Reference Manual

Sourcer Modbus client

Index

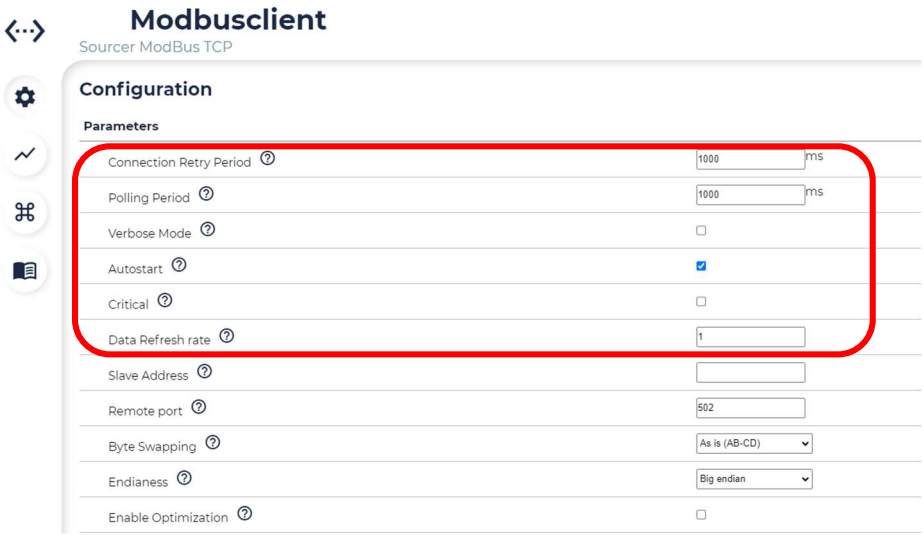
1 Introduction

The sourcer Modbus allows users to connect UDS Atom quickly and easily to any Modbus compatible device. Some of the main functionalities supported by the module include:

- TCP and Serial communications: Modbus TCP, Modbus RTU, and RTU over TCP.
- Unlimited connections: multiple connections can be run in parallel to provide the fastest overall communications.
- Poll on demand: poll can depend on a tag property, if needed.
- Automatic reconnection
- Function Codes Supported: listed in the following table.

Function Name	Code	Hex
Read Discrete Inputs	02	0x02
Read Coils	01	0x01
Write Single Coils	05	0x05
Read Input Register	04	0x04
Read Holding Register	03	0x03
Write Single Register	06	0x06

2 General configuration

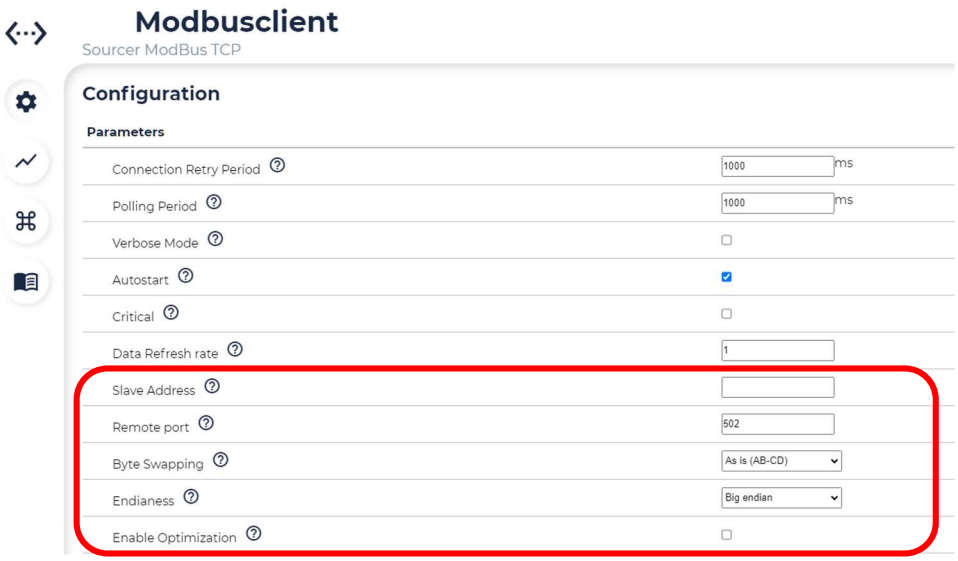


Every sourcer have a common set of parameters (highlighted in red on the above figure)

