Weidmüller

UC20-WL2000-IOT / UC20-WL2000-AC

Application Note for Modbus/TCP implementation in Node-RED

Abstract:
In this application note we will describe how to read and write data from Modbus/TCP Coupler using a u-control controller (Web or IOT) with Node-RED as a master.

Hardware reference

No.	Component name	Article No.	Hardware / Firmware version
1	UC20-WL2000-IOT	1334990000	FW: 1.6.0 (or higher)
2	UC20-WL2000-AC	1334950000	FW: 1.6.0 (or higher)
3	UR20-FBC-MOD-TCP	1334930000	FW: 2.0.7 (or higher)
4	UR20-FBC-MOD-TCP-V2	2476450000	FW: 2.0.7 (or higher)

Software reference

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

File reference

No.	Name	Description	Version
1	AN0009-UC20-WL2000 ModbusTCP master implementation in Node-RED.zip	The file contains a screenshot of a Node-RED flow and a .json file, related to this application note	-

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 4
2 Requirements 5
3 Configuration Modbus Coupler 6
4 Configuration IOT Controller 7
5 Modbus TCP Write 12
5.1 Write Single Coil 13
5.2 Write Single Holding Register 17
5.3 Write Multiple Coils 20
5.4 Write Multiple Holding Registers 21
6 Modbus TCP Read 24
6.1 Read Coils 24
6.2 Read Holding Registers 26
7 Additional information / ‘expert’ examples 28
7.1 Read Example 28
7.2 Write Example 33

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.

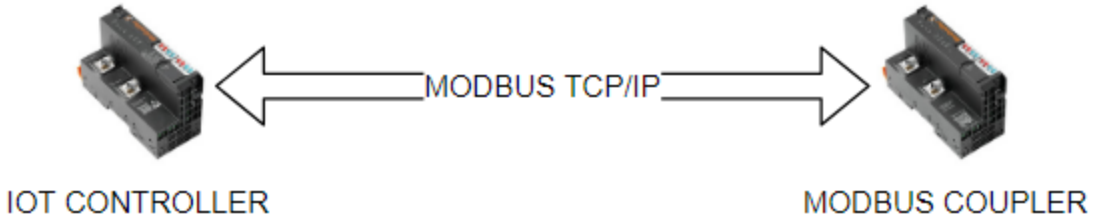
Disclaimer
This Application Note / Quick Start Guide / Example Program does not relieve you of the obligation to handle it safely during use, installation, operation and maintenance. Each user is responsible for the correct operation of his control system. By using this Application Note / Quick Start 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 Requirements

The connection diagram that has been used as an example is the following:



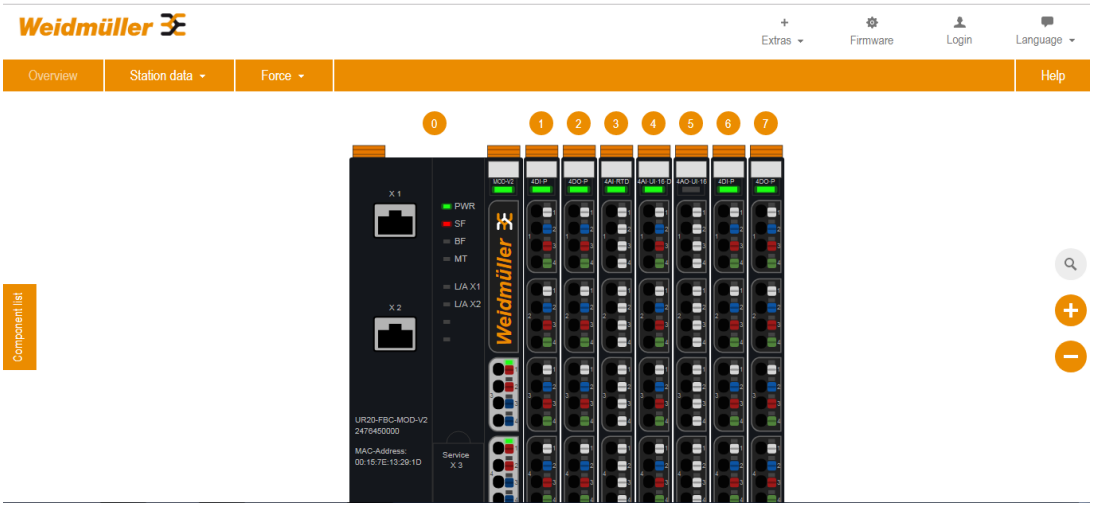
You need at least 1 digital output module which is connected as the first module at the Modbus coupler in order to do this tutorial. In this example we will use a 8DO module, but you can also use 4DO or 16DO.

3 Configuration Modbus Coupler

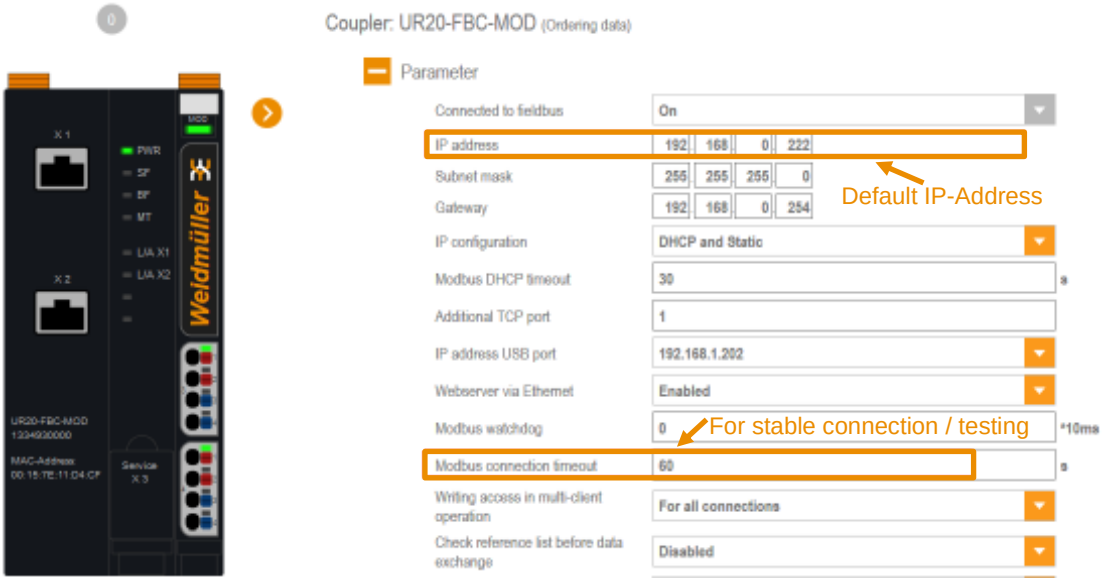
In order to read and write, no further configuration on the Modbus coupler is needed.
The purpose of accessing this interface through web browser is to check the results of writing and reading → Monitoring.

Default IP via USB: **192.168.1.202**

If you cannot establish a USB-connection, you can try with 192.168.**X**.202, where X must be a number between 1 to 5 (inclusive).



Check the LAN IP of Modbus Coupler (Default: **192.168.0.222**). Modbus connection timeout can be set to **60** for stable connection and testing.



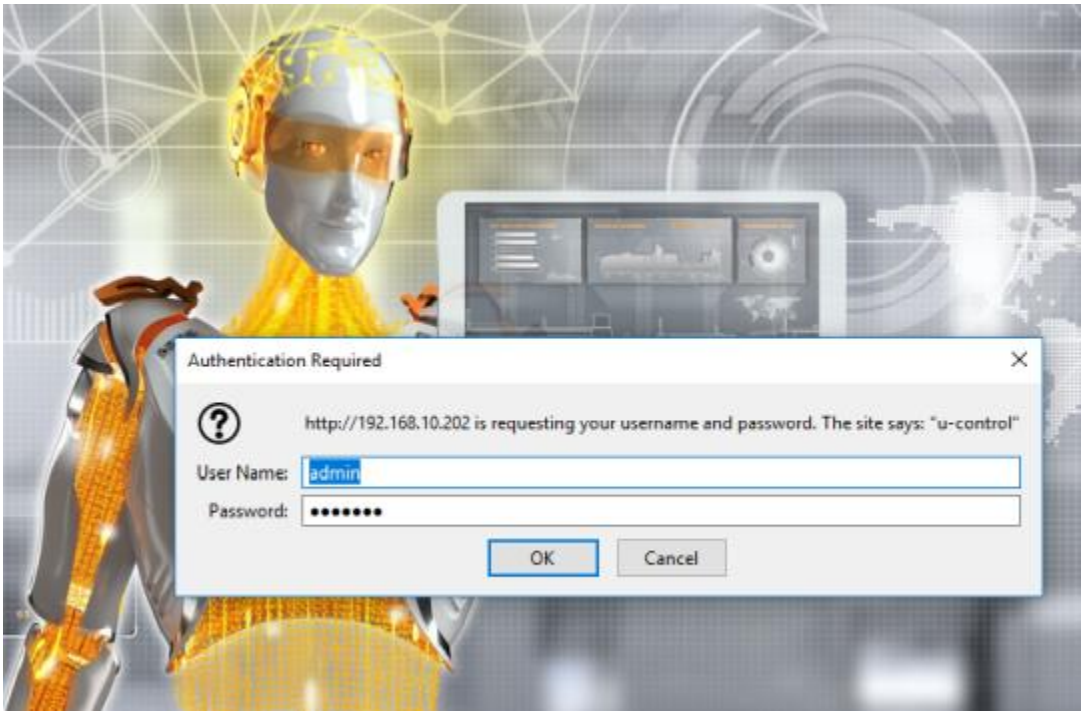
If LAN IP is given through a DHCP-Server or changed manually, you must use this instead.

