number of "logical ones (1s)" has to be odd.</li> <li>• <b>Even (E)</b> - the parity bit is set so that the number of "logical ones (1s)" has to be even.</li> <li>• <b>Space (s)</b> - the parity bit will always be a binary 0.</li> <li>• <b>Mark (M)</b> - the parity bit will always be a binary 1.</li> </ul>
Flow control	None   RTS/CTS   Xon/Xoff; default: <b>None</b>	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. <ul style="list-style-type: none"> <li>• <b>RTS/CTS</b> - 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.</li> <li>• <b>Xon/Xoff</b> - 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.</li> </ul>
<b>RS485:</b>		
Full Duplex	off   on; default: <b>off</b>	Enables RS485 full duplex.

## Modbus Slave Device Configuration

---

This section is used to create slave instances that the Master (this RUT955 device) will be querying with requests.

#### MODBUS SLAVE DEVICE CONFIGURATION

NAME	MODBUS SERIAL DEVICE	FREQUENCY	TIMEOUT
This section contains no values yet			

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.

#### ADD NEW INSTANCE

NEW CONFIGURATION NAME:  MODBUS SERIAL MASTER INSTANCE NAME:

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 RUT955 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:

#### SLAVE DEVICE CONFIGURATION

Enabled

Name

Serial device

Slave ID

Always reconnect

Number of timeouts

Frequency

Delay

Period

Timeout

Field	Value	Description
Enabled	off   on; default: <b>off</b>	Turns communication with the slave device on or off.
Name	string; default: <b>none</b>	Slave device's name, used for easier management purposes.
Slave device	<i>slave device instance</i> ; default: <b>none</b>	Specifies which serial device will be used on this slave.

Slave ID	integer [0..255]; default: <b>none</b>	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: <b>off</b>	Create new connection after every Modbus request.
Number of timeouts	integer [0..10]; default: <b>1</b>	Skip pending request and reset connection after number of request failures.
Frequency	Period   Schedule; default: <b>Period</b>	
Delay	integer [0..999]; default: <b>0</b>	Wait in milliseconds after connection initialization.
Period	integer [1..99999]; default: <b>none</b>	Interval at which requests are sent to the slave device.
Timeout	integer [1..60]; default: <b>1</b>	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:

^ ADD NEW INSTANCE

NAME



The new Request Configuration should become visible in the list:

v REQUESTS CONFIGURATION

NAME	DATA TYPE	FUNCTION	FIRST REGISTER NUMBER	REGISTER COUNT / VALUES	BRACKETS
Demo	16bit INT, high byte first	Read coils (1)	1	1	off on off on

Field	Value	Description
Name	string; default: <b>Unnamed</b>	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: <b>16bit INT, high byte first</b>	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: <b>Read holding registers (3)</b>	Specifies the type of register being addressed by a Modbus request.
First Register	integer [0..65535]; default: <b>1</b>	First Modbus register from which data will be read.
Register Count / Values	integer [1..2000]; default: <b>1</b>	Number of Modbus registers that will be read during the request.
Remove Brackets	off   on; default: <b>off</b>	Removes the starting and ending brackets from the request (only for read requests).
off/on slider	off   on; default: <b>off</b>	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:

^ REQUESTS CONFIGURATION

NAME	DATA TYPE	FUNCTION	FIRST REGISTER	REGISTER COUNT / VALUES	NO BRACKETS
Demo	<input type="text" value="16bit INT, high byte first"/>	<input type="text" value="Read holding registers (3)"/>	<input type="text" value="1"/>	<input type="text" value="1"/>	<input type="checkbox"/> off <input checked="" type="checkbox"/> on <span style="float: right; border: 1px solid red; padding: 2px;"> <input checked="" type="checkbox"/> off <input type="checkbox"/> on             </span>

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:

^ REQUEST CONFIGURATION TESTING

Requests

[2,3,4,5,6,7,8,9,10]

**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 RUT955 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.

^ MODBUS MASTER ALARMS

FUNCTION	REGISTER	CONDITION	VALUE	ACTION
<i>No Modbus Master alarms created yet</i>				

