Weidmüller

UR20-FBC-MOD/ u-remote

Quick Start Guide for Modbus- Understanding Coils and Register

Abstract:
The differences between coils and registers repeatedly causes misunderstandings and errors when using Modbus communication. In this Quick Start Guide we want to illustrate the differences and special features using Modbus with u-remote.

Hardware reference

No.	Component name	Article No.	Hardware / Firmware version
1	UC20-WL2000-IOT	1334990000	HW 01.23.00 / FW 1.5.0
2	UR20-FBC-MOD-TCP	1334930000	HW 01.xx.xx / FW 2.0.7
3	UR20-4DO-P	1315220000	-
4	UR20-8DO-P	1315240000	-
5	UR20-16DO-P	1315250000	-

Software reference

No.	Software name	Article No.	Software version
1	Node-RED	-	0.18.7

File reference

No.	Name	Description	Version
1	-	-	-

Contact
Weidmüller Interface GmbH & Co. KG
Klingenbergstraße 26
32758 Detmold, Germany
www.weidmueller.com

For any further support please contact your local sales representative:
https://www.weidmueller.com/countries

Content

1 Warning and Disclaimer

Warning
Controls may fail in unsafe operating conditions, causing uncontrolled operation of the controlled devices. Such hazardous events can result in death and / or serious injury and / or property damage. Therefore, there must be safety equipment provided / electrical safety design or other redundant safety features that are independent from the automation system.

Disclaimer
This Application Note / Quick Start Guide / Example Program does not relieve you of the obligation to handle it safely during use, installation, operation and maintenance. Each user is responsible for the correct operation of his control system. By using this Application Note / Quick Start Guide / Example Program prepared by Weidmüller, you accept that Weidmüller cannot be held liable for any damage to property and / or personal injury that may occur because of the use.

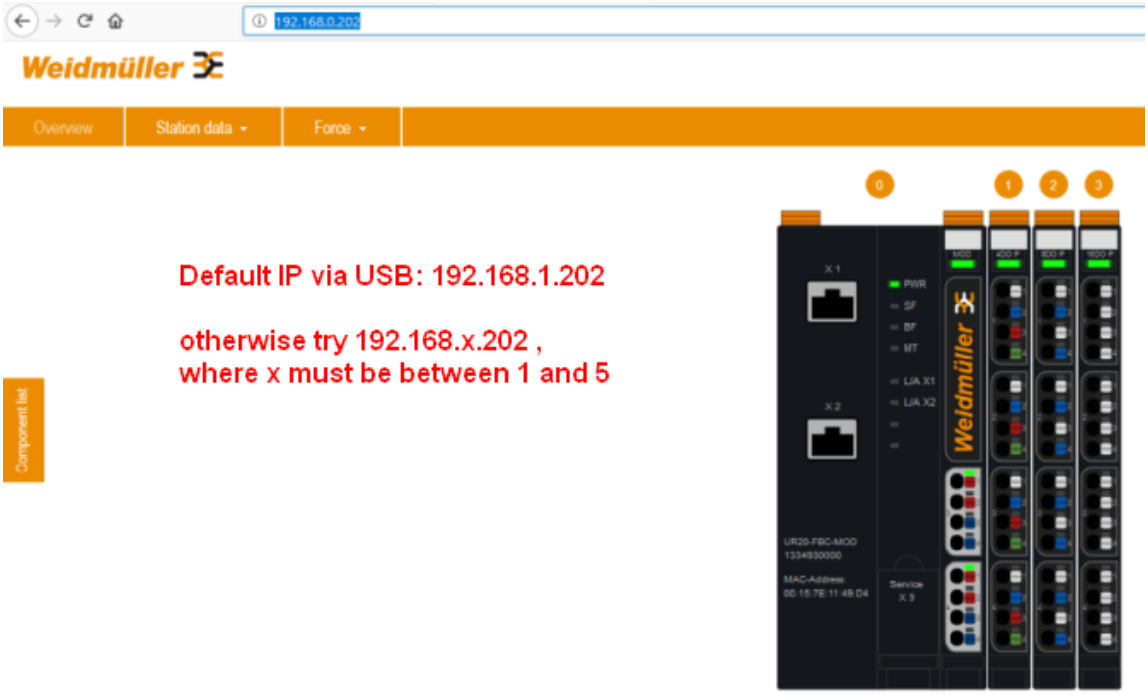
Note
The given descriptions and examples do not represent any customer-specific solutions, they are simply intended to help for typical tasks. The user is responsible for the proper operation of the described products. Application notes / Quick Start Guides / Example Programs are not binding and do not claim to be complete in terms of configuration as well as any contingencies. By using this Application Note / Quick Start Guide / Example Program, you acknowledge that we cannot be held liable for any damages beyond the described liability regime. We reserve the right to make changes to this application note / quick start guide / example at any time without notice. In case of discrepancies between the proposals Application Notes / Quick Start Guides / Program Examples and other Weidmüller publications, like manuals, such contents have always more priority to the examples. We assume no liability for the information contained in this document. Our liability, for whatever legal reason, for damages caused using the examples, instructions, programs, project planning and performance data, etc. described in this Application Note / Quick Start Guide / Example is excluded.

