RUT955 Modbus (legacy WebUI)

The information in this page is updated in accordance with firmware version **RUT9XX_R_00.06.09.5**.

Note: *this user manual page is for RUT955's old WebUI style available in earlier FW versions.* ***Click here*** *for information based on the latest FW version.*

☐

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 chapter of the user manual provides an overview of the Modbus page for RUT955 devices.

Modbus TCP

Modbus TCP provides users with the possibility to set or get system parameters. The Modbus daemon acts as slave device. That means it accepts connections from a master (client) and sends out a response or sets some system related parameter in accordance with the given query.

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	yes | no; default: **none**	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.
Allow Remote Access	yes | no; default: **no**	Allows remote Modbus connections by adding an exception to the device's firewall on the port specified in the field above.
Keep persistent connection	yes | no; default: **no**	If enabled, the connection will not be closed after each completed Modbus request.
Connection timeout	integer [1..60]; default: **0**	Timeout in seconds after which the connection will be closed. Use **0** to use default value provided by Operating System.
Enable custom register block	yes | no; default: **no**	Allow custom register block

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	32 bit unsigned integer
Analog input value	141	142	2	32 bit 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 integer
GPS satellite count	181	182	2	32 bit 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 (4 PIN connector)	324	325	1	16 bit unsigned integer
Digital open collector output (4 PIN connector)	325	326	1	16 bit unsigned integer
IMSI	348	349	16	Text

Set Parameters

The Modbus daemon can also set some device parameters. These parameters and explanations on how to use them are described in the table below:

value to set	register address	register number	register value	description
Hostname	7	8	Hostname	Changes hostname
Device name	71	72	Device name	Changes device name
Digital output 1 (DOUT1) (ON/OFF*)	201	202	1 | 0	Changes the state of the open collector (OC) output
Digital output 2 (DOUT2) (ON/OFF*)	202	203	1 | 0	Changes the state of the relay output
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 • 1 - switch to SIM1 • 2 - switch to SIM2 • 0 - switch from the the SIM card opposite of the one currently in use (SIM1 → SIM2 or SIM2 → SIM1
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: • 1 - to set APN for SIM1 • 2 - to set APN for SIM2
Switch PIN4 state (ON/OFF*)	325	326	1 | 0	Turns PIN4 state ON or OFF

* All ON/OFF commands only accept **0** and **1** values, which represent the following:

- 1 - ON
- 0 - OFF

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. You can create a maximum of 10 slave configurations.

Slave device configuration

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	yes | no; default: **no**	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	ip; default: **none**	Slave device's IP address.
Port	integer [0..65535]; default: **none**	Slave device's Modbus TCP port.
Period	integer [1..6400]; default: **60**	Interval at which requests are sent to the slave device.
Timeout	integer [1..30]; default: **5**	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. You can create a maximum of 64 request configurations for each slave device.

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 figure below is an example of the Requests configuration section and the table below provides information contained in the fields of that section:

Field	Value	Description
Name	string; default: **Unnamed Parameter**	Request name. Used for easier management purposes.
Data type	Hex | Ascii | 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, Byte order 1,2,3,4 | 32bit float, Byte order 4,3,2,1 | 32bit float, Byte order 2,1,4,3 | 32bit float, Byte order 3,4,1,2; default: **16bit INT, high byte first**	How read data will be stored.

Function	1 | 2 | 3 | 4 | 5 | 6 | 15 | 16; default: **3**	A function code specifies the type of register being addressed by a Modbus request. The codes represent these functions: • **1** - read Coil Status • **2** - read Input Status • **3** - read Holding Registers • **4** - read Input Registers • **5** - force Single Coil • **6** - preset Single Register • **15** - force Multiple Coils • **16** - force Multiple Registers
First Register	integer [1..65536]; default: **1**	First Modbus register number from which data will be read. **Note** - RUT9XX Modbus Master uses register numbers, which value is +1 higher than address value.
Number of Registers	integer [1..2000]; default: **none**	Number of Modbus registers that will be read during the request.
Enabled	yes | no; default: **no**	Turns the request on or off.
Test	- (interactive button)	Generates a Modbus request according to given parameters in order to test the request configuration. You must first save the configuration before you can use the Test button.
Delete	- (interactive button)	Deletes the request.
Add	- (interactive button)	Adds a new request configuration.