**ADD**

< BACK SAVE & APPLY

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

^ ALARM CONFIGURATION

Enabled

Function Code

Compared condition data type

First register number

Values

Condition

Action frequency

Redundancy protection

Action

IP address

Port

Timeout

ID

Modbus function

Executed action data type

First register number

Values

BACK SAVE & APPLY

Field	Value	Description
Enabled	off   on; default: <b>off</b>	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: <b>Read Coil Status (1)</b>	Modbus function used for this alarm's Modbus request. The Modbus TCP Master (this RUT955 device) perform this request as often as specified in the 'Period' field in <a href="#">Slave Device Configuration</a> .
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: <b>16bit INT, high byte first</b>	Select data type that will be used for checking conditions.
First register number	integer [1..65536]; default: <b>none</b>	Number of the Modbus coil/input/holding-register/input-register to read from.
Values	various; default: <b>none</b>	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: <b>More than</b>	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: <b>Every trigger</b>	Describes how frequently the specified action will be taken.
Redundancy protection	off   on; default: <b>off</b>	Protection against executing a configured action too often.
<a href="#">Redundancy protection period</a>	integer [1..86400]; default: <b>none</b>	Duration to activate redundancy protection for, measured in seconds. This field becomes visible only when 'Redundancy protection' is turned on.
Action	SMS   MODBUS Write Request  Trigger output; default: <b>MODBUS Write Request</b>	Action that will be taken if the condition is met. Possible actions: <ul style="list-style-type: none"> <li>• <b>SMS</b> - sends and SMS message to a specified recipient(s).</li> <li>• <b>Modbus Request</b> - sends a Modbus Write request to a specified slave.</li> <li>• <b>Trigger output</b> - changes state of selected I/O output pin.</li> </ul>
<a href="#">SMS: Message</a>	string; default: <b>none</b>	SMS message text.
<a href="#">SMS: Phone number</a>	phone number; default: <b>none</b>	Recipient's phone number.
<a href="#">MODBUS Write Request: Timeout</a>	integer [1..30]; default: <b>5</b>	Maximum time to wait for a response.
<a href="#">MODBUS Write Request: ID</a>	integer [1..255]; default: <b>none</b>	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: <b>Set Single Coil (5)</b>	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: <b>Bool</b>	Select data type that will be used for executing action.
MODBUS Write Request: First register number	integer [0..65535]; default: <b>none</b>	Begins reading from the register specified in this field.
MODBUS Write Request: Values	integer [0..65535]; default: <b>none</b>	Register/Coil values to be written (multiple values must be separated by space character).
Trigger output: Output	Open collector output   Relay output   4PIN output; default: <b>Open collector output</b>	Selects which output will be triggered.
Trigger output: I/O Action	Turn On   Turn Off   Invert; default: <b>Turn On</b>	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 RUT955) 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.

Enable  off on

Host

Port

Request topic

Response topic

QoS

Username

Password

Client ID

Keepalive

Use TLS/SSL  off on

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

## ADD NEW INSTANCE

DEVICE ID	DEVICE NAME	ADD
<input type="text" value="Demo"/>	<input type="text" value="rs232"/>	<input type="button" value="ADD"/>

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

## RS232 DEVICES SERIAL GATEWAY CONFIGURATION

Enable

Device

Baud rate

Data bits

Stop bits

Parity

Flow control

SAVE & APPLY

Field	Value	Description
Enable	off   on; default: <b>off</b>	Enables this Serial Gateway instance configuration.
Name	string; default: <b>none</b>	Name of the gateway instance. Used for management purposes only.
Device	RS232   RS485   USB RS232 interface; default: <b>RS232</b>	Specifies which serial port will be used for serial communication.
Baud rate	300   1200   2400   4800   9600   19200   38400   57600   115200   230400   460800   921600   1000000   3000000; default: <b>9600</b>	Serial data transmission rate (in bits per second).
Data bits	5   6   7   8; default: <b>8</b>	Number of data bits for each character.
Stop bits	1   2; default: <b>1</b>	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: <b>None</b>	<p>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.</p> <ul style="list-style-type: none"> <li>• <b>None (N)</b> - no parity method is used.</li> <li>• <b>Odd (O)</b> - the parity bit is set so that the number of "logical ones (1s)" has to be odd.</li> <li>• <b>Even (E)</b> - the parity bit is set so that the number of "logical ones (1s)" has to be even.</li> <li>• <b>Space (s)</b> - the parity bit will always be a binary 0.</li> <li>• <b>Mark (M)</b> - the parity bit will always be a binary 1.</li> </ul>
Flow control	None   RTS/CTS   Xon/Xoff; default: <b>None</b>	<p>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.</p> <ul style="list-style-type: none"> <li>• <b>RTS/CTS</b> - 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.</li> <li>• <b>Xon/Xoff</b> - 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.</li> </ul>
RS485: Full Duplex	off   on; default: <b>off</b>	Enables RS485 full duplex.