- Connection Retry Period: Time before trying to re-open the connection after a failed attempt, displayed in milliseconds.
- Polling Period: The polling period (in milliseconds) is the amount of time that passes before the sourcer communicates updated values to UDS Atom.
- Verbose Mode: When this option is set to true, the sourcer provides additional information as to what the sourcer is doing in the logbook.
- Autostart: When **Autostart** is set to true, the sourcer will start automatically after UDS Atom startup. If set to false, the sourcer will be in **Stand-by** state.
- Critical: When this option is set to true, the sourcer will send a notification when it's down.
- Data Refresh rate: This parameter determines the rate which data is sent to the hub. This parameter is tied to **Poll Period**. For example, setting a **Data Refresh Rate** to 10 with a **Poll Period** to 2000 means that a bunch of 10 samples will be sent every 20 seconds.

3 Modbus TCP

3.1 Specific configuration



Modbus TCP client have some specific parameters among the followings:

- Slave Address: Hostname or IP address of the target device.
- Remote Port: Determines which TCP Port to connect to. Valid values range from 1 to 65535.
- Read Timeout: Specifies the read operation timeout.
- Write Timeout: Specifies the write operation timeout.
- Byte swapping: Defines how bytes of registers values are interpreted. Allowed values are **"As is (AB-CD)"** or **"Byte Swap (BA-DC)"**
- Endianess: Defines how 2 consecutive registers are interpreted as a float. Allowed values are **"Big endian"** or **"Little endian"**
- Enable Optimization: If True, the sourcer try to optimize Read Functions by grouping range of registers to read.

3.2 Tags configuration

3.2.1 Input tags

Input tags are defined in a JSON array of objects like the following:

```
[
  {
    "Channel": "T_OUTLET",
    "Label": "Temperature",
    "Address": "%S400005",
    "Unit": "°C",
    "HighPrio": false,
    "Metadata":
      {
        "MyText":"information about data"
      }
  }
]
```

Each tag is defined by:

Parameter	Type	Required	Description
Channel	string	NO	Unique Id of the channel (optional). if this field is missing, Channel = Address field value
Label	string	NO	Caption of the channel. A more human readable name for the channel. If empty, field is like Channel field
Address	string	**YES**	Address of the register to read
Unit	string	NO	Unit of the read value
HighPrio	boolean	NO	If **true** result of the virtual channel will be computed at high priority by pushers
Metadata	JSON object	NO	Free JSON object to put information on the channel

Addresses are formatted as followed:

Register Type	Address Format
Coils (Code 01)	%0XXXXX
Discrete Register (Code 02)	%1XXXXX
U16 Holding Register (Code 03)	%4XXXXX
I16 Holding Register (Code 03)	%S4XXXXX
U32 Holding Register (Code 03)	%D4XXXXX
I32 Holding Register (Code 03)	%SD4XXXXX
Float Holding Register (Code 03)	%F4XXXXX
U16 Input Register (Code 04)	%3XXXXX
I16 Input Register (Code 04)	%S3XXXXX
U32 Input Register (Code 04)	%D3XXXXX
I32 Input Register (Code 04)	%SD3XXXXX
Float Input Register (Code 04)	%F3XXXXX

NOTE: The registers tagged U32, I32 or Float read 2 consecutive registers from the Address to make a U32, I32 or a Float based on IEEE754.

