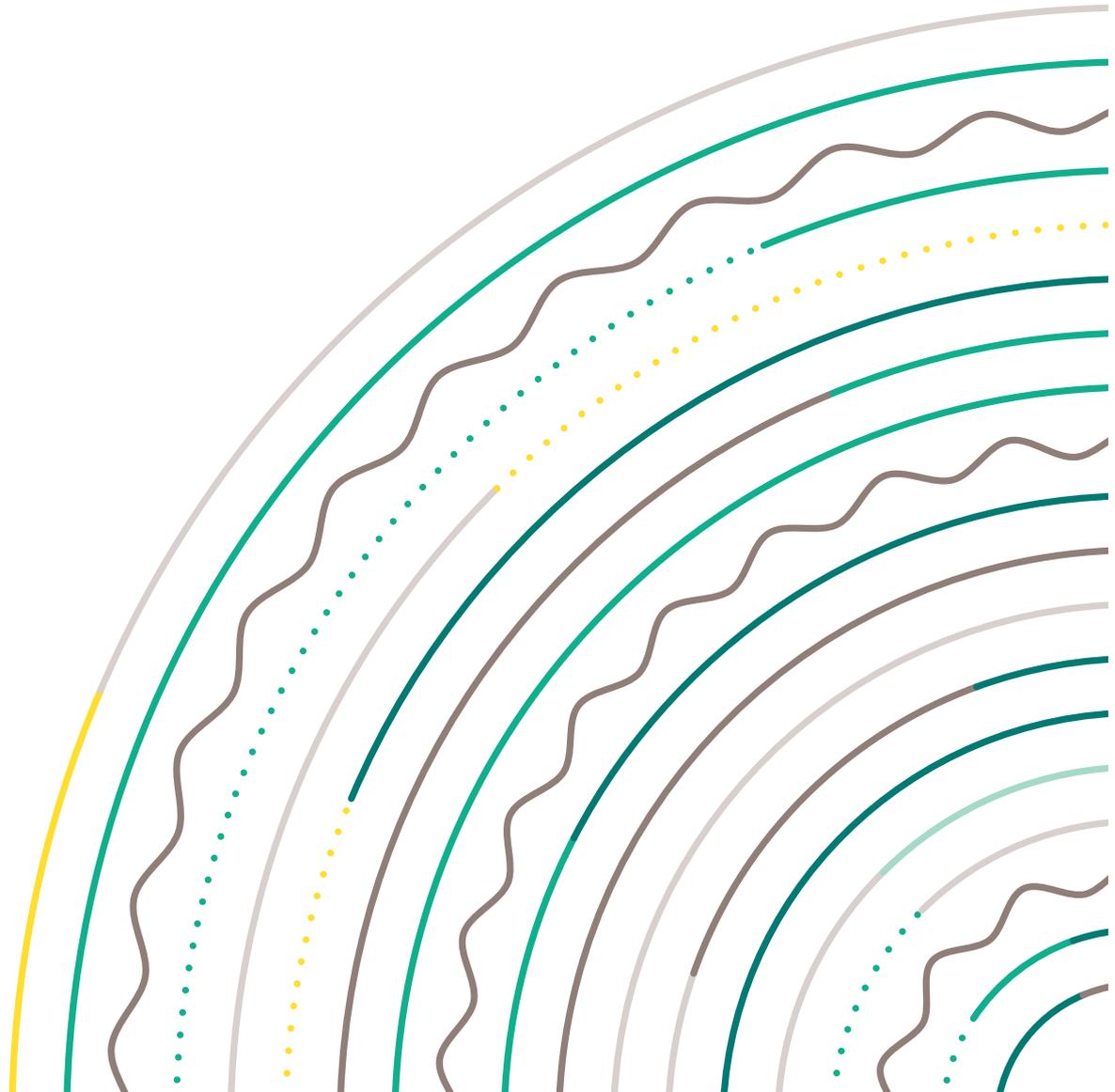


# Lifedomus

## Modbus

31/01/2018

Version 1.4



# The Modbus TCP protocol, 'Master' mode

## Table of contents

1	Prerequisites .....	3
2	Modbus connector .....	3
2.1	Adding a Modbus TCP connector.....	3
2.2	Configuring a Modbus TCP connector .....	3
3	Modbus device .....	4
3.1	Device properties .....	4
3.1.1	Configuring a property's 'read' field.....	4
3.1.2	Configuring a property's 'write' field .....	4
3.2	Device variables .....	5
3.2.1	Variable management .....	5
3.2.2	Configuring a variable .....	6
3.3	Device commands.....	8
4	Creating the 'Design Studio' interface.....	9

