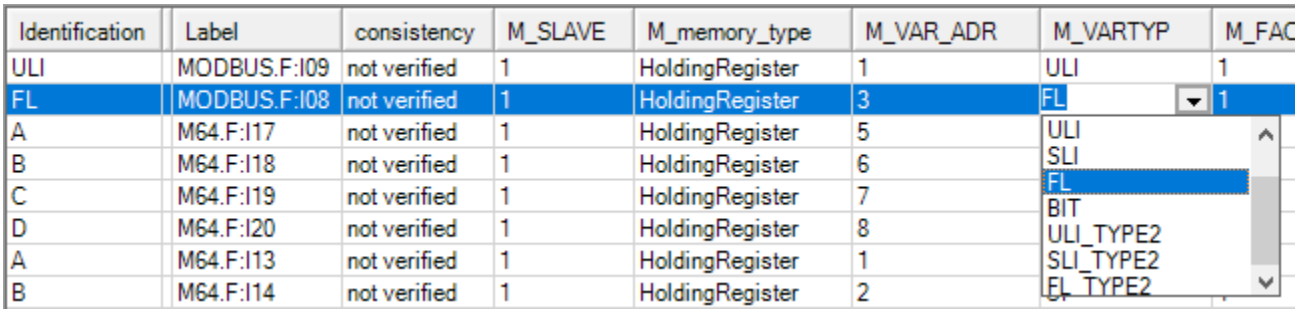
TT211004 – FUP - Modbus 32 Bit Register

1. Modbus 32-bit register are supported natively in our system. The types that we support are ULI, SLI and FL. We implement 32-bit floating point data using the IEEE-754 standard.



Identification	Label	consistency	M_SLAVE	M_memory_type	M_VAR_ADR	M_VARTYP	M_FAC
ULI	MODBUS.F:I09	not verified	1	HoldingRegister	1	ULI	1
FL	MODBUS.F:I08	not verified	1	HoldingRegister	3	FL	1
A	M64.F:I17	not verified	1	HoldingRegister	5		
B	M64.F:I18	not verified	1	HoldingRegister	6		
C	M64.F:I19	not verified	1	HoldingRegister	7		
D	M64.F:I20	not verified	1	HoldingRegister	8		
A	M64.F:I13	not verified	1	HoldingRegister	1		
B	M64.F:I14	not verified	1	HoldingRegister	2		

2. The Modbus protocol was designed to operate with devices of 16-bit register length (e.g. UI and SI, 2 bytes of data). Special considerations are required when implementing 32-bit data elements, or 4 bytes of data.
3. First is the address (M_VAR_ADR). All 32-bit register Modbus data use 2 addresses. In the example below, the first register “ULI” use address 1 and 2, and the second register “FL” use address 3 and 4.

Identification	Label	consistency	M_SLAVE	M_memory_type	M_VAR_ADR	M_VARTYP
ULI	MODBUS.F:I09	not verified	1	HoldingRegister	1	ULI
FL	MODBUS.F:I08	not verified	1	HoldingRegister	3	FL

4. For long integers ‘Unsigned INT32’, this require 32 bits, and are implemented as two consecutive 16-bit registers. The range is 0 to 4294967295, which is called ‘unsigned INT32’. For this, we use “ULI” in “M_VARTYP”.
5. Alternatively negative values can be stored if the instrument is defined that way, and is then called ‘INT32’ which has the range -2147483648 to 2147483647. We use “SLI” in “M_VARTYP” for this type.
6. Single precision floating point values (binary32) are defined by 32 bits (4 bytes), and are implemented as two consecutive 16-bit registers. For this, we use “FL” in “M_VARTYP”.
7. Please refer to the device Modbus datasheet from the manufacture for details. Make sure you use the correct type and address for the integration. If they’re not clear in the datasheet, you can use 3rd party Modbus software (e.g. Modbus Poll) to check for the correct settings.
8. Please also note that our Modbus address (M_VAR_ADR) starts from 0. Many Modbus devices in the market use address that that start from 1. If you can’t get correct readings with the address specify in the datasheet, you can try using “Address – 1” in “M_VAR_ADR”.
9. The second thing is the order of the 4 bytes. Please read the following description in our manual.