3.2.2 Output tags

Output tags are defined in a JSON array of objects like the following:

```
  {
    "Label": "Actuator",
    "Address": "%F412500",
  }
]
```

Each output is defined by:

Parameter	Type	Required	Description
Label	string	**YES**	Name given to the output tag
Address	string	**YES**	Address of the register to write

Addresses are formatted as followed:

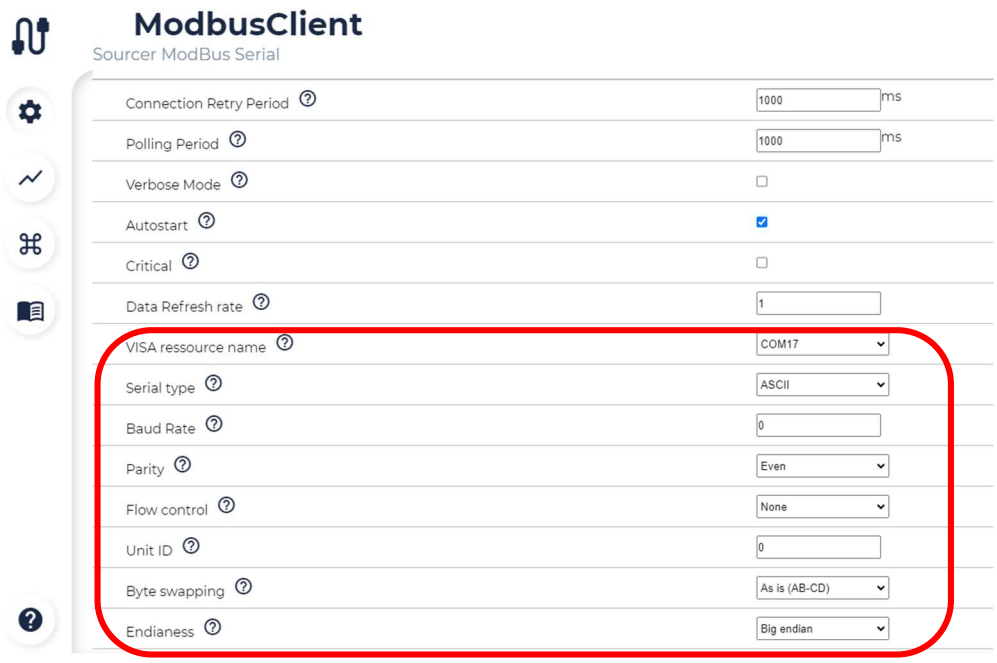
Register Type	Address Format
Coils (Code 05)	0XXXXX
U16 Holding Register (Code 06)	%4XXXXX
I16 Holding Register (Code 06)	%S4XXXXX
U32 Holding Register (Code 06)	%D4XXXXX
I32 Holding Register (Code 06)	%SD4XXXXX

Float Holding Register (Code 06)	%F4XXXXX

The functions tagged U32, I32 or Float write a U32, I32 or a float (interpreted as an I32 in IEEE754 encoding) into 2 consecutive registers from the Address.

4 Modbus serial

4.1 Specific configuration



Modbus TCP client have some specific parameters among the followings:

- VISA resource name: Serial port as displayed by the operating system
- Serial type: Specifies the Modbus working mode. Allowed values are **RTU** or **ASCII**
- Baud rate: Serial port transfer speed in milliseconds. Valid values are 110, 300, 1200, 2400, 4800, 9600, 19200, 38400, 57600, or 115200.
- Parity: Specifies the type of parity for data. Valid values are None, even or odd.
- Flow Control: Allowed values are **None**, **DTR/DSR**, **XON/XOFF**, **RTS/CTS**, **XON/XOFF & DTR/DSR**, **XON/XOFF & RTS/CTS**
- Unit ID: Slave Address you want to connect.
- Byte swapping: Defines how bytes of registers values are interpreted. Allowed values are **"As is (AB-CD)"** or **"Byte Swap (BA-DC)"**
- Endianess: Defines how 2 consecutive registers are interpreted as a float. Allowed values are **"Big endian"** or **"Little endian"**
- Enable Optimization: If True, the sourcer try to optimize Read Functions by grouping range of registers to read.