Alarm configuration

Alarms are a way of setting up automated actions when some Modbus values meet user specified conditions. The figure below is an example of the Alarm configuration page and the table below provides information on fields that it contains:

Field	Value	Description
Enabled	yes | no; default: **no**	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 in Modbus request.
Register	integer [0..65535]; default: **none**	Number of the Modbus coil/input/holding register/input register that will be read.
Condition	More than | Less than | Equal to | Not Equal to; default: **Equal to**	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.
Value	various; default: **none**	The value against which the read data will be compared.

Action	SMS | **Trigger output** | **Modbus Request**; default: **SMS**	Action that will be taken if the condition is met. Possible actions: • **SMS** - sends and SMS message to a specified recipient(s). • **Trigger output** - changes the state of a specified output(s). • **Modbus Request** - sends a Modbus request to a specified slave.
SMS: Message	string; default: **none**	SMS message text.
SMS: Phone number	phone number; default: **none**	Recipient's phone number.
Trigger output: Output	Open collector output | Relay output | Both; default: **Open collector output**	Which output(s) will be triggered.
Trigger output: I/O Action	Turn On | Turn Off | Invert; default: **Turn On**	Action that will taken on the specified output.
Modbus Request: IP address	ip | host; default: **none**	Modbus slave's IP address.
Modbus Request: Port	integer [0..65535]; default: **none**	Modbus slave's port.
Modbus Request: Timeout	integer [1..30]; default: **5**	Maximum time to wait for a response.
Modbus Request: ID	integer [1..255]; default: **none**	Modbus slave ID.
Modbus Request: Modbus function	Read Coil Status (1) | Read Input Status (2) | Read Holding Registers (3) | Read Input Registers (4) | Force Single Coil (5) | Preset Single Register (6) | Force Multiple Coils (15) | Force Multiple Registers (16); default: **Force Single Coil (5)**	A function code specifies the type of register being addressed by a Modbus request.
Modbus Request: First register	integer [0..65535]; default: **none**	Begins reading from the register specified in this field.
Modbus Request: Number of registers	integer [0..65535]; default: **none**	The number of registers that will be read from the first register.

Modbus Serial Master

The **Modbus Serial Master** page is used to configure the router as a Modbus RTU master. Modbus RTU (remote terminal unit) is a serial communication protocol mainly used in communication via RS232 or RS485 serial interfaces.

RS232

This section is used to configure the Modbus RTU master's RS232 serial interface settings. Refer to the figure and table below for information on RS232 configuration.

Field	Value	Description
Enabled	yes | no; default: **no**	Turns Modbus RTU via RS232 on or off.
Baud rate	300 | 1200 | 2400 | 4800 | 9600 | 19200 | 38400 | 57600 | 115200; default: **115200**	Serial data transmission rate (in bits per second).
Data bits	5 | 6 | 7 | 8; default: **8**	Number of data bits for each character.
Parity	None | Even | Odd; default: **Even**	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.
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.
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.

RS485

This section is used to configure the Modbus RTU master's RS485 serial interface settings. Refer to the figure and table below for information on RS485 configuration.

Field	Value	Description

Enabled	yes | no; default: **no**	Turns Modbus RTU via RS485 on or off.
Baud rate	300 | 1200 | 2400 | 4800 | 9600 | 19200 | 38400 | 57600 | 115200; default: **115200**	Serial data transmission rate (in bits per second).
Data bits	5 | 6 | 7 | 8; default: **8**	Number of data bits for each character.
Parity	None | Even | Odd; default: **Even**	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.
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.
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.

Slaves

The **Slaves** section is used to configure new Modbus slave devices. A Modbus slave is an entity that can be called upon by a Modbus master in order to obtain some type of information from it.

To create a new Modbus slave, enter a custom name for it and click the 'Add' button. Then click the 'Edit' button next to the slave in order to enter its configuration window.

Slave settings

The **Settings** section is used to configure the main parameters of the Modbus slave. Refer to the figure and table below for additional information.

Field	Value	Description
Enabled	yes | no; default: **no**	Turns the slave on or off.
Slave ID	integer [1..255]; default: **1**	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.
Frequency settings period	Period | Schedule; default: **Period**	Specifies whether request frequency should happen every x amount of seconds (*Period*) or on a set schedule (*Schedule*).
Period/Schedule	integer [1..9999]/; default: **10**	Interval (in minutes) at which requests are sent to the slave device. Or Shedule (crontab-like, three fields (HH MM SS); e.g.. *0,12* * *).