## 1 Prerequisites

To fully understand this documentation, a few prerequisites are necessary:

- ✓ Addition, configuration and use of a connector in 'Config Studio'
- ✓ Creating a generic Lifemodus device
- ✓ Modbus protocol
- ✓ Basics on logic binary operations to understand and use the 'mask'

## 2 Modbus connector

Communication between Lifedomus and a Modbus bus requires the use of a Modbus TCP/IP gateway.

As with all the other protocols, communication is done via a Lifemodus connector. Currently, Lifemodus behaves as a **'master'** on the bus. A 'slave' mode will be implemented subsequently.

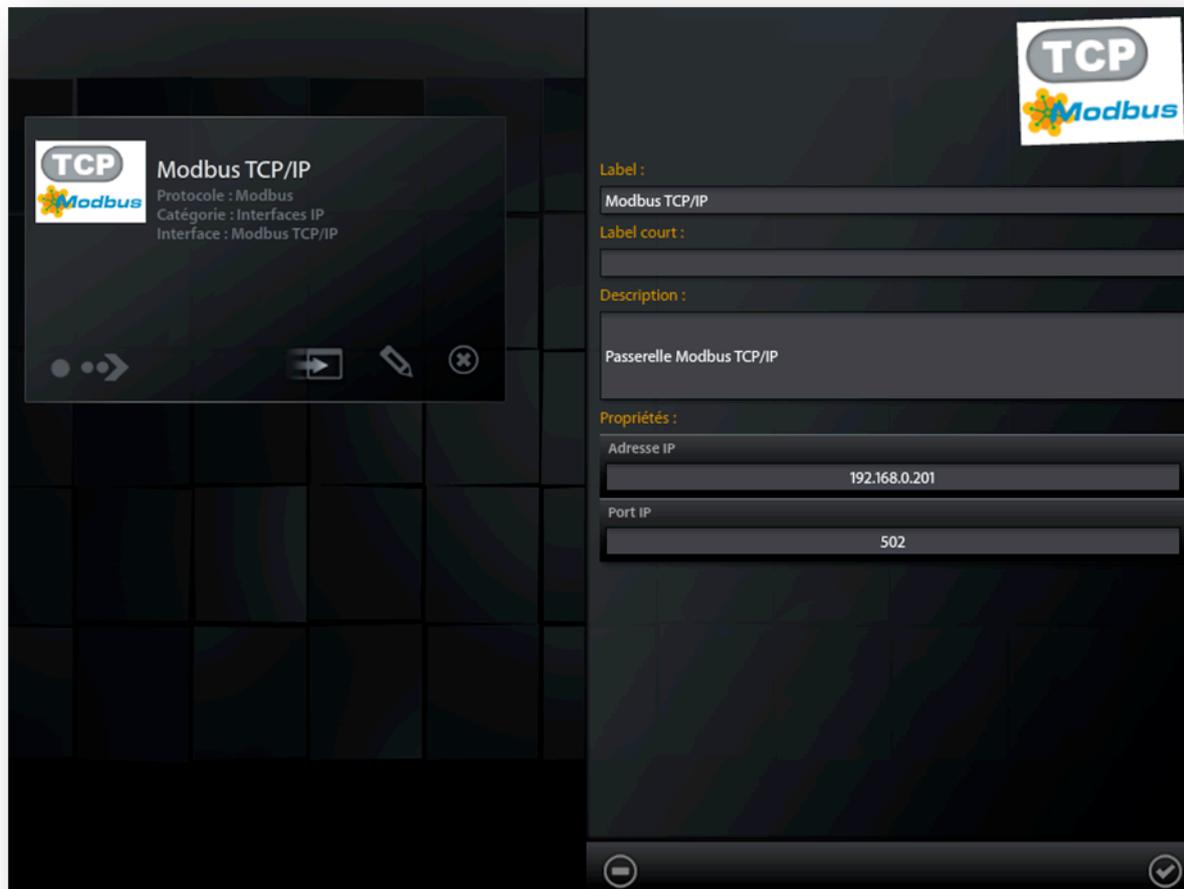
### 2.1 Adding a Modbus TCP connector

A 'Modbus TCP/IP' connector can be added in the same way as any other connector.

