

# 13 Modbus

- 13.1 Introduction
- 13.2 Modbus Master
- 13.3 Modbus Slave

# 13.1 Introduction

Modbus is a communication protocol for use with its programmable logic controllers (PLCs). Modbus has become a de facto standard communication protocol and is now a commonly available means of connecting industrial electronic devices. It was developed for industrial applications, is relatively easy to deploy and maintain compared to other standards, and places few restrictions other than size on the format of the data to be transmitted.

Modbus enables communication among many devices connected to the same network, for example, a system that measures temperature and humidity and communicates the results to a computer. Modbus is often used to connect a supervisory computer with a remote terminal unit (RTU) in supervisory control and data acquisition (SCADA) systems. Many of the data types are named from industry usage of Ladder logic and its use in driving relays: a single-bit physical output is called a coil, and a single-bit physical input is called a discrete input or a contact.

WCC Lite supports both Modbus Master and Slave protocols. One can select between transmission over TCP/IP or serial connection (RS-485/RS232). Bytes to transmit can either be encoded according to both RTU and ASCII parts of the standard.

# 13.2 Modbus Master

Modbus communication contains a single Master and may include more than 1, but not more than 247 devices. To gather data from peripheral devices, the master device requests a cluster of slave devices for data. If any device understands that this message is addressed to it, it will reply with data. As no timestamp is sent along with data, having recent data requires frequent polling. WCC Lite can be configured to acquire data periodically in custom-defined intervals.

## Configuring datapoints

Modbus Master in WCC Lite has to be configured via Excel. This configuration contains two Excel sheets where parameters have to be filled in - Devices and Signals

### Modbus Master parameters for the Devices tab

Parameter	Type	Description	Required		Default Value (when not specified)	Range	
			TCP	RTU/A SCII		Min	Max
name	string	User-friendly name for a device	Yes	Yes			
description	string	Description of a device	No	No			
device_alias	string	Alphanumeric string to identify a device	Yes	Yes			
enable	boolean	Enabling/disabling of a device	No	No	1	0	1
protocol	string	Protocol to be used	Yes	Yes		Modbus RTU, Modbus TCP	
ip	string	The IP address of the TCP slave device	Yes	-			
port	integer	TCP communication port	Yes	-	502		
bind_address	string	The IP address of the network adapter used to connect to the slave device (Default: "0.0.0.0")	No	No	0.0.0.0		
id	integer	Modbus Slave ID	Yes	Yes			
mode	string	Choosing between RTU ("rtu"), ASCII ("ascii") and TCP("tcp") modes. ASCII is the same as RTU, but with ASCII symbols.	No	No	TCP (for TCP) RTU (for Serial)	rtu, ascii, tcp	
timeout_ms	integer	Response timeout in milliseconds	No	No	10000		
device	string	Communication port ("PORT1"/"PORT2")	-	Yes		PORT1	PORT2
baudrate	integer	Communication speed, baud/s	-	No	9600	300, 600, 1200, 2400, 4800, 9600, 19200, 38400, 57600, 115200	

databits	integer	Data bit count for communication	-	No	8	6	9
stopbits	integer	Stop bit count for communication	-	No	1	1	2
parity	string	Communication parity option	-	No	none	none, even, odd	
flowcontrol	string	Number of requests, before the link is considered lost (device status signals are changed) and reconnect attempt will be issued	-	No	none	none	
scan_rate_ms	integer	If provided and positive - all jobs will have similar scan rates - all reads and writes will be executed within this timeframe (parameter scan_rate_ms in the Signals tab will be ignored)	No	No	300		
retry_count	integer	Number of requests, before the link is considered lost (device status signals are changed) and reconnect attempt will be issued	No	No	3		
serial_delay	integer	RS485 delay between read and write operations in milliseconds	-	No	50		
keep_alive_timeout	integer	Time interval for sending a keep-alive packet (in seconds)	No	-	60		
modbus_multi_write	boolean	Use 15/16 functions to write 1 register/coil (Default: 0)	No	No	0	0	1
comm_restart_delay	integer	Time delay between disconnecting from the slave device and restarting the connection (in milliseconds) (Default: 500)	No	-	500		
update	boolean	Enable to keep updating the tags even if they have the same value.	No	No	0	0	1

