# HOWTO use NodeRED® on C4-8CO Controller

Raspberry Pi is a trademark of the Raspberry Pi Foundation.
More information under www.raspberrypi.org

it´s all about perfection


STRIVE IN PERFECTION
IN WHATEVER YOU
DO
TAKE THE BEST THAT
EXISTS AND MAKE IT
BETTER
WHEN IT DOES NOT
EXIST. DESIGN IT.
Sir Henry Royce

# CONTENT

In this application note you will find ...

# PREREQUISITES

Raspberry Pi is a trademark of the Raspberry Pi Foundation.
More information under www.raspberrypi.org

# PREREQUISITES

We assume that the reader is familiar how to use WINDOWS® operating system, how to configure a LINUX® Ethernet interface, how to use a remote desktop program or SSH console to configure LINUX®. Also we assume that the reader is able to install and open NodeRED® in a browser.

Furthermore we assume, that the reader is able to create a correct NodeRED® flow and that the reader is able to writhe a JavaScript script. If not, please consult the internet or book a education workshop. RESI is in no way responsible, if you or your customer cannot use the given advice here, because of lack of education in your or their staff!

With the purchase of a IoT Controller from RESI, you have not purchased the right of free education or free consulting from RESI!

RESI delivers IoT controllers with the ability to run NodeRED® on it, but RESI is not liable for any functional problems, software errors, law suits or other issues which results out of using NodeRED® on our devices in your project or machinery!

IMPORTANT SAFETY NOTES
Important hint:

Before you start with the installation and the initial setup of the device, you have to read this document and the attached installation guide and the actual manual for the device very carefully. You have to follow all the herein given information very accurate!

- Only authorized and qualified personnel are allowed to install and setup the device!
- The connection of the device must be done in de-energized state!
- Do not perform any electrical work while the device is connected to power!
- Disable and secure the system against any automatic restart or power on procedure!
- The device must be operated with the defined voltage level!
- Supply voltage jitters must not exceed the technical specifications and tolerances given in the technical manuals for the product. If you do not obey this issue, the proper performance of the device cannot be guaranteed. This can lead to fail functions of the device and in worst case to a complete breakdown of the device!
- You have to obey the current EMC regulations for wiring!
- All signal, control and supply voltage cables must be wired in a way, that no inductive or capacitive interference or any other severe electrical noise disturbance may interfere with the device. Wrong wiring can lead to a malfunction of the device!
- For signal or sensor cables you have to use shielded cables, to avoid damages through induction!
- You have to obey and to apply the current safety regulations given by the ÖVE, VDE, the countries, their control authorities, the TÜV or the local energy supply company!
- Obey country-specific laws and standards!
- The device must be used for the intended purpose of the manufacturer!
- No warranties or liabilities will be accepted for defects and damages resulting from improper or incorrect usage of the device!
- Subsequent damages, which results from faults of this device, are excluded from warranty and liability!
- Only the technical data, wiring diagrams and operation instructions, which are part to the product shipment are valid!
- The information on our homepage, in our datasheets, in our manuals, in our catalogues or published by our partners can deviate from the product documentation and is not necessarily always actual, due to constant improvement of our products for technical progress!
- In case of modification of our devices made by the user, all warranty and liability claims are lost!
- The installation has to fulfill the technical conditions and specifications (e.g. operating temperatures, power supply, ...) given in the devices documentation!
- Operating our device close to equipment, which do not comply with EMC directives, can influence the functionality of our device, leading to malfunction or in worst case to a breakdown of our device!
- Our devices must not be used for monitoring applications, which solely serve the purpose of protecting persons against hazards or injury, or as an emergency stop switch for systems or machinery, or for any other similar safety-relevant purposes!
- Dimensions of the enclosures or enclosures accessories may show slight tolerances on the specifications provided in these instructions!
- Modifications of this documentation is not allowed!
- In case of a complaint, only complete devices returned in original packing will be accepted!

# Install NodeRED®
# on RESI-T4/C4 controller

Raspberry Pi is a trademark of the Raspberry Pi Foundation.
More information under www.raspberrypi.org

it´s all about perfection

# Install NodeRED® on RESI-T4/C4 controller

Please search in the internet for a tutorial or more information, how to install NodeRED on a Raspberry Pi. We do not want to write yet another manual for the installation of NodeRED.

Open with VNCViewer the Raspberry Desktop or connect your monitor direct to HDMI and keyboard+mouse to the USB interface of our C4/T4 controllers. But do NOT login with root user. Choose your local user like pi. In the desktop choose Settings → Add/remove Software. Enter in the search field Options NodeRED. Select the package and click OK. After a while NodeRED is installed.

We are using NodeRED Version 4.x based on NodeJS version 18 on a 64 Bit OS (bookworm)

## WHERE IS NODE RED?

Determine the exact location of the node-red command.
If you have done a global install of node-red, then on Linux/OS X the node-red command will probably be either: /usr/bin/node-red or /usr/local/bin/node-red. The command which node-red can be used to confirm the location.

If you have done a local install, it will be node_modules/node-red/bin/node-red, relative to where you ran npm install from.

## INSTALL PROCESS MANAGER2

Install pm2 to start/stop the NodeRED system

```
sudo npm install -g pm2
```

## HOWTO START/STOP NodeRED

We are using pm2 to manually start or stop NodeRED after system power on.

```
pm2 start /usr/bin/node-red -- -v
```
or
```
pm2 start /usr/local/bin/node-red -- -v
```


You can get info from NodeRED with
```
pm2 info node-red
pm2 logs node-red
```

You can stop NodeRED with
```
pm2 stop node-red
```

# Install NodeRED® on RESI-T4/C4 controller

HOWTO AUTOMATICALLY START NODE RED AT SYSTEM STARTUP?
We are using pm2 to automatically start NodeRED after system power on.
First start NodeRED with

```
pm2 start /usr/bin/node-red -- -v
```
or
```
pm2 start /usr/local/bin/node-red -- -v
```

Then save the current setup with
```
pm2 save
```
and
```
pm2 startup
```

Then execute the shown command as described:
```
sudo env PATH=$PATH:/usr/bin /usr/local/lib/node_modules/pm2/bin/pm2 startup systemd -u resi --hp /home/resi
```

Finished!

# Install NodeRED® on RESI-T4/C4 controller

```
resi@RESI-C4:~ $ pm2 startup
[PM2] Init System found: systemd
[PM2] To setup the Startup Script, copy/paste the following command:
sudo env PATH=$PATH:/usr/bin /usr/local/lib/node_modules/pm2/bin/pm2 startup systemd -u resi --hp /home/r
esi
resi@RESI-C4:~ $ sudo env PATH=$PATH:/usr/bin /usr/local/lib/node_modules/pm2/bin/pm2 startup systemd -u
resi --hp /home/resi
[PM2] Init System found: systemd
Platform systemd
Template
[Unit]
Description=PM2 process manager
Documentation=https://pm2.keymetrics.io/
After=network.target

[Service]
Type=forking
User=resi
LimitNOFILE=infinity
LimitNPROC=infinity
LimitCORE=infinity
Environment=PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/local/games:/usr/games
:/usr/bin:/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin
Environment=PM2_HOME=/home/resi/.pm2
PIDFile=/home/resi/.pm2/pm2.pid
Restart=on-failure

ExecStart=/usr/local/lib/node_modules/pm2/bin/pm2 resurrect
ExecReload=/usr/local/lib/node_modules/pm2/bin/pm2 reload all
ExecStop=/usr/local/lib/node_modules/pm2/bin/pm2 kill

[Install]
WantedBy=multi-user.target

Target path
/etc/systemd/system/pm2-resi.service
Command list
[ 'systemctl enable pm2-resi' ]
[PM2] Writing init configuration in /etc/systemd/system/pm2-resi.service
[PM2] Making script booting at startup...
[PM2] [-] Executing: systemctl enable pm2-resi...
Created symlink /etc/systemd/system/multi-user.target.wants/pm2-resi.service → /etc/systemd/system/pm2-re
si.service.
[PM2] [v] Command successfully executed.
+---------------------------------------+
[PM2] Freeze a process list on reboot via:
$ pm2 save

[PM2] Remove init script via:
$ pm2 unstartup systemd
```

# Our RESI-C4-32DI24RO16AIOX RESI-C4-32DI24RO, RESI-C4-32DI24RO-2E controller

Raspberry Pi is a trademark of the Raspberry Pi Foundation.
More information under www.raspberrypi.org

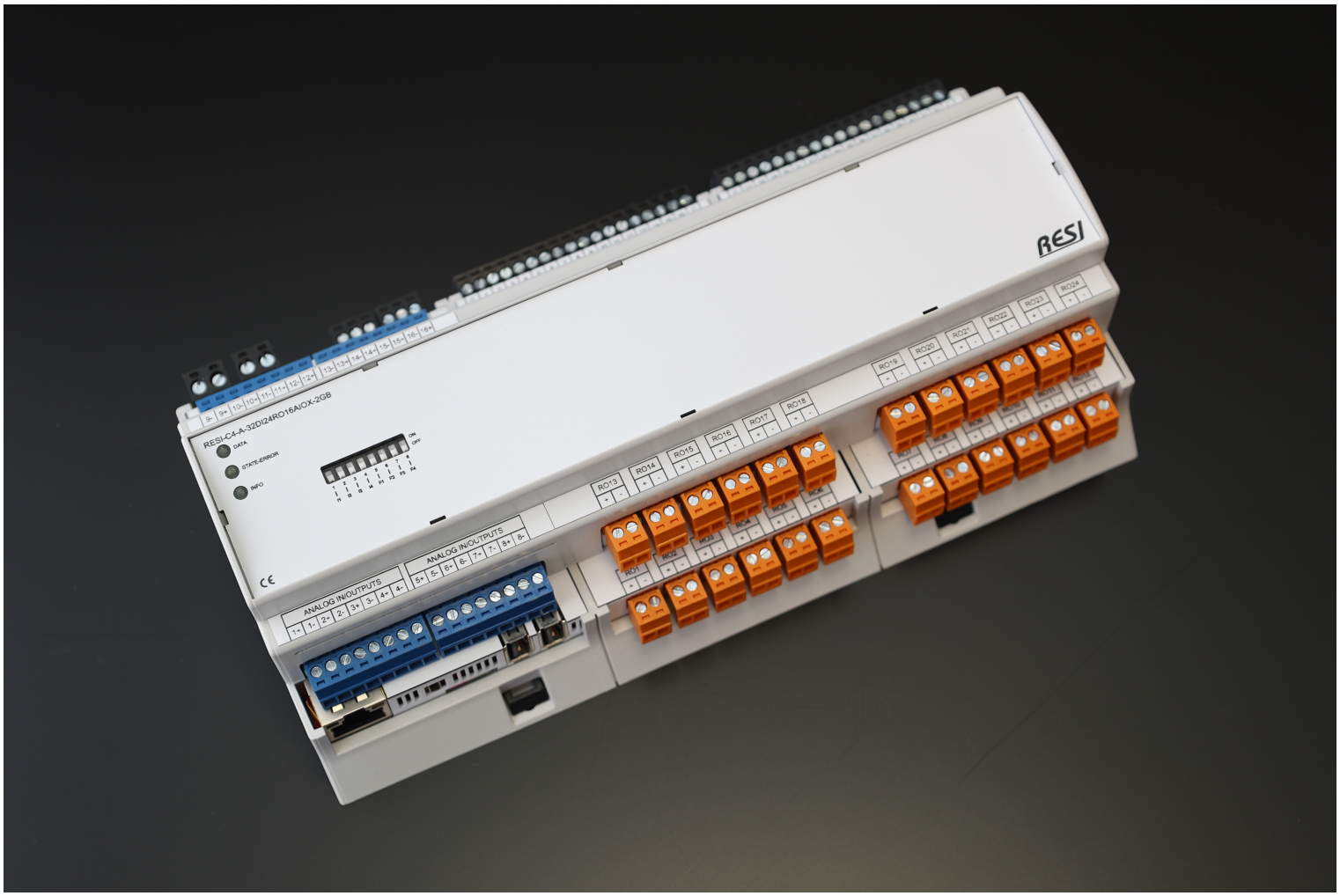
it´s all about perfection

# Our RESI-C4-A-32DI24RO16AIOX-xGB controllers

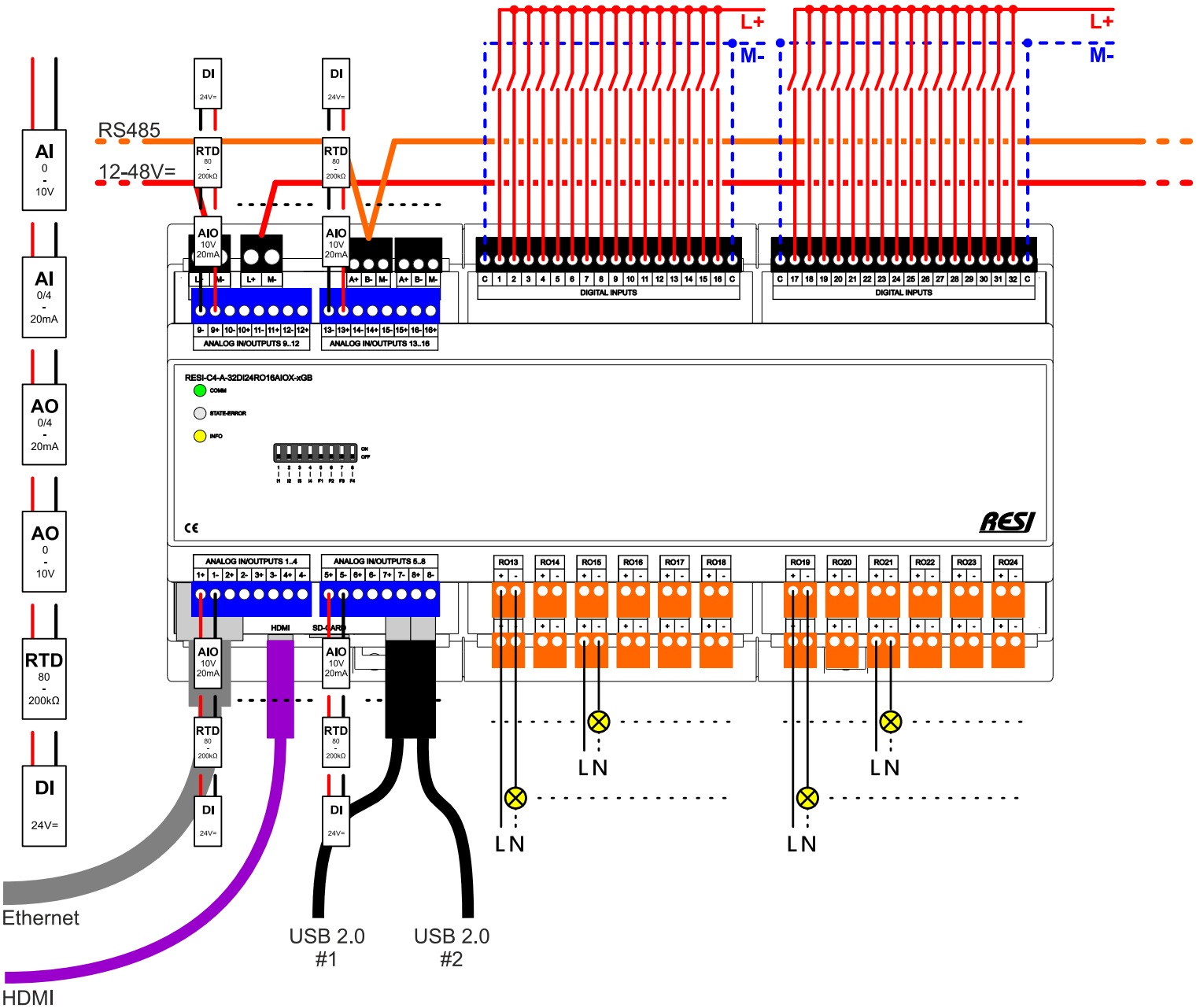
HIGHLIGHTS

- Raspberry Compute Module 4
- 2GB or 4GB or 8GB RAM
- 1xSD-CARD Slot with 32GB card
- 1xEthernet Interface
- 2xUSB 2.0
- 1xMicro-HDMI for 4k Monitor
- 1xRS485
- 8-pin DIP switch for software use
- 4 Status LEDs for Software use (green, white, red and yellow)

- 32 digital inputs for ≦48V= signals
- 24 form A relay outputs for ≦30V=, ≦250V=, ≦6A
- 16 universal analog inputs & outputs. Every channel can be configured to a different functionality:
  - Analog Input: 0-10V or 2-10V or 0-20mA or 4-20mA
  - Analog Output: 0-10V or 2-10V or 0-20mA or 4-20mA
  - Resistor Input: 0-1MΩ, PT100, PT1000, NI1000-DIN43760 linearisation and temperature calculation in °C, °F or °K
  - Digital input: 24V logic input or dry contact

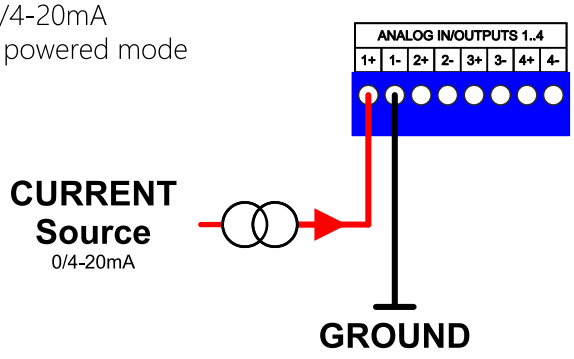


# RESI-C4-A-32DI24RO16AIOX-xGB
# Scematic

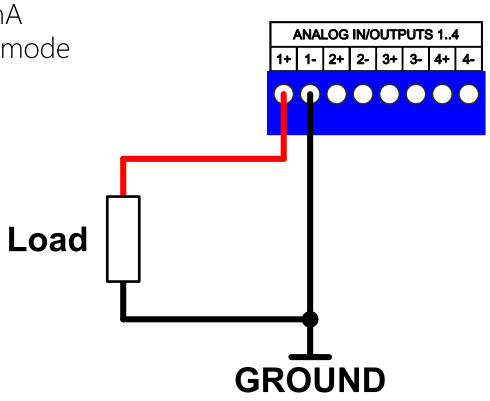


# RESI-C4-A-32DI24RO16AIOX-xGB
# Universal analog IOs - possibilities

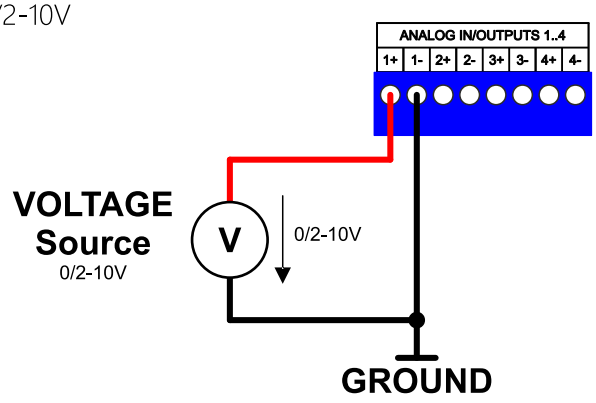
## ANALOG INPUT
0-20mA/4-20mA
External powered mode




## ANALOG INPUT
0-20mA/4-20mA
Loop powered mode

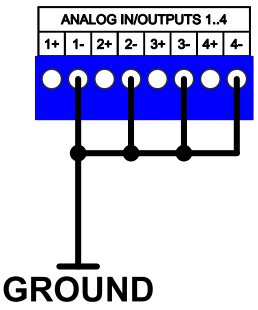



## ANALOG INPUT
0-10V/2-10V

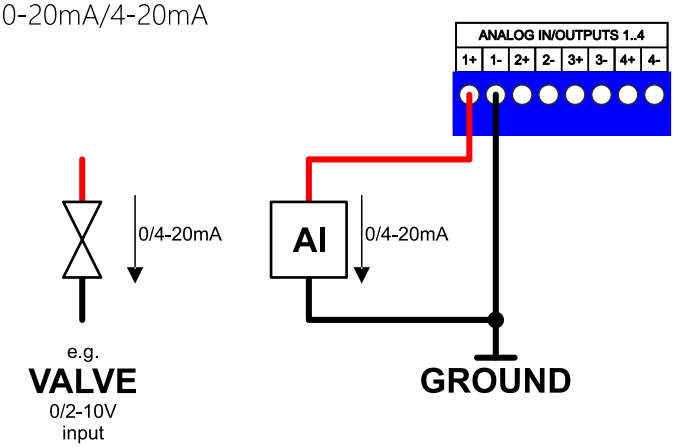



## ANALOG GROUND
All analog grounds are internally tied together!

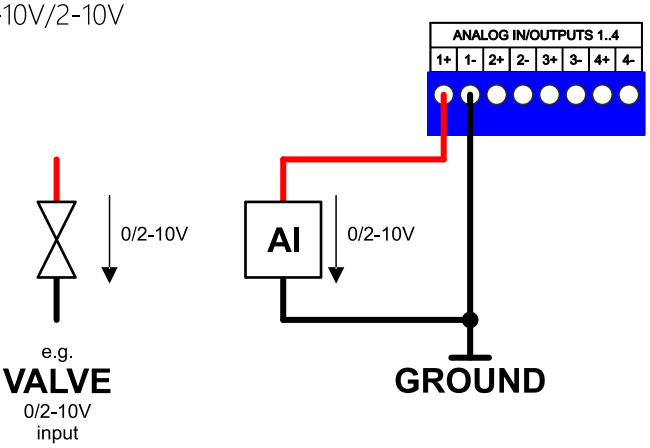
This affects to all AIOX inputs! But this analog ground is galvanically isolated from the M- of the module!