### 2.2 Configuring a Modbus TCP connector

The configuration elements for a Modbus TCP are as follows:

- IP address: The IP address of the Modbus TCP gateway
- IP port: The port of the Modbus TCP gateway (by default: 502)

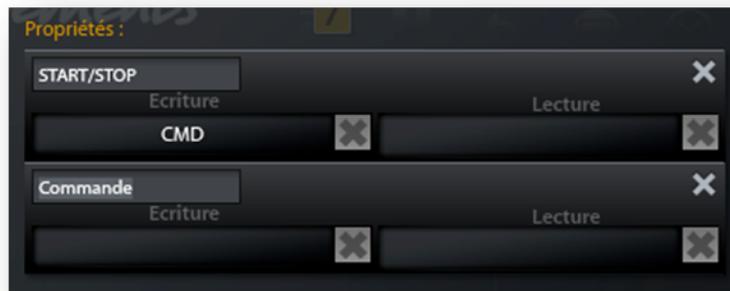


### 3 Modbus device

A Modbus connector exclusively operates with 'Universal' Lifemodus devices. A '(Generic) device' must be created and associated with the previously created Modbus connector.

#### 3.1 Device properties

To add a property to the new device, use the  icon in the bottom right hand corner. A property will read and write data in one or more registers of the Modbus slave.



##### 3.1.1 Configuring a property's 'read' field

When clicking on a property's 'read' field, a new window giving you access to the created variants appears. You can create new variables.

Select a **variable** and confirm the window to associate the property's read field with this variable. A state feedback associated with the value of this variable then becomes accessible in Lifemodus.

*Variable configuration procedure and details are covered in the [Device variables](#) section in this document*

##### 3.1.2 Configuring a property's 'write' field

When clicking on a property's 'write' field, a new window giving you access to the created variants and commands appears.