Slave requests

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 figure below is an example of the Requests configuration section and the table below provides information contained in the fields of that section:

Field	Value	Description
Enabled	yes | no; default: **no**	Turns the request on or off.
Function	Read Coil | Read Discrete Input | Read Holding Registers | Read Input Registers; default: **Read Holding Registers**	Modbus function used in Modbus request.
First Register	integer [1..65536]; default: **1**	First Modbus register from which data will be read.
Number of Registers	integer [1..2000]; default: **none**	Number of Modbus registers that will be read during the request/

Slave alarms

Alarms are a way of setting up automated actions when some Modbus values meet user specified conditions. The figure below is an example of the Alarm configuration page and the table below provides information on fields that it contains:

Field	Value	Description
Enabled	yes | no; default: **no**	Turns the alarm on or off.
Function	Read Coil | Read Discrete Input | Read Holding Registers | Read Input Registers; default: **Read Holding Registers**	Modbus function used in Modbus request.
Register	integer [1..65536]; default: **1**	Number of the Modbus coil/input/holding register/input register that will be read.
Condition	More than | Less than | Equal to | Not equal to; 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.
Value	integer [0..65535]; default: **0**	The value against which the read data will be compared.
Action	SMS | Trigger output | Modbus request; default: **SMS**	Action that will be taken if the condition is met. Possible actions: • **SMS** - sends and SMS message to a specified recipient(s). • **Trigger output** - changes the state of a specified output(s). • **Modbus Request** - sends a Modbus request to a specified slave.

Modbus Data to Server

The Modbus **Data to Server** function provides you with the possibility to set up senders that transfer data collected from Modbus slaves to remote servers. To add a new data sender, enter the server's address, specify the data sending period and click the "Add" button:

Data sender configuration

When you add a new data sender, you will be redirected to its configuration window. The figure below is an example of that window and the table below provides information on the fields that it contains:

Field	Value	Description
Enabled	yes | no; Default: **no**	Turns the data sender ON or OFF
Name	string; Default: **none**	Data sender's name. used for easier management purposes
Protocol	**HTTP(S)** | **MQTT**; Default: **HTTP(S)**	Data sending protocol

JSON format	json string; Default: **{"ID":"%i", "TS":"%t","ST":"%s","VR":"%a"}**	Provides the possibility to fully customize the JSON segment
Segment count	1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | All; Default: **1**	Max segment count in one JSON string sent to server.
URL / Host / Connection string	host | ip; Default: **none**	Address of the server to which the data will be sent. **Important note**: when using HTTPS, remember to add the ***https://*** prefix before the URL.
Period	integer [1..6400]; Default: **none**	Data sending frequency (in seconds)
HTTP(S): Data filtering	All data | By slave ID | By slave IP; Default: **All data**	Which data this sender will transfer to the server
HTTP(S): Retry on fail	yes | no; Default: **no**	Specifies whether the data sender should retry failed attempts
HTTP(S): Custom header	string; Default: **no**	Adds a custom header(s) to HTTP requests
MQTT: Port	integer [0..65535]; Default: **none**	Port used to connect to host.
MQTT: Keepalive	integer [1..640]; Default: **none**	MQTT keepalive period in seconds.
MQTT: Topic	string; Default: **none**	Write topic to which your data will be sent.
MQTT: QoS	0 | 1 | 2; Default: **0**	This field defines the guarantee of delivery for specific message. Possible values are: • At most once (0) • At least once (1) • Exactly once (2)
MQTT: Use TLS	yes | no; Default: **no**	Turns TLS authentication on or off.

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.

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	string; default: **request**	MQTT topic for sending requests.
Response topic	string; default: **response**	MQTT topic for subscribing to responses.
Username	string; default: **none**	Username for authentication to the MQTT broker. Leave empty if you do not use client authentication.
Password	string; default: **none**	Password for authentication to the MQTT broker. Leave empty if you do not use client authentication.

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** - timeoutfor Modbus TCP connection, in seconds. Range [1..999].
- **Slave ID** - Modbus TCP slave ID. Range [1..255].
- **Modbus function** - Only these are supported at the moment:
  - **3** - read holding registers;
  - **6** - write to a single holding register;
  - **16** - write to multiple holding registers.
- **Register number** - number 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
```

See also

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