## ANALOG OUTPUT
0-20mA/4-20mA



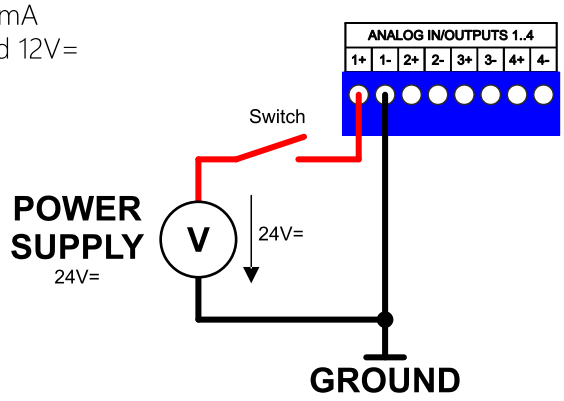

## ANALOG OUTPUT
0-10V/2-10V



# RESI-C4-A-32DI24RO16AIOX-xGB Universal analog IOs - possibilities

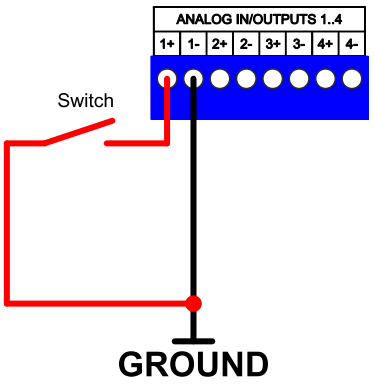
## DIGITAL INPUT

≦40V, 1.8mA
Threshold 12V=
Logic



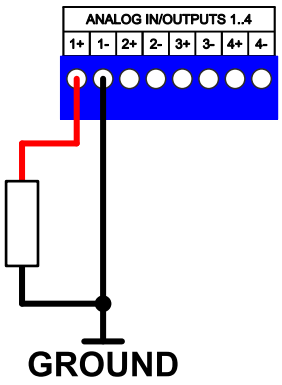

## DIGITAL INPUT

Loop powered
4mA




## RESISTOR INPUT

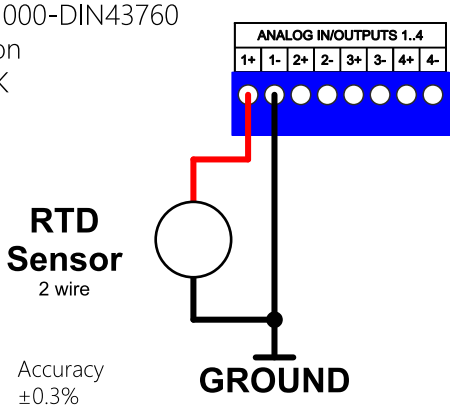
0-1MΩ




| Measurement Range | Accuracy |
|---|---|
| 0-80Ω | ±0.5%±0.5Ω |
| 80-200Ω | ±0.3% |
| 200-1kΩ | ±0.2% |
| 1k-10kΩ | ±0.2% |
| 10k-20kΩ | ±0.3% |
| 20k-100kΩ | ±0.8% |
| 100k-200kΩ | ±1.0% |
| 200k-1MΩ | ±8% |

## RESISTOR INPUT

PT100, PT1000, NI1000-DIN43760
internal linearisation
Output: °C,°F or °K




| Sensor type | Accuracy |
|---|---|
| PT100 | ±0.3% |
| PT1000 | ±0.2% |
| NI1000-DIN43760 | ±0.2% |

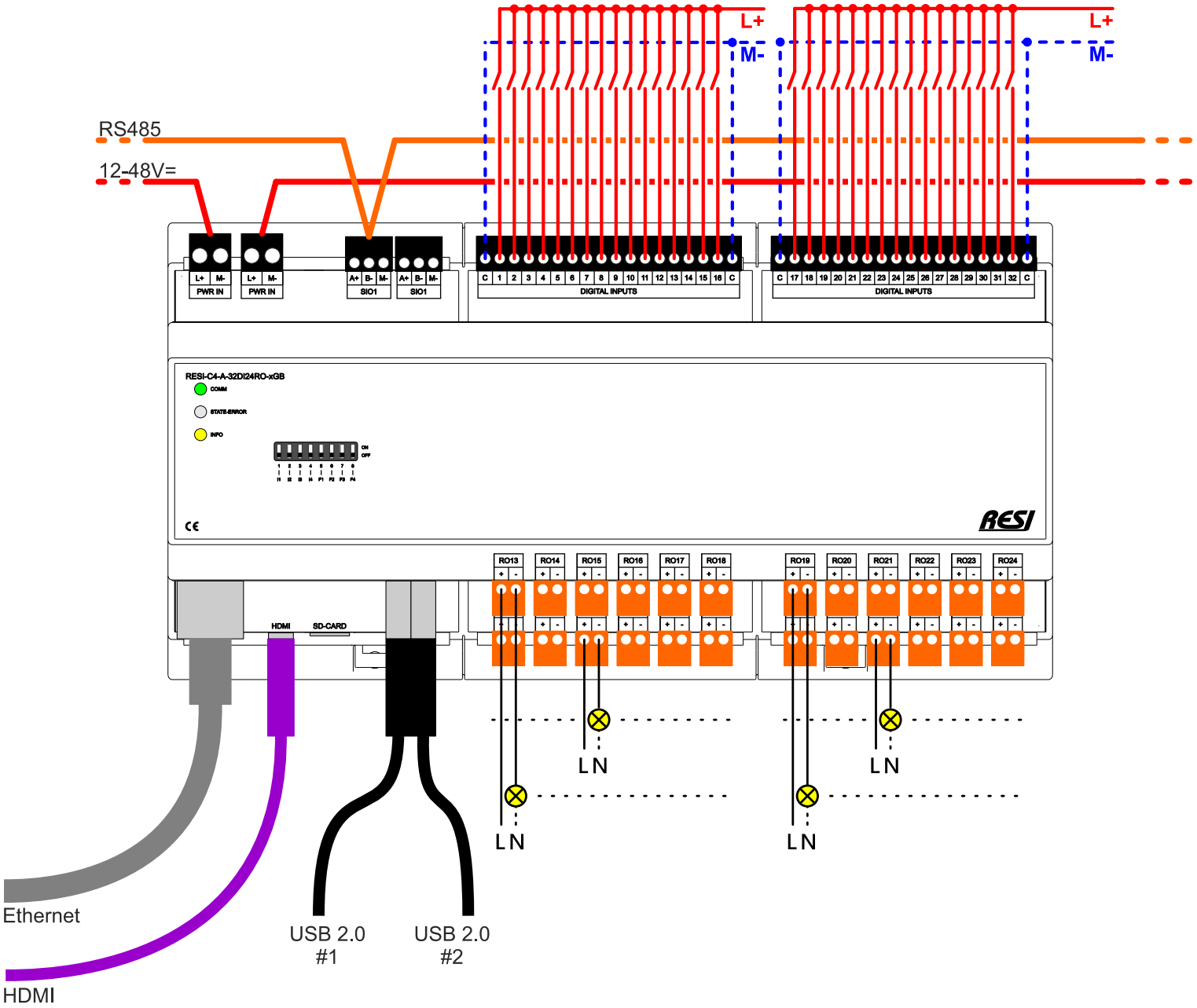
# Our RESI-C4-A-32DI24RO-xGB controllers

HIGHLIGHTS

- Raspberry Compute Module 4
- 2GB or 4GB or 8GB RAM
- 1xSD-CARD Slot with 32GB card
- 1xEthernet Interface
- 2xUSB 2.0
- 1xMicro-HDMI for 4k Monitor
- 1xRS485
- 8-pin DIP switch for software use
- 4 Status LEDs for Software use (green, white, red and yellow)

- 32 digital inputs for ≦48V= signals
- 24 form A relay outputs for ≦30V=, ≦250V=, ≦6A

# RESI-C4-A-32DI24RO-xGB
# Scematic

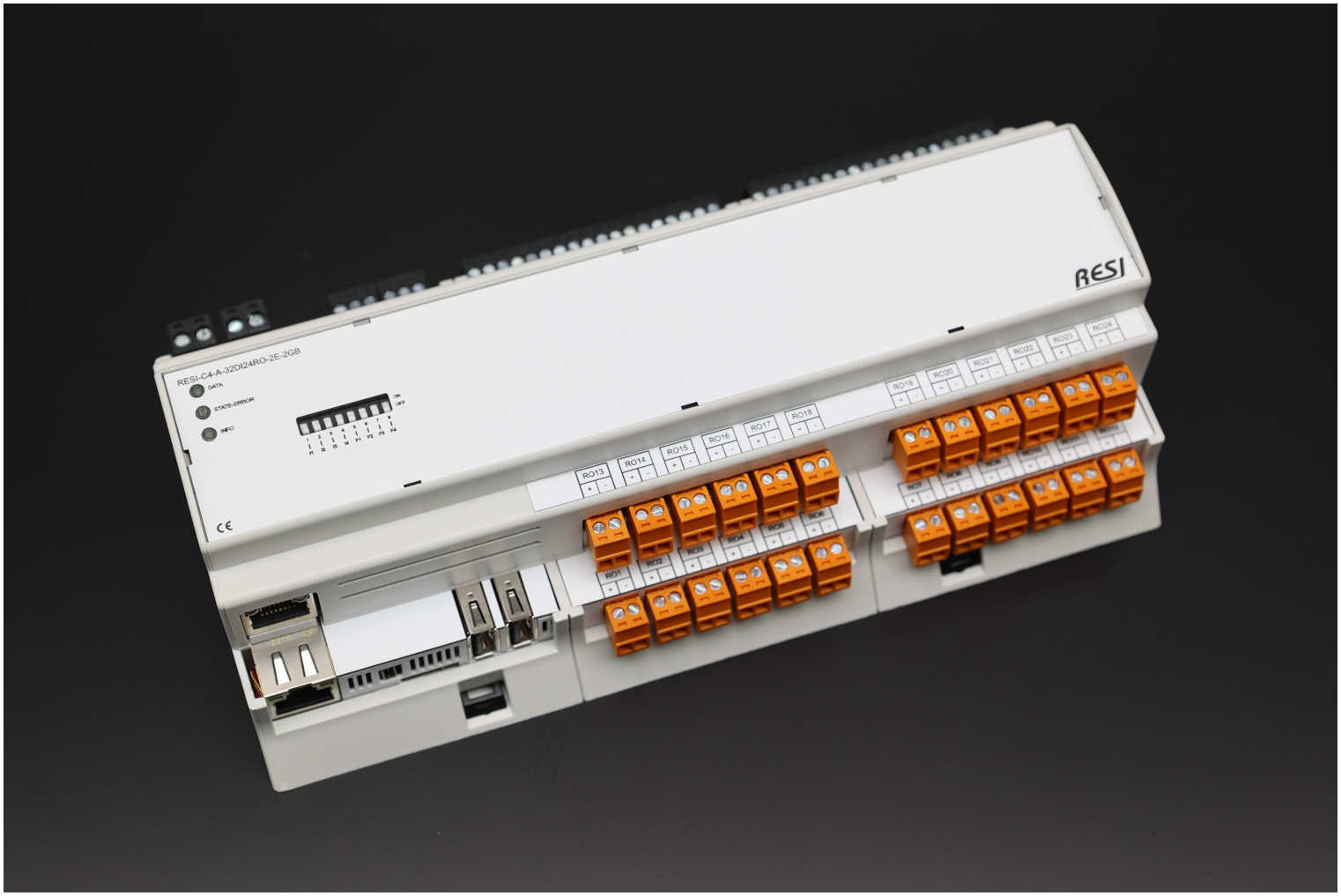


# Our RESI-C4-A-32DI24RO16-2E-xGB controllers

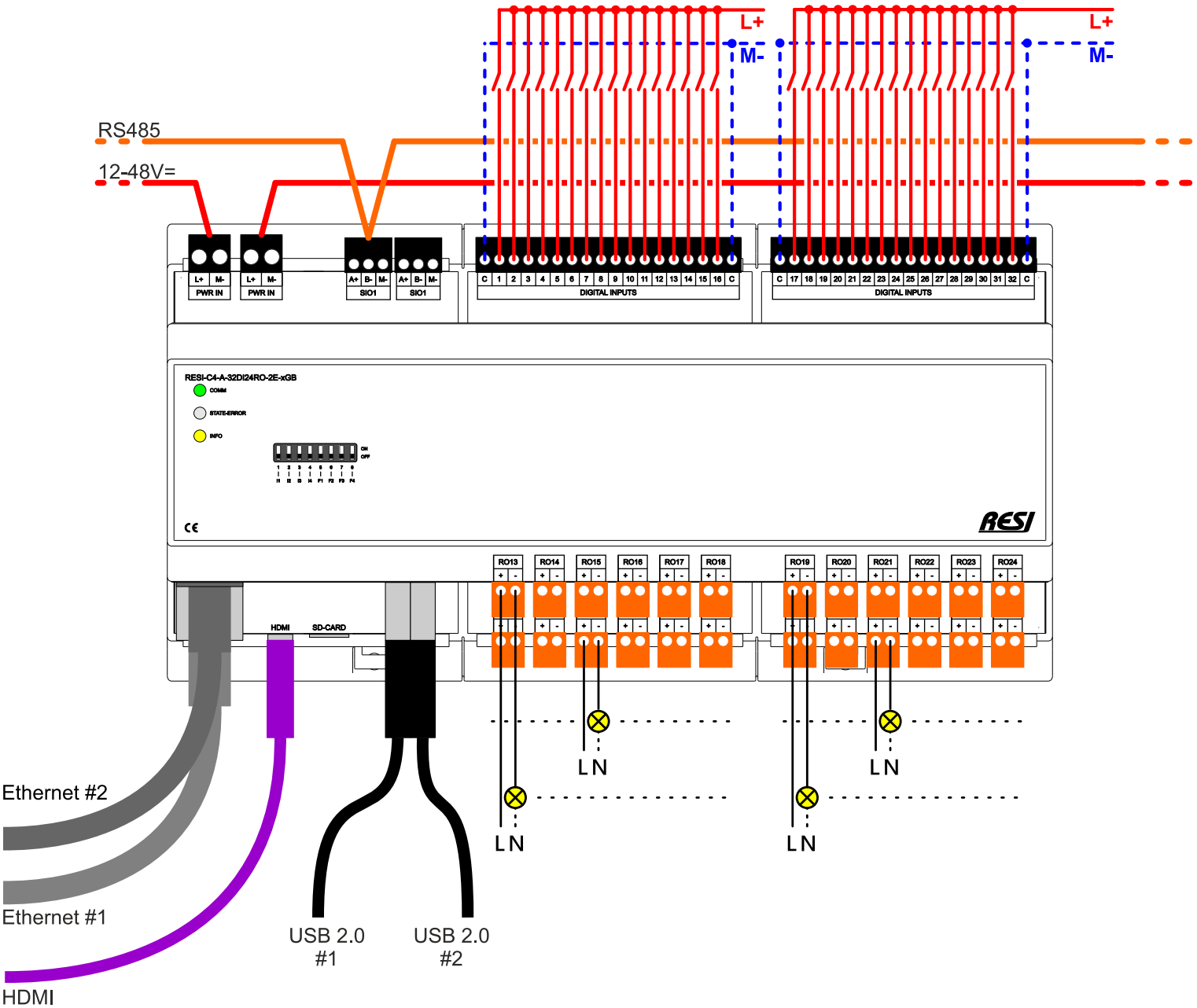
HIGHLIGHTS

- Raspberry Compute Module 4
- 2GB or 4GB or 8GB RAM
- 1xSD-CARD Slot with 32GB card
- 2xEthernet Interface for router or bridge appllications
- 2xUSB 2.0
- 1xMicro-HDMI for 4k Monitor
- 1xRS485
- 8-pin DIP switch for software use
- 4 Status LEDs for Software use (green, white, red and yellow)

- 32 digital inputs for ≦48V= signals
- 24 form A relay outputs for ≦30V=, ≦250V=, ≦6A



# RESI-C4-A-32DI24RO-2E-xGB
# Scematic



# Install NodeRED® components

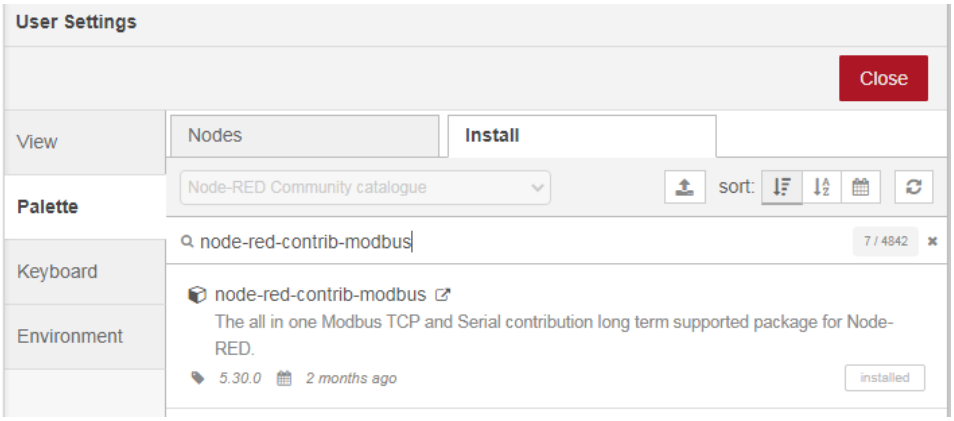
Raspberry Pi is a trademark of the Raspberry Pi Foundation.
More information under www.raspberrypi.org

it´s all about perfection

# Install NodeRED® components node-red-contrib-modbus

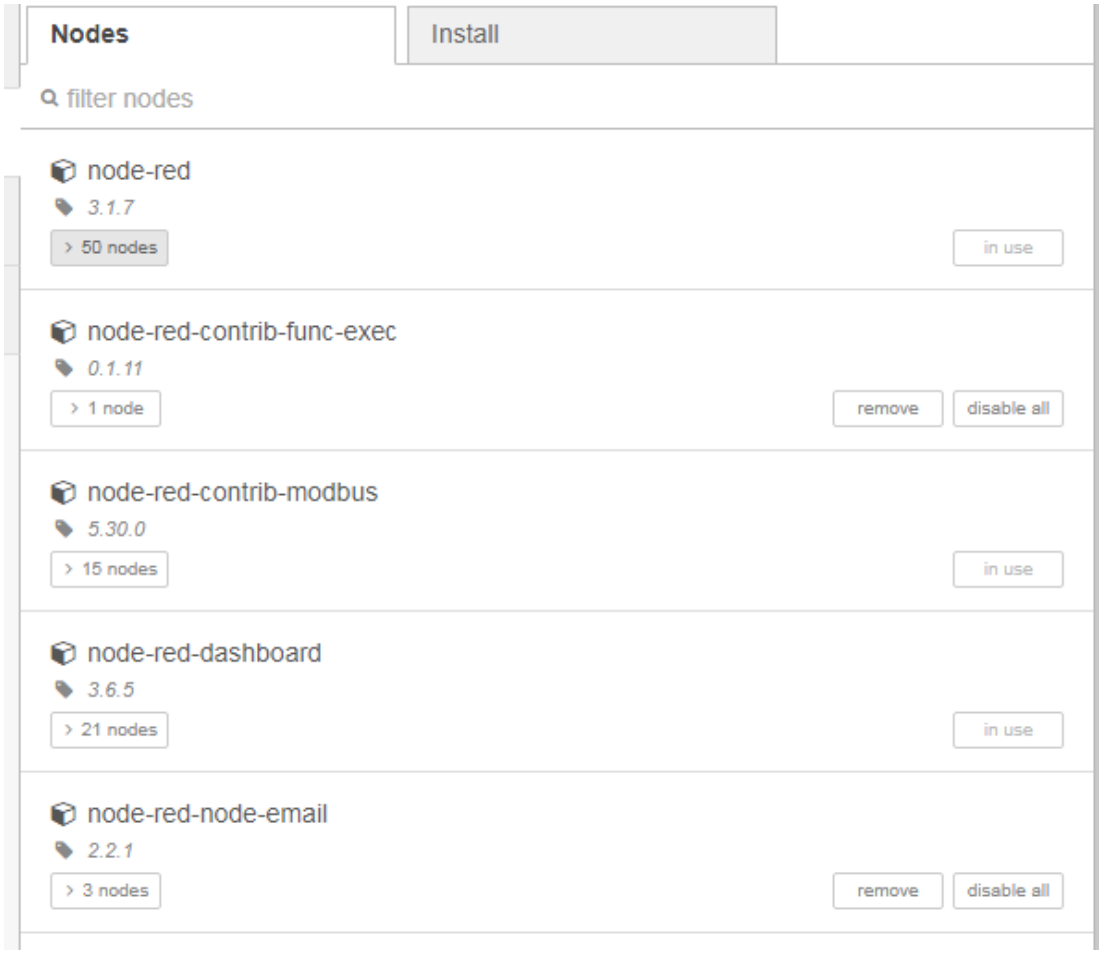
INSTALL node-red-contrib-modbus

Open menu "Manage palette" and select tab install. Enter node-red-contrib-modbus in the search field. You should see an similar screen:




Click on Install. We have installed this component already.