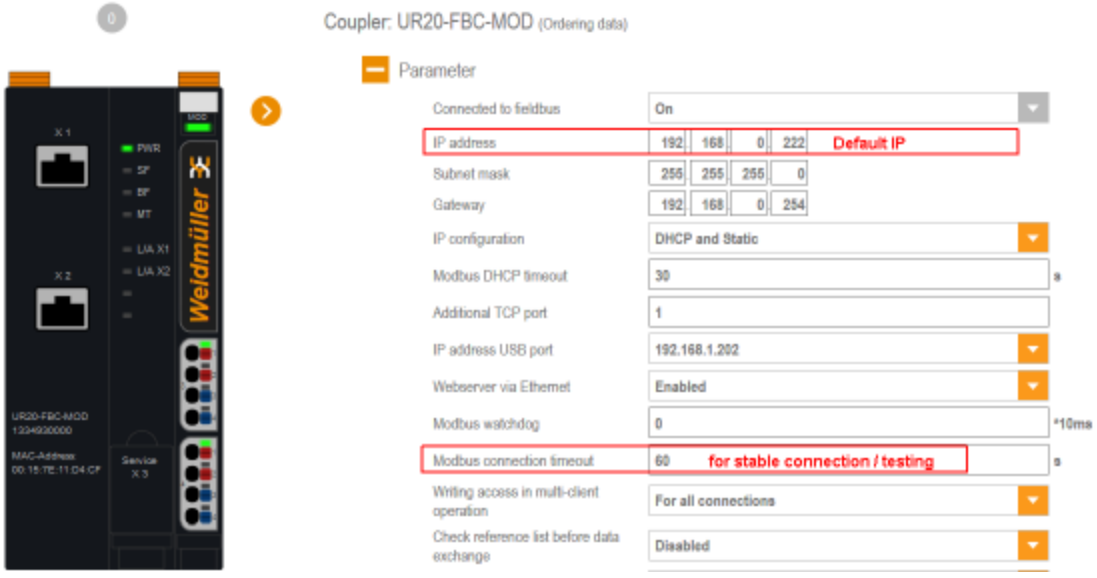
Security notes
In order to protect equipment, systems, machines and networks against cyber threats, it is necessary to implement (and maintain) a complete state-of-the-art industrial security concept. The customer is responsible for preventing unauthorized access to his equipment, systems, machines and networks. Systems, machines and components should only be connected to the corporate network or the Internet if necessary and appropriate safeguards (such as firewalls and network segmentation) have been taken.

2 Connection to Modbus Coupler

In order to read and write no further configuration on Modbus coupler is needed. The purpose of accessing this interface through web browser is to check the results of writing and reading → Monitoring.
Attention: For understanding it is not necessary to have a test setup, we only use this to take screenshots in order to illustrate commands and results.

- Default IP via USB : **192.168.1.202**

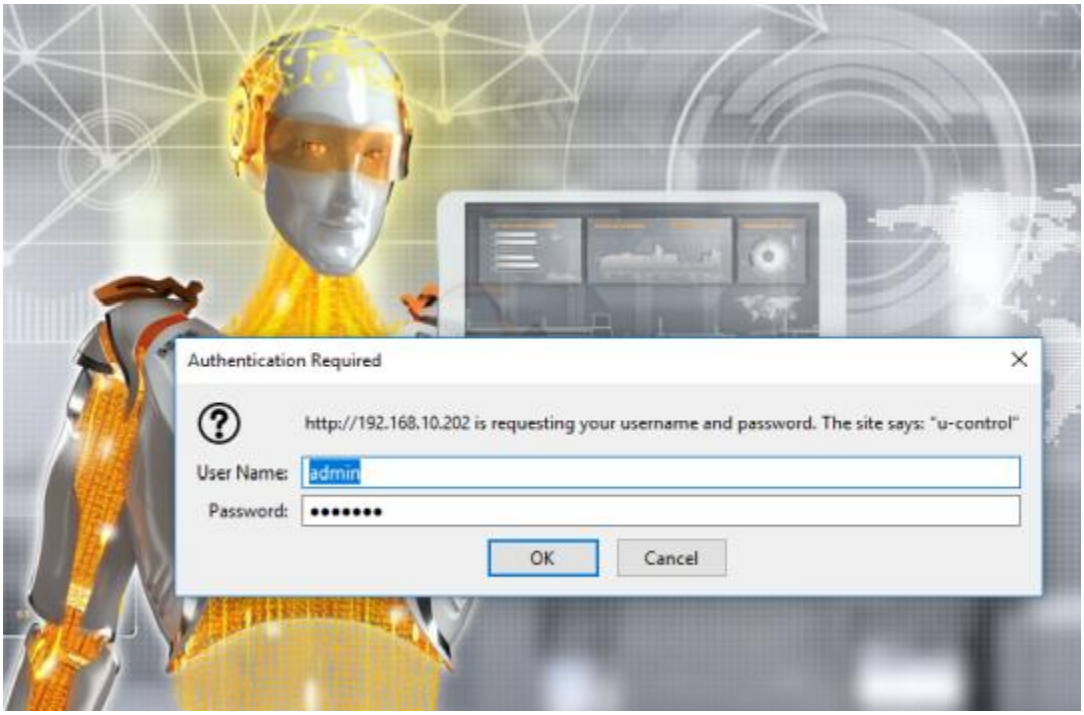




3 Configuring with IOT Controller

1) Access the IOT web server or configuration interface through any browser.

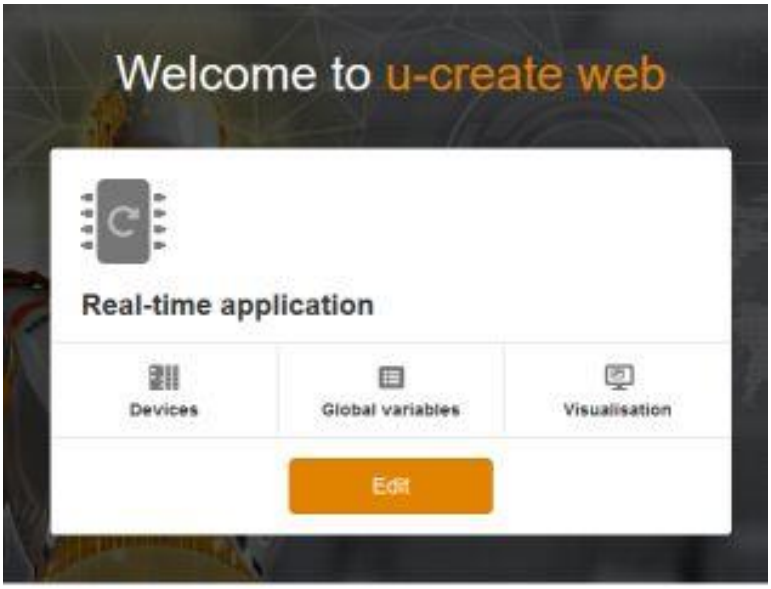
- Default IP via USB: **192.168.10.202**