4 Configuration IOT Controller

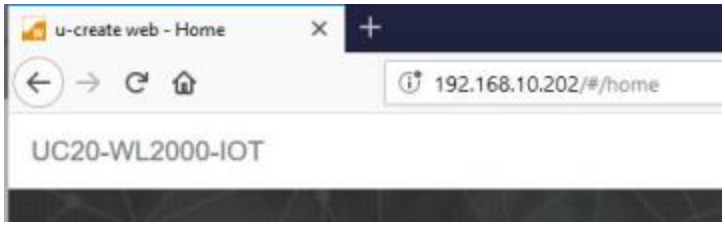
- Access the IOT web server or configuration interface through any browser.

  Default IP via USB: **192.168.10.202**

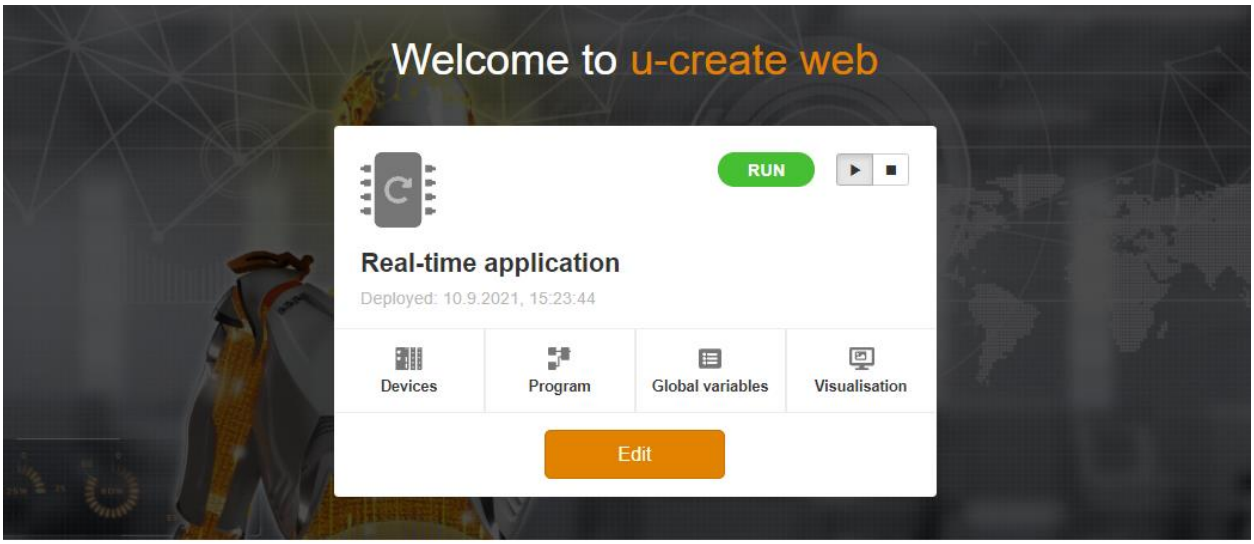
Maybe you must first log in with <admin> and <Detmold>.

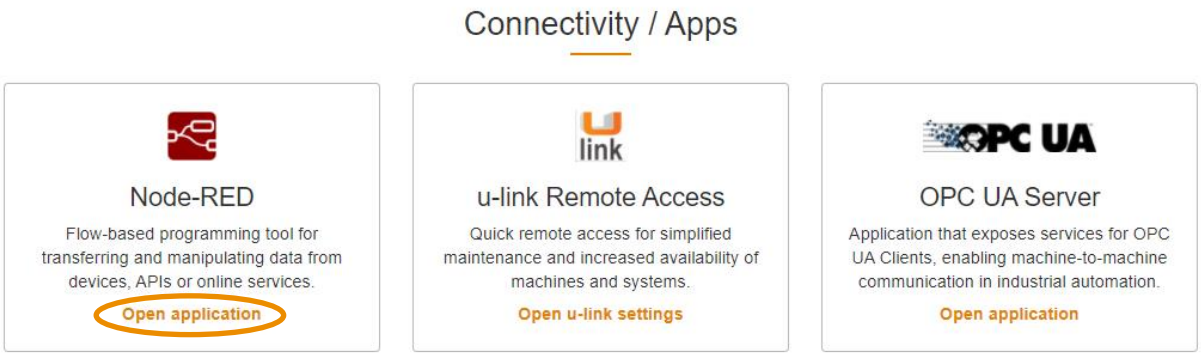


Starting from the software version 1.2.0, web-interface of UC20-WL2000-IOT and UC20-WL2000-AC have the same design. The main difference is in the enabled functionalities, which depend on the controller type.
Your type of controller is displayed in the upper left corner.



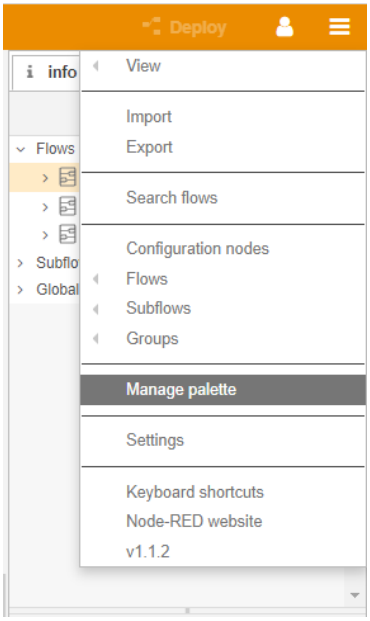
- Start the Node-RED with **Open application**



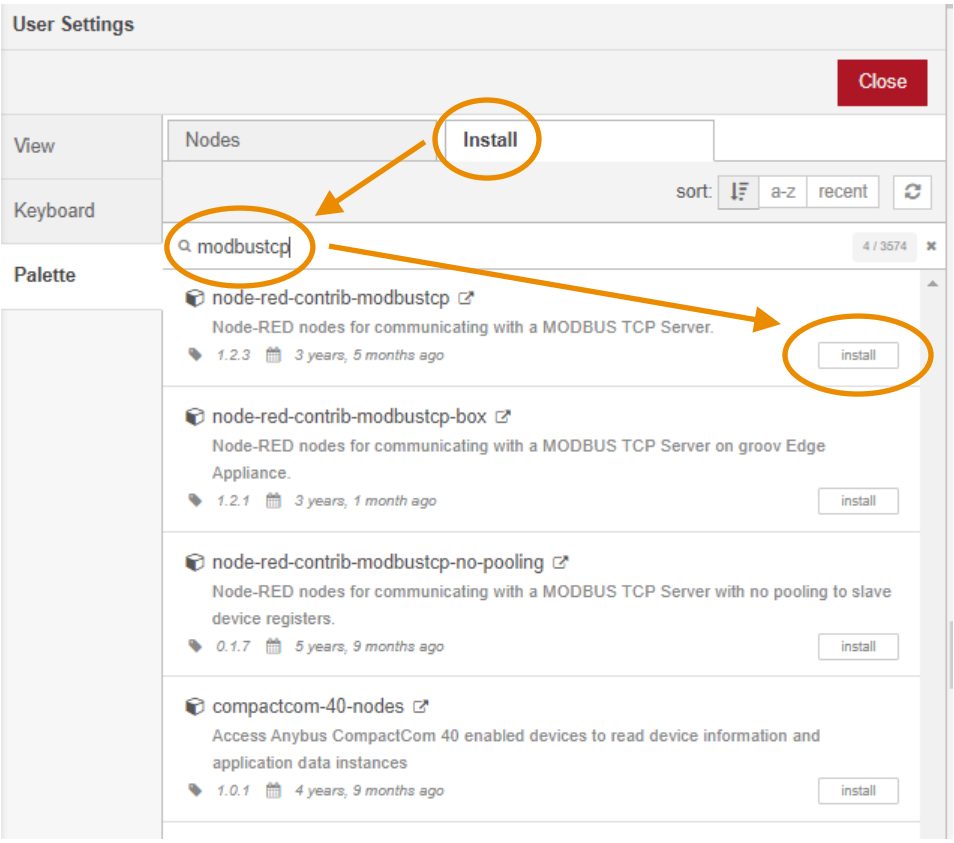


We want to make a connection to a Modbus coupler. This is special, because Modbus/TCP nodes are not installed by default. It is necessary to add them first.

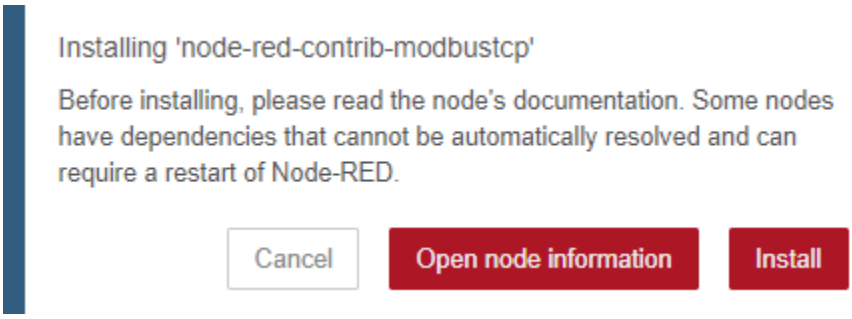
1. Go to menu and click on **Manage palette**.