We have additionally installed the components node-red-dashboard for creating a simple UI



# Important NodeRED® nodes explained

Raspberry Pi is a trademark of the Raspberry Pi Foundation.
More information under www.raspberrypi.org

it´s all about perfection

# Important NodeRED® nodes explained

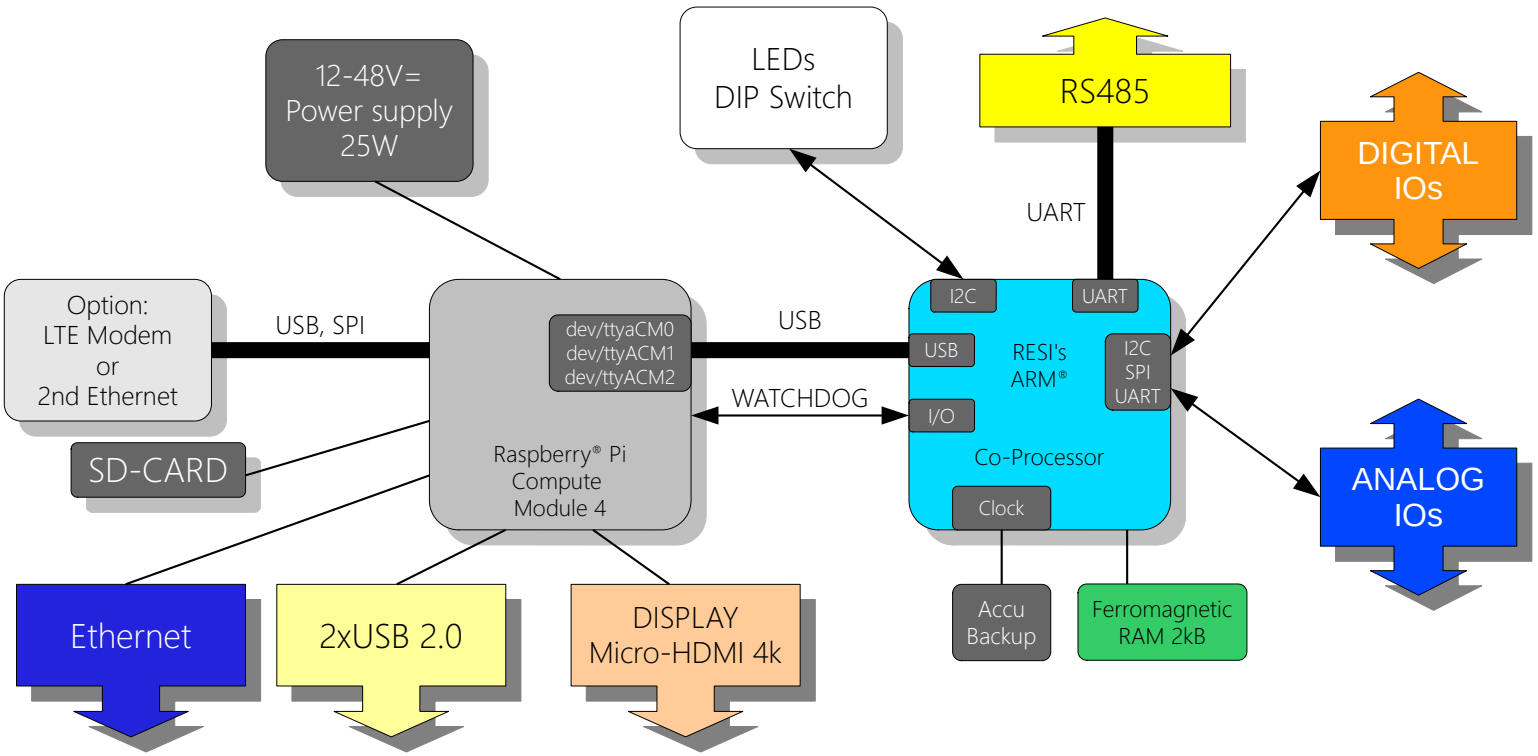
BASIC PRINCIPLE OF RESI-C4 controllers
All of our RESI-C4 controller are based on the following hardware architecture. We have extended the Compute Module 4 with an ARM co-processor,, which is connected via USB to the LINUX system. All time critical actions are driven by the ARM firmware and not by LINUX. Therefore there is no real time issue or latency problem due LINUX in handling all IOs, RS485 and other peripheral units.

The most important issue is the fact that there are three additional interfaces available in LINUS named dev/ttyACM0 to dev/ttyACM2

dev/ttyACM0: This is the internal connection to the ARM co-processor using simple ASCII text strings master slave protocol.
LINUX is the ASCII master and the ARM co-processor is the ASCII slave.

dev/ttyACM1: This is the internal connection to the ARM co-processor using MODBUS/RTU protocol. LINUX is the MODBUS/RTU master and the ARM co-processor is the MODBUS/RTU slave.

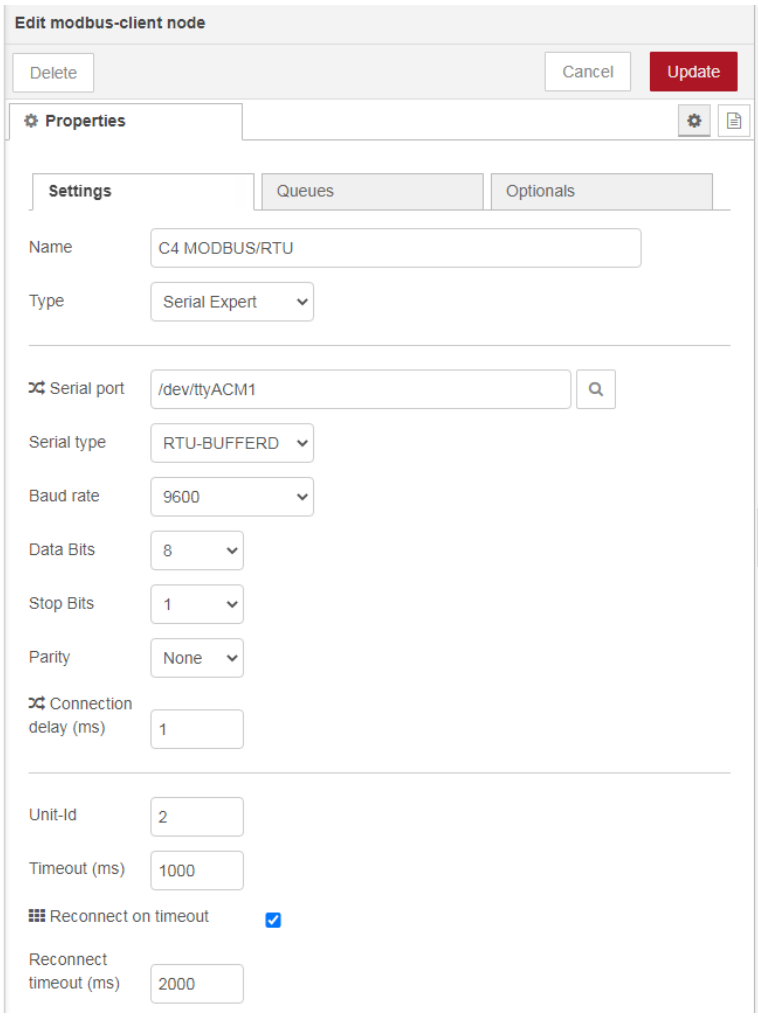
dev/ttyACM2: This is the internal connection to the ARM co-processor for using the native RS485 interface like it was directly coupled to the LINUX. The direction switching is done in real-time from the ARM co-processor.

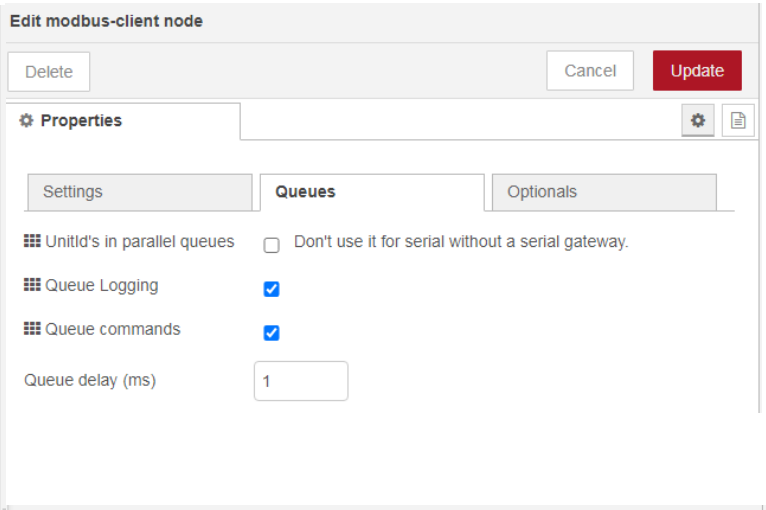


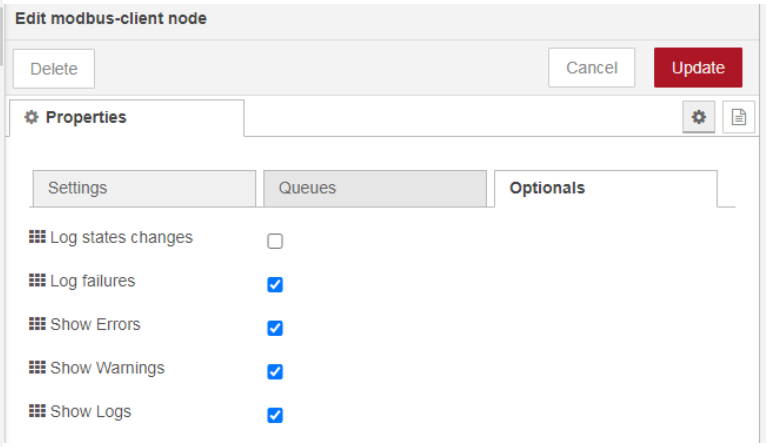
# Important NodeRED® nodes explained

Configuration node: modbus-client node for internal ARM communcation

This node is responsible for communicating with MODBUS/RTU master protocol to our internal ARM co-processor and therefore all IOs can be handled by this interface. But also the DIP switch and the LEDs are handled over this interface. Important is only the device /dev/ttyACM1. The baudrate parameters are not important due to the fact that this is a direct USB communication.



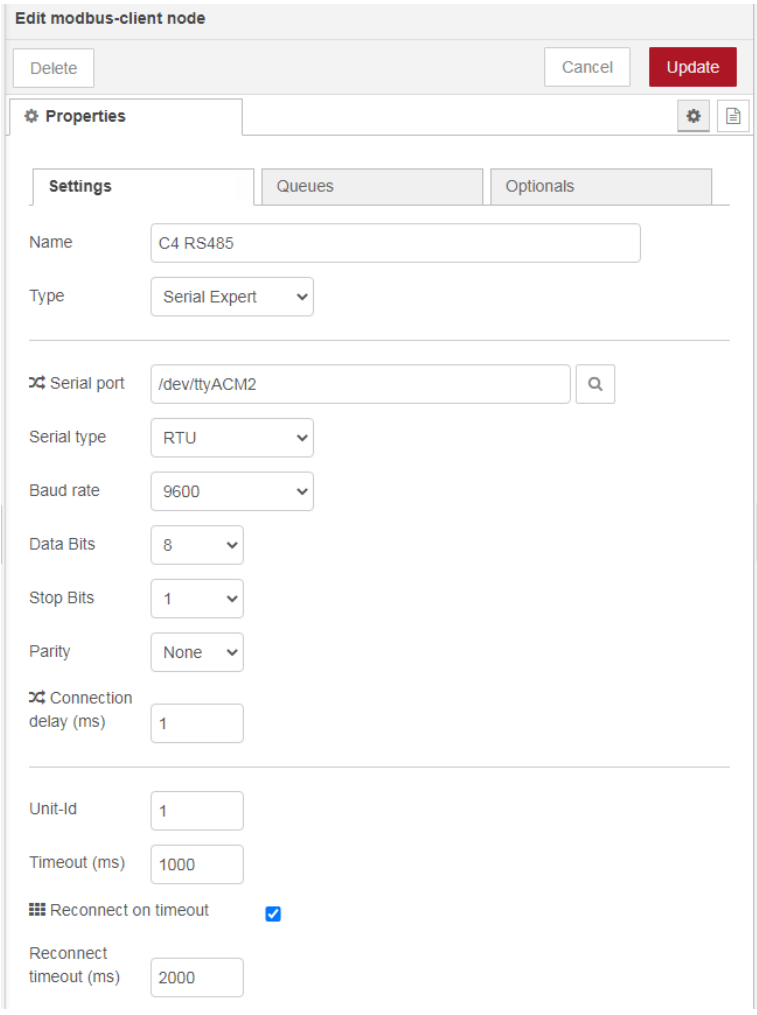



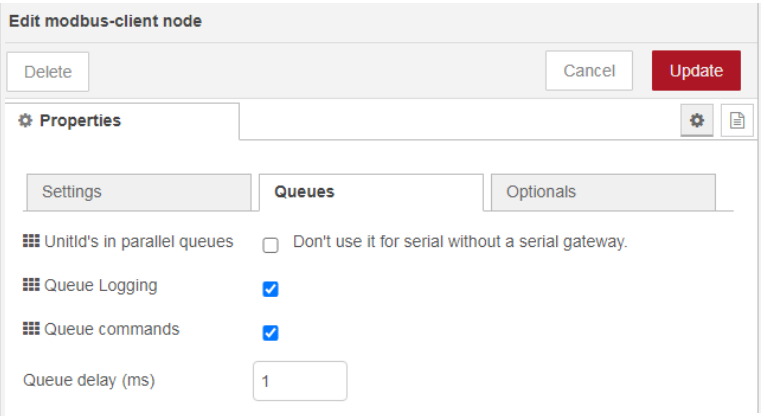



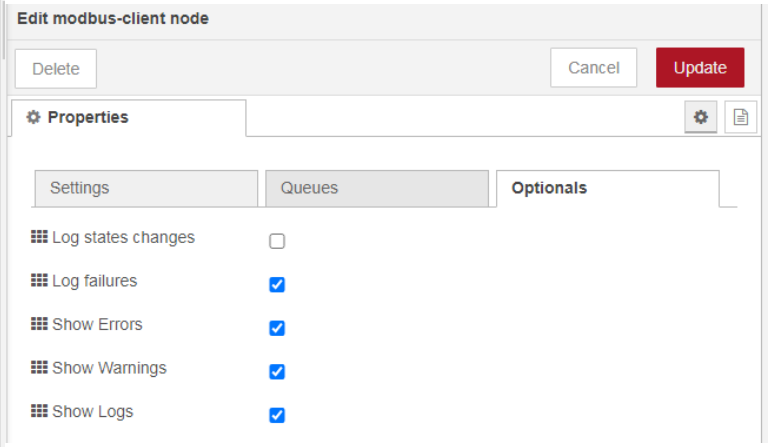
# Important NodeRED® nodes explained

Configuration node: modbus-client node for external RS485 connected devices

This node is responsible for communicating with MODBUS/RTU master protocol with IO modules or other MODBUS devices connected th the RS4895 on our C4 controllers. Important is the correct device **/dev/ttyACM2.** The baud rate settings depend on the connected IO modules or MODBUS/RTU slave devices. So adopt them correctly for your used devices!