### Modbus Master parameters for the Signals tab

Parameter	Type	Description	Required		Default Value (when not specified)	Range	
			TCP	RTU/A SCII		Min	Max
signal_name	string	User-friendly signal name	Yes	Yes			
device_alias	string	Alphanumeric string to identify a device	Yes	Yes			

signal_alias	string	Unique alphanumeric name of the signal to be Yes used	Yes	Yes			
enable	boolean	Enabling/disabling of an individual signal	No	No	1	0	1
job_todo	string	Request to send according to Modbus specification without device address and checksum. This field can be identical on several tags to fetch them in a single request	Yes	Yes			
tag_job_todo	string	Similar format to the job_todo field. Address and length must be a subset of the job field. Defines the individual tag's register(s) or coil(s). Can be described in HEX or DEC formats	Yes	Yes			
number_type	string	Type of a number. Each allowed type can be prefixed by a single or a combination of swap prefixes (SW8, SW16, SW32), e.g. SW16.UNSIGNED32, SW32.SW16.SW8.DOUBLE	Yes	Yes		DIGITAL, SIGNED8, UNSIGNED8, SIGNED16, UNSIGNED16, SIGNED32, UNSIGNED32, FLOAT, DOUBLE	
log	integer	If the log parameter is 1, the signal will be visible in the events log tab, if 0, signals will not be logged.	No	No	0		
pulse_short_time_ms	integer	The time interval for short output pulse to stay active	No	No	0		
pulse_long_time_ms	integer	The time interval for a long output pulse to stay active	No	No	0		
periodic_update_ms	integer	Signal value will be published periodically according to the value set.	No	No			



Note: If the number type is more than 16 bits (this means FLOAT, SIGNED32, DOUBLE etc.), modbus multiwrite should be enabled (1).

## Device status signals

Modbus Master has an additional signal which can be configured to show communication status. It is used to indicate if the slave device has disconnected from the master (WCC Lite). To configure such signal for Modbus protocol, job\_todo and tag\_job\_todo fields with string values are required. For Modbus master required parameters for the status signal will be **signal\_name**, **device\_alias**, **signal\_alias**, **number\_type**, **job\_todo** and **tag\_job\_todo**. Job\_todo value must be *device\_status* and for tag\_job\_todo there are 4 variations: communication\_status, device\_running, device\_error, unknown\_error. Each signal has 4 possible values and is based on the same logic. If the signal returns the value of 0, it means an unknown error has appeared, 1 – device or protocol connection is on and working properly, 2 – device is off or protocol is disconnected, 3 – error or service is down.

Different device vendors can have different implementations of a Modbus protocol stack. A register table can be one of the primary differences. WCC Lite Modbus Master transmits the most significant word (byte) first, however, devices from some vendors might require transmitting the least significant word (byte) first. If that is the case, make sure to switch bytes as needed. To find out more about setting a correct number format, one should consult a section

`number_type`.

Modbus job or tag (as a task to be completed) can be built in two different formats - the user can select a more convenient way for him:

- hexadecimal format with every single byte separated by a | symbol. Device address, bytes containing output information and CRC (LRC) bytes should be excluded from the message;
- decimal format containing function number, first address and address count, separated by ; symbol. All other information should be excluded from the message;

`job_todo` can group several `tag_job_todo`'s. That way one Modbus message can be used to extract several tags. Grouping is accomplished dynamically meaning that if several identical jobs are found, their tags are grouped automatically.

## Debugging a Modbus Master application

If the configuration for Modbus Master is set up, a handler for the protocol will start automatically. If the configuration is missing or contains errors, the protocol will not start. It is done intentionally to decrease unnecessary memory usage.

Modbus Master command line debugging options

`modbus-master`

```
-h [ -help ] Display help information
-V [ -version ] Show version
-d<debug level> Set debugging level
-c [ -config ] Config path
-e [ -redis ] Show redis debug information
```

If Modbus Master does not work properly (e.g. no communication between devices, data is corrupted, etc.), a user can launch a debug session from the command line interface and find out why the link is not functioning properly. To launch a debugging session, a user should stop the Modbus-master process and run the Modbus-master command with respective flags as shown above.

To run a debug session:

- **Step 1:** Service must be stopped by entering the following command into the wclite:

```
/etc/init.d/modbus-master stop
```

- **Step 2:** After the service is stopped it must be started with the preferred configuration file (JSON files found in the /etc/ folder) and a debug level 7:

```
modbus-master -c /etc/modbus-master/modbus-master.json -d7
```

Additional output forming options described in the table above.

- **Step 3:** Once the problem is diagnosed normal operations can be resumed with the following command:


```
/etc/init.d/modbus-master start
```

# 13.3 Modbus Slave

WCC Lite can act as one (or several) of slave devices in a communication line. This can be used to transmit data to SCADA systems or other RTU devices. It can reply to messages from Modbus Master with matching devices and register addresses.

## Configuring datapoints

Modbus Slave in WCC Lite has to be configured via Excel. This configuration contains two Excel sheets where parameters have to be filled in - Devices and Signals

 If TCP/IP is used as a transmission medium, only devices with IPs predefined in the host column are allowed to connect. All other connections are rejected