2. Go to I**nstall** tab, search the nodes and click **Install**.



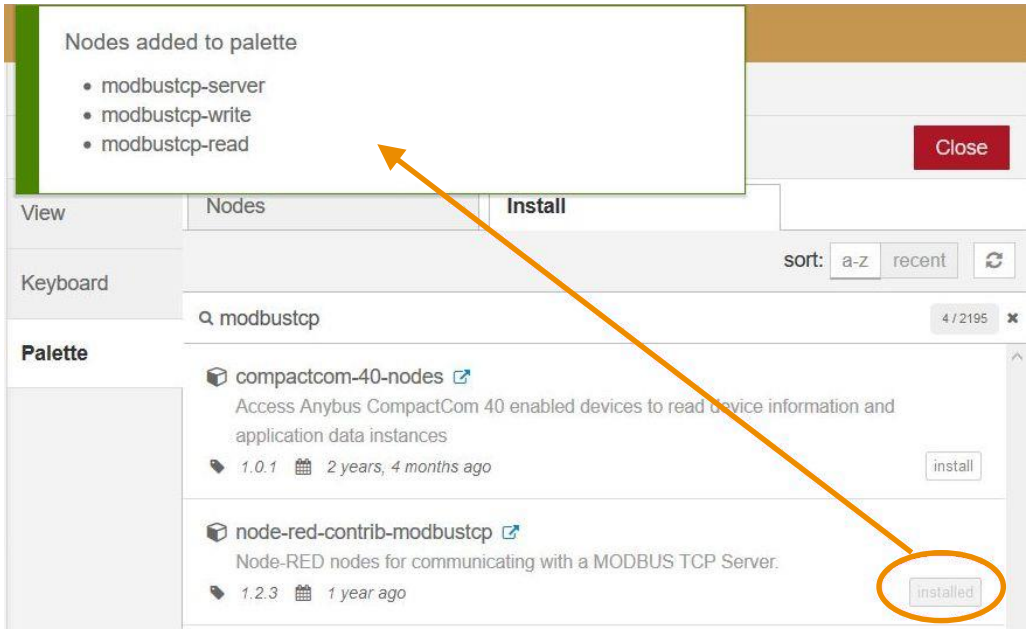
3. Click **Install** again.



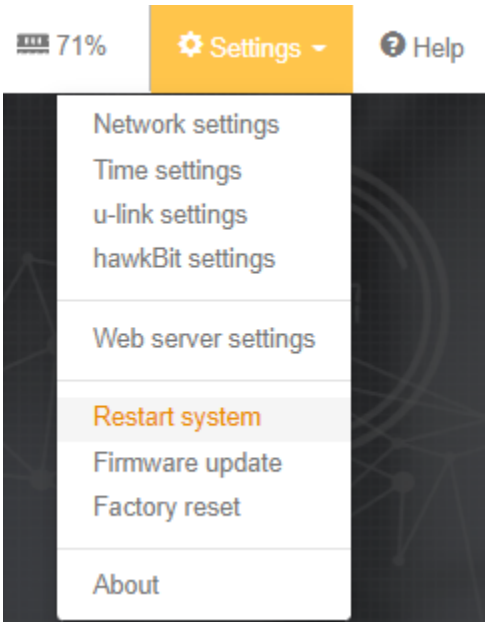
If an error message occurs, please check the network settings (Default gateway / DNS servers).
For the installation it is necessary that the controller has a direct internet connection!

This is different to the information in your browser, because the browser runs on your PC and uses this connection for searching and displaying the nodes.

If the installation was successful, you will get the following information:

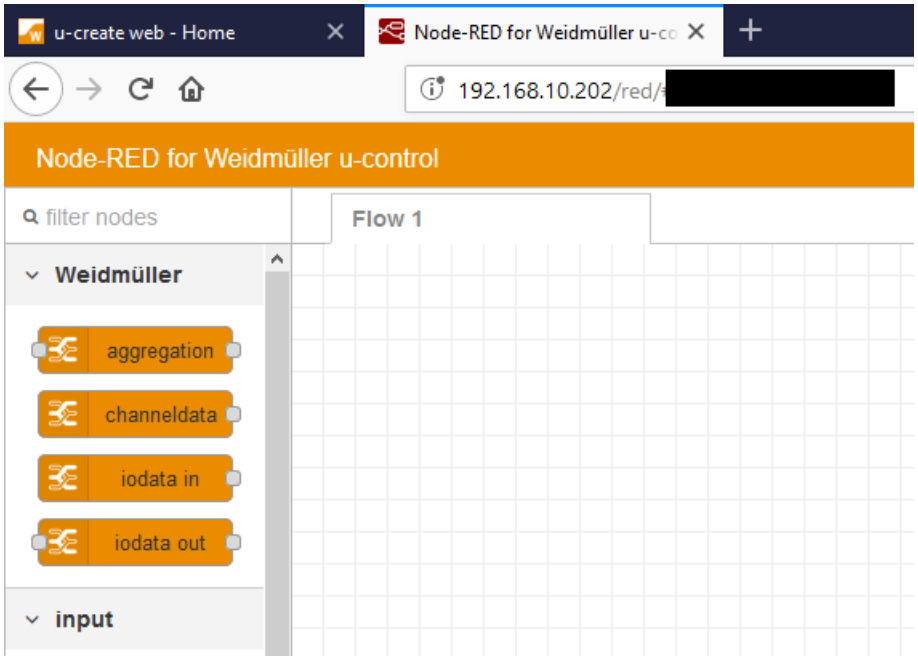


- Perhaps it is better to make a restart in order to take effect of the changes.



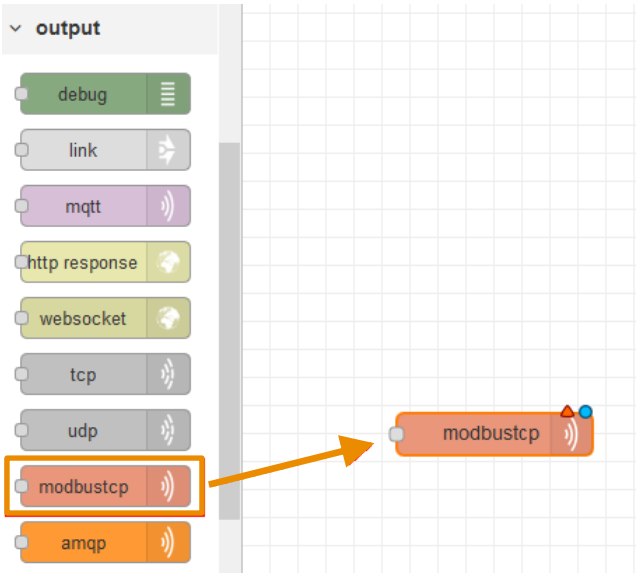
5 Modbus TCP Write

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



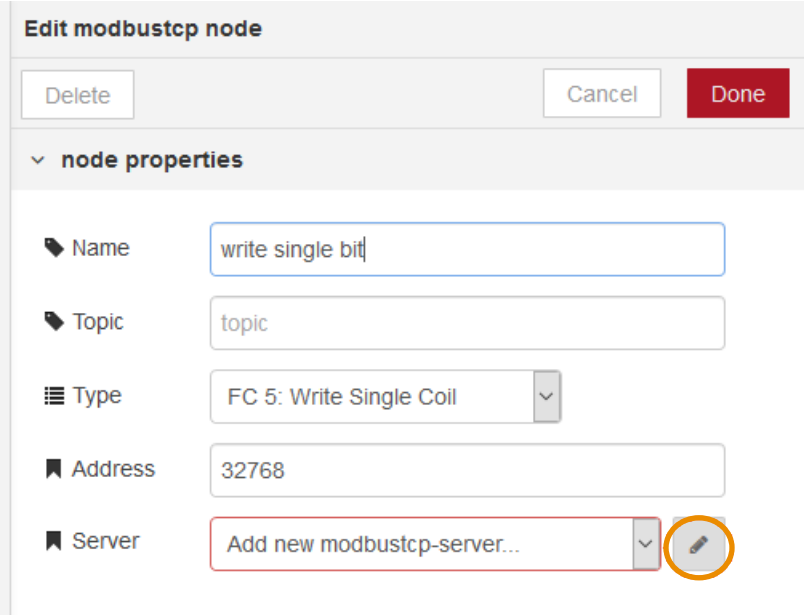
At this point it is now possible to establish a connection to the Modbus coupler and perform read and write actions.

The node used for writing data to the Modbus coupler is located in **output -> Modbus/TCP.**



5.1 Write Single Coil

- The **modebustcp** node must be configured (double-click) as shown on the following image:



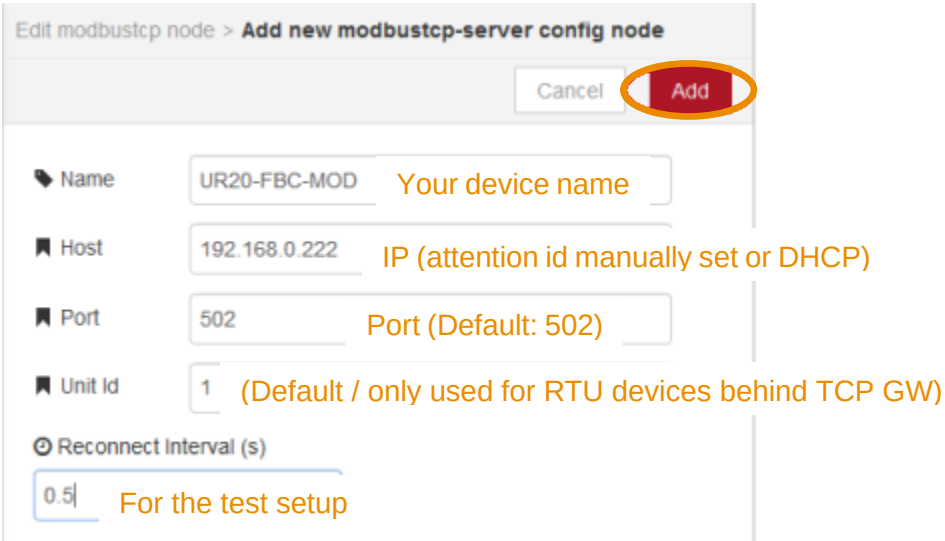
name shown on the node

the type of data to be written, in this case set to **FC 5: Write Single Coil**

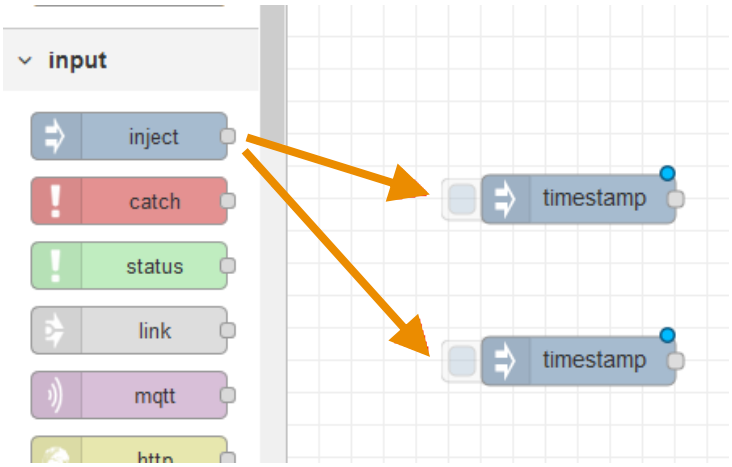
in Modbus writable output bits start at address **32768** (0x8000h)