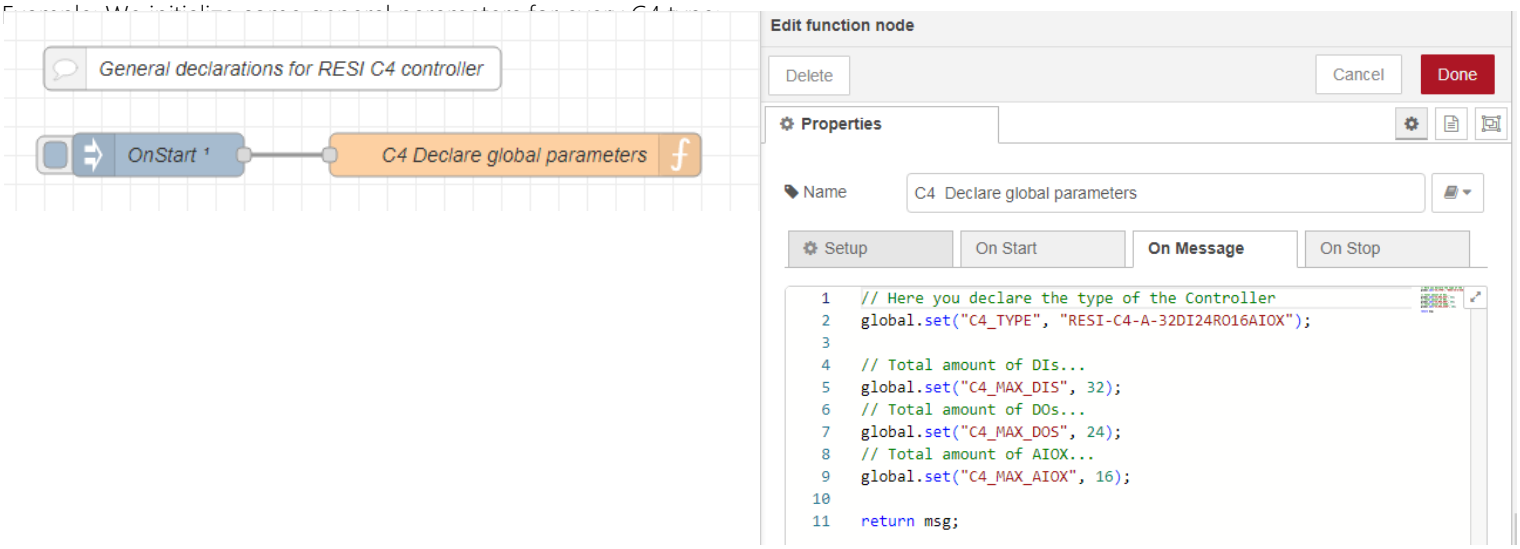



# Important NodeRED® nodes explained

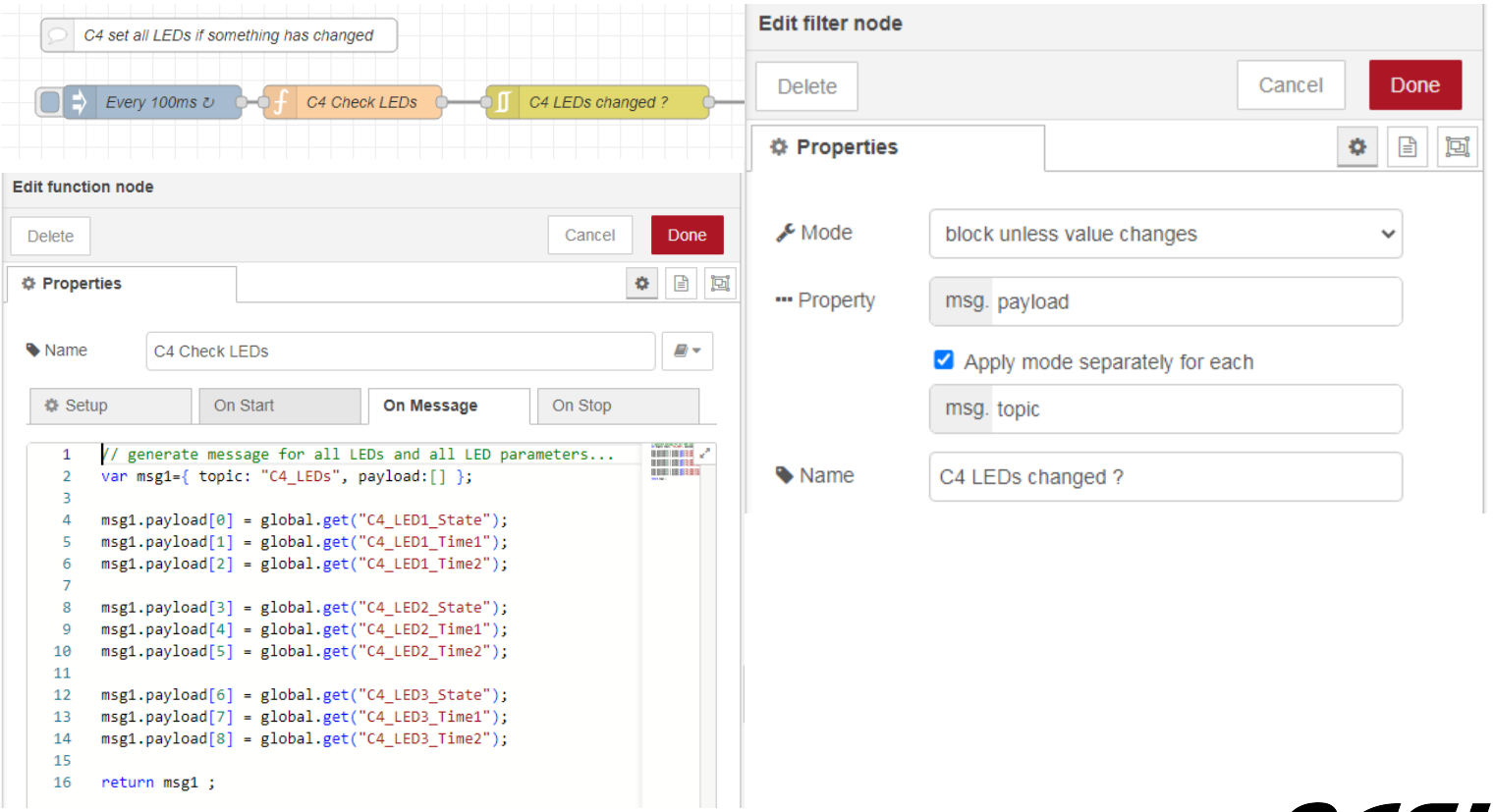
## global.set and global.get

In our software we always use global variables representing all C4 IOs or internal parameters. So we use in our function nodes very often global.get to read the current value of a global variable. Also we use global.set to define a new value for a global variable. So this allows us to use all IOs and C4 features in all flows in our Node-RED project. Check the section context data → Global to see the actual values of all global variables.




## Wait for change node

To trigger MODBUS writes or UI updates only if there is a change in values, we use very often the filter node. This node will trigger its output only, if the incoming message has changed. Therefore we have to prepare in a function node preceding the filter node a message with all relevant data in payload, which we want to monitor for a change.



# Important NodeRED® nodes explained

## Modbus-Read node

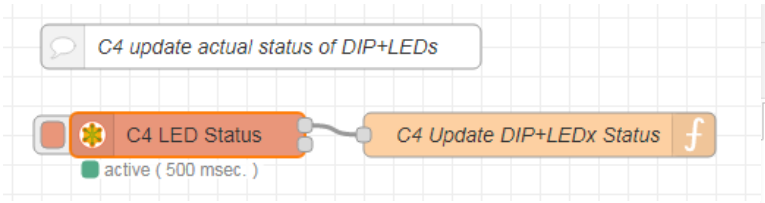
With this node we read MODBUS holding registers or inputs or coils from our ARM co-processor. The UnitID of our ARM is always 1 and the poll rate is setup according to the importance of the values we want to refresh.

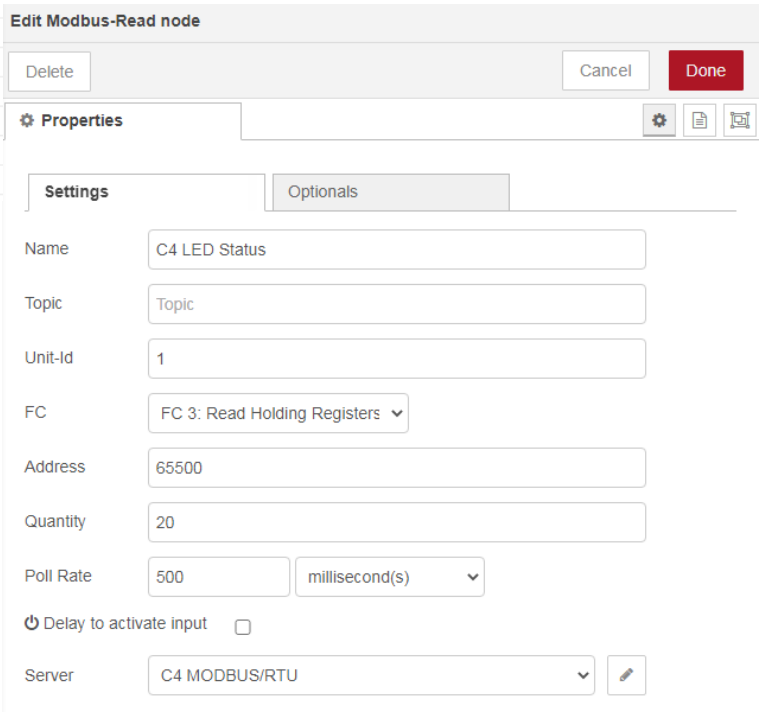
**address** defines the start index. But be carefully This is a index starting with 0. In our documentation for all ASCII commands and MODBUS registers (e.g. for the RESI-C4-32DI24RO16AIOX-xGB this document is called **RESI-L-C4-A-32DI12RO16AIOX-xGB-MODBUS+ASCII-EN.pdf**) we use the definition:

| | |
|---|---|
| 3x65501 | This is the index meaning a MODBUS holding register (3x → FC3) according to MODBUS nomenclature It starts with 1! |
| 4x65501 | This is the index meaning a MODBUS input register (4x → FC4) according to MODBUS nomenclature. It starts with 1! |
| I:65500 | This is the index you have to use in NodeRED. It starts with 0 |

**Quantity** defines the amount of registers we will read with one command. For holding and input registers this must be ≦125. For coils or input bits ≦2000.

So the FC can be FC3: Read holding register or FC4: read input register to read 16 bit register. But be aware you can only write to FC4 holding registers!
Or it can be FC1: Read coil status or FC2: Read input status to read 1-bit coils or inputs.



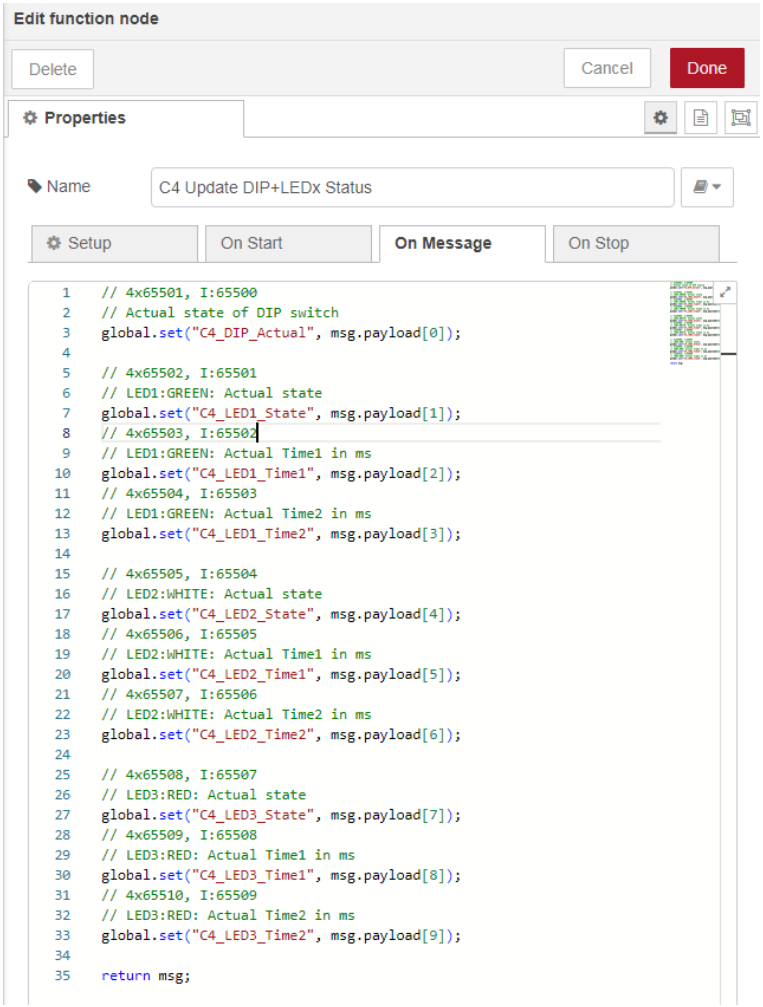

Sample from our MODBUS register documentation for the DIP switch

| DIP SWITCH STATUS | | | | | | |
|---|---|---|---|---|---|---|
| DIP SWITCH | 3x65501<br>4x65501<br>I:65500 | ???? | | | UINT16<br>R/O | |

Returns the actual setting of the Dip switches.
Bit 0: DIP Switch 1 (=0:OFF, =1:ON)
Bit 1: DIP Switch 2 (=0:OFF, =1:ON)
Bit 2: DIP Switch 3 (=0:OFF, =1:ON)
Bit 3: DIP Switch 4 (=0:OFF, =1:ON)
Bit 4: DIP Switch 5 (=0:OFF, =1:ON)
Bit 5: DIP Switch 6 (=0:OFF, =1:ON)
Bit 6: DIP Switch 7 (=0:OFF, =1:ON)
Bit 7: DIP Switch 8 (=0:OFF, =1:ON)

# Important NodeRED® nodes explained

In the succeeding node we have to handle the message with the data from the registers, we have read from the MODBUS. The msg.payload represents an array (in our sample 20 elements) with 16 bit values. Every entry in the array represents a 16 bit value from 0 to 65535, meaning the content of the holding or input register we have read-out.



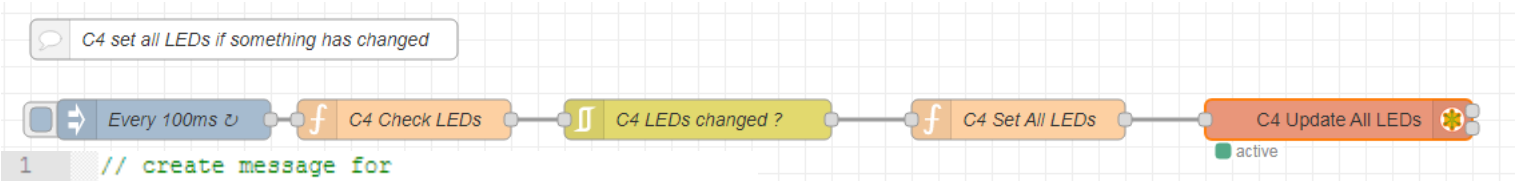
# Important NodeRED® nodes explained

Modbus-flex-write node

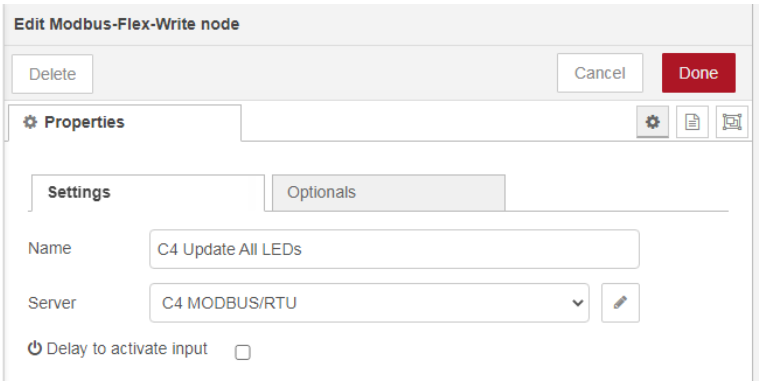
With this node we write MODBUS holding registers or coils to our ARM co-processor. The UnitID of our ARM is always 1.
To use this flex write node, we have to prepare a special message. The payload represents a structure with the following parts:
'fc': stands for the function code we want to use. 16 is write multiple holding registers.
The node supports the following write actions:

5: Force Single Coil to write one bit
6: Preset Single Register to write one 16-bit holding register
15: Force Multiple Coils: to write multiple bits and
16: Preset Multiple Registers to write multiple 16-bit holding registers.

'unitid' is always 1 for our ARM co-processor.
'address' is the starting index in the holding registers.
Again use the I:ddddd entry from our tables, because this function starts with index 0.
'quantity' is the amount of holding registers we want to write.
value is an array of 16-bit values from 0 to 65535 representing the new values for every holding register.
In our sample we use 9 new values for the three LEDs.



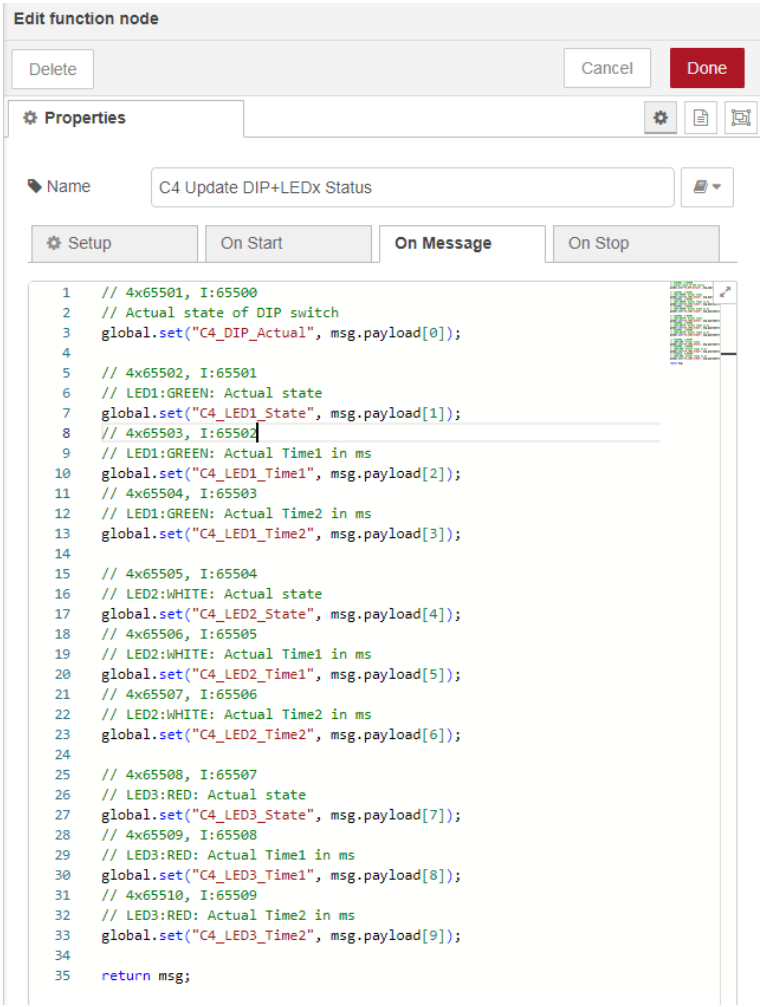

```
// create message for
// MODBUS WRITE MULTIPLE HOLDING REGISTERS
let C4_LEDs=[ 0,0,0, 0,0,0, 0,0,0];

// 4x65502, I:65501
// LED1:GREEN: Actual state
C4_LEDs[0]=global.get("C4_LED1_State");
// 4x65503, I:65502
// LED1:GREEN: Actual Time1 in ms
C4_LEDs[1]=global.get("C4_LED1_Time1");
// 4x65504, I:65503
// LED1:GREEN: Actual Time2 in ms
C4_LEDs[2]=global.get("C4_LED1_Time2");

// 4x65505, I:65504
// LED2:WHITE: Actual state
C4_LEDs[3]=global.get("C4_LED2_State");
// 4x65506, I:65505
// LED2:WHITE: Actual Time1 in ms
C4_LEDs[4]=global.get("C4_LED2_Time1");
// 4x65507, I:65506
// LED2:WHITE: Actual Time2 in ms
C4_LEDs[5]=global.get("C4_LED2_Time2");

// 4x65508, I:65507
// LED3:RED: Actual state
C4_LEDs[6]=global.get("C4_LED3_State");
// 4x65509, I:65508
// LED3:RED: Actual Time1 in ms
C4_LEDs[7]=global.get("C4_LED3_Time1");
// 4x65510, I:65509
// LED3:RED: Actual Time2 in ms
C4_LEDs[8]=global.get("C4_LED3_Time2");
```




```
msg.payload = {
    value: [
        C4_LEDs[0], C4_LEDs[1], C4_LEDs[2],
        C4_LEDs[3], C4_LEDs[4], C4_LEDs[5],
        C4_LEDs[6], C4_LEDs[7], C4_LEDs[8]
    ],
    // WRITE MULTIPLE REGISTERS
    'fc': 16,
    // INTERNAL UnitID of C4
    'unitid': 1,
    // START ADDRESS: 4x65502,I:65501
    'address': 65501,
    // 3 LEDS with 3 vaklues each...
    'quantity': 9
}
return msg;
```

# Important NodeRED® nodes explained

In the succeeding node we have to handle the message with the data from the registers, we have read from the MODBUS. The msg.payload represents an array (in our sample 20 elements) with 16 bit values. Every entry in the array represents a 16 bit value from 0 to 65535, meaning the content of the holding or input register we have read-out.