Maybe you must first login with the following details:
User Name: admin
Password: Detmold

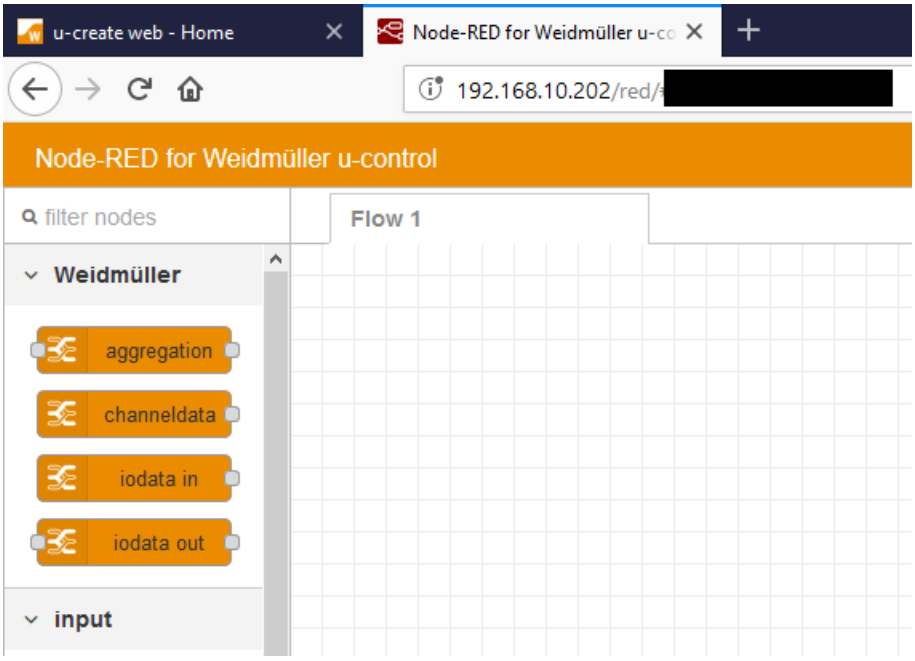
Since software version 1.2.0 web-interface at UC20-WL2000-IOT and UC20-WL2000-AC have the same design.
Differences are in the functionalities which are enabled depending on the controller type.

2) Start the Node-RED by clicking on Open application.



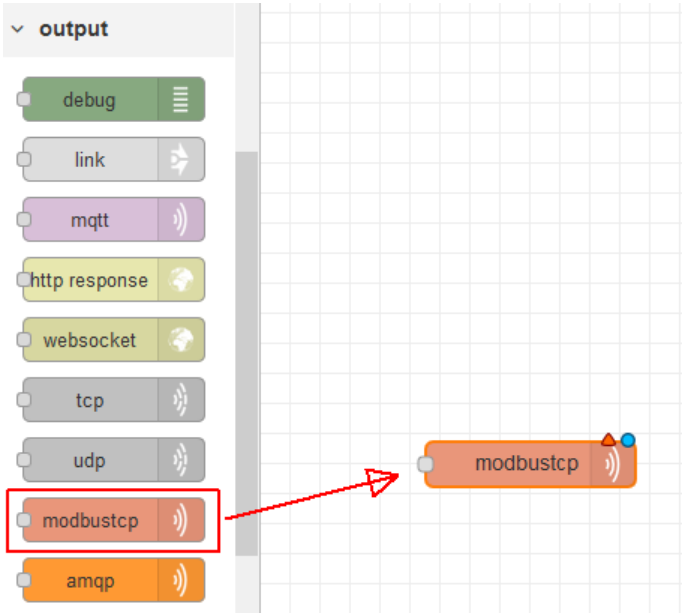
In order to do some Modbus communication, it is necessary to install the Modbus/TCP nodes.
If not done yet, you have to add them first.
This is not the part of this Quick Start Guide; you may refer Application Note AN0009 for more information.

3) Access the IOT web server and open the Node-RED application again.