Server: here is where the Modbus/TCP server is configured

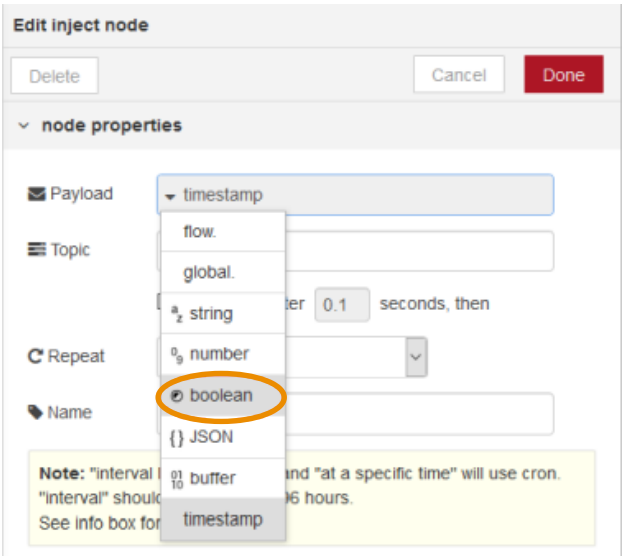
- Once you click the **edit** button you can fill in the server settings:

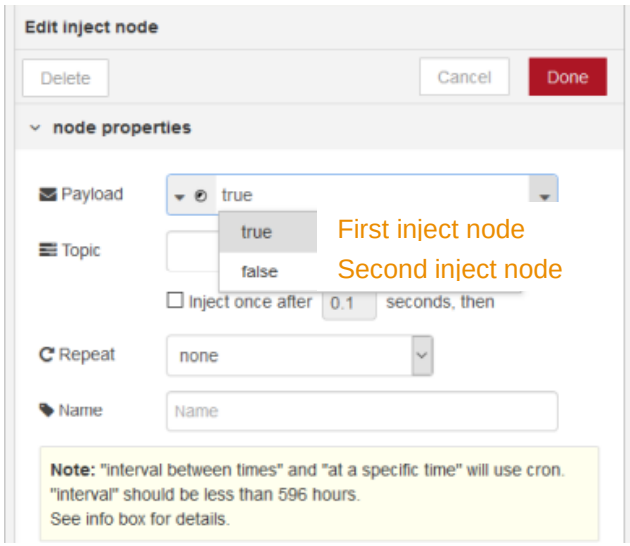


This example shows how to use the node **inject** to pass a true (1) or false (0) value to a single coil address:

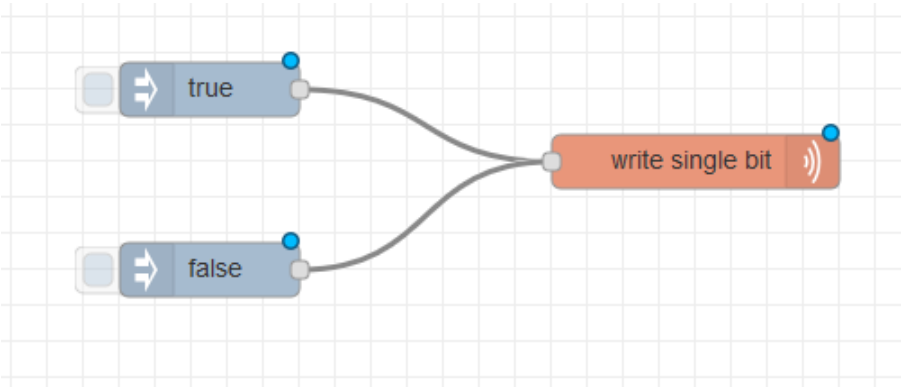


- Drag them on the Flow. To start editing double-click each of them.

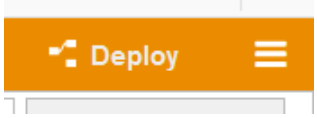




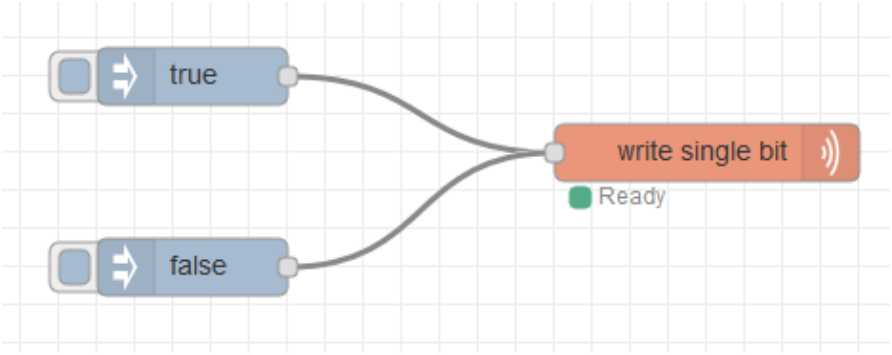
- Implement connections between the nodes as follows.



- Deploy your changes to the controller.



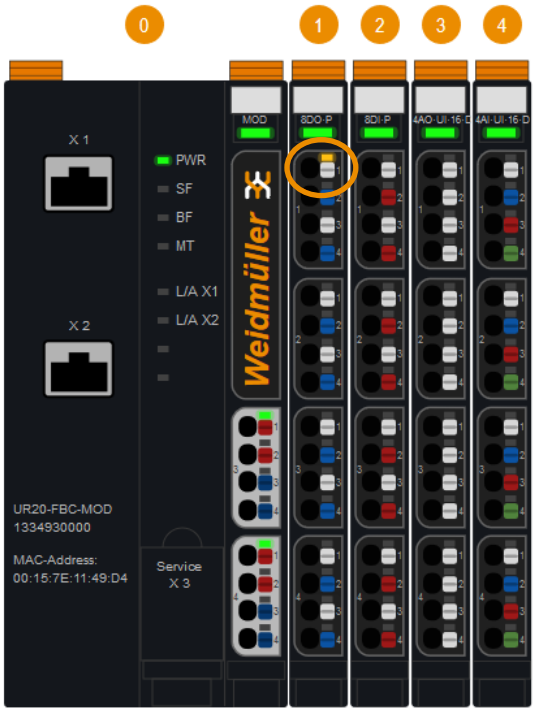
Once the changes are deployed the node shows a green little square indicating the connection to the Modbus server.



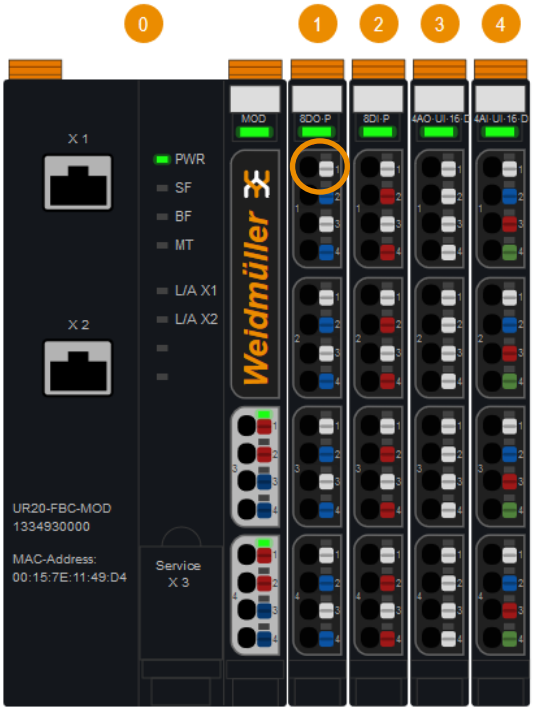
At this point it is possible to write a single coil.

- Check the web interface of your Modbus/TCP coupler via USB or LAN.