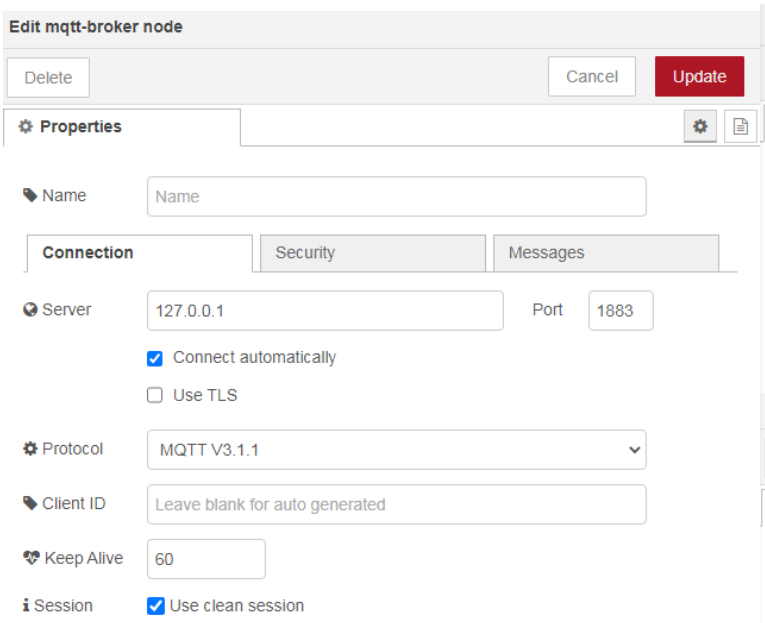


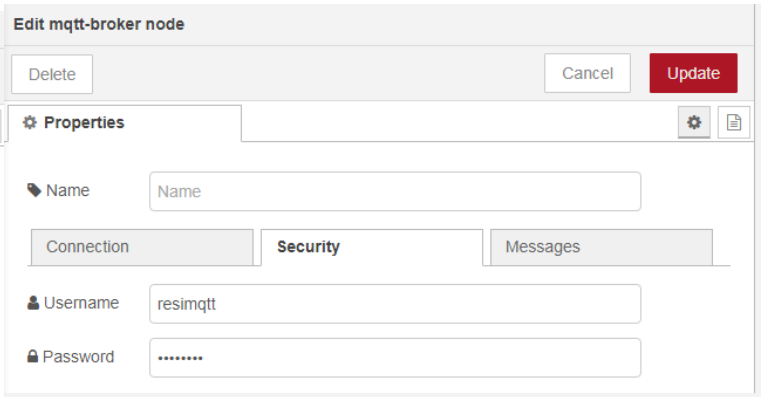
# Important NodeRED® nodes explained

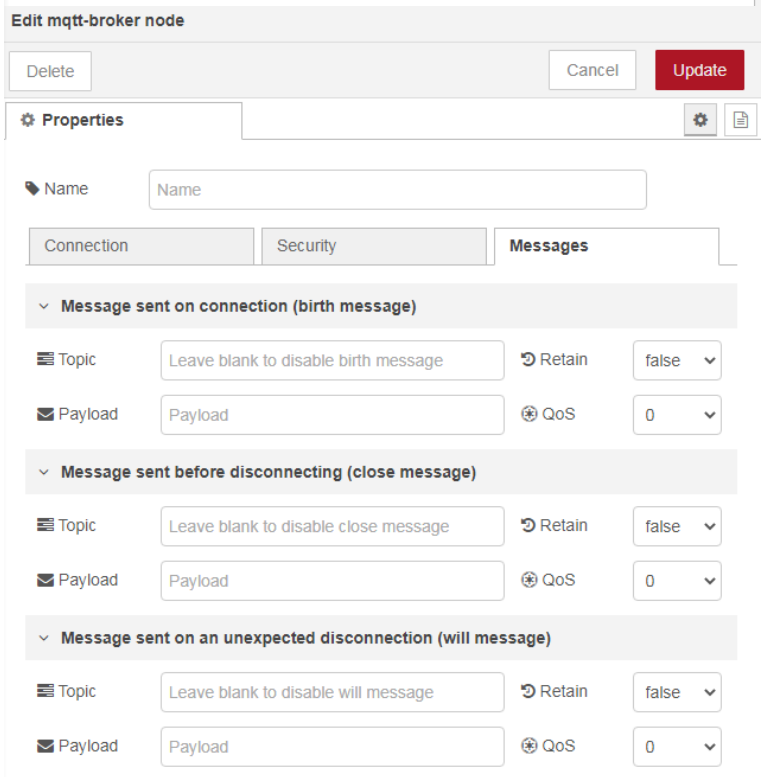
## Configuration node: MQTT broker

In many of our samples we use the MQTT protocol to send actual status from our IOs to the cloud or to receive some control messages to switch digital or analog outputs. Therefore we use mosquitto MQTT broker under LINUX as the MQTT broker. The SW is installed on the C4 locally. We use no special security. But if you use this for real cloud connections, use all available security features for the MQTT connection and VPN to protect your communication against unwanted intruders!

user: resiMQTT
password: r4MQTT



# Important NodeRED® nodes explained

### Receive and test MQTT messages

Please consult the internet, how the mosquitto server is configured and used.
To test the MQTT broker use the command to view all incoming messages of your MQTT server:

```
mosquitto_sub -t RESI_C4/# -d -u resimqtt -P r4MQTT
```

To switch the digital output DO2 ON, use this command

```
mosquitto_pub -t RESI_C4/DigitalOutputs/Cx_DO2 -u resimqtt -P r4MQTT -m 1
```

To switch the digital output DO2 OFF, use this command

```
mosquitto_pub -t RESI_C4/DigitalOutputs/Cx_DO2 -u resimqtt -P r4MQTT -m 0
```

To switch the digital outputs DO1 to DO8  ON or OFF, use this command

```
mosquitto_pub -t RESI_C4/DigitalOutputs/Cx_DO1 -u resimqtt -P r4MQTT -m 1
mosquitto_pub -t RESI_C4/DigitalOutputs/Cx_DO1 -u resimqtt -P r4MQTT -m 0
...
mosquitto_pub -t RESI_C4/DigitalOutputs/Cx_DO8 -u resimqtt -P r4MQTT -m 1
mosquitto_pub -t RESI_C4/DigitalOutputs/Cx_DO8 -u resimqtt -P r4MQTT -m 0
```

As soon as NodeRED sends a new MQTT status you will receive similar logging output:

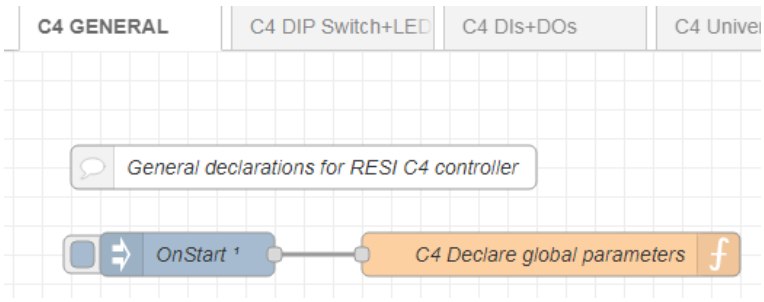
```
0
Client (null) received PUBLISH (d0, q0, r0, m0, 'RESI_C4/DigitalOutputs/C4_DO04', ... (1 bytes))
0
Client (null) received PUBLISH (d0, q0, r0, m0, 'RESI_C4/DigitalOutputs/C4_DO05', ... (1 bytes))
1
Client (null) received PUBLISH (d0, q0, r0, m0, 'RESI_C4/DigitalOutputs/C4_DO06', ... (1 bytes))
0
Client (null) received PUBLISH (d0, q0, r0, m0, 'RESI_C4/DigitalOutputs/C4_DO07', ... (1 bytes))
0
Client (null) received PUBLISH (d0, q0, r0, m0, 'RESI_C4/DigitalOutputs/C4_DO08', ... (1 bytes))
0
Client (null) received PUBLISH (d0, q0, r0, m0, 'RESI_C4/DigitalOutputs/C4_DO01', ... (1 bytes))
1
Client (null) received PUBLISH (d0, q0, r0, m0, 'RESI_C4/DigitalOutputs/C4_DO02', ... (1 bytes))
0
Client (null) received PUBLISH (d0, q0, r0, m0, 'RESI_C4/DigitalOutputs/C4_DO03', ... (1 bytes))
0
Client (null) received PUBLISH (d0, q0, r0, m0, 'RESI_C4/DigitalOutputs/C4_DO04', ... (1 bytes))
0
Client (null) received PUBLISH (d0, q0, r0, m0, 'RESI_C4/DigitalOutputs/C4_DO05', ... (1 bytes))
1
Client (null) received PUBLISH (d0, q0, r0, m0, 'RESI_C4/DigitalOutputs/C4_DO06', ... (1 bytes))
0
Client (null) received PUBLISH (d0, q0, r0, m0, 'RESI_C4/DigitalOutputs/C4_DO07', ... (1 bytes))
0
Client (null) received PUBLISH (d0, q0, r0, m0, 'RESI_C4/DigitalOutputs/C4_DO08', ... (1 bytes))
1
```

# NodeRED®
# sample flows explained

# NodeRED® flow
# C4-GENERAL+C4-DIP Switch+LEDs

## C4 GENERAL

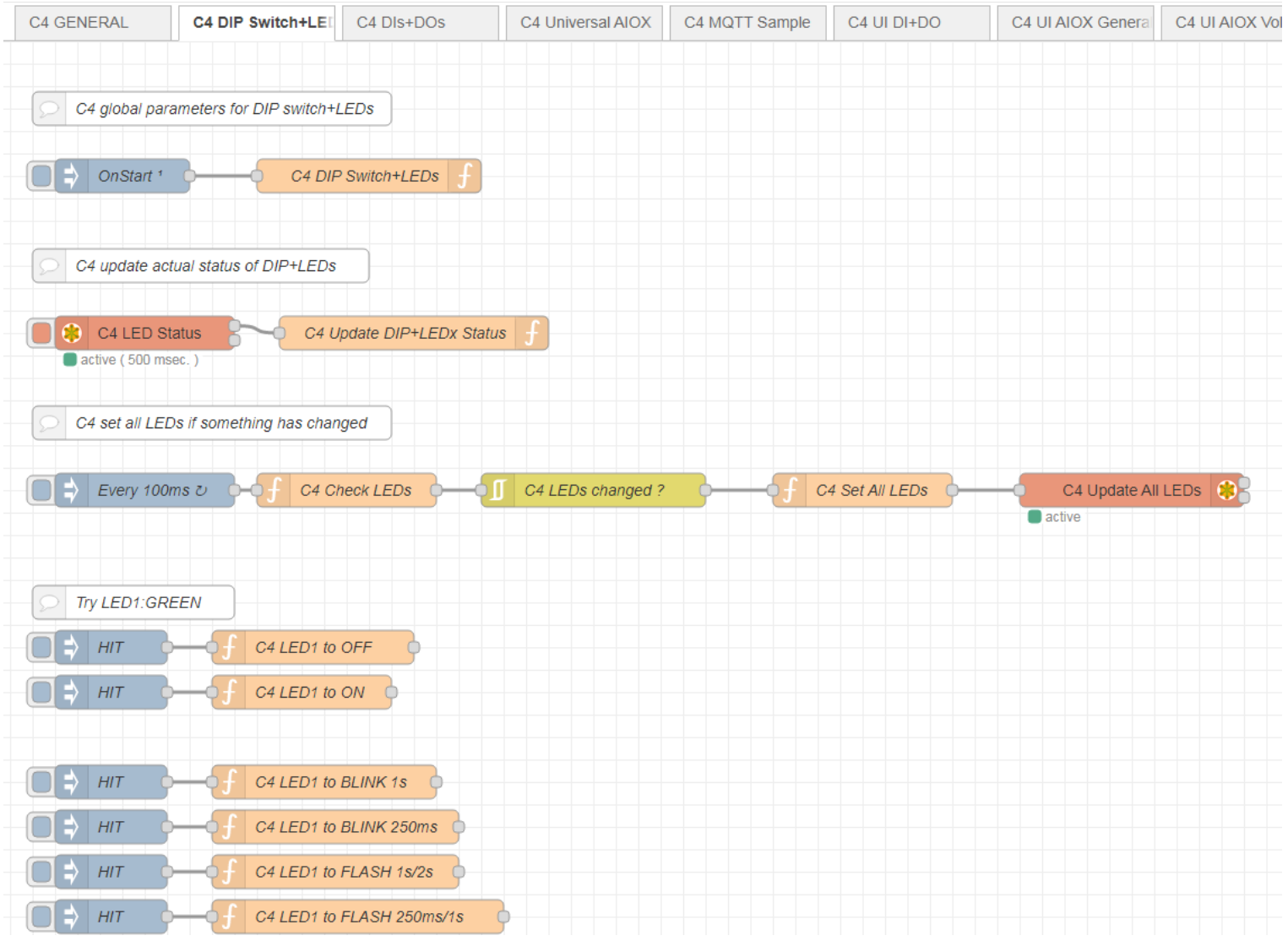
With this flow we define general parameters for the C4 controller like the exact type and the amount of IOs.




```
// Here you declare the type of the Controller
global.set("C4_TYPE", "RESI-C4-A-32DI24RO16AIOX");

// Total amount of DIs...
global.set("C4_MAX_DIS", 32);
// Total amount of DOs...
global.set("C4_MAX_DOS", 24);
// Total amount of AIOX...
global.set("C4_MAX_AIOX", 16);

return msg;
```

## C4 DIP Switch+LEDs

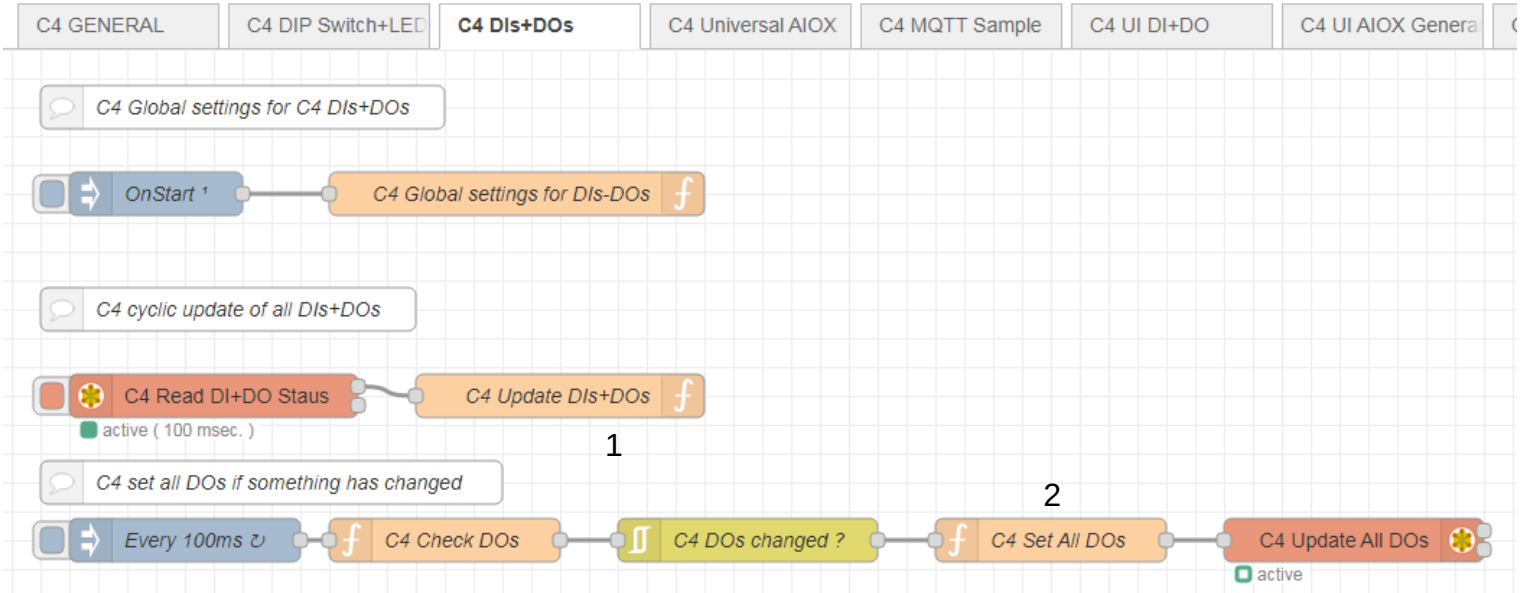
With this flow we read every 500ms the actual status for the DIPs witch and LEDs from the Co-processor and update the global variables. Also we check  every 100ms, if there is a new mode to write to the LEDs. If there is a change we update all 3 LEDs with one MODBUS write cycle. To test the LEDs, there are test flows for every LED to set a specific mode interactively.



# NodeRED® flow
# C4-DIs+DOs

C4 DIs+DOs

With this flow we read every 100ms the current status of all digital inputs & outputs. Then we update the global variables C4_DIxx, representing for every digital input the actual state with 0 or 1. Also we update the actual DOs in the global variables C4_DOxx.
Then we check every 100ms if there is a need to update the digital output state. If yes, we prepare a message with the new status for all DOs and use a flex write to update the ARM co-processor.




1

```
var i;
let Word=0,Bit=0,V=0;

// Register list for C4-A-32DI24RO16AIOX
// 4x10001,I:10000: Change counter
// 4x10002,I:10001: DIGITAL INPUTS DI1-DI16
// 4x10003,I:10002: DIGITAL INPUTS DI17-DI32
// 4x10004,I:10003: DIGITAL OUTPUTS DO1-DO16
// 4x10005,I:10004: DIGITAL OUTPUTS DO17-DO24

// n DIs
for (i = 0; i < global.get("C4_MAX_DIS");i++) {
    Word=~~(i/16);
    Bit=~~(i%16);
    // to take register 4x10002,I:10001 or 4x10003,I:10002
    V=(msg.payload[1+Word]&(1<<Bit)) ? 1:0;
    global.set("C4_DI"+String(i+1).padStart(2, '0'),V);
}

// n ROs
for (i = 0; i < global.get("C4_MAX_DOS"); i++) {
    Word=~~(i/16);
    Bit=~~(i%16);
    // to take register 4x10004,I:10003 or 4x10005,I:10004
    V=(msg.payload[3+Word]&(1<<Bit)) ? 1:0;
    global.set("C4_DO"+String(i+1).padStart(2, '0')+"_Actual",V);
}

return msg ;
```

2

```
var i;

// for C4-A-32DI24RO16AIOX controller ...
// for C4-A-32DI24RO controller ...
if (global.get("C4_TYPE")=="RESI-C4-A-32DI24RO16AIOX" ||
    global.get("C4_TYPE")=="RESI-C4-A-32DI24RO")
{
    // Our controller offers 24 Relay outputs
    // therefore we need 2 16 Bit registers
    // 4x10004,I:10003: OUTPUTS DO1-DO16
    // 4x10005,I:10004: OUTPUTS DO17-DO24
    let C4_DOS=[ 0,0];
    let Word=0,Bit=0;

    // 24RO controller
    C4_DOS[0] =C4_DOS[1]=0;

    // Check for new value of C4_DOxx
    for (i=0;i<global.get("C4_MAX_DOS");i++)
    {
        Word=~~(i/16);
        Bit=~~(i%16);
        if (global.get("C4_DO" +String(i+1).padStart(2, '0'))==1)
        {
            C4_DOS[Word] = C4_DOS[Word]|(1<<Bit);
        }
    }

    msg.payload = {
        'value': [ C4_DOS[0], C4_DOS[1] ],
        // WRITE MULTIPLE HOLDING REGISTERS
        'fc': 16,
        // Our C4 MODBUS UnitID
        'unitid': 1,
        // 4x10004,I:10003: OUTPUTS DO1-DO16
        // 4x10005,I:10004: OUTPUTS DO17-DO24
        'address': 10003 ,
        // for the 24 DOs or 2 registers...
        'quantity': 2
    }
}
else
{
    msg.payload=null;
}
return msg;
```

# NodeRED® flow
# C4-universal AIs+AOs

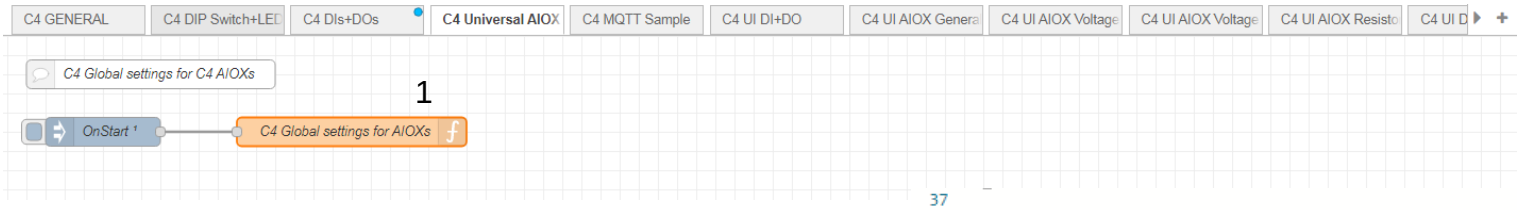
C4 Universal AIOX

With this flow we handle the universal analog inputs and outputs. This flow needs some global variables. xx stands for the number of the AIO: 01 to 16 for 16AIOX

C4_AIOXxx_TYPE: The type of the universal analog input/output. Allowed are the following types:

=0: UNUSED
=1: VOLTAGE INPUT[0-10V]
=2: VOLTAGE INPUT[2-10V]
=3: VOLTAGE OUTPUT[0-10V]
=4: VOLTAGE OUTPUT[2-10V]
=5: CURRENT INPUT LOOP POWERED[0-20mA]
=6: CURRENT INPUT LOOP POWERED[4-20mA]
=7: CURRENT INPUT EXTERNAL POWERED[0-20mA]
=8: CURRENT INPUT EXTERNAL POWERED[4-20mA]
=9: CURRENT OUTPUT[0-20mA]
=10: CURRENT OUTPUT[4-20mA]
=11: DIGITAL INPUT LOGIC 24V=
=12: DIGITAL INPUT LOOP POWERED
=13: RESISTANCE MEASUREMENT

At the end of the function node you can defined the startup types for the universal IOs. It will be stored in the controller even if the controller looses power.



