

UPDU Modbus TCP

Riedo Networks Ltd
Switzerland

Revision: 3.0.0-545f6b60

Date: October 9, 2023

Contents

Overview **3**

 Protocol compliancy 3

 Data types and conventions 3

 Reserved and invalid registers 3

Supported commands **4**

 Read Input Registers (0x04) 4

Device configuration **5**

 Enable Modbus TCP 5

 Print Register Map 5

Holding Register Map (Version 1.0) **6**

 Global Object Mapping 6

 PDU Information Object 6

 Sensor Object 6

 RCM Object 7

 Power Object 7

Overview

The UPDU features a *Modbus TCP* server which must be enabled before usage. The *Modbus TCP* protocol allows accessing all measurement values of the device using a very simple and efficient protocol.

The current implementation of the *Modbus TCP* protocol allows reading all relevant values of a device. Writing settings or values is currently not supported.

Protocol compliance

The implementation follows the specifications of the following protocol specifications published by the Modbus Organization:

- MODBUS Application Protocol Specification V1.1b3
- MODBUS Messaging on TCP/IP Implementation Guide V1.0b

Exceptions and conventions to the above mentioned specifications are described below.

Data types and conventions

In order to overcome the 16 bit upper size limit for values defined by the *Modbus* specification, a few additional datatypes have been implemented.

- `bitfield` : A single Modbus register representing 16 individual bits.
- `uint16` : A single Modbus register holding a 16-bit unsigned integer.
- `uint32` : 2 Modbus registers to hold a 32-bit unsigned integer.
- `uint64` : 4 Modbus registers to hold a 64-bit unsigned integer.

For any integers spanning over multiple Modbus registers, the data is returned in big-endian format. Thus, the lowest register address contains the most significant bits.

Important: The Modbus server supports reading multiple registers with a single read command. Care must be taken when reading values which span over multiple Modbus registers. Data consistency is only guaranteed for a response to a single read command. It is therefore important to ensure that multi-register values are always obtained with a single read instruction.

Reserved and invalid registers

In addition to the specified and valid registers, certain addresses contain reserved registers. Reserved registers can be read but will not contain valid information. These are listed as followed:

- `res` : A single register reserved for future or internal use.
- `res[n]` : Multiple (n) registers reserved for future or internal use.

In addition to reserved registers, the global register map defines invalid ranges which are not to be accessed. Any operation on these register addresses will result in returning an error according to the Modbus protocol specification.

Supported commands

The following sections describe the currently supported commands. Note that future firmware versions may support additional commands. For details on how to use the commands, refer to the Modbus standards or the documentation of the controller (e.g. PLC).

Read Input Registers (0x04)

This command is used to read from 1 to 125 continuous input registers from an UPDU.

Device configuration

In order to use the Modbus TCP server with a UPDU, the device has to be configured accordingly. The following is a quick start using commands which are documented in the *CLI Reference Manual*.

Enable Modbus TCP

To enable the service, connect to the CLI console and configure the device to enable Modbus TCP:

```
updu> configure
updu(config)# modbus/tcp
updu(config-modbus/tcp)# enabled
updu(config-modbus/tcp)# end
Leaving configuration mode. Use the "write" command to save changes.
updu> write
Configuration saved.
```

Print Register Map

As the actual register map depends on the UPDU model, the CLI allows to print a list with the start address of the object registers:

```
updu> show modbus/tcp
Hex   Dec   # Object      Description
0x0000    0    8          PDU Information
0x0200  512 16 Sensor1     Sensors
0x0210  528 16 Sensor2     Sensors
0x0220  544 16 Sensor3     Sensors
0x0400 1024 32 PDU         PDU Total Power
0x0500 1280 32 Inlet      Inlet Power
0x0600 1536 32 WireL      Wire Power
0x1000 4096 32 Module0      Module Power
0x2000 8192 32 Outlet0.1      Outlet Power
0x2020 8224 32 Outlet0.2      Outlet Power
0x2040 8256 32 Outlet0.3      Outlet Power
0x2060 8288 32 Outlet0.4      Outlet Power
0x2080 8320 32 Outlet0.5      Outlet Power
0x20a0 8352 32 Outlet0.6      Outlet Power
0x20c0 8384 32 Outlet0.7      Outlet Power
0x20e0 8416 32 Outlet0.8      Outlet Power
```

Legend:

Hex/Dec: Register address in hexadecimal/decimal notation
 #: Number of registers
 Object: Corresponding PDU object
 Description: Field or register object description

To print a complete register map with all registers, append the `detail` parameter:

```
updu-101718> show modbus/tcp detail
Hex   Dec   # Object      Description
0x0000    0    1          Modbus/TCP register map version
0x0001    1    1          Number of inlet objects
...
```

Holding Register Map (Version 1.0)

The holding registers are read-only registers which can be obtained by using the Read Input Registers command.