4.2 Tags configuration

4.2.1 Input tags

Input tags are defined in a JSON array of objects like the following:

```
[
  {
    "Channel": "T_OUTLET",
    "Label": "Temperature",
    "Address": "%S400005",
    "Unit": "°C",
    "HighPrio": false,
    "Metadata":
```

```
        {
          "MyText":"information about data"
        }
    }
]
```

Each tag is defined by:

Parameter	Type	Required	Description
Channel	string	NO	Unique Id of the channel (optional). if this field is missing, Channel = Address field value
Label	string	NO	Caption of the channel. A more human readable name for the channel. If empty, field is like Channel field
Address	string	**YES**	Address of the register to read
Unit	string	NO	Unit of the read value
HighPrio	boolean	NO	If **true** result of the virtual channel will be computed at high priority by pushers
Metadata	JSON object	NO	Free JSON object to put information on the channel

Addresses are formatted as followed:

Register Type	Address Format
Coils (Code 01)	%0XXXXX
Discrete Register (Code 02)	%1XXXXX
U16 Holding Register (Code 03)	%4XXXXX
I16 Holding Register (Code 03)	%S4XXXXX
U32 Holding Register (Code 03)	%D4XXXXX
I32 Holding Register (Code 03)	%SD4XXXXX
Float Holding Register (Code 03)	%F4XXXXX
U16 Input Register (Code 04)	%3XXXXX
I16 Input Register (Code 04)	%S3XXXXX
U32 Input Register (Code 04)	%D3XXXXX
I32 Input Register (Code 04)	%SD3XXXXX
Float Input Register (Code 04)	%F3XXXXX

NOTE: The registers tagged U32, I32 or Float read 2 consecutive registers from the Address to make a U32, I32 or a Float based on IEEE754

4.2.2 Output tags

Output tags are defined in a JSON array of objects like the following:

```
  {
    "Label": "Actuator",
    "Address": "%F412500",
  }
]
```

Each output is defined by:

Parameter	Type	Required	Description
Label	string	**YES**	Name given to the output tag
Address	string	**YES**	Address of the register to write

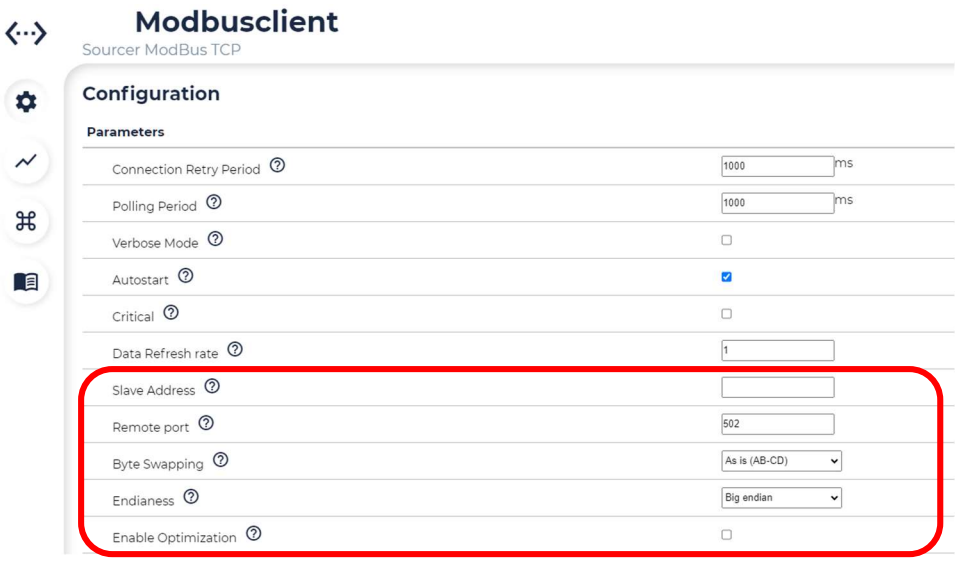
Addresses are formatted as followed:

Register Type	Address Format
Coils (Code 05)	0XXXXX
U16 Holding Register (Code 06)	%4XXXXX
I16 Holding Register (Code 06)	%S4XXXXX
U32 Holding Register (Code 06)	%D4XXXXX
I32 Holding Register (Code 06)	%SD4XXXXX
Float Holding Register (Code 06)	%F4XXXXX

The functions tagged U32, I32 or Float write a U32, I32 or a float (interpreted as an I32 in IEEE754 encoding) into 2 consecutive registers from the Address.