TRUE the led at the first channel turns on:

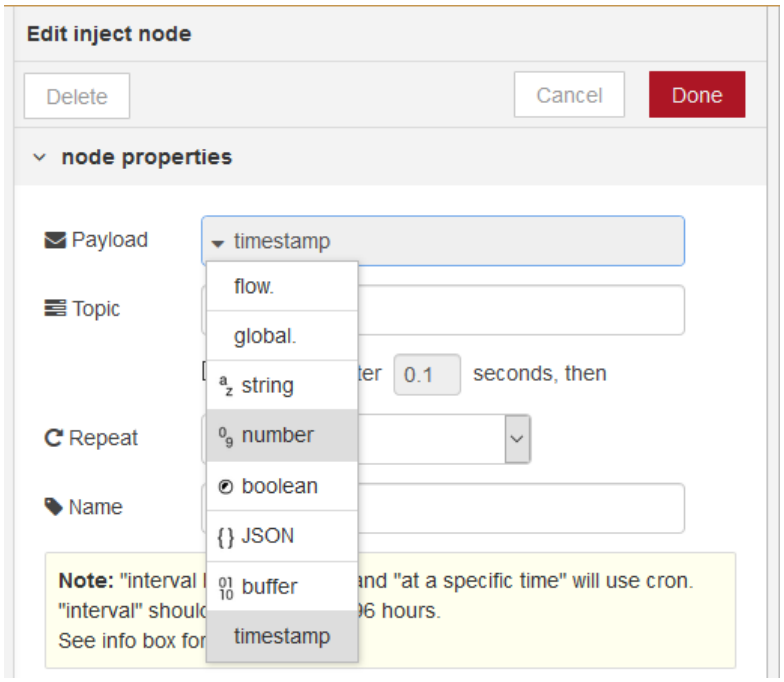


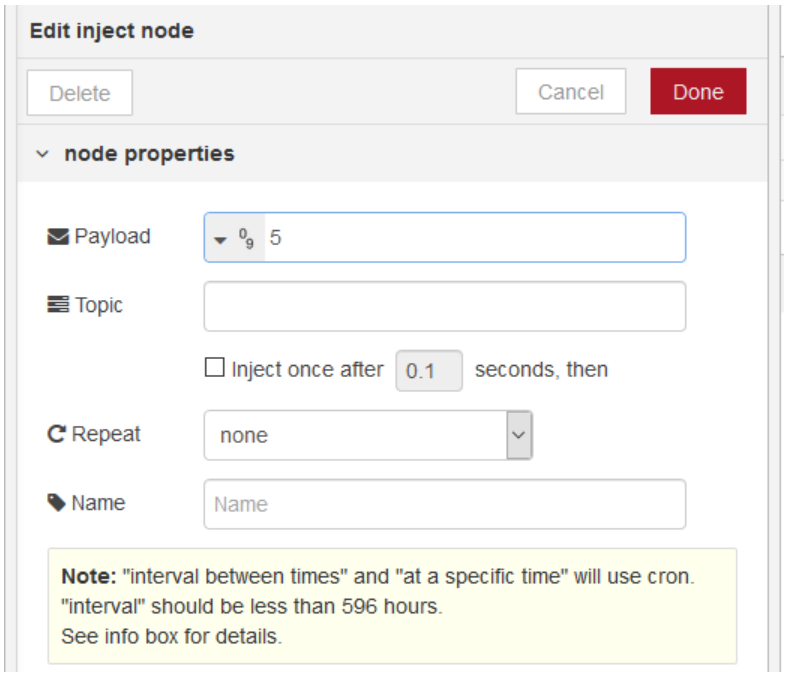
FALSE the led at the first channel turns off:

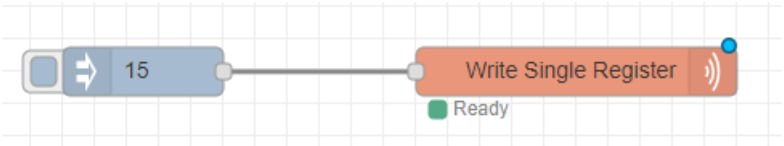


5.2 Write Single Holding Register

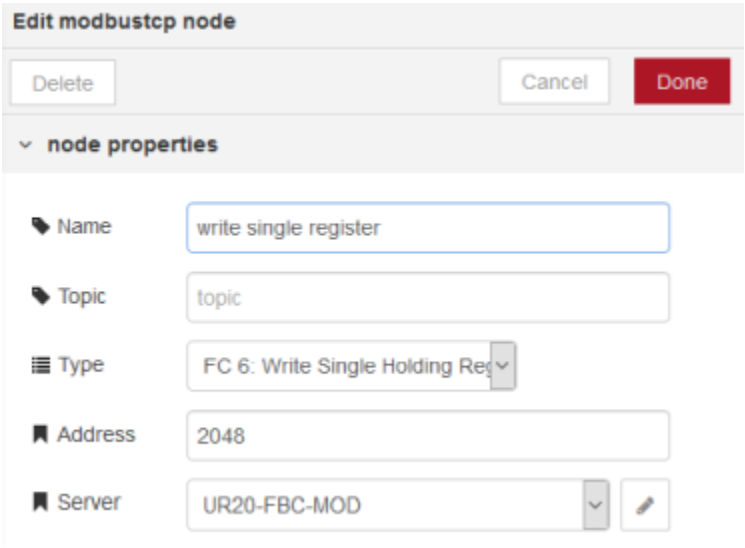
For this data type the injection must be a `number`:







Inside the **modbustcp** node the `Type` needs to be changed to `Write Single Holding Register`:



Register addresses are different from bit addresses!

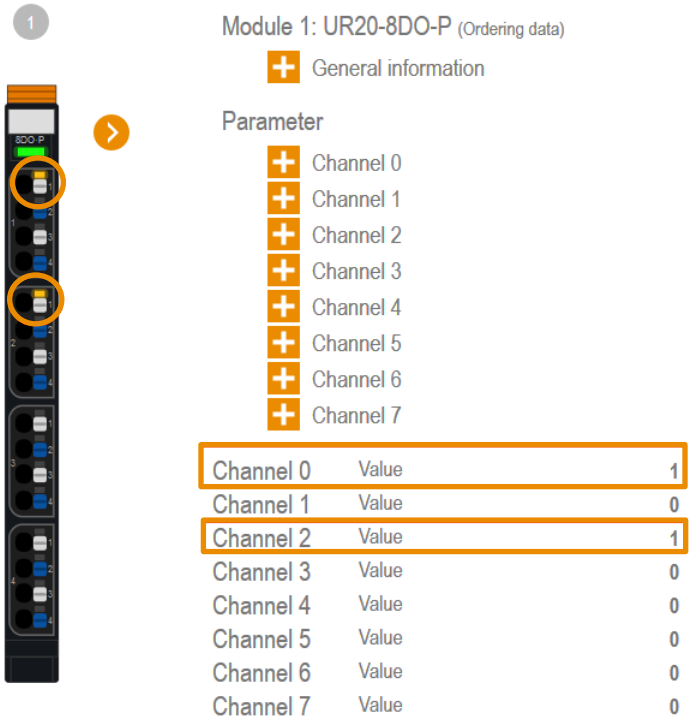
Packed process data for outputs
- begins at 0x0800h (2048dez)
- data width is module-dependent
- byte granularly

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

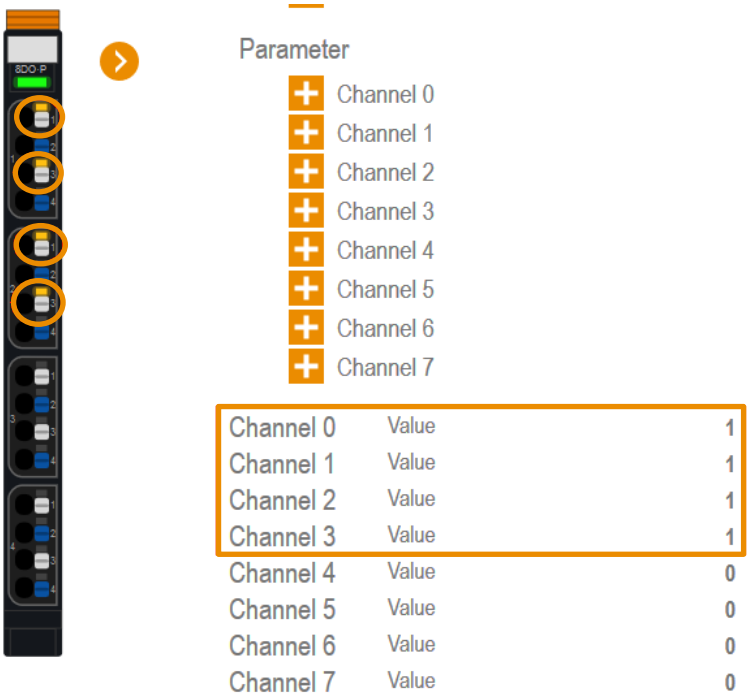
One register has 16 bits. If a write action is performed, 16 bits are written.

Keep this in mind!

Writing 5 changes the state of channels 0 and 2. [decimal 5 = binary 0101 (LSB is right)]:

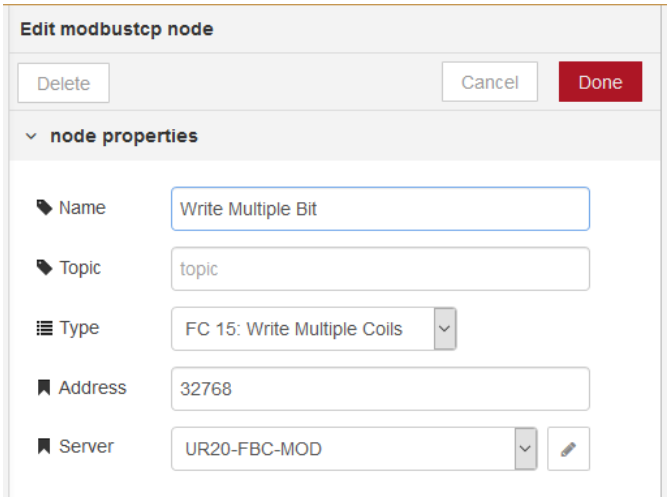


Writing 15 changes the state of channels 0 and 3. [decimal 15 = binary 1111 (LSB is right)]:

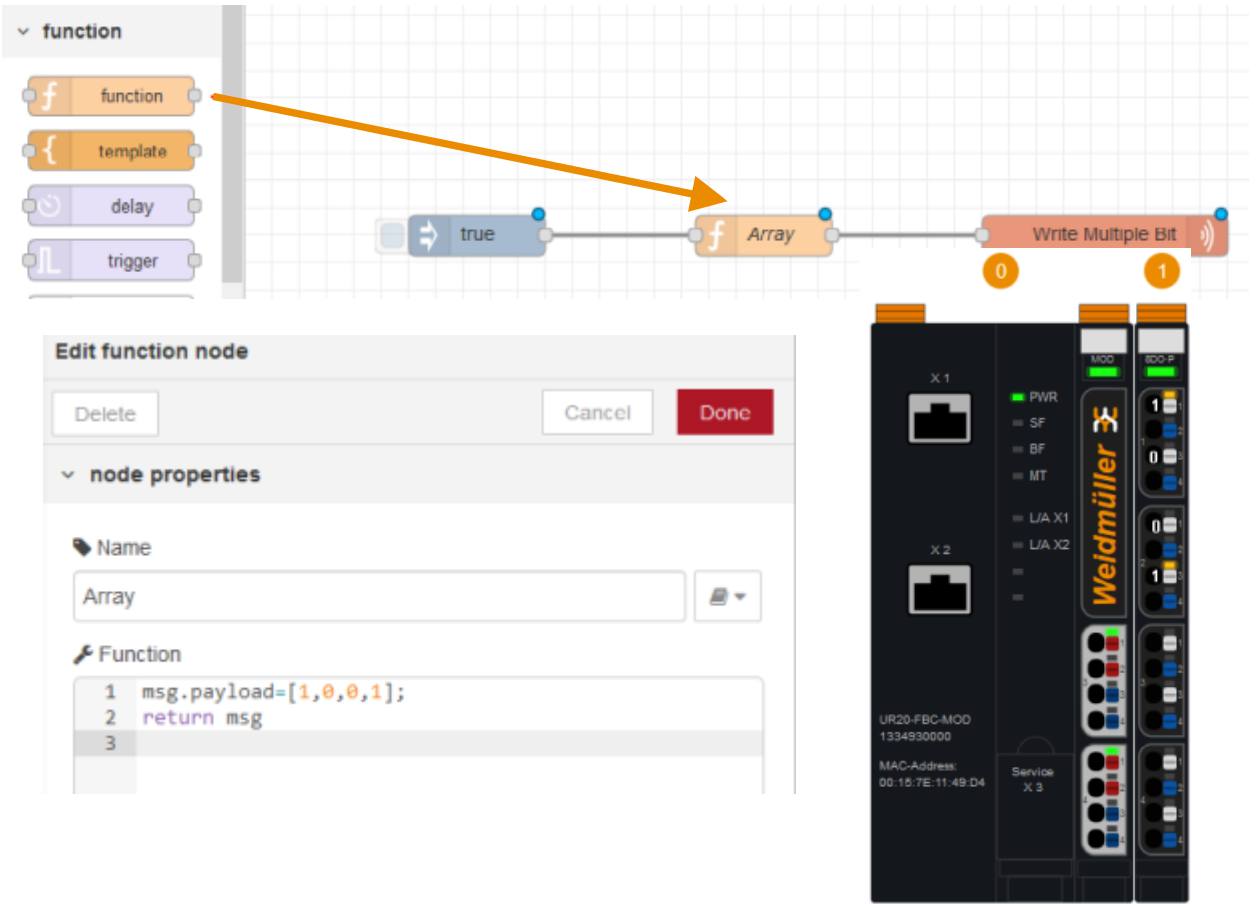


5.3 Write Multiple Coils

This type of writing is the same as single coil but with multiple values.
The **inject** node should be an array of binary values:

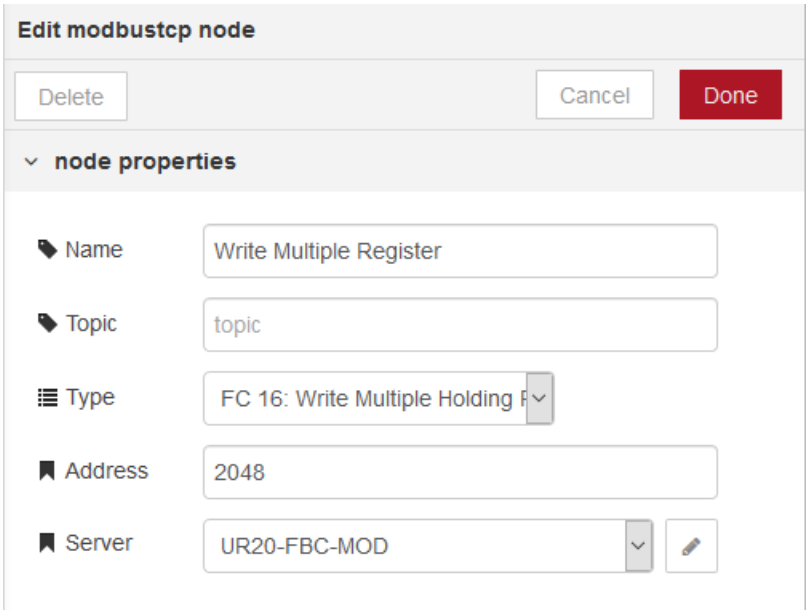


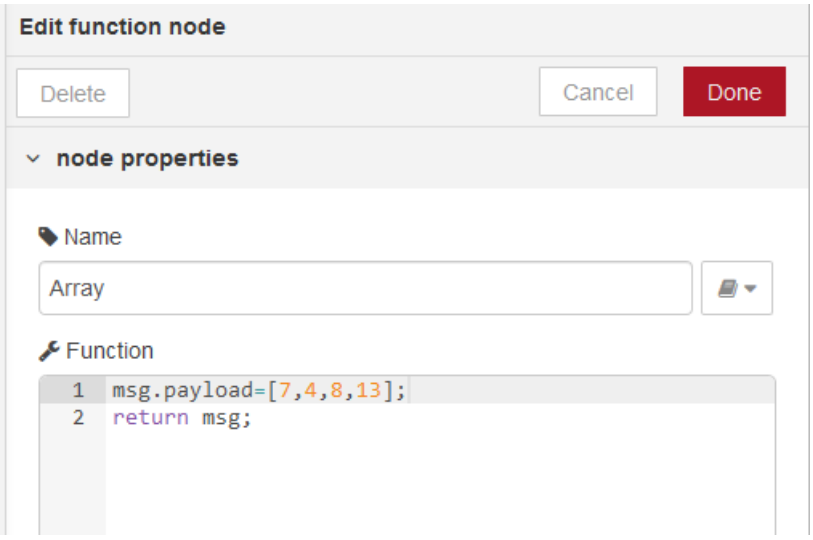
The **Array** function node defines and returns an msg object which contains an array in `msg.payload`:



5.4 Write Multiple Holding Registers

This case shows write operations to 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 neighbored registers of the same type.

But: we are using the register addresses for packed process data, so it depends on the module arrangement as well!

How would the result look like if we have three 8DO modules?

Remember, for two 8DO modules we need 2 bytes (1 byte for each module) which are allocated in 1 register.
This means, writing 1 register controls 2 8DO modules.

For beginners this is very difficult to understand but an important aspect of the Modbus communication.

Process data

Channels	1 8DO-P	2 8DO-P	3 8DO-P
0	1	0	0
1	1	0	0
2	1	0	1
3	0	0	0
4	0	0	0
5	0	0	0
6	0	0	0
7	0	0	0

1st register (16 bit)

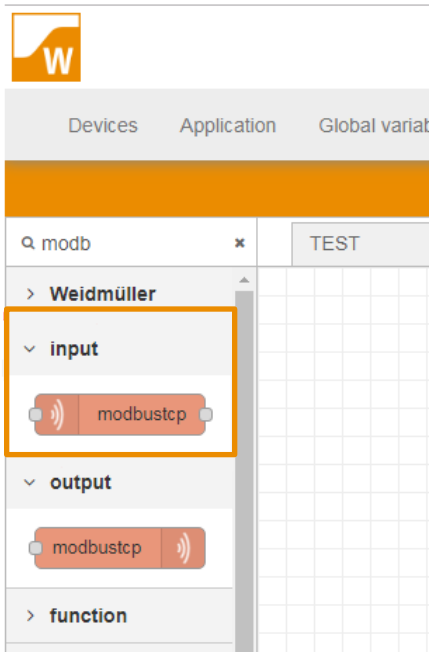
2nd register (16 bit)

7 from Array

4 from Array

6 Modbus TCP Read

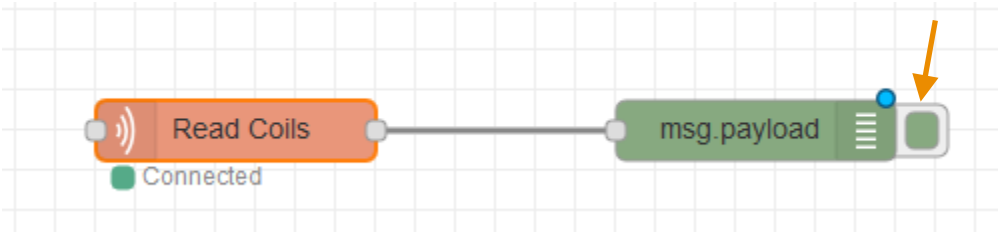
There is no big difference between writing of coils/registers and reading.
Because of this we will use our Modbus server, which we have already configured in the first chapter, and show only the differences without detailed explanations.
The node used for writing data to the Modbus coupler is located in **Input -> Modbus/TCP**:



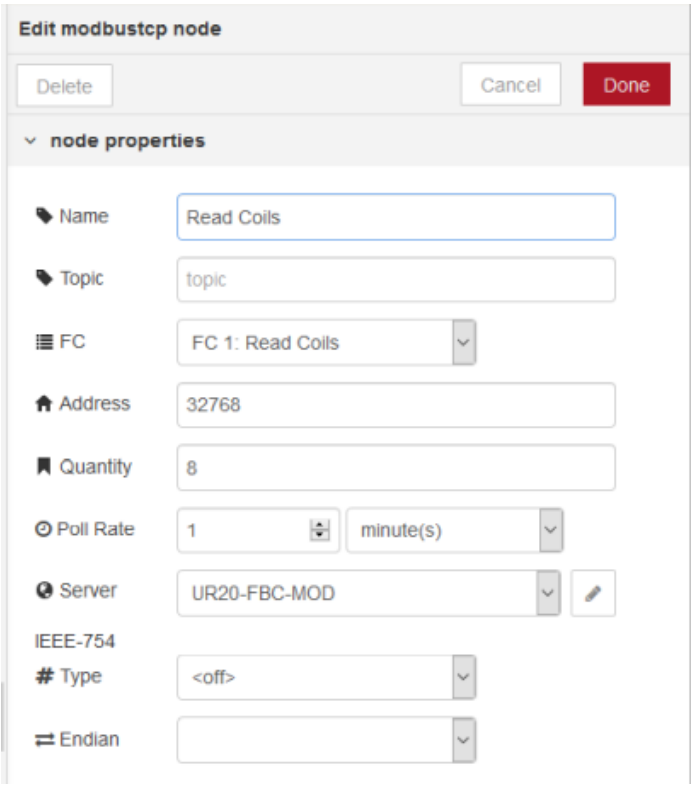
6.1 Read Coils

This configuration shows how to read from **Modbus** node and print it in the debug view:

You need to activate the **debug** node in order to see the output:

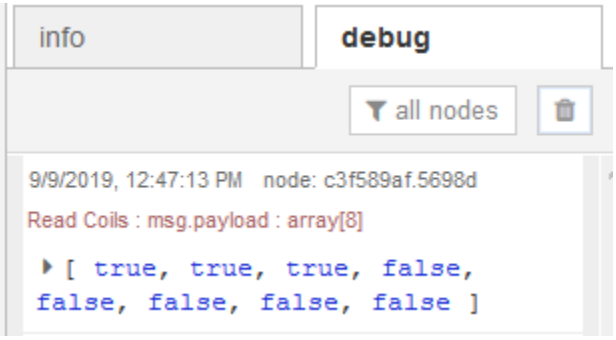


The Modbus node needs to be set to `Read Coils`:



`Quantity`**:** Number of coils/bits that would be read starting from the `Address` value.

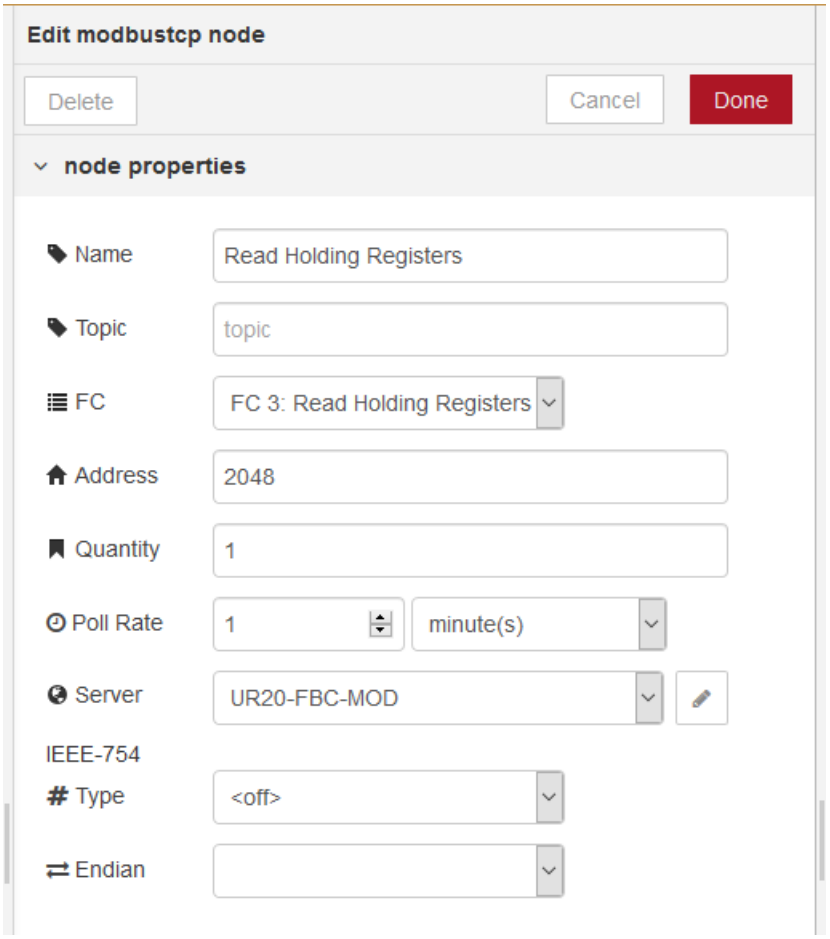
The output of this configuration would be an array that shows the state of the channels:



This output shows that channels 0, 1 and 2 are enabled (value 1 or true) while the other channels are disabled (value 0 or false).

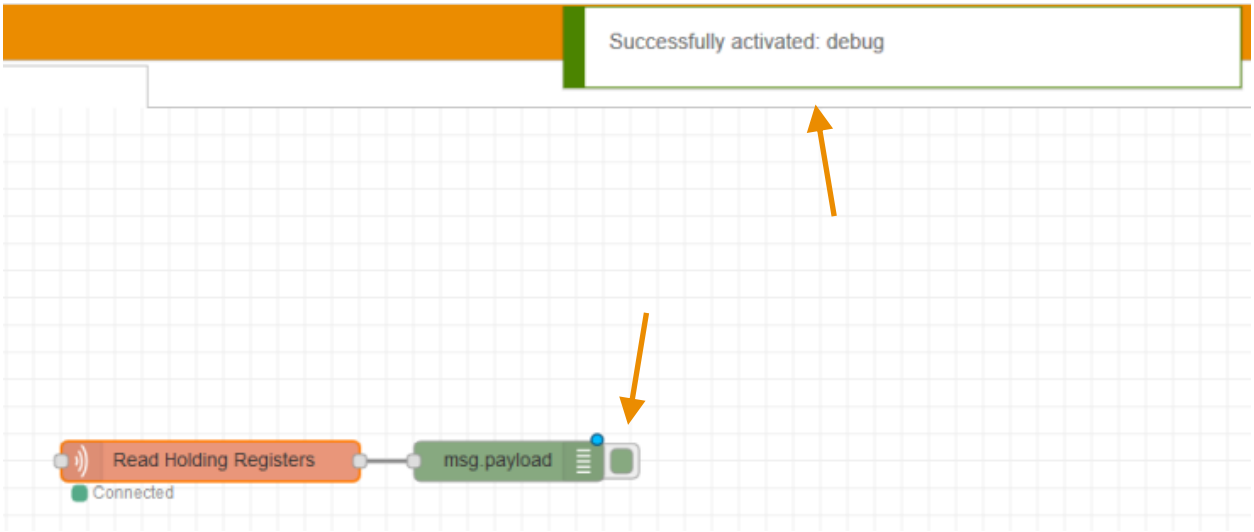
6.2 Read Holding Registers

Settings for this configuration are as follows.

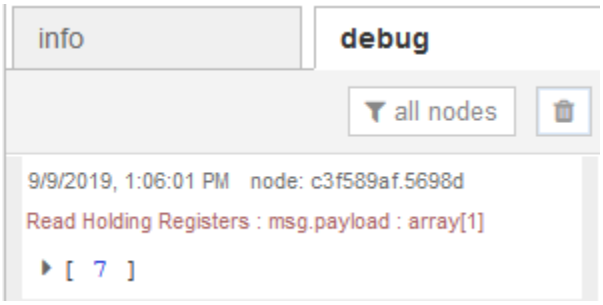


FC should be set to `Read Holding Registers` and the `Address` to `2048` as it is the start of output registers (packed process data of outputs).
`Type` must be set to `off` if the decimal value is desired. Otherwise, the log would show a number according to the standard IEEE-754.

Activate the **debug** node:



The following image shows the register value from the first holding register:



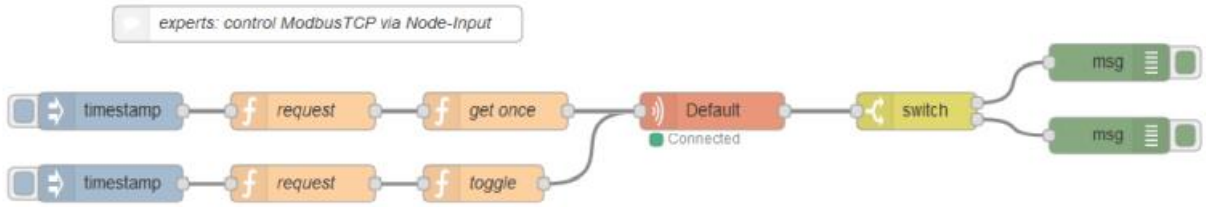
7 Additional information / ‘expert’ examples

All examples within the previous chapters are only for demonstrating the structure and function of the Modbus nodes.
For production use it is recommended **not** to use several nodes for read / write functions which are connected to the same Modbus/TCP coupler.

To avoid timing issues and errors when using more read/write requests, it is recommended to configure one **Modbus/TCP** node and assign the desired properties to the `msg.payload` object (address, function code, quantity, aso…) via a **function** node.

7.1 Read Example

Example flow:



This is a small example flow which describes how to read coils and register dynamically by different instances of one Modbus/TCP connection.
The Modbus/TCP connection has a default configuration with “Poll rate” = 0, which leads to the fact, that this configuration is not executed itself.
Minimizing the use of the same configuration in different nodes leads to a more stable system.
Main advantage is that you only have one active Modbus connection to the Modbus/TCP Server.
In the example two functions are implemented, in order to define whether it is a single request or a periodical one.

Following pictures will show configuration of the used nodes from left to right.