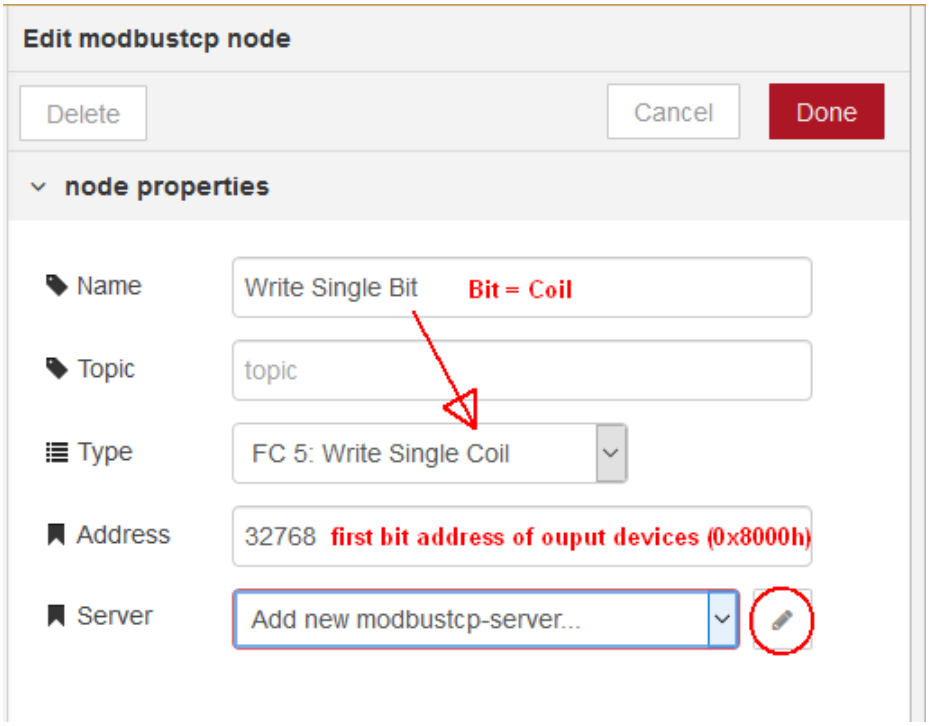


3.1 Modbus TCP Write

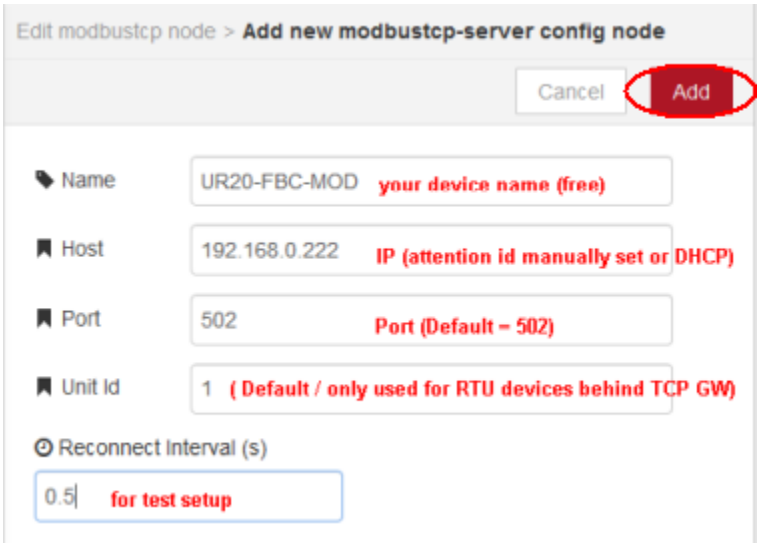
The node used for writing data to the Modbus Coupler is output -> modbustcp.



1) The modbustcp node must be configured (double-click) as shown on the Image:

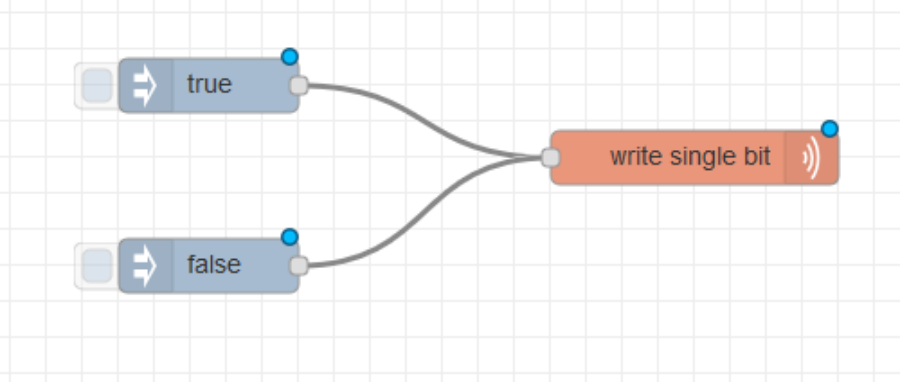


2) Once you click the edit button you can fill the server settings:

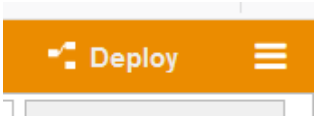


3.2 Write Single Coil

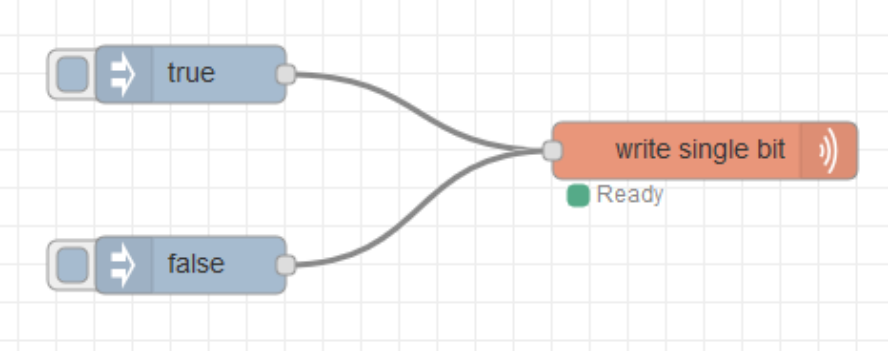
1) Drag connections between the nodes and your example should look like this.



2) Deploy your changes to the controller.



Once the changes are deployed the node shows a green little square indicating the connection to the Modbus server.



At this point it is possible to write a Single Coil.

3) Check web interface of Modbus/TCP Coupler via USB or LAN.

If it looks like this, then everything is fine.

4) Let us have a look at the Process data in the web interface.