Global Object Mapping

The following object map specifies the address ranges to read values from a UPDU. Depending on the model, a certain number of object are present.

| Start | End | Description |
|--------|--------|----------------------------|
| 0x0000 | 0x0007 | PDU Information object |
| 0x0008 | 0x01ff | (Reserved range) |
| 0x0200 | 0x02ff | Sensor objects (16) |
| 0x0300 | 0x03ff | RCM objects (16) |
| 0x0400 | 0x041f | PDU Total Power object |
| 0x0420 | 0x04ff | (Reserved range) |
| 0x0500 | 0x05ff | Inlet Power objects (8) |
| 0x0600 | 0x07ff | Wire Power objects (16) |
| 0x0800 | 0x0fff | Branch Power objects (64) |
| 0x1000 | 0x17ff | Module Power objects (64) |
| 0x1800 | 0x1fff | (Reserved range) |
| 0x2000 | 0x3fff | Outlet Power objects (256) |
| 0x4000 | 0xffff | (Invalid range) |

PDU Information Object

The *PDU Information Object* lists the active version of the register map along with the number of available objects per type.

Object total size: 8 Registers

| Offset | Type | Description |
|--------|--------|----------------------|
| 0x00 | uint16 | Register Map Version |
| 0x01 | uint16 | Number of inlets |
| 0x02 | uint16 | Number of wires |
| 0x03 | uint16 | Number of branches |
| 0x04 | uint16 | Number of modules |
| 0x05 | uint16 | Number of outlets |
| 0x06 | uint16 | Number of RCMs |
| 0x07 | uint16 | Number of sensors |

Sensor Object

The values of external sensors connected to the device.

Object total size: 16 Registers

Register map:

| Offset | Type | Description |
|--------|----------|--------------|
| 0x00 | bitfield | Capabilities |
| 0x01 | bitfield | Status |
| 0x02 | bitfield | Monitoring |

| Offset | Type | Description |
|--------|---------|---------------------------|
| 0x03 | res | |
| 0x04 | uint16 | Temperature in 0.1 deg-C |
| 0x05 | uint16 | Relative humidity 0.1 %RH |
| 0x06 | res[10] | |

Bitfields

The capabilities, status and monitoring bitfields contain one bit per functionality in this object:

- **Capabilities:** A set bit (1) indicates that a certain functionality supported.
- **Status:** A set bit (1) indicates that the corresponding value is valid. A cleared (0) bit thus indicates an issue with obtaining the measurement.
- **Monitoring:** A set bit (1) indicates that a monitoring condition for the corresponding value is currently active. A clear bit (0) thus indicates that the measurement values are within the good range or no rules have been configured for the value.

The bits are as follows:

- Bit 0: Sensor provides temperature
- Bit 1: Sensor provides relative humidity

RCM Object

| Offset | Type | Description |
|--------|----------|--------------------------------|
| 0x00 | bitfield | Capabilities |
| 0x01 | bitfield | Status |
| 0x02 | bitfield | Monitoring |
| 0x03 | res | |
| 0x04 | uint16 | Residual RMS current in 0.1 mA |
| 0x05 | uint16 | Residual DC current in 0.1 mA |
| 0x06 | res[10] | |

The capabilities, status and monitoring bitfields contain one bit per functionality in this object:

- The bit in the capabilities bitset tells if the functionality is supported (1) or not (0).
- The bit in the status bitset tells if the corresponding field is currently filled with valid data (1) or not (0).
- The bit in the monitoring bitset tells if the corresponding field is currently OK (0) or if a condition is active (1).

The bits are as follows:

- Bit 0: RCM provides residual RMS current
- Bit 1: RCM provides residual DC current

Power Object

| Offset | Type | Description |
|--------|----------|--------------|
| 0x00 | bitfield | Capabilities |
| 0x01 | bitfield | Status |
| 0x02 | bitfield | Monitoring |
| 0x03 | res | |

| Offset | Type | Description |
|--------|--------|----------------------------------|
| 0x04 | uint32 | RMS current in mA |
| 0x06 | uint32 | RMS voltage in mV |
| 0x08 | uint32 | Active power in W |
| 0x0a | uint32 | Apparent power in var |
| 0x0c | uint32 | Reactive power in VA |
| 0x0e | uint16 | Power factor in per mill |
| 0x0f | uint16 | Frequency in 0.001 Hz |
| 0x10 | uint64 | Positive active energy in Wh |
| 0x14 | uint64 | Positive reactive energy in varh |
| 0x18 | uint64 | Negative reactive energy in varh |
| 0x1c | res[4] | |

The capabilities, status and monitoring bitfields contain one bit per functionality in this object:

- The bit in the capabilities bitset tells if the functionality is supported (1) or not (0).
- The bit in the status bitset tells if the corresponding field is currently filled with valid data (1) or not (0).
- The bit in the monitoring bitset tells if the corresponding field is currently OK (0) or if a condition is active (1).

The bits are as follows:

- Bit 0: Object provides RMS current
- Bit 1: Object provides RMS voltage
- Bit 2: Object provides active power
- Bit 3: Object provides reactive power
- Bit 4: Object provides apparent power
- Bit 5: Object provides power factor
- Bit 6: Object provides frequency
- Bit 7: Object provides positive active energy
- Bit 8: Object provides positive reactive energy
- Bit 9: Object provides negative reactive energy