1. **inject** node - normal trigger with timestamp / no configuration

2. '**request' function** node - 2 variants – for coil and register functions

   a. reading 8 Bits from starting address 32768 with function code 1 every 1000ms

Function

```
msg.payload = {
    "name": "myTestRequest1",
    "topic": "8DI Bits",
    "address": 32768,
    "quantity": 8,
    "dataType": "FC1",
    "interval": 1000,
}

return msg;
```

   b. reading 2 register from starting address 2048 with function code 3 every 2000ms

Function

```
msg.payload = {
    "name": "myTestRequest2",
    "topic": "HoldRegister",
    "address": 2048,
    "quantity": 2,
    "dataType": "FC3",
    "interval": 2000,
}

return msg;
```

3. **function** node **'get once'** - data is read / written only once

Function

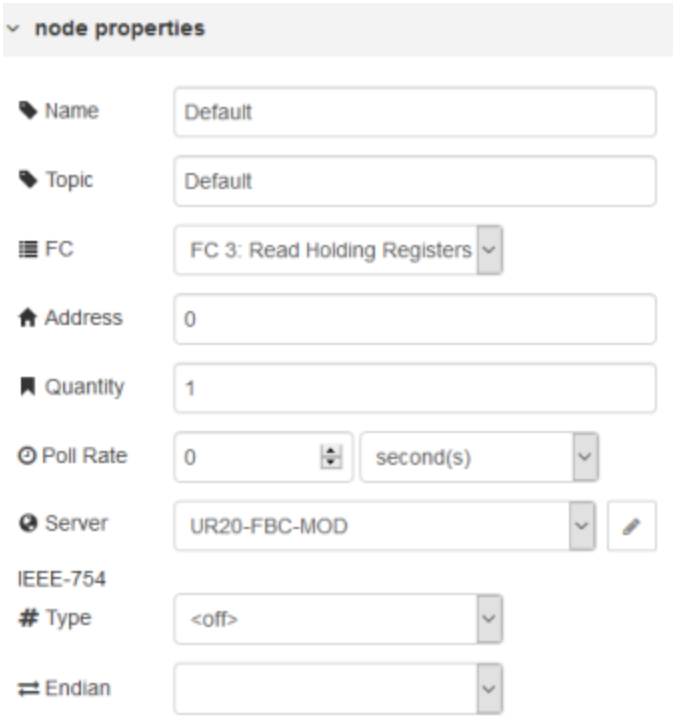
```
node.send(RED.util.cloneMessage(msg));
msg.kill = true;

return msg;
```

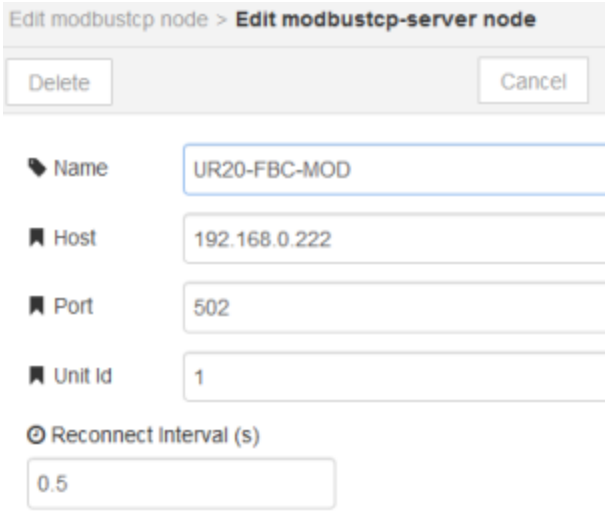
4. **function** node **‘toggle’** - first trigger activates / second trigger deactivates

Function

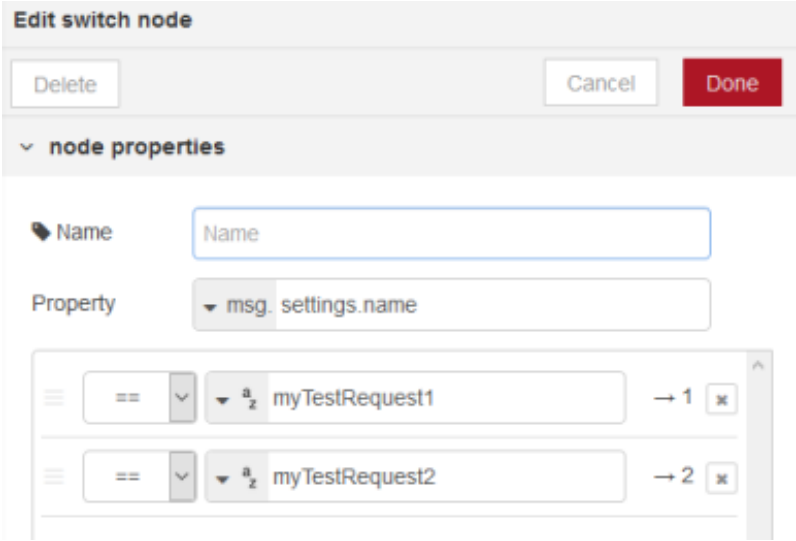
```
let toggle = context.get('toggleState')||1;

if (toggle < 0){

    msg.kill = true
    node.warn('deactivated');

}else{

    node.warn('activated');

}

toggle = -toggle;
context.set('toggleState', toggle);

return msg;
```

5. **Modbus/TCP** node - only default configuration (example UR20-FBC-MOD)





6. **switch** node - split requests to different debug messages



7. **Debug** node(s) - example for debug output messages

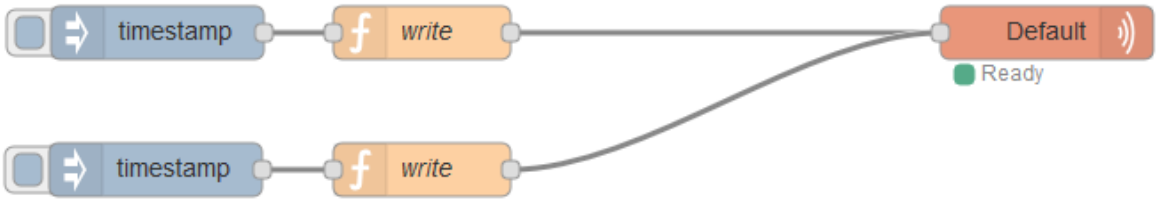
```
2/27/2020, 11:02:53 AM   node: fe17bb05.1e80b8
8DI Bits : msg : Object
▾ object
  ▸ settings: object
    topic: "8DI Bits"
  ▾ payload: array[8]
      0: 0
      1: 0
      2: 0
      3: 0
      4: 0
      5: 0
      6: 0
      7: 0
    _msgid: "f86a8a2.795bd78"
```

```
2/27/2020, 11:04:12 AM   node: toggle
function : (warn)
"activated"
2/27/2020, 11:04:12 AM   node: ad3962be.d021d
HoldRegister : msg : Object
▸ { settings: object, topic:
"HoldRegister", payload:
array[2], _msgid:
"c75edc73.b497a" }
2/27/2020, 11:04:14 AM   node: ad3962be.d021d
HoldRegister : msg : Object
▾ object
  ▸ settings: object
    topic: "HoldRegister"
  ▾ payload: array[2]
      0: 0
      1: 0
    _msgid: "4b8681ae.f0235"
2/27/2020, 11:04:15 AM   node: toggle
function : (warn)
"deactivated"
```

See additional `warn` messages for activating and deactivating

7.2 Write Example

Example flow:



1. **inject** node - normal trigger with timestamp / no configuration
2. **'write' function** node - 2 variants – for coil and register functions
   a. write a number 1 (dec) to starting address 32768 with function code 5

Name

write

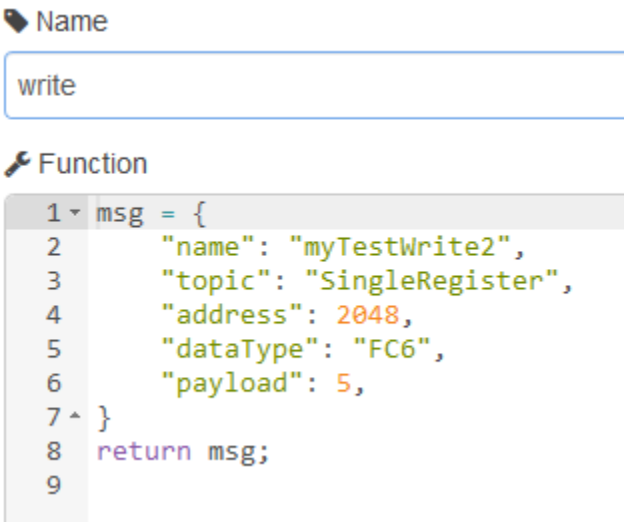
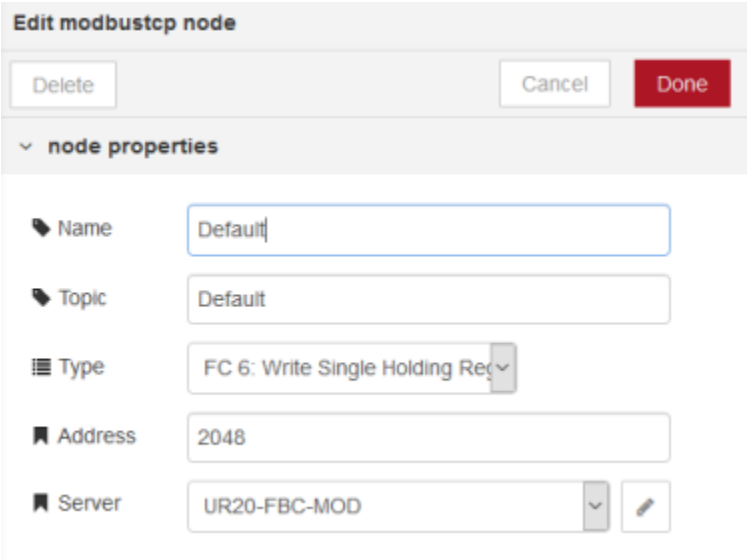
Function

```
msg = {
    "name": "myTestWrite1",
    "topic": "SingleCoil",
    "address": 32768,
    "dataType": "FC5",
    "payload": 1,
}
return msg;
```

allowed syntax for `dataType`:

- dataType
  - "Coil", "FC5", "FC 5", 5
  - "HoldingRegister", "FC6", "FC 6", 6
  - "Coils", "FC15" "FC 15", 15
  - "HoldingRegisters", "FC16", "FC 16", 16

b. Write a number 5 (dec) to starting address 2048 with function code 6



3. **Modbus/TCP** node - only default configuration (example UR20-FBC-MOD)