Process data bit addresses in our test setup

Channels	1 4DO-P	2 8DO-P	3 16DO-P
0	1 32768	0 32776	0 32784
1	0 32769	0 32777	0 32785
2	0 32770	0 32778	0 32786
3	0 32771	0 32779	0 32787
4	32772	0 32780	0 32788
5	32773	0 32781	0 32789
6	32774	0 32782	0 32790
7	32775	0 32783	0 32791
8			0 32792
9			0 32793
10			0 32794
11			0 32795
12			0 32796
13			0 32797
14			0 32798
15			0 32799

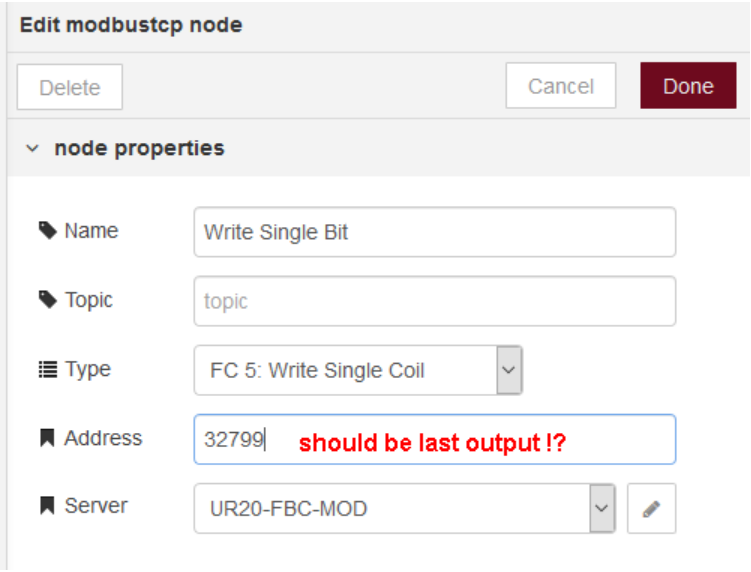
As you can see, the 4DO module also have 8 bits reserved because 1 Byte is the minimum which is allocated for this module.

For use with function codes 1, 2, 5, 15 (bit addressing) we are talking about packed process data.

Packed process data for outputs,
- begins at 0x8000h (32768dez).
- data width is module dependent.
- byte granularly.

There is no “unpacked” process data for bit addressing!
Keep this in mind!

With different module arrangement, e.g. the 4DO also reserves 16 bits. See explanation at multiple register chapter for further information.

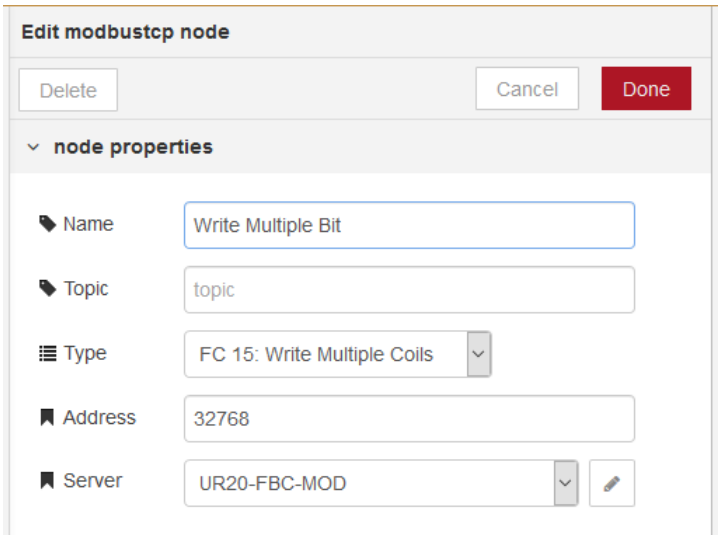


If it looks like this, then everything is fine.

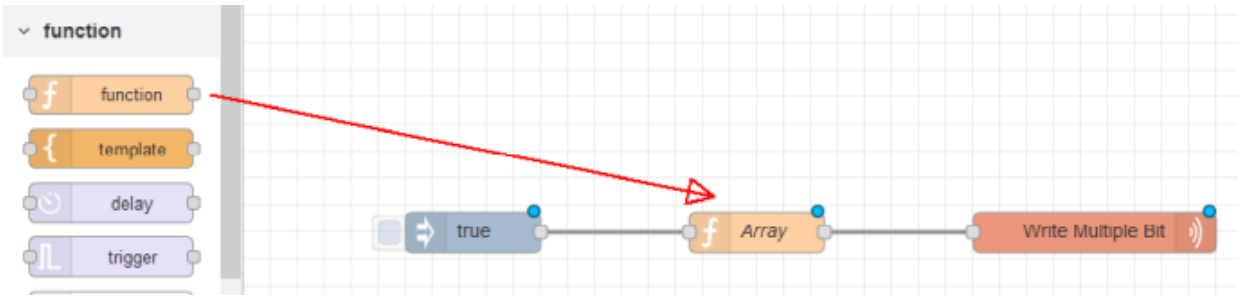
3.3 Write Multiple Coils

This type of writing is similar to single coil but writes several coils at once.

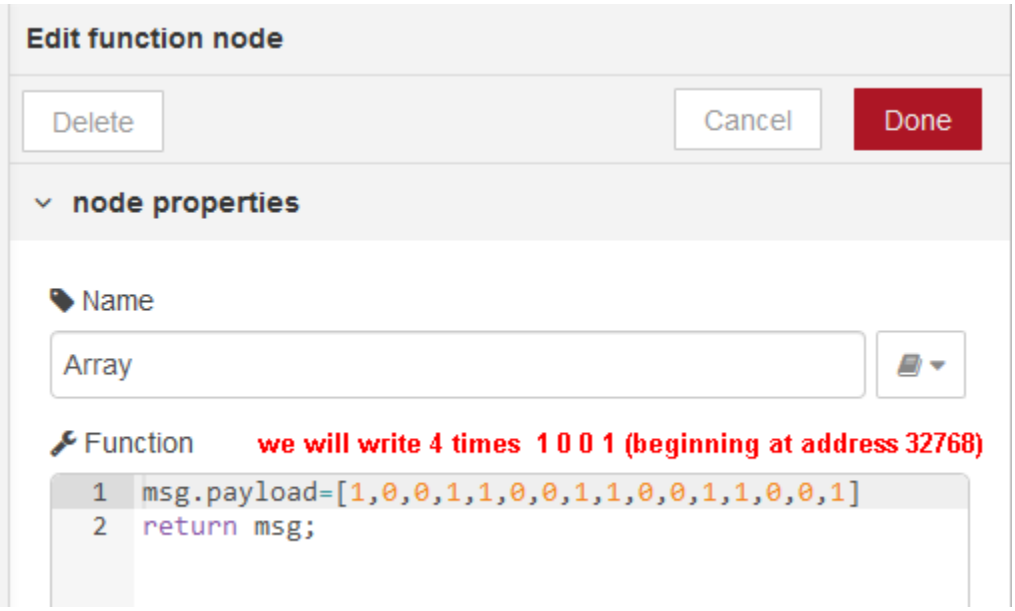
1) We start at the same address.