Modbus data byte format

The Modbus variable types FL, ULI and SLI take up more than one register. For Modbus variable types that take up more than one individual register, the Modbus protocol does not standardize the order in which the data bytes must be saved in the registers. This decision is up to the manufacturer of the respective device.

The number 1 billion (10^9) is saved in the memory as a hexadecimal 0x3B9ACA00. It can be entered in the Modbus register as [0x3B9A][0xCA00] (standard) or also [0xCA00][0x3B9A] (type2).

The corresponding documentation specifies the data format in which these Modbus variable types are saved for the devices to be connected. If no information is provided about this, you have to check the data format yourself by checking the plausibility of the read out data. If necessary, the data format must be adjusted in the system integration (FL_TYPE2, ULI_TYPE2 or SLI_TYPE2).

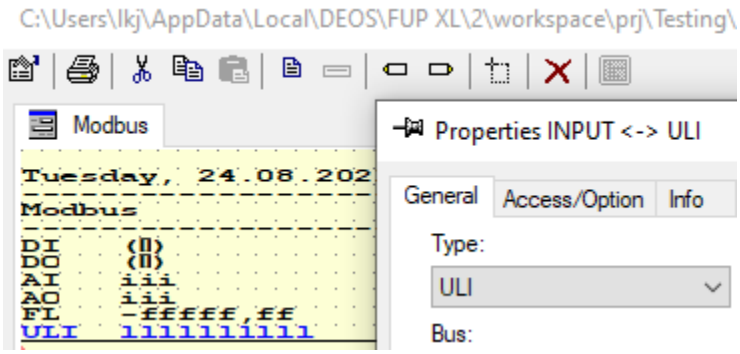
10. Modbus itself does not define floating-point data types, but it is widely accepted that it implements 32-bit floating-point data using the IEEE-754 standard. However, the IEEE standard has no clear definition of the byte order. Therefore, the most important consideration when dealing with 32-bit data is that the data be addressed in the correct order. This is the same for 32-but integer.
11. The following table shows two adjacent 16-bit registers conversion to a 32-bit floating point or 32-bit integer values.

Function Keyword	Swap Mode	Source Bytes	Target Bytes
2.i16-1.i32	N/A	[ab][cd]	[abcd]
2.i16-1.i32-s	byte and word swap	[ab][cd]	[dcba]
2.i16-1.i32-sb	byte swap	[ab][cd]	[badc]
2.i16-1.i32-sw	word swap	[ab][cd]	[cdab]

12. In our system, we have a selection for the order, which is called FL_TYPE2, ULI_TYPE2 and SLI_TYPE2.

Identification	Label	consistency	M_SLAVE	M_memory_type	M_VAR_ADR	M_VARTYP
ULI	MODBUS.F:I09	not verified	1	HoldingRegister	12	ULI_TYPE2
FL	MODBUS.F:I08	not verified	1	HoldingRegister	10	FL_TYPE2

13. In most of the Modbus device datasheet, it is always not clear of what the order is for the 32-bit registers. So you must try it out using 3rd party software (e.g. Modbus Poll) if you're not sure which type should be used for the integration.
14. If you still can't figure it out correctly, you can try to use "TYPE2" first in FUP and test it out in the controller. It it's not correct, then try without "TYPE2".
15. Finally is the type we use in the FUP page. Most of the time, we use the same type in both the FUP page and in Modbus integration. For example, we use type "ULI" in the FUP page for the point "ULI".



16. But sometimes we need to use different types. For example, the below power meter use UI and ULI in the Modbus registers.

Identification	Label	consistency	M_SLAVE	M_memory_type	M_VAR_ADR	M_VARTYP	M_FACTOR
kWh	MODBUS.F:I20	not verified	41:192.168.170.253:502	HoldingRegister	0	ULI_TYPE2	0.1
Voltage	MODBUS.F:I07	not verified	41:192.168.170.253:502	HoldingRegister	2	UI	0.01
Current	MODBUS.F:I18	not verified	41:192.168.170.253:502	HoldingRegister	3	ULI_TYPE2	0.001

17. But since we use a factor of less than 1 in "M_FACTOR" (e.g 0.1), now it's converted to a floating number, so we have to use type "FL" in the FUP page to see the decimal places.