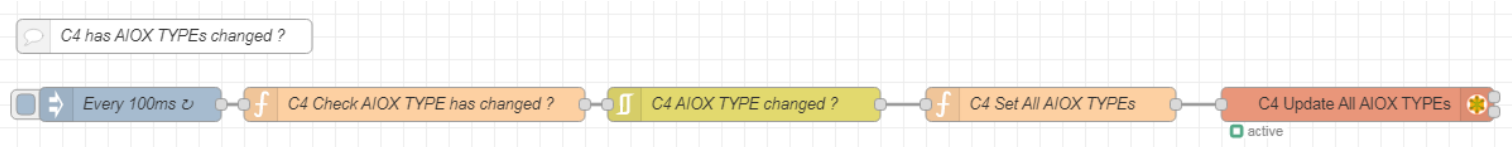

1

```
var i,nr;

// n AIOXs ....

global.set("C4_AIOX_ISONLINE");


for (i=0;i<global.get("C4_MAX_AIOX");i++)
{
  nr = String(i + 1).padStart(2, '0');
  // type of AIOX: 0=Unused


  global.set("C4_AIOX" + nr + "_TYPE", 0);

  // Output value in Volt for AIOX
  global.set("C4_AIOX" + nr + "_VO", 0);

  // Input value in Volt for AIOX
  global.set("C4_AIOX" + nr + "_VI", 0);

  // Input value in mA for AIOX
  global.set("C4_AIOX" + nr + "_CI", 0);

  // Input value in Ohm for AIOX
  global.set("C4_AIOX" + nr + "_OHM", 0);

  // Input value in PT100 °C for AIOX
  global.set("C4_AIOX" + nr + "_PT100_C", 0);

  // Input value in PT1000 °C for AIOX
  global.set("C4_AIOX" + nr + "_PT1000_C", 0);

  // Input value in NI1000-DIN43760 °C for AIOX
  global.set("C4_AIOX" + nr + "_NI1000_DIN43760_C", 0);
}

// possible IO types for AIOXx:
// =0: UNUSED
// =1: VOLTAGE INPUT[0-10V]
// =2: VOLTAGE INPUT[2-10V]
// =3: VOLTAGE OUTPUT[0-10V]
// =4: VOLTAGE OUTPUT[2-10V]
// =5: CURRENT INPUT LOOP POWERED[0-20mA]
// =6: CURRENT INPUT LOOP POWERED[4-20mA]
// =7: CURRENT INPUT EXTERNAL POWERED[0-20mA]
// =8: CURRENT INPUT EXTERNAL POWERED[4-20mA]
// =9: CURRENT OUTPUT[0-20mA]
// =10: CURRENT OUTPUT[4-20mA]
// =11: DIGITAL INPUT LOGIC 24V=
// =12: DIGITAL INPUT LOOP POWERED
// =13: RESISTANCE MEASUREMENT

// Configure your AIOX types here ...
global.set("C4_AIOX01_TYPE", 3);
global.set("C4_AIOX02_TYPE", 1);
global.set("C4_AIOX03_TYPE", 3);
global.set("C4_AIOX04_TYPE", 1);
global.set("C4_AIOX05_TYPE", 13);
global.set("C4_AIOX06_TYPE", 13);
global.set("C4_AIOX07_TYPE", 13);
global.set("C4_AIOX08_TYPE", 13);
global.set("C4_AIOX09_TYPE", 3);
global.set("C4_AIOX10_TYPE", 1);
global.set("C4_AIOX11_TYPE", 3);
global.set("C4_AIOX12_TYPE", 1);
global.set("C4_AIOX13_TYPE", 3);
global.set("C4_AIOX14_TYPE", 1);
global.set("C4_AIOX15_TYPE", 3);
global.set("C4_AIOX16_TYPE", 1);

return msg;
```

# NodeRED® flow
# C4-universal AIs+AOs

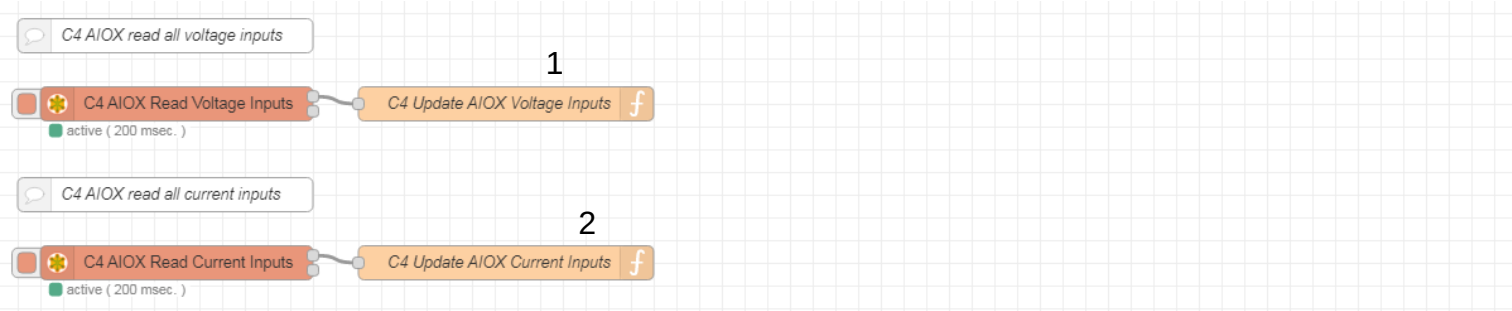
C4 has AIOX TYPESs changed ?

With this flow we check every 100ms if there is a new AIOX type. If yes, we update all AIOX types with one flex write to 16 holding registers.




The next flows read cyclically the current configured AIOX types and a status information, if the AIOX component is working correctly and is online. The current configured AIOX types are stored in the global variables C4_AIOXxx_TYPE_Actual. The Online status of the AIOX components is save in C4_AIOX_ISONLINE.


The next flows read the actual input voltage for all 16 AIOX and save the values into C4_AIOXxx_VI as a numeric value in Volt.
The same we do for the current inputs: We read the actual input current for all 16 AIOX and save the values into C4_AIOXxx_CI as a numeric value in mA.




1

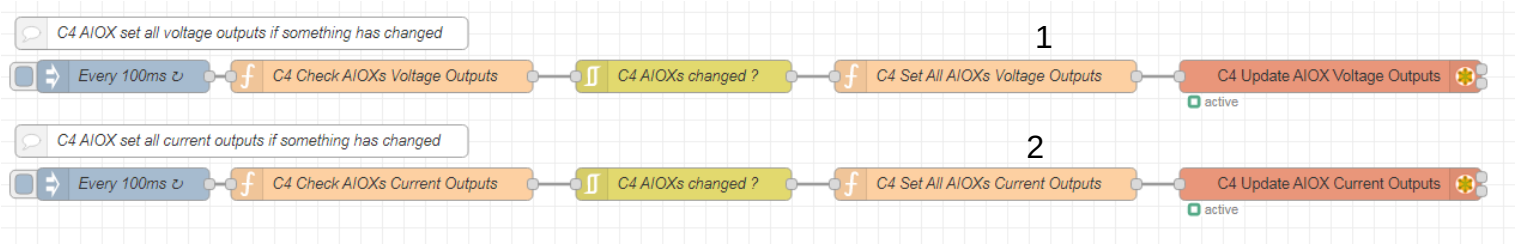
```
var i;

// Register list for C4-A-32DI24RO16AIOX
// 4x40017,I:40016: VOLTAGE INPUT in VOLT*100
// 4x40018,I:40017: VOLTAGE INPUT in VOLT*100
// ...
// 4x40032,I:40031: VOLTAGE INPUT in VOLT*100

// n AIOX
for (i = 0; i < global.get("C4_MAX_AIOX");i++) {
    let V=msg.payload[i]/100.0;
    global.set("C4_AIOX"+String(i+1).padStart(2, '0')+"_VI",V);
}

return msg ;
```

2

```
var i;

// Register list for C4-A-32DI24RO16AIOX
// 4x40097,I:40096: CURRENT INPUT in MILLIAMPERE*100
// 4x40098,I:40097: CURRENT INPUT in MILLIAMPERE*100
// ...
// 4x40111,I:40110: CURRENT INPUT in MILLIAMPERE*100

// n AIOX
for (i = 0; i < global.get("C4_MAX_AIOX");i++) {
    let V=msg.payload[i]/100.0;
    global.set("C4_AIOX"+String(i+1).padStart(2, '0')+"_CI",V);
}

return msg ;
```

# NodeRED® flow
# C4-universal AIs+AOs

Set voltage and current outputs.
the next two flows will update every 100ms the voltage and current outputs. Therefore we check, if the contents of the global variables C4_AIOXxx_VO or C4_AIOXxx_CO have changed. if yes, we update either all 16 voltage outputs or all 16 current outputs with a MODBUS flex write node.




For both updates we have to prepare the message with the correct MODBUS flex write parameters.and with the correct 16 bit value for the holding registers.

1

```
var i;
let V;
// for C4-A-32DI24RO16AIOX controller ...
// for C4-A-32DI24RO controller ...
if (global.get("C4_TYPE")=="RESI-C4-A-32DI24RO16AIOX")
{
    // 4x40049,I:40048: VOLTAGE OUTPUT AIOX1 VOLTS*100
    // 4x40050,I:40049: VOLTAGE OUTPUT AIOX2 VOLTS*100
    // ...
    // 4x40064,I:40063: VOLTAGE OUTPUT AIOX16 VOLTS*100

    msg.payload = {
        'value': [ ],
        // WRITE MULTIPLE HOLDING REGISTERS
        'fc': 16,
        // Our C4 MODBUS UnitID
        'unitid': 1,
        'address': 40048,
        // for the 16 AIOX...
        'quantity': 16
    }

    for (i=0;i<global.get("C4_MAX_AIOX");i++)
    {
        V=Math.trunc(global.get("C4_AIOX" +String(i+1).padStart(2, '0')+"_VO")*100);
        msg.payload.value.push(V);
    }
}
else
{
    msg.payload=null;
}
return msg;
```

2

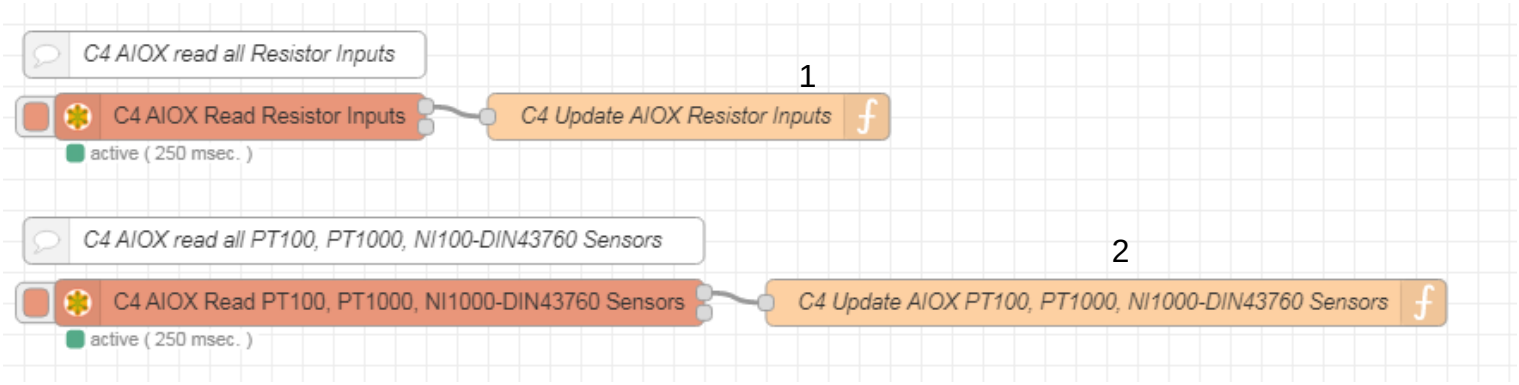
```
var i;
let V;
// for C4-A-32DI24RO16AIOX controller ...
// for C4-A-32DI24RO controller ...
if (global.get("C4_TYPE")=="RESI-C4-A-32DI24RO16AIOX")
{
    // 4x40129,I:40128: CURRENT OUTPUT AIOX1 MILLIAMPERE*100
    // 4x40130,I:40129: CURRENT OUTPUT AIOX1 MILLIAMPERE*100
    // ...
    // 4x40144,I:40143: CURRENT OUTPUT AIOX1 MILLIAMPERE*100

    msg.payload = {
        'value': [ ],
        // WRITE MULTIPLE HOLDING REGISTERS
        'fc': 16,
        // Our C4 MODBUS UnitID
        'unitid': 1,
        'address': 40128,
        // for the 16 AIOX...
        'quantity': 16
    }

    for (i=0;i<global.get("C4_MAX_AIOX");i++)
    {
        V=Math.trunc(global.get("C4_AIOX" +String(i+1).padStart(2, '0')+"_CO")*100);
        msg.payload.value.push(V);
    }
}
else
{
    msg.payload=null;
}
return msg;
```

# NodeRED® flow
# C4-universal AIs+AOs

Read resistor inputs

the next two flows will update every 250ms the resistor inputs. In the first flow we read all 16 resistor input values and save the in the global variables C4_AIOXxx_OHM. Then we read 48 consecutive registers with the calculated temperature for PT100, PT1000 or NI1000-DIN43760 sensors in °CELSIUS. After that we save the PT100 sensor values to C4_AIOXxx_PT100_C, the PT1000 sensor values to C4_AIOXxx_PT1000_C and the NI1000-DIN43760 sensor values to C4_AIOXxx_NI1000_DIN43760_C.

Which kind of variable you use in your application depends on the type of RTD sensor, you have connected to the AIOX input.




1

```
var i;

// Register list for C4-A-32DI24RO16AIOX
// 4x41501,I:41500: RESISTOR INPUT in OHM*100
// 4x41502,I:41501: RESISTOR INPUT in OHM*100
// ...
// 4x41531,I:41530: RESISTOR INPUT in OHM*100

// n AIOX
for (i = 0; i < global.get("C4_MAX_AIOX");i++) {
    let V=(msg.payload[i*2+1]+(msg.payload[i*2]<<16))/100.0;
    global.set("C4_AIOX"+String(i+1).padStart(2, '0')+"_OHM",V);
}

return msg ;
```

2

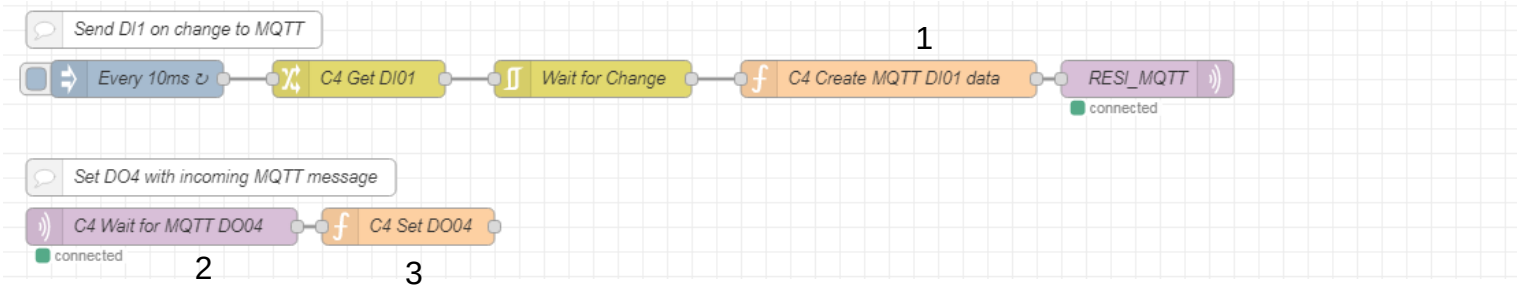
```
var i;
let V,sint16;

// Register list for C4-A-32DI24RO16AIOX
// 4x41049,I:41048: PT100 INPUT 1 in °C*100
// 4x41050,I:41049: PT100 INPUT 2 in °C*100
// ...
// 4x41064,I:41063: PT100 INPUT 16 in °C*100

// 4x41065,I:41064: PT1000 INPUT 1 in °C*100
// 4x41066,I:41065: PT1000 INPUT 2 in °C*100
// ...
// 4x41080,I:41079: PT1000 INPUT 16 in °C*100

// 4x41081,I:41080: NI1000-DIN43760 INPUT 1 in °C*100
// 4x41082,I:41081: NI1000-DIN43760 INPUT 2 in °C*100
// ...
// 4x41096,I:41095: NI1000-DIN43760 INPUT 16 in °C*100

// n AIOX
for (i = 0; i < global.get("C4_MAX_AIOX");i++) {
    // PT100...

    if (msg.payload[i]>32767) sint16=msg.payload[i]-65536;
    else sint16=msg.payload[i];
    V=sint16/100.0;
    global.set("C4_AIOX"+String(i+1).padStart(2, '0')+"_PT100_C",V);

    // PT1000...
    if (msg.payload[i+16]>32767) sint16=msg.payload[i+16]-65536;
    else sint16=msg.payload[i+16];
    V=sint16/100.0;
    global.set("C4_AIOX"+String(i+1).padStart(2, '0')+"_PT1000_C",V);

    // NI1000-DIN43760...
    if (msg.payload[i+32]>32767) sint16=msg.payload[i+32]-65536;
    else sint16=msg.payload[i+32];
    V=sint16/100.0;
    global.set("C4_AIOX"+String(i+1).padStart(2, '0')+"_NI1000_DIN43760_C",V);
}

return msg ;
```

# NodeRED® flow
# MQTT messages for DIs-DOs

MQTT messages

The next flows demonstrate howto send/receive messages via MQTT. The first node will check, if DI01 has changed its state. If yes, a new MQTT message will be generated and send to the MQTT broker.