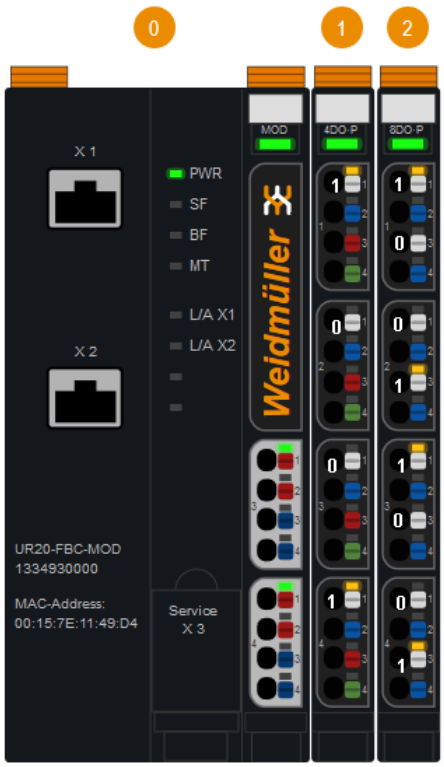


2) The inject node should be an array of binary values.



The array function only sends an array payload.





Process data

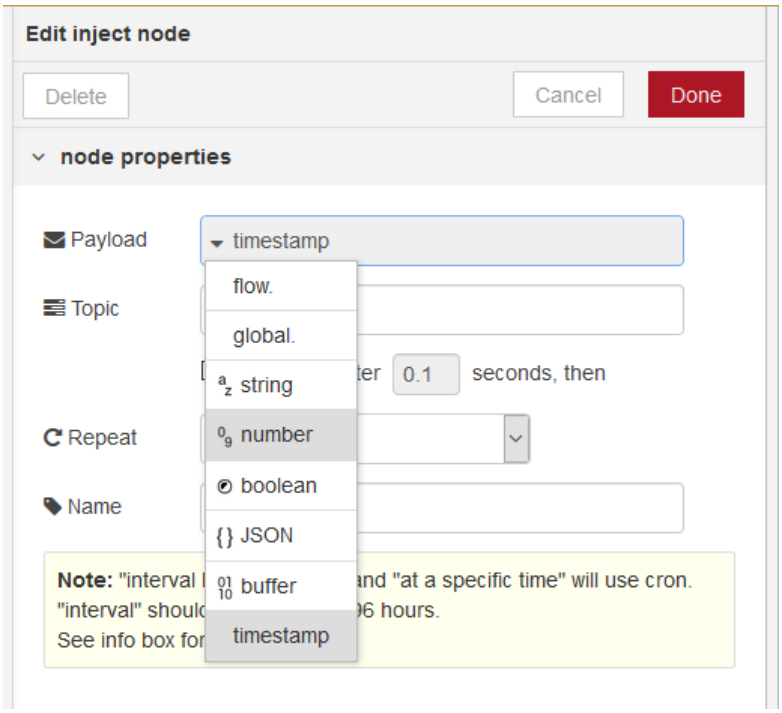
Channels	1 4DO-P	2 8DO-P	3 16DO-P
0	1	1	0
1	0	0	0
2	0	0	0
3	1	1	0
4		1	0
5		0	0
6		0	0
7		1	0
8			0
9			0
10			0
11			0
12			0
13			0
14			0
15			0

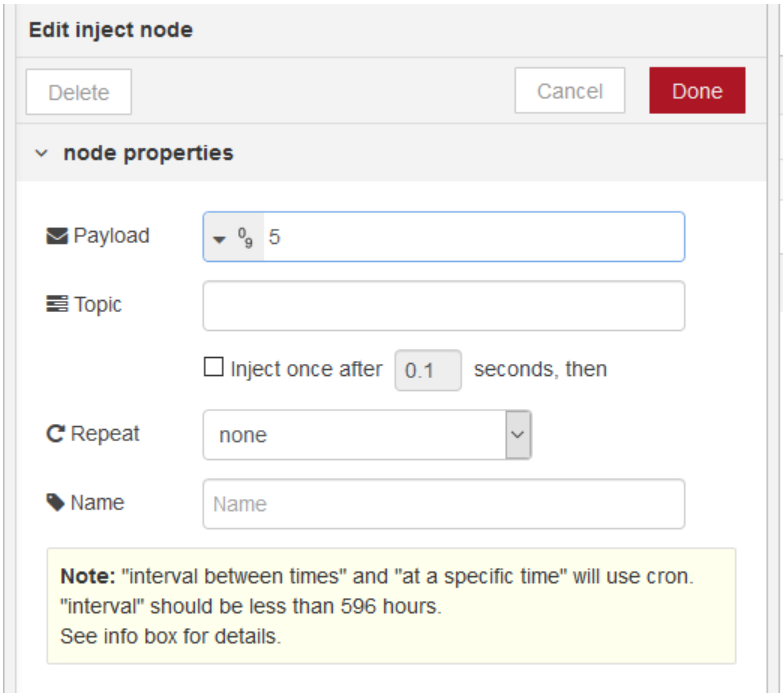
1.
2.
3.
4.