## 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<sup>64</sup>]. 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 RUT955.

---

### 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 RUT955), 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.

∨ ADD NEW INSTANCE

---

NEW CONFIGURATION NAME

Demo

DEVICE NAME

rs232

ADD

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

Enable  off  on  
 Name   
 Device   
 Baud rate   
 Data bits   
 Stop bits   
 Parity   
 Flow control   
 Listening ip   
 Port   
 Slave ID configuration type   
 Slave ID   
 CRC verification  off  on  
 Echo  off  on

Field	Value	Description
Enable	off   on; default: <b>off</b>	Enables this Modbus TCP over Serial Gateway instance configuration.
Name	string; default: <b>none</b>	Name of the gateway instance. Used for management purposes only.
Device	RS232   RS485   USB RS232 interface; default: <b>RS232</b>	Specifies which serial port will be used for serial communication.
Baud rate	300   1200   2400   4800   9600   19200   38400   57600   115200   230400   460800   921600   1000000   3000000; default: <b>9600</b>	Serial data transmission rate (in bits per second).
Data bits	5   6   7   8; default: <b>8</b>	Number of data bits for each character.
Stop bits	1   2; default: <b>1</b>	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: <b>None</b>	<p>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.</p> <ul style="list-style-type: none"> <li>• <b>None (N)</b> - no parity method is used.</li> <li>• <b>Odd (O)</b> - the parity bit is set so that the number of "logical ones (1s)" has to be odd.</li> <li>• <b>Even (E)</b> - the parity bit is set so that the number of "logical ones (1s)" has to be even.</li> <li>• <b>Space (s)</b> - the parity bit will always be a binary 0.</li> <li>• <b>Mark (M)</b> - the parity bit will always be a binary 1.</li> </ul>
Flow control	None   RTS/CTS   Xon/Xoff; default: <b>None</b>	<p>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.</p> <ul style="list-style-type: none"> <li>• <b>RTS/CTS</b> - 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.</li> <li>• <b>Xon/Xoff</b> - 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.</li> </ul>
Listening IP	ip; default: <b>none</b>	<p>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.</p>
Port	integer [0..65535]; default: <b>none</b>	<p>Port number to listen for incoming connections.</p>
Slave ID configuration type	<p>User defined   Obtained from TCP; default: <b>User defined</b></p>	<p>Specifies whether slave IDs are user defined or automatically obtained from TCP.</p>
Slave ID	<p>integer; default: <b>none</b></p>	<p>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 <i>Slave ID configuration</i> field.</p> <p>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: <i>10-20</i></p> <p>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: <i>6,50,100</i></p>
Permitted slave IDs	<p>range of integers; default: <b>1-247</b></p>	<p>Read <i>Slave ID</i> field description.</p>
CRC verification	<p>off   on; default: <b>off</b></p>	<p>Checks if sent serial message is not disturbed.</p>

Echo	off   on; default: <b>off</b>	Turns RS232 echo on or off. RS232 echo is a loopback test usually used to check whether the RS232 cable is working properly.
<a href="#">RS485: Full Duplex</a>	off   on; default: <b>off</b>	Enables RS485 full duplex.

## 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.

IP FILTER

---

INTERFACE	ALLOW IP
This section contains no values yet	

ADD NEW INSTANCE

---

INTERFACE	ALLOW IP
<input type="text" value="LAN"/>	<input type="text"/>

Then enter the IP address and save.

IP FILTER

---

INTERFACE	ALLOW IP
lan	<input type="text" value="0.0.0.0/24"/> <input type="button" value="+"/>

## See also

- [Monitoring via Modbus](#) - detailed examples on how to use Modbus TCP