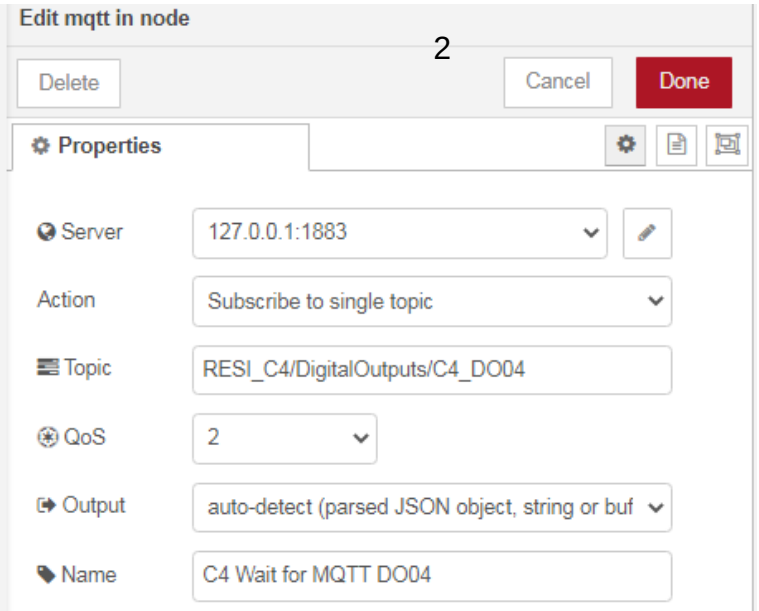
The second flow waits until a MQTT message will arrive. We check strictly for the message RESI_C4/DigitalOutputs/C4_DO04. If yes, we check the payload and set the digital output DO04 to the new arrived state. It's enough to set the global variable here, because the flow C4 DIs+DOs will check the change and update via MODBUS the outputs.




1

```
var msg1={
    topic:"RESI_C4/DigitalInputs/C4_DI01",
    payload: msg.payload
}

return msg1;
```

2



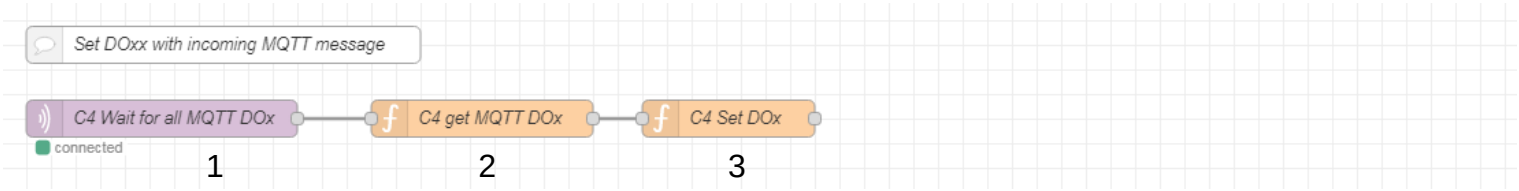

3

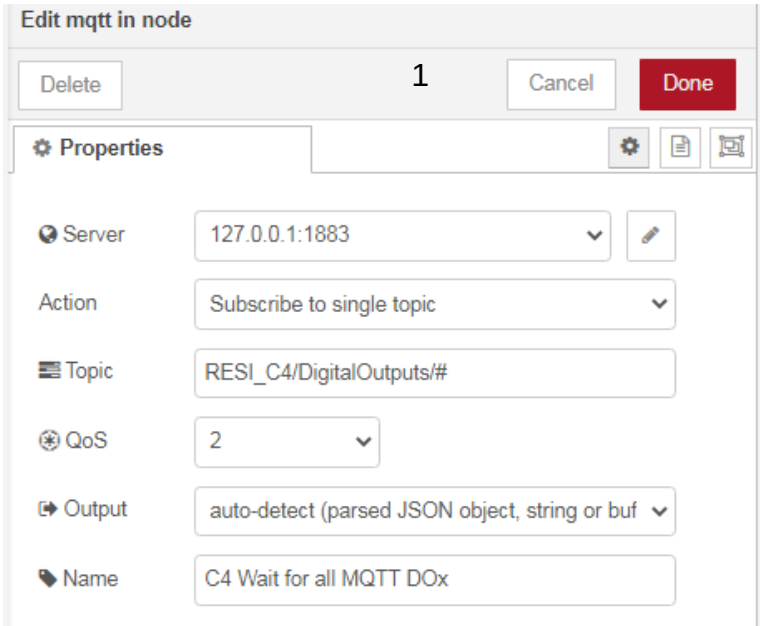
```
if (msg.payload == 0 || msg.payload == 1) {
    global.set("C4_DO04", msg.payload);
}

return msg;
```

# NodeRED® flow
# MQTT messages for DIs-DOs

MQTT messages

The next flows will trigger, whenever a MQTT message in the form RESI_C4/DigitalOutputs/.... will receive. The next node will then split the string and use the last word of the MQTT message as topic. The node C4 Set DOx will then check of the topic has the string format DOxx. If yes, the payload is checked for 0 or 1 to update the global variabel for the affected digital output. Again the flow C4 DIs+DOs will then update the digital output via MODBUS. We have only to write to the global variable here.




```
if (msg.topic.startsWith("RESI_C4/DigitalOutputs/")
{
    let DOx=msg.topic.split("/").pop();
    msg.topic=DOx;
}
return msg;
```

2

2

```
let ok=false;

if (msg.topic.startsWith="C4_DO")
{
  let DOnr=msg.topic.substring(5,100);
  if (Number.isInteger(parseInt(DOnr)))
  {
    let DOx=parseInt(DOnr,10);
    if (DOx>=1 && DOx<global.get("C4_MAX_DOS"))
    {
      ok=true;
    }
    if ((msg.payload == 0 || msg.payload == 1) && ok)
    {
      var DOxNAME="C4_DO"+String(DOx).padStart(2, '0');
      global.set(DOxNAME, msg.payload);
    }
  }
}
return msg;
```

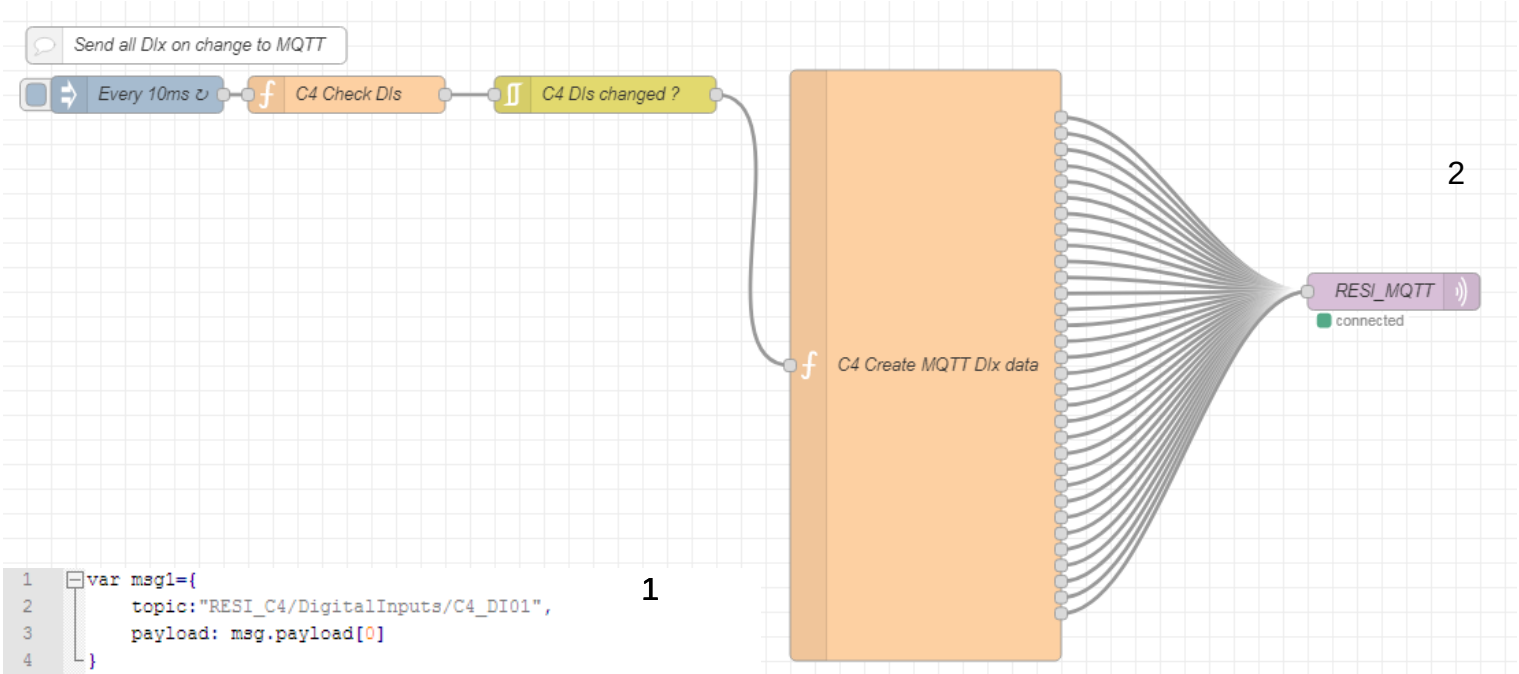
3

# NodeRED® flow
# MQTT messages for DIs+DOs

MQTT messages

The last flow will send new MQTT state messages for all digital inputs as soon as one digital input has changed its value. Therefore we use a function block with 32 output knots. For every output knto we prepare a new message named msg1 to msg32. The topc of every message defines the MQTT message in the form **RESI_C4/DigitalInputs/C4_DI01** to **RESI_C4/DigitalInputs/C4_DI32.** The payload is the current state of the digital input and is retrieved from the incoming message with an payload array of 32 elements, which we have build in the nodes before. Last but not least the mqtt-out node will then send all 32 messages to the MQTT broker.

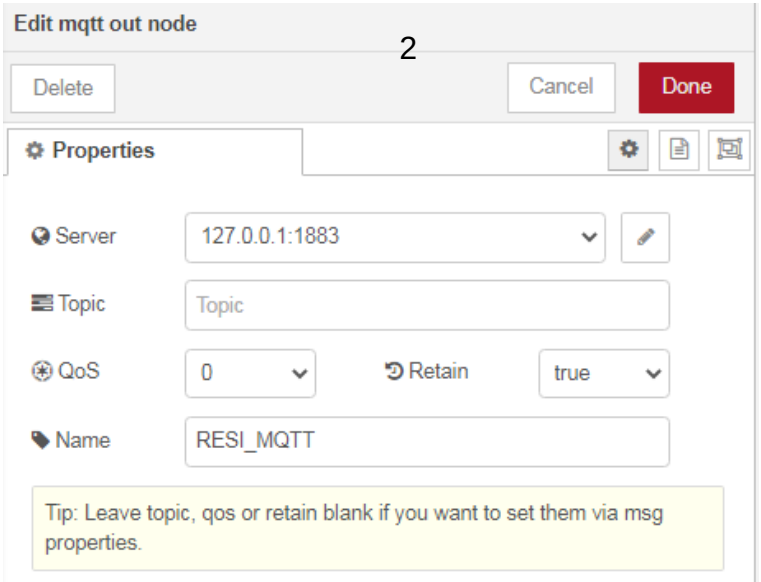
1




2

```
var msg1={
    topic:"RESI_C4/DigitalInputs/C4_DI01",
    payload: msg.payload[0]
}

var msg2={
    topic:"RESI_C4/DigitalInputs/C4_DI02",
    payload: msg.payload[1]
}

// do the same for all 32 digital inputs
....

var msg30={
    topic:"RESI_C4/DigitalInputs/C4_DI30",
    payload: msg.payload[29]
}

var msg31={
    topic:"RESI_C4/DigitalInputs/C4_DI31",
    payload: msg.payload[30]
}

var msg32={
    topic:"RESI_C4/DigitalInputs/C4_DI32",
    payload: msg.payload[31]
}


return [ msg1, msg2, msg3, msg4, msg5, msg6, msg7, msg8,
         msg9, msg10,msg11, msg12, msg13, msg14, msg15,
         msg16, msg17, msg18, msg19, msg20, msg21, msg22,
         msg23, msg24, msg25, msg26, msg27, msg27, msg29,
         msg30, msg31, msg32  ];
```

1

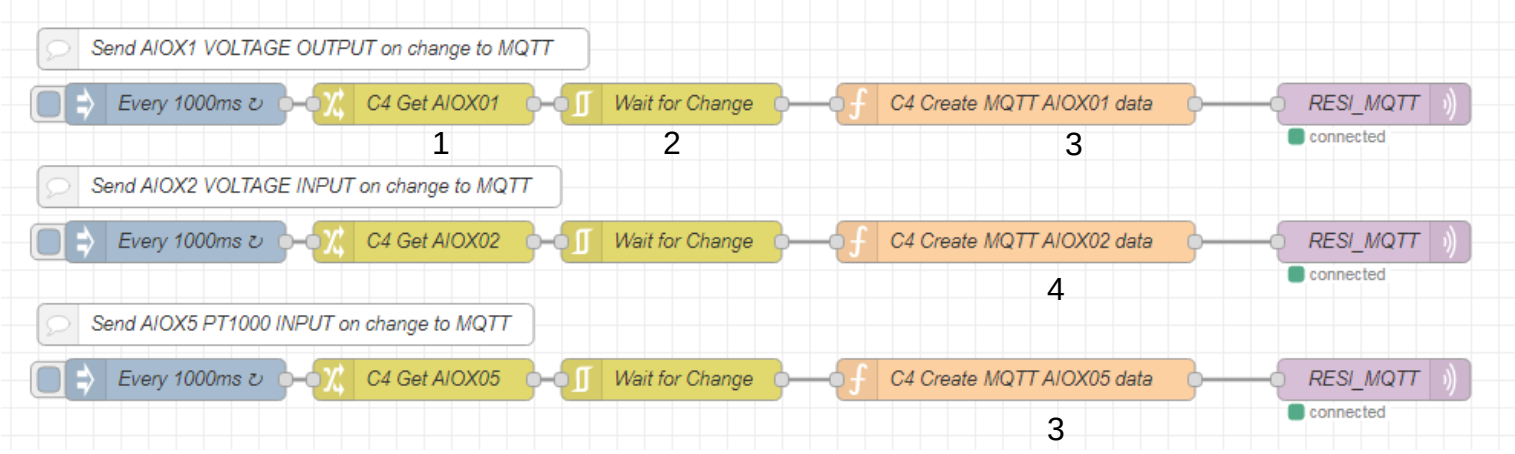


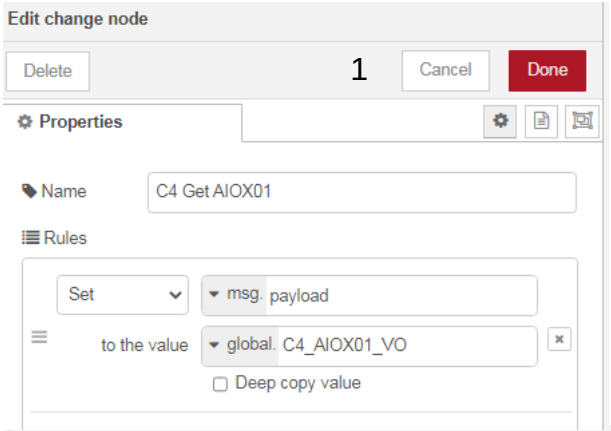

2

# NodeRED® flow
# MQTT messages for AIs+AOs

MQTT messages

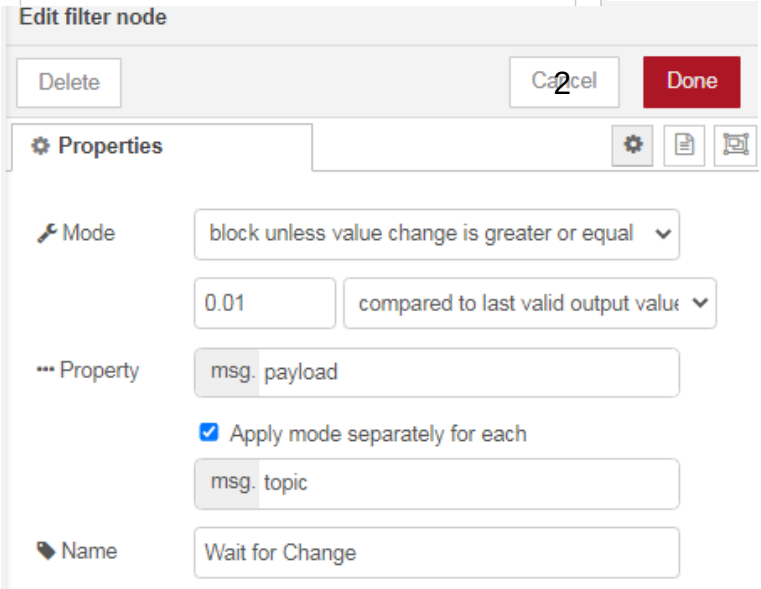
This three flows will send MQTT messages in case of a change of the corresponding global analog variable. Therefore we use the filter to detect a delta of +/-0.1 to the output value. Only in this case we trigger the send of the MQTT message.




```
var msg1={
    topic:"RESI_C4/AnalogInOutputs/VoltageOutputs/C4_VO01",
    payload: msg.payload
}

return msg1;
```

3




```
var msg1={
    topic:"RESI_C4/AnalogInOutputs/VoltageInputs/C4_VI01",
    payload: msg.payload
}

return msg1;
```

4

```
var msg1={
    topic:"RESI_C4/AnalogInOutputs/PT1000Inputs/C4_PT1000_C_05",
    payload: msg.payload
}

return msg1;
```

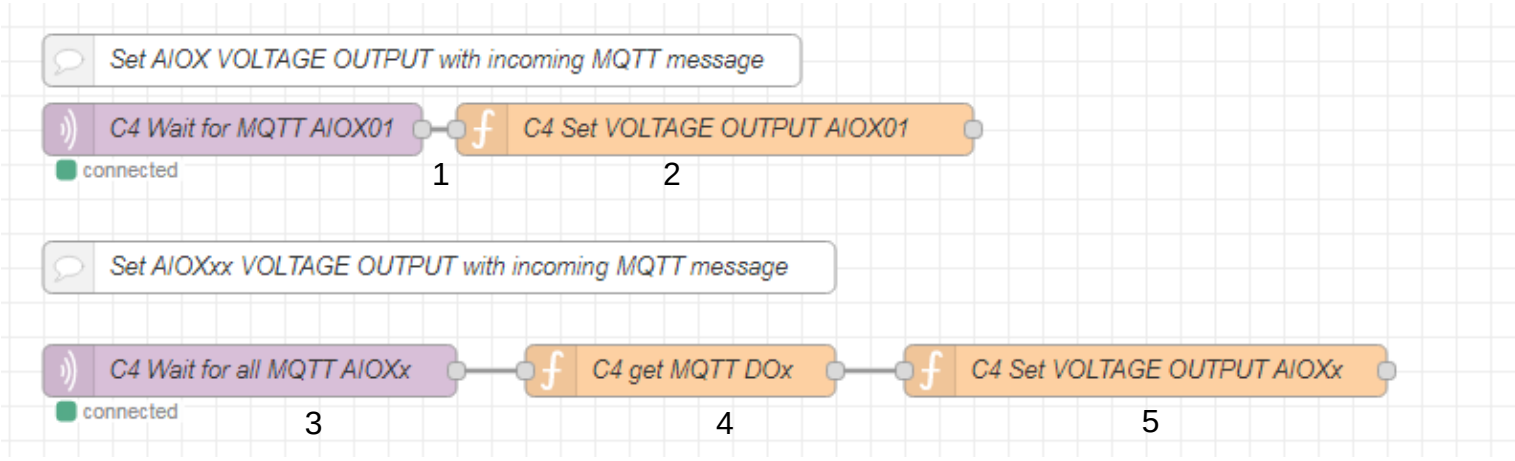
5

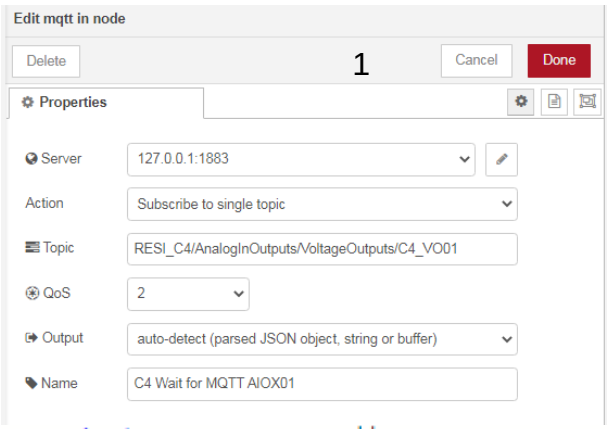
# NodeRED® flow
# MQTT messages for AIs+AOs

MQTT messages

The first flow will wait for the incoming message **RESI_C4/AnalogInOutputs/VoltageOutputs/C4_VO01.** If this message is received, we check if the payload is within 0-10V for the voltage output. Then we set the global variable **C4_AIOX01_VO** to the received set point. Again the flow C4 Universal AIOX will check the change and update via MODBUS the correct registers in the controller.