ignored, because
- 4DO has only 4 outputs
- but reserves 8 Bits

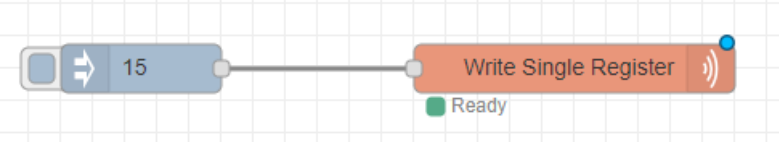
3.4 Write Single Holding Register

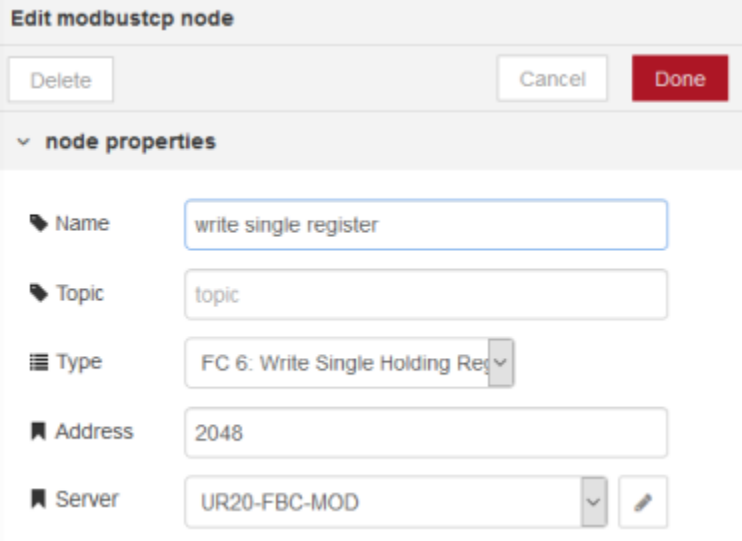
For this data type the inject must be a number.





The type set to the modbus node is Writing Single Holding Register.





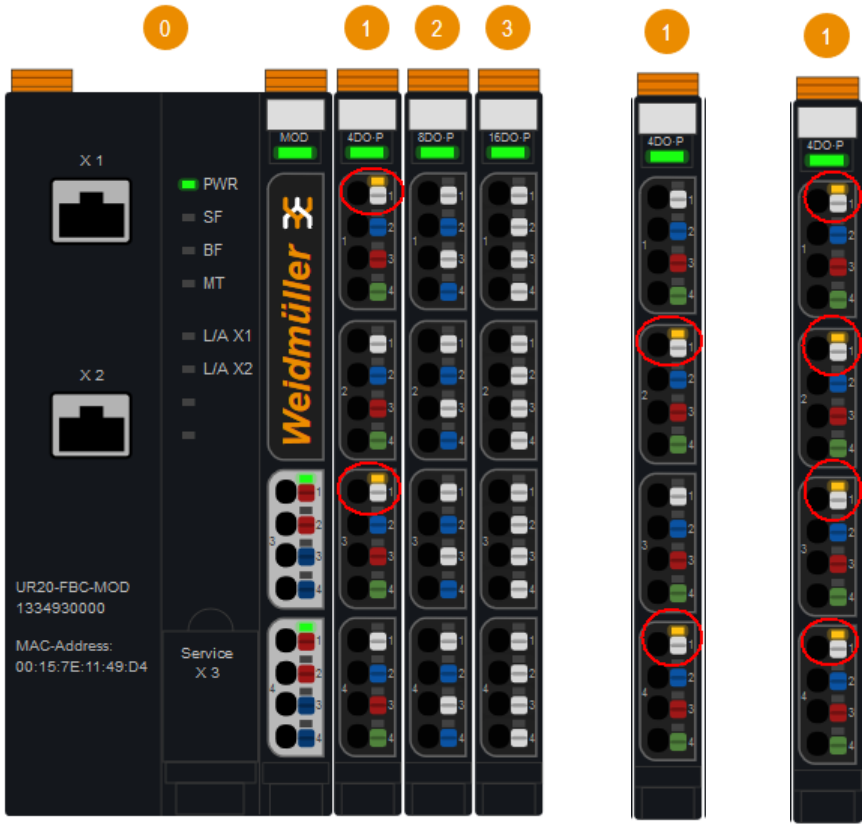
Register addresses are different to bit addresses.

Packed process data for outputs,
- begins at 0x0800h (2048dez).
- data width is module dependent.
- byte granularly.

Unpacked process data for outputs,
- begins at 0x9000h (36864dez).
- data width is module dependent.
- per module are 32 registers reserved / 0x20h offset per module.

One register has 16 bits. If a write action is performed, 16 bits are written.

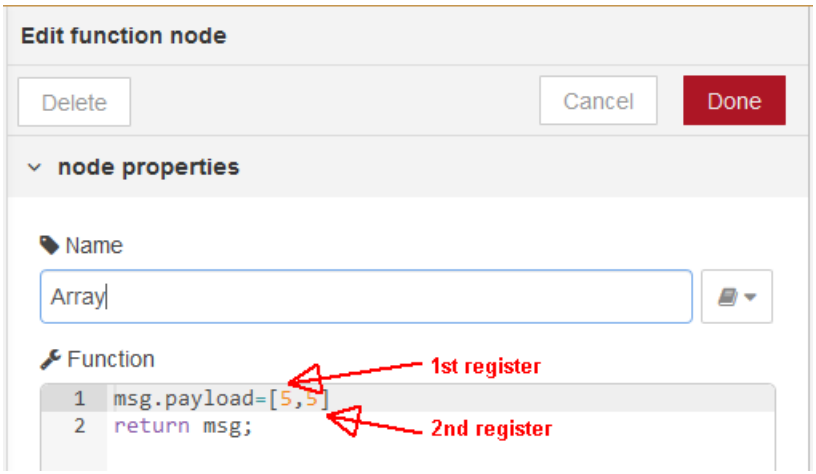
After writing a 5,
channels 0 and 2 are turned on which makes a binary 5, 1010.