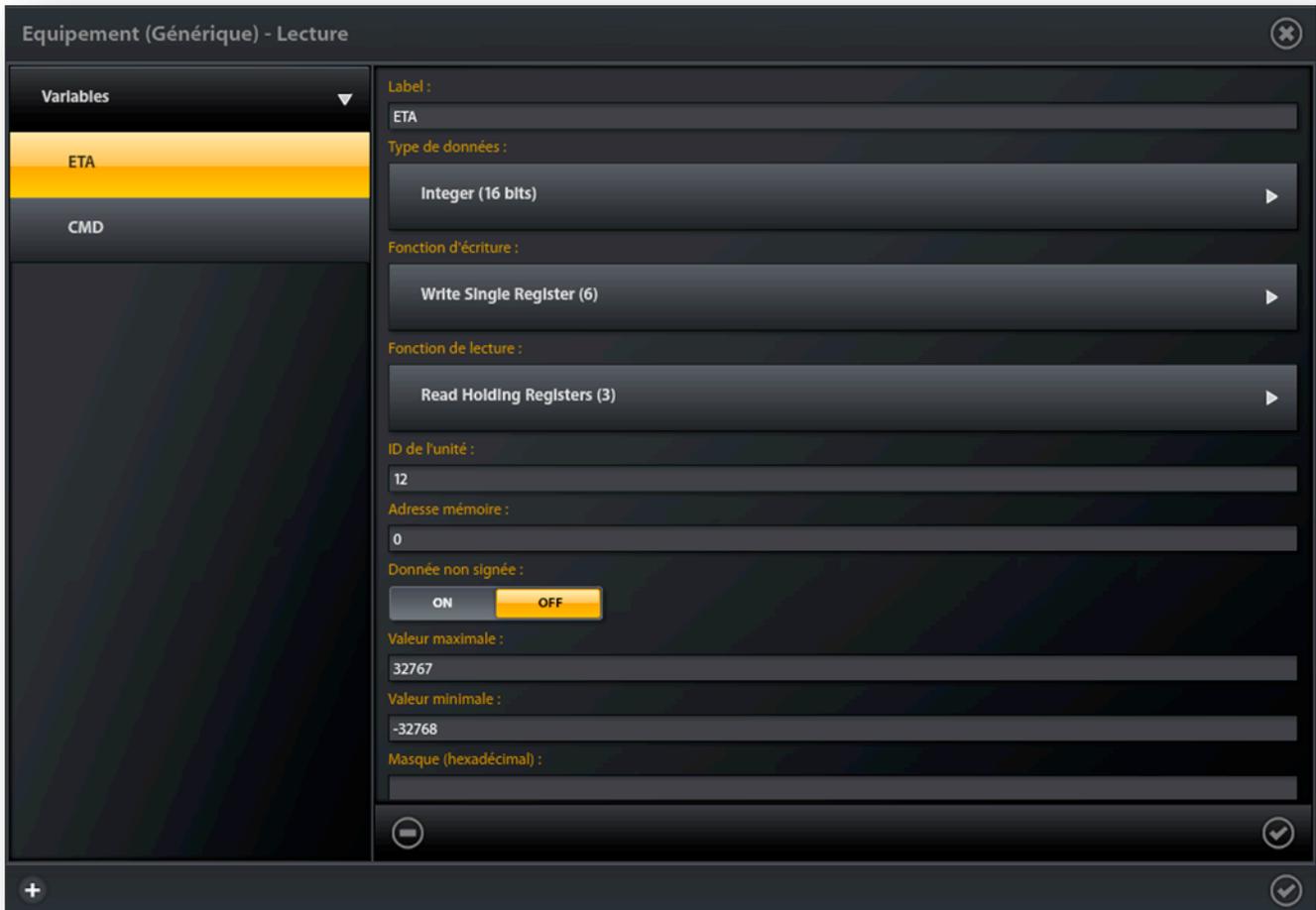
Select a **variable** and confirm the window to associate the property's write field with this variable. An action allowing you to write in the variable now becomes available in Lifemodus.

Select a **command** and confirm the window to associate the property's write field with this command. An action that executes the variable now becomes available in Lifemodus.

*Command configuration procedure and details are covered in the [Device commands](#) section in this document*

## 3.2 Device variables

A Modbus variable is a data item in Lifemodus representing a memory zone (one or several registers) of a bus slave.



From the variable management screen shown above, you can add, configure and delete Modbus variables. The new variables are specific to the device being edited.

### 3.2.1 Variable management

You can add variables by selecting the 'Variables' list and clicking on  (bottom left).

With the  button, you can save the changes to the variable and its association with the previously selected property (read or write).

With the  button, you can delete the selected variable.

### 3.2.2 Configuring a variable

Variables are configured by selecting them from the variable list. Use the configuration items to accurately determine the way in which the memory zones are read and interpreted.

- **Label:** The label is simply used to add a title to the variable.
- **Data type:** The type of data you want to use in Lifemodus: boolean, integer, floating point (IEEE 754) or hexadecimal value. 16 bit values or less are read and written from a single register, whereas 32 bit values are read or written from two consecutive registers.
- The **read function** and **write function** are Modbus functions that read and write the memory zones associated with this variable from the bus. These functions are generally covered in the manufacturer's documentation.
- **Unit ID** or 'UnitID' (1 to 255): This field is specific to the Modbus TCP protocol and identifies a slave on the bus.
- **Memory address** (1 to 65535): The memory address is the address of the register you want to read or write. When a 32 bit data type is selected, the read and written registers are those indicated in the 'memory address' field and the following register.  
Example: an Integer (32bits) read or written on the 2202 memory address will actually be read or written on the 2202 and 2203 addresses.
- **Unsigned data:** When this field is active, the data will be considered as unsigned, namely a positive value. If you change the value, this will update the fields 'Minimum value' and 'Maximum value'.
- **Minimum and maximum value:** These values are defined by default depending on the selected '**data type**' and on whether or not '**signed data**' is enabled. Examples:
  - A Boolean value is 0 (false) or 1 (true)
  - A signed Integer (16 bits) has a value between -32768 and 32767
  - An unsigned Integer (16 bits) has a value between 0 and 65535

The default minimum and maximum values are those read and written in Lifemodus. Though these limit values cannot be exceeded, they can be restricted.