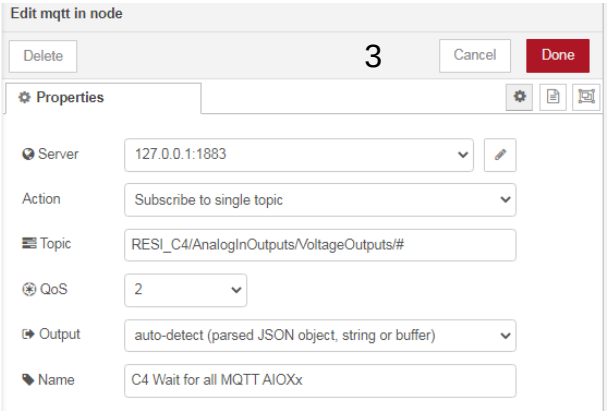
The second flow will wait for messages starting with **RESI_C4/AnalogInOutputs/VoltageOutputs/.** If such a message is received it will retrive the string after the last / delimiter. if the string has the format C4_VO01 to C4_VO16 and the payload is a numeric value between 0 and 10V, the global variable for voltage output **C4_AIOXxx_VO** is updated with the received set point. The real update is done by the flows on C4 Universal AIOX.




```
if (msg.payload >= 0 || msg.payload <= 10.00) {
    global.set("C4_AIOX01_VO", msg.payload);
}

return msg;
```

2




```
if (msg.topic.startsWith("RESI_C4/AnalogInOutputs/VoltageOutputs/"))
{
    let VOx=msg.topic.split("/").pop();
    msg.topic=VOx;
}
return msg;
```

4

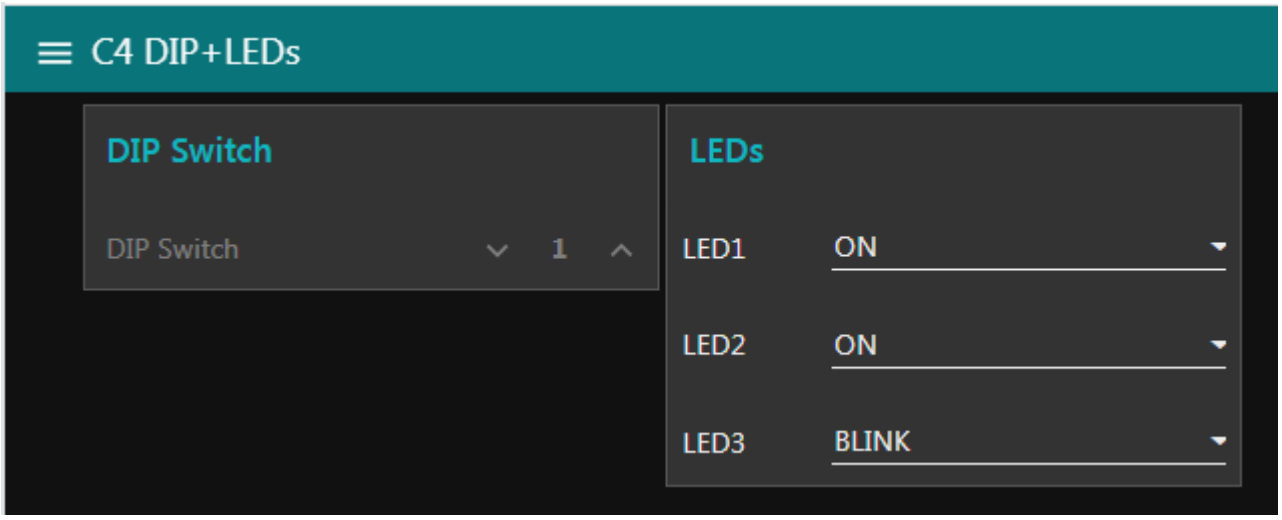
```
let value=0.;

node.log(msg.topic);
if (msg.topic.startsWith="C4_VO")
{
  let VOnr=msg.topic.substring(5,100);
  if (Number.isInteger(parseInt(VOnr,10)))
  {
    let VOx=parseInt(VOnr,10);
    if (VOx>=1 && VOx<global.get("C4_MAX_AIOX"))
    {
      ok=true;
    }
  if (!isNaN(parseFloat(msg.payload)))
  {
    value=parseFloat(msg.payload);
  }
  else
  {
    ok=false;
  }

    if ((value >= 0 || value <= 10.0) && ok)
    {
      var VOxNAME="C4_AIOX"+String(VOx).padStart(2, '0')+"_VO";
      global.set(VOxNAME, value);
    }
  }
}
return msg;
```

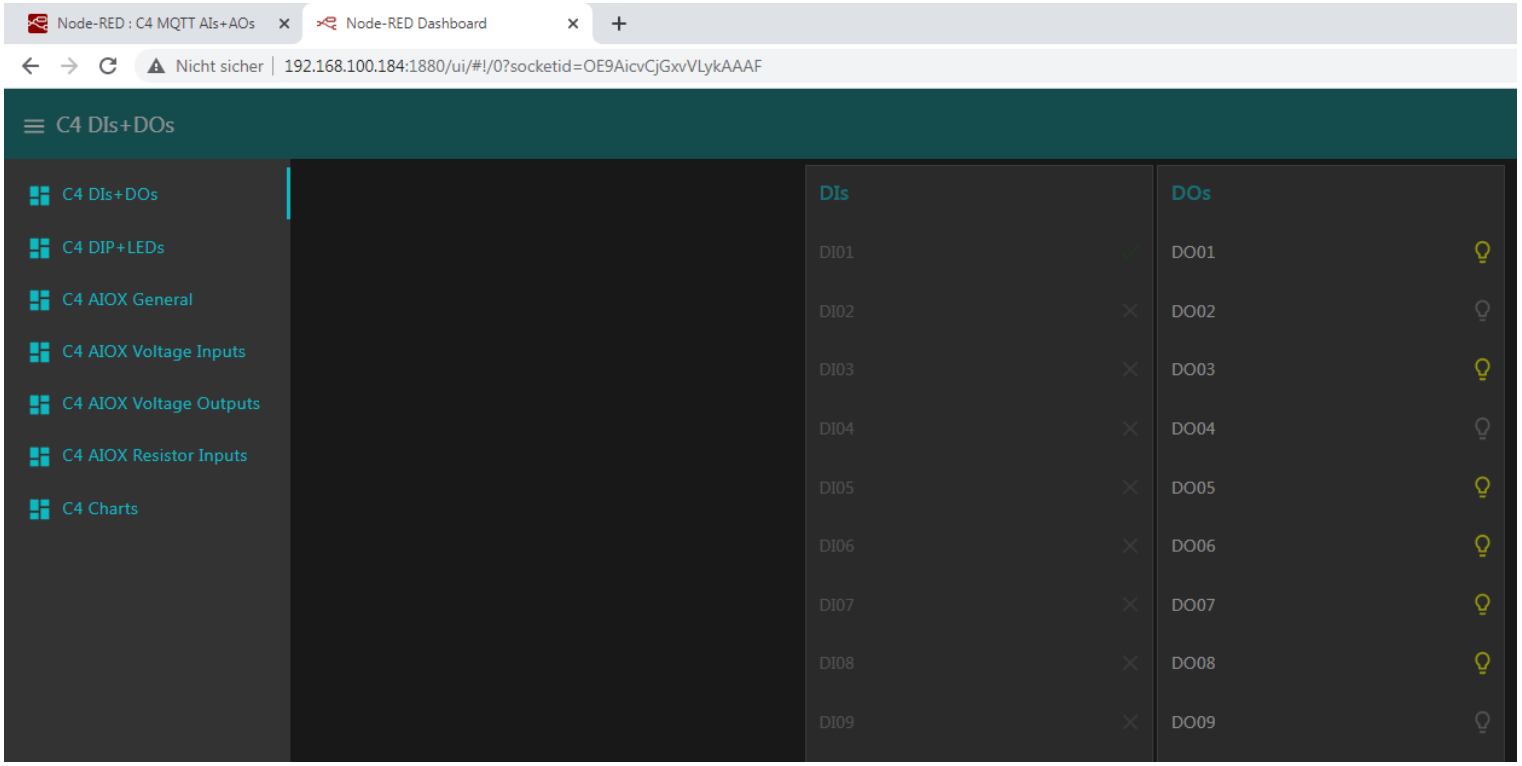
5

# NodeRED® flow dashboard UI

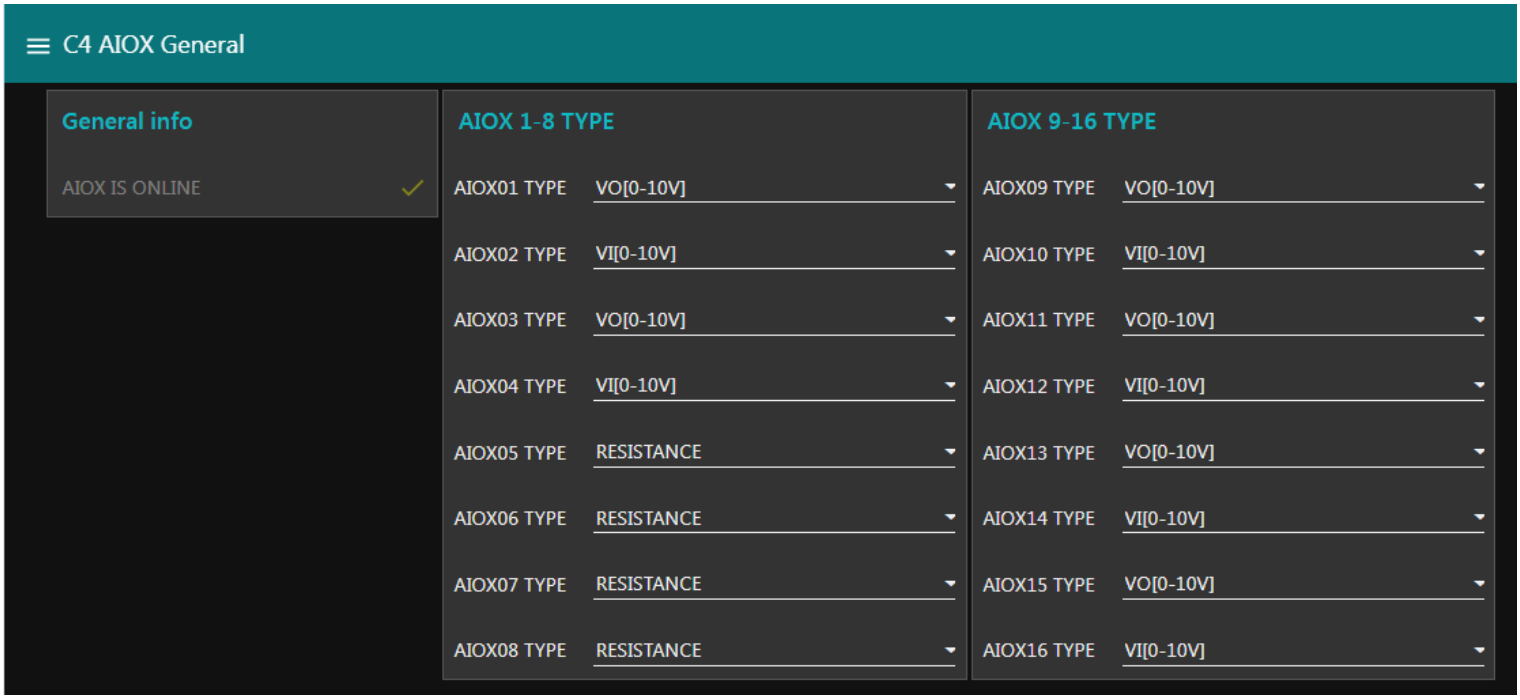
nodered dashboard UI

We added also flows for a dashboard for all function. So you can interactively test all IOs.



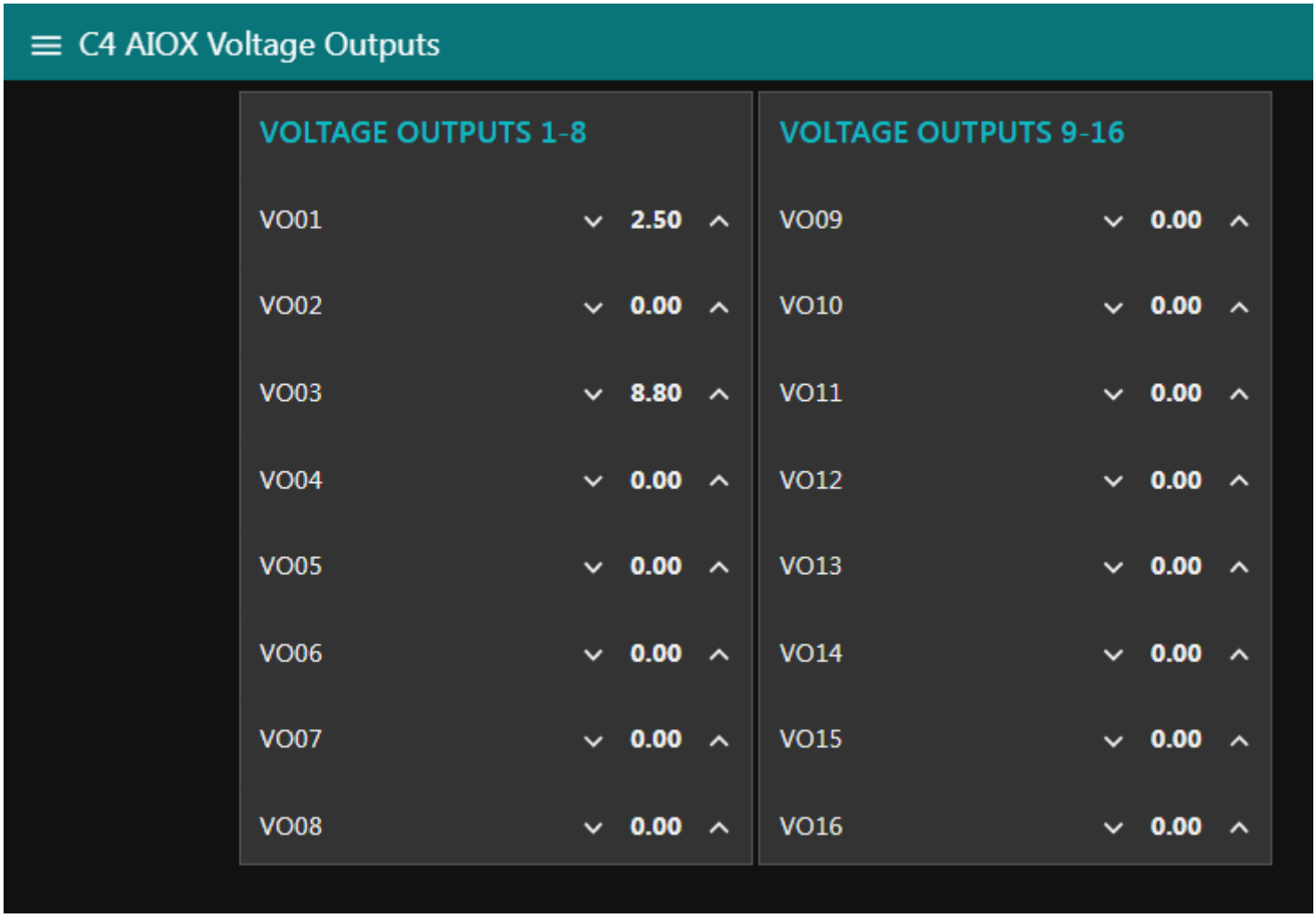



# NodeRED® flow dashboard UI

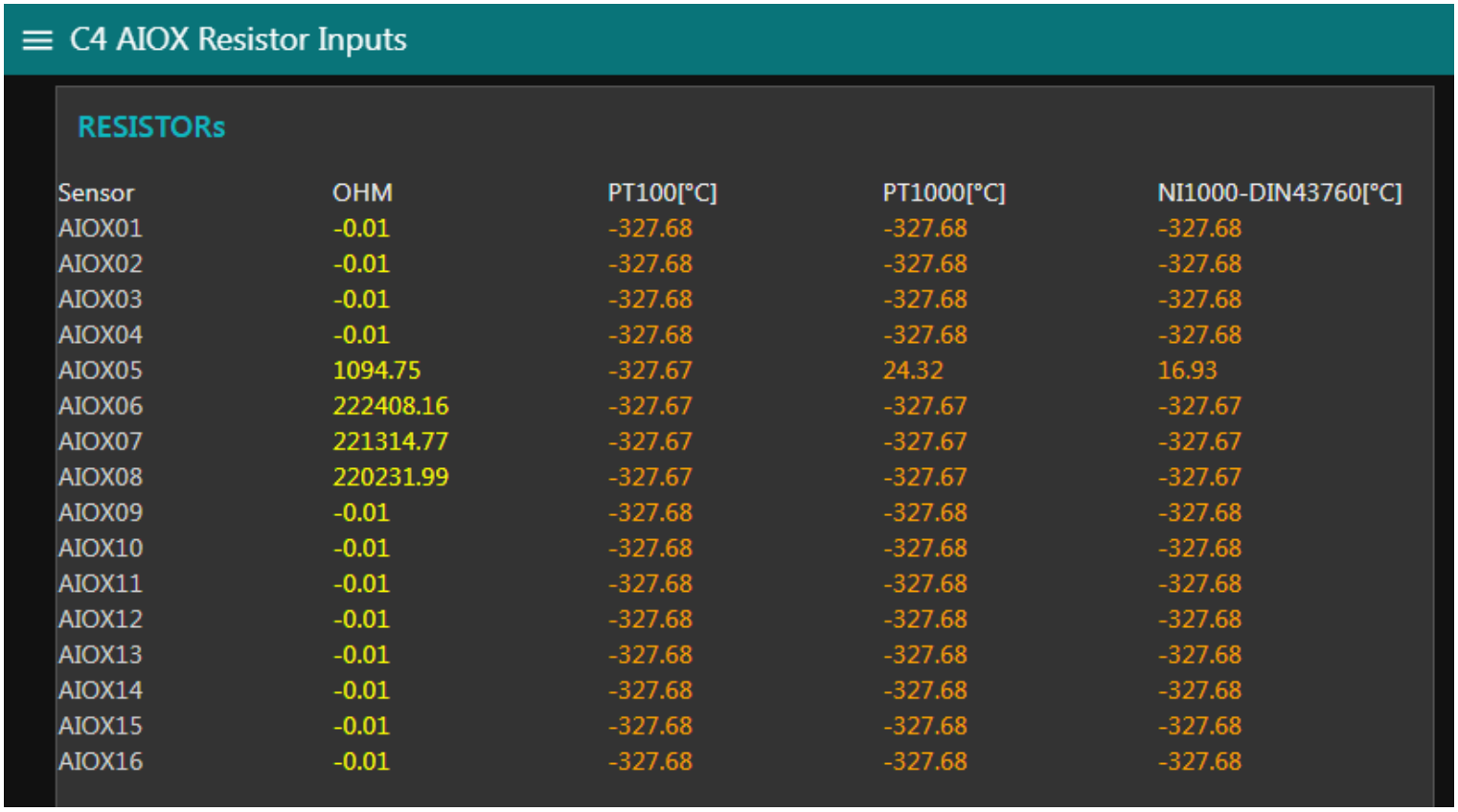


# NodeRED® flow dashboard UI

# NodeRED® flow dashboard UI



# NodeRED® flow dashboard UI



RESI Informatik & Automation GmbH
Altenmarkt 29, A-8551 Wies, AUSTRIA
help@RESI.cc  www.RESI.cc