### Modbus Slave parameters for Devices tab

Parameter	Type	Description	Required		Default Value (when not specified)	Range	
			TCP	RTU/A SCII		Min	Max
name	string	User-friendly name for a device	Yes	Yes			
description	string	Description of a device	No	No			
device_alias	string	Alphanumeric string to identify a device	Yes	Yes	unknown		
enable	boolean	Enabling/disabling of a device	No	No	1	0	1
protocol	string	Protocol to be used	Yes	Yes		Modbus serial Slave, Modbus TCP Slave	
host	string	Space-separated host IP addresses of master device	Yes	-			
port	integer	TCP port to listen for incoming connections	Yes	-			
bind_address	string	The IP address of the network adapter used to connect to the slave device (Default: "0.0.0.0")	No	No	0.0.0.0		
keep_alive_timeout	integer	The minimum time a connection can be idle without being closed in seconds	No	No	60		
mode	string	Choosing between RTU ("rtu"), ASCII ("ascii") and TCP("tcp") modes. ASCII is the same as RTU, but with ASCII symbols.	No	No	TCP (for TCP) RTU (for Serial)	rtu, ascii, tcp	
device	string	Communication port ("PORT1"/"PORT2")	-	Yes		PORT1	PORT2
baudrate	integer	Communication speed, baud/s	-	Yes	9600	300, 600, 1200, 2400, 4800, 9600, 19200, 38400, 57600, 115200	
databits	integer	Data bit count for communication	-	Yes	8	6	9
stopbits	integer	Stop bit count for communication	-	Yes	1	1	2
parity	string	Communication parity option	-	Yes	none	none, even, odd	

flowcontrol	string	The communication device's flow control option.	-	No	none	none
-------------	--------	---	---	----	------	------

## Modbus Slave parameters for the Signals tab

Parameter	Type	Description	Required		Default Value (when not specified)	Range	
			TCP	RTU/ ASCII		Min	Max
signal_name	string	User-friendly signal name	Yes	Yes			
device_alias	string	Alphanumeric string to identify a device	Yes	Yes			
signal_alias	string	Unique alphanumeric name of the signal to be used	Yes	Yes			
enable	boolean	Enabling/disabling an individual signal	No	No	1	0	1
number_type	string	Type of a number (FLOAT, DOUBLE, DIGITAL, etc.). This defines the size that will be read.	Yes	Yes			
log	integer	If the log parameter is equal to 1, the signal values and attributes will be seen in the events log.	No	No	0		
slave_id	integer	Address of a slave device	Yes	Yes			
function	integer	Function number	Yes	Yes		1, 2, 3, 4	
register_address	integer	Register address	Yes	Yes			
periodic_update_ms	integer	Signal value is published periodically according to the value set.	No	No	-	-	-


## Device status signals

Modbus slave has an additional signal which can be configured to show communication status. It indicates if the master device has disconnected from the slave (WCC Lite). To configure such signal for Modbus protocol, **job\_todo** and **tag\_job\_todo** fields with string values are required. For Modbus slave required parameters for the status, the signal will be **signal\_name device\_alias, signal\_alias, number\_type, slave\_id, function, register\_address, job\_todo** and **tag\_job\_todo**. **job\_todo** value must be *device\_status* and for **tag\_job\_todo** there are 4 variations: *communication\_status*, *device\_running*, *device\_error*, *unknown\_error*. Each signal has 4 possible values and is based on the same logic. If the signal returns the value of 0, it means an unknown error has appeared, 1 – device or protocol connection is on and working properly, 2 – device is off or protocol is disconnected, 3 – error or service is down.

## Mapping values to registers

Internally stored values aren't organised in a register-like order, therefore mapping should be done by the user. This mapping includes setting the address of the device WCC Lite is simulating as well as the function number, register number and how many 16-bit registers are used to store a value. These values should be set in `common_address`, `function`, `info_address` and `size` columns respectively in the Excel configuration.

To find out how many registers should be used for storing values, and how values can have their values swapped, a user should consult a section [number\\_type](#).

 If a Modbus master device requests data from a register that is mapped but doesn't yet have an initial value, an **ILLEGAL DATA ADDRESS** error code will be returned. The same error code is returned if the requested size of the value is bigger than defined or if the register is not configured at all.

## Debugging a Modbus Slave application

If the configuration for Modbus Slave is set up, a handler for the protocol will start automatically. If the configuration is missing or contains errors, the protocol will not start. It is done intentionally to decrease unnecessary memory usage.

Modbus Slave command line debugging options

To run a debug session:

- **Step 1:** Service must be stopped by entering the following command into the wccLite:

```
/etc/init.d/modbus-slave stop
```

- **Step 2:** After the service is stopped it must be started with the preferred configuration file (JSON files found in the /etc/ folder) and a debug level 7:


```
modbus-slave -c /etc/modbus-slave/modbus-slave.json -d7
```


Additional output forming options described in the table below.

- **Step 3:** Once the problem is diagnosed normal operations can be resumed with the following command:

```
/etc/init.d/modbus-slave start
```

```
-h [ -help ] Display help information  
-V [ -version ] Show version  
-d<debug level> Set debugging level  
-c [ -config ] Config path  
-e [ -redis ] Show redis debug information
```

 If Modbus Slave does not work properly (e.g. no communication between devices, data is corrupted, etc.), a user can launch a debug session from the command line interface and find out why the link is not functioning properly.

 To launch a debugging session, a user should stop `modbus-slave` process and run `modbus-slave` command with respective flags as shown above.