- **Mask (hexadecimal):** This field allocates a mask to the registers. It must have a hexadecimal format.  
Example: FFF7, 0xFFF7 or 16#FFF7
  - **Read:** To determine the final value available in Lifemodus, a logic 'AND' is entered between the mask and the value read on the bus.
  - **Write:** Only bits on '1' in the mask will be changed in the target registers.

Using the mask adds great power when you use data.

*Practical examples are available in the [Mask examples and specific features](#) section in this document.*

- **Invert bytes:** The manufacturer's documentation usually specifies whether this parameter must be used. It is used to invert the heavy byte and the lightweight byte when reading and writing the register. A register has 16 bits, i.e. 2 bytes. If a register's value is 0x7F00, inverting the bytes will read: 0x007F.
- **Invert words:** A word has 16 bits (i.e. the size of a register). Inverting words is useful when reading and writing several registers, when the selected data type is Integer (32 bits) or Float (32 bits).  
Example: Registers no.2202 and no.2203 are used to read and write a 32 bit integer. Without word inversion, register no.2202 contains heavy bits and the sign bit (if the value is signed) whereas register no.2203 contains lightweight bits. With word inversion, register no.2202 contains lightweight bits and register no.2203 contains the heavy bits and sign bit.
- **Coefficient:** This field is a multiplying coefficient applied to the value read from the bus (divided when writing).  
Read: 'Lifemodus value' = 'Bus value' x 'Coefficient'  
Write: 'Bus value' = 'Lifemodus value' / 'Coefficient'
- **Unit:** This optional field is used for selecting the value unit in Lifemodus.

## Mask examples and distinctive features

Using the mask, you can select some bits in the register when reading and writing from the bus. It can be used for Integer (16 or 32 bits), Hexadecimal or Boolean type data.

The hexadecimal and binary values are used in the following examples to help you understand how to use the mask when reading and writing from the bus.

### 3.2.2.1 Using the mask with 'integer' or 'hexadecimal' values in Lifedomus

#### Example:

- Register #3303 contains the following value: **0x12FF** (0001 0010 1111 1111).
- The mask is: **0x007F** (0000 0000 0111 1111).
- Reading the variable will produce a state feedback value of 'mask' AND 'bus value':  
**0x007F** (0000 0000 0111 1111) AND **0x12FF** (0001 0010 1111 1111) = **0x007F** (0000 0000 0111 1111)
- Writing the value **0x0000** (0000 0000 0000 0000) on the variable will produce the writing of the value:  
**0x1280** (0001 0010 1000 0000) on the bus:
- A new reading of the variable will produce the state feedback value:  
**0x007F** (0000 0000 0111 1111) AND **0x1280** (0001 0010 1000 0000) = **0x0000** (0000 0000 0000 0000)
- Writing the value **0x0F22** (0000 1111 0010 0010) on the variable will produce the writing of:  
**0x12A2** (0001 0010 1010 0010) on the bus:

### 3.2.2.2 Using the mask with 'boolean' values in Lifedomus

Use of the mask with a boolean type value can be common in Modbus. In fact, several boolean values 'TRUE/FALSE', 'Enabled/Disabled', ... are often saved in the same register. Each bit represents a boolean value ('TRUE' if the bits in question are '1', otherwise 'FALSE').