Results for writing 5 10 15

3.5 Write Multiple Holding Registers

This case also writes holding registers passed through an array:



Keep in mind that we are talking about registers.
The first array value is set to the first register, the next values of the array to the adjacent registers of the same type.

But we are using the register addresses for packed process data, so it depends on the module arrangement also.

How would the result look like if we change the arrangement of our DO-modules?

Example of an optimal module arrangement

Product	Input data	Output data	Number of input registers	Number of output registers	Remarks
UR20-FBC-MOD	–	–	0	0	
UR20-4AO-UI	–	8 Byte	0	4	allocates 4 registers
UR20-4AI-UI	8 Byte	–	4	0	allocates 4 registers
UR20-4DI-P	1 Byte	–	1/2	0	allocates low byte of a register
UR20-8DI-x	1 Byte	–	1/2	0	allocates high byte of a register
UR20-16DI-x	2 Byte	–	1	0	allocates 1 register
Total			6	4	

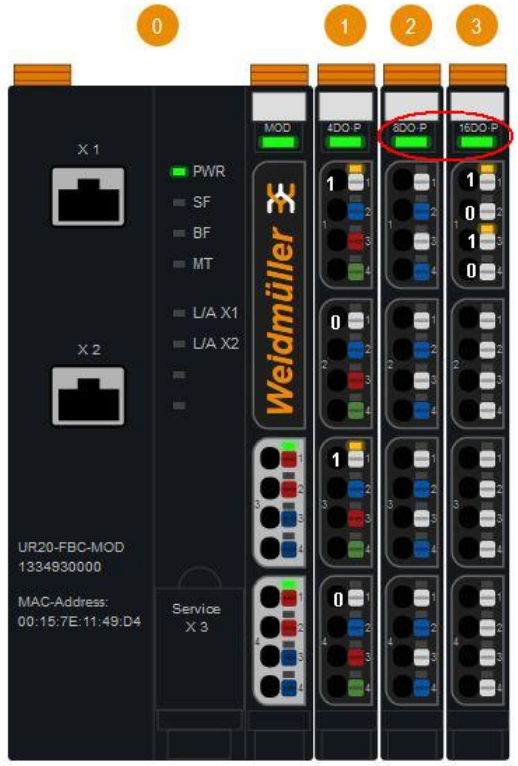
Example of a suboptimal module arrangement

Product	Input data	Output data	Number of input registers	Number of output registers	Remarks
UR20-FBC-MOD	–	–	0	0	
UR20-4DI-P	1 Byte	–	1	0	allocates 1 register
UR20-4AO-UI	–	8 Byte	0	4	allocates 4 registers
UR20-4AI-UI	8 Byte	–	4	0	allocates 4 registers
UR20-8DI-x	1 Byte	–	1	0	allocates 1 register
UR20-16DI-x	2 Byte	–	1	0	allocates 1 register
Total			7	4	

So, in one case with 8DO behind 4DO module both need 1 Byte and allocate ½ register each.
For that writing 1 register would control both modules (4DO and 8DO).
If we change 8DO and 16DO, our first 4DO will also need 1 Byte but allocate 1 register.
For that writing 1 register would only control the first 4DO module.

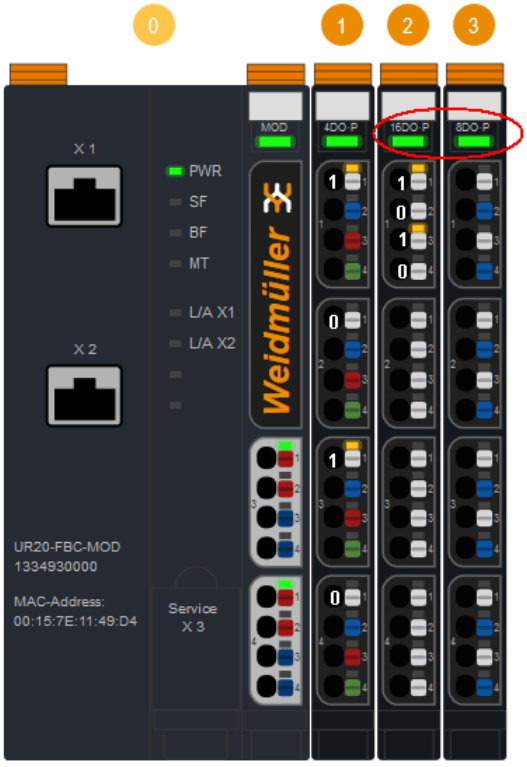
For beginners this is very difficult to understand, but very important to understand Modbus communication with u-remote.

Hint:
If you want to avoid this, use the register addresses for (unpacked) process data.
There you have a fixed offset for each module to address it directly.



Process data

Channels	8bit = 1/2 register 1 4DO-P	8bit = 1/2 register 2 8DO-P	16bit = 1 register 3 16DO-P
0	1	0	1
1	0	0	0
2	1	0	1
3	0	0	0
4		0	0
5		0	0
6		0	0
7		0	0
8	1st written register		0
9			0
10			0
11	written 5dez = 1 0 1 0 bin		0
12			0
13			0
14		2nd written register	0
15			0



Process data

Channels	8bit = 1 Register 1 4DO-P	16bit = 1 Register 2 16DO-P	8bit = Register depends on following module min: 1/2 register max: 1 register 3 8DO-P
0	1	1	0
1	0	0	0
2	1	1	0
3	0	0	0
4		0	0
5		0	0
6	1st written register	0	0
7		0	0
8		0	
9		0	
10		0	
11		0	
12		0	
13		0	
14		0	
15		0	

2nd written register