5 Modbus RTU over TCP

5.1 Specific configuration



Modbus TCP client have some specific parameters among the followings:

- Gateway Address: Hostname or IP address of the target device.
- Remote Port: Determines which TCP Port to connect to. Valid values range from 1 to 65535.
- Read Timeout: Specifies the read operation timeout.
- Write Timeout: Specifies the write operation timeout.
- Byte swapping: Defines how bytes of registers values are interpreted. Allowed values are **"As is (AB-CD)"** or **"Byte Swap (BA-DC)"**
- Endianess: Defines how 2 consecutive registers are interpreted as a float. Allowed values are **"Big endian"** or **"Little endian"**
- Enable Optimization: If True, the sourcer try to optimize Read Functions by grouping range of registers to read.

5.2 Tags configuration

5.2.1 Input tags

Input tags are defined in a JSON array of objects like the following:

```
[
  {
    "Slave Address":1,
    "Slave Paramaters":
    [
      {
        "Channel": "T_OUTLET",
        "Label": "Temperature",
        "Address": "%S400005",
        "Unit": "°C",
        "HighPrio": false,
        "Metadata":
          {
            "MyText":"information about data"
          }
```

```
        }
      ]
    }
]
```

Each slave to be read is defined by:

Parameter	Type	Required	Description
Slave Address	numeric	**YES**	RTU slave address
Slave Parameters	Array of JSON objects	**YES**	List of channels to read from the RTU slave

Each **Slave Parameters** is defined by:

Parameter	Type	Required	Description
Channel	string	NO	Unique Id of the channel (optional). if this field is missing, Channel = Address field value
Label	string	NO	Caption of the channel. A more human readable name for the channel. If empty, field is like Channel field
Address	string	**YES**	Address of the register to read
Unit	string	NO	Unit of the read value
HighPrio	boolean	NO	If **true** result of the virtual channel will be computed at high priority by pushers
Metadata	JSON object	NO	Free JSON object to put information on the channel

Addresses are formatted as followed:

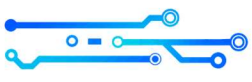
Register Type	Address Format
Coils (Code 01)	%0XXXXX
Discrete Register (Code 02)	%1XXXXX
U16 Holding Register (Code 03)	%4XXXXX
I16 Holding Register (Code 03)	%S4XXXXX
U32 Holding Register (Code 03)	%D4XXXXX
I32 Holding Register (Code 03)	%SD4XXXXX
Float Holding Register (Code 03)	%F4XXXXX
U16 Input Register (Code 04)	%3XXXXX
I16 Input Register (Code 04)	%S3XXXXX
U32 Input Register (Code 04)	%D3XXXXX
I32 Input Register (Code 04)	%SD3XXXXX
Float Input Register (Code 04)	%F3XXXXX

NOTE: The registers tagged U32, I32 or Float read 2 consecutive registers from the Address to make a U32, I32 or a Float based on IEEE754.

5.2.2 Output tags

Output tags are defined in a JSON array of objects like the following:

```
[
  {
    "Slave Address":1,
    "Slave Paramaters":
    [
      {
```

```
        "Label": "Temperature",
        "Address": "%S400005",
}
    ]
  }
]
```

Each output is defined by:

Parameter	Type	Required	Description
Slave Address	numeric	**YES**	RTU slave address
Slave Parameters	Array of JSON objects	**YES**	List of channels to read from the RTU slave

Each **Slave Parameters** is defined by:

Parameter	Type	Required	Description
Label	string	**YES**	Name given to the output tag
Address	string	**YES**	Address of the register to write

Addresses are formatted as followed:

Register Type	Address Format
Coils (Code 05)	0XXXXX
U16 Holding Register (Code 06)	%4XXXXX
I16 Holding Register (Code 06)	%S4XXXXX
U32 Holding Register (Code 06)	%D4XXXXX
I32 Holding Register (Code 06)	%SD4XXXXX
Float Holding Register (Code 06)	%F4XXXXX

The functions tagged U32, I32 or Float write a U32, I32 or a float (interpreted as an I32 in IEEE754 encoding) into 2 consecutive registers from the Address.

UDS ATOM

Dedicated software
for OT/IT convergence

Sales Contact

sales@udsatom.com

Technical Support

support@udsatom.com