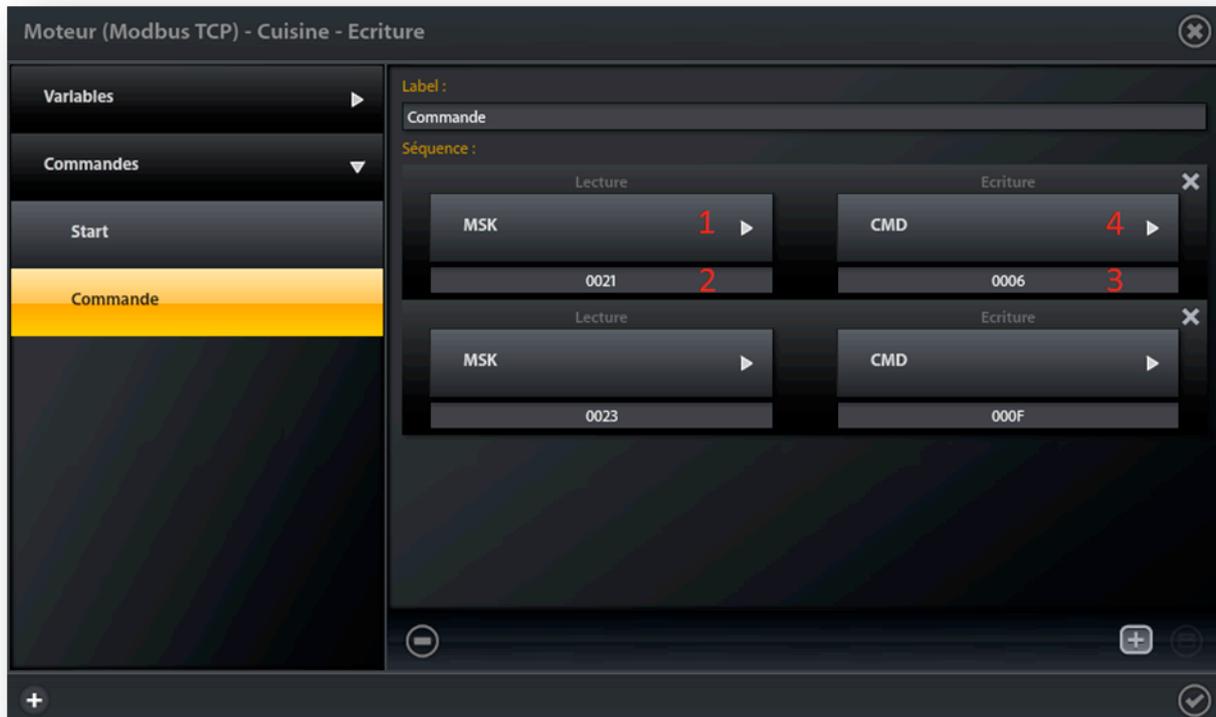
#### Example:

- Register #3303 contains the following value: **0x12FF** (0001 0010 1111 1111).
- The mask is: **0x0080** (0000 0000 1000 0000)
- **0x0080** (0000 0000 1000 0000) AND **0x12FF** (0001 0010 1111 1111) = **0x0080** (0000 0000 1000 0000)  
As the result is the same as the mask, reading the variable will produce the following state feedback value: **TRUE**
- Writing the **FALSE** value will write on the bus: 'invert mask' AND 'bus value'  
**0x0F22** (1111 1111 0111 1111) AND **0x12FF** (0001 0010 1111 1111) = **0x127F** (0001 0010 0111 1111)
- **0x0080** (0000 0000 1000 0000) AND **0x127F** (0001 0010 0111 1111) = **0x0080** (0000 0000 0000 0000)  
As the result is different to the mask, reading the variable will produce the following value: **FALSE**
- Writing the **TRUE** value will write on the bus: 'mask' AND 'bus value'  
**0x0080** (0000 0000 1000 0000) AND **0x127F** (0001 0010 0111 1111) = **0x12FF** (0001 0010 1111 1111)

### 3.3 Device commands

The command management screen adds, configures and deletes Modbus commands.

Like with variables these commands are specific to the device being edited.



Each item in the sequence of a command is executed in the sequence entered and works as follows: If the left hand variable (1) equals the value indicated on the left (2), then the value indicated on the right (3) is written in the right hand variable (4). Note that if this read field (2) is empty, the condition will be validated and writing will be executed.

Example: These commands are used in some cases such as starting a motor: If the motor is switched off, it is switched on, if it is stopped it is started, then its speed is determined.

## 4 Creating the 'Design Studio' interface

As Modbus devices are generic devices in Lifemodus, there is no default widget in Design Studio. The edit mode is therefore required to build the supervision interface.

*Refer to 'Design Studio's 'What I See' and 'What I Do' documentations*