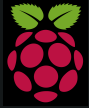


# HOWTO use NodeRED<sup>®</sup> on C4-8CO Controller



Raspberry Pi is a trademark of the Raspberry Pi Foundation.  
More information under [www.raspberrypi.org](http://www.raspberrypi.org)



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

Install NodeRED®  
on RESI-T4/C4 controller

Our RESI-C4-A-32DI24RO16AIOX-xGB  
controller

Our RESI-C4-A-32DI24RO-xGB  
controller

Our RESI-C4-A-32DI24RO-2E-xGB  
controller

Install NodeRED®  
components

Important NodeRED® nodes  
explained

NodeRED® sample flows  
explained

AN-5

AN-7

AN-11

AN-15

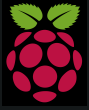
AN-17

AN-19

AN-21

AN-32

# PREREQUISITES



Raspberry Pi is a trademark of the Raspberry Pi Foundation.  
More information under [www.raspberrypi.org](http://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 write 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

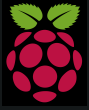
Important hint:

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](http://www.raspberrypi.org)

it's all about perfection \_\_\_\_\_

**RESI**

# 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
```

```
resi@RESI-C4:~$ pm2 start /usr/local/bin/node-red
[PM2] Starting /usr/local/bin/node-red in fork_mode (1 instance)
[PM2] Done.
```

id	name	mode	u	status	cpu	memory
0	node-red	fork	0	online	0%	36.0mb

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<sup>®</sup> 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=$PATH:/usr/bin /usr/local/lib/node_modules/pm2/bin/pm2 startup systemd -u resi --hp /home/resi
resi@RESI-C4:~$ 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

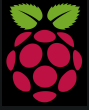
[Install]
WantedBy=multi-user.target

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](http://www.raspberrypi.org)

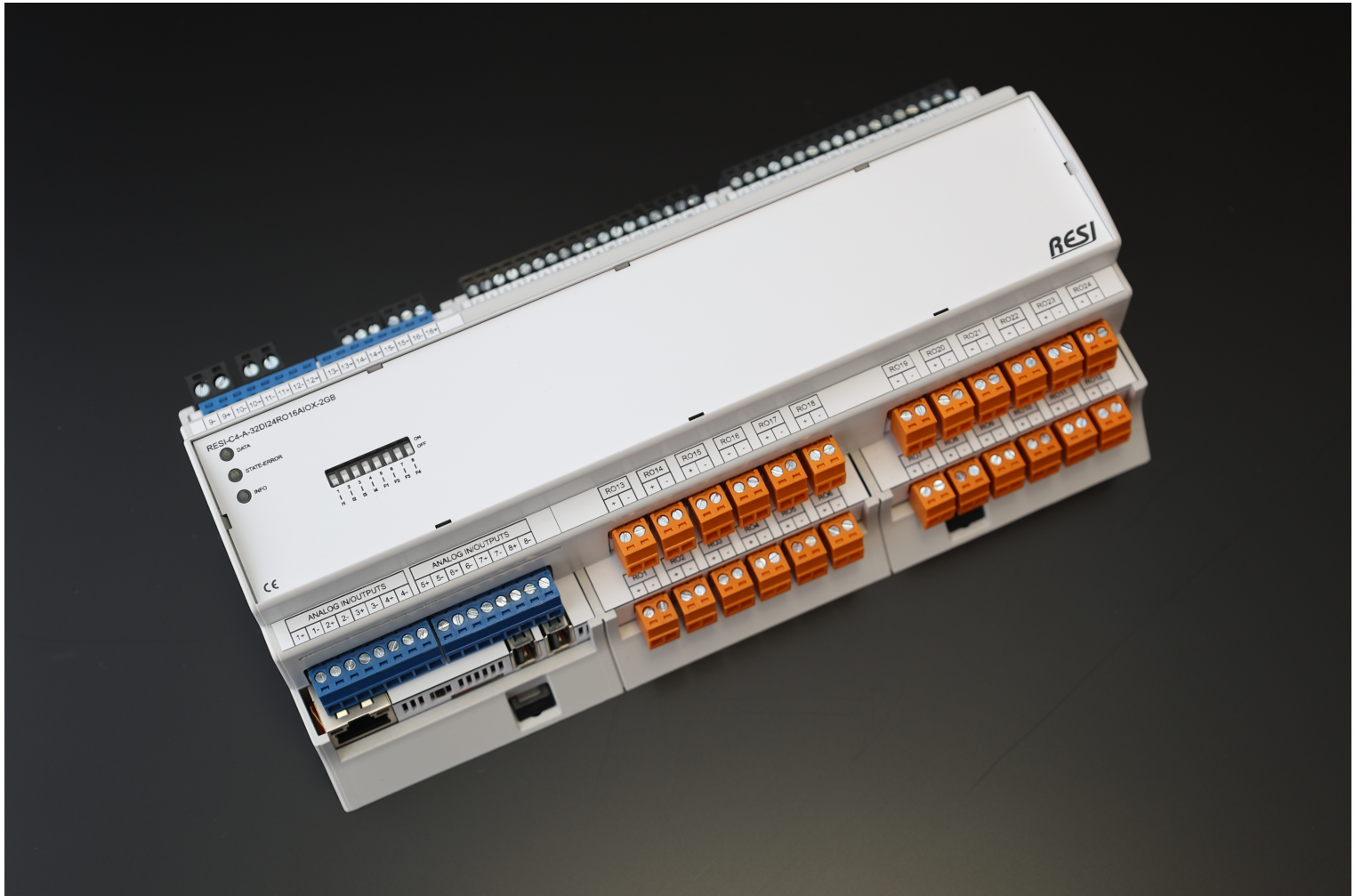
it's all about perfection \_\_\_\_\_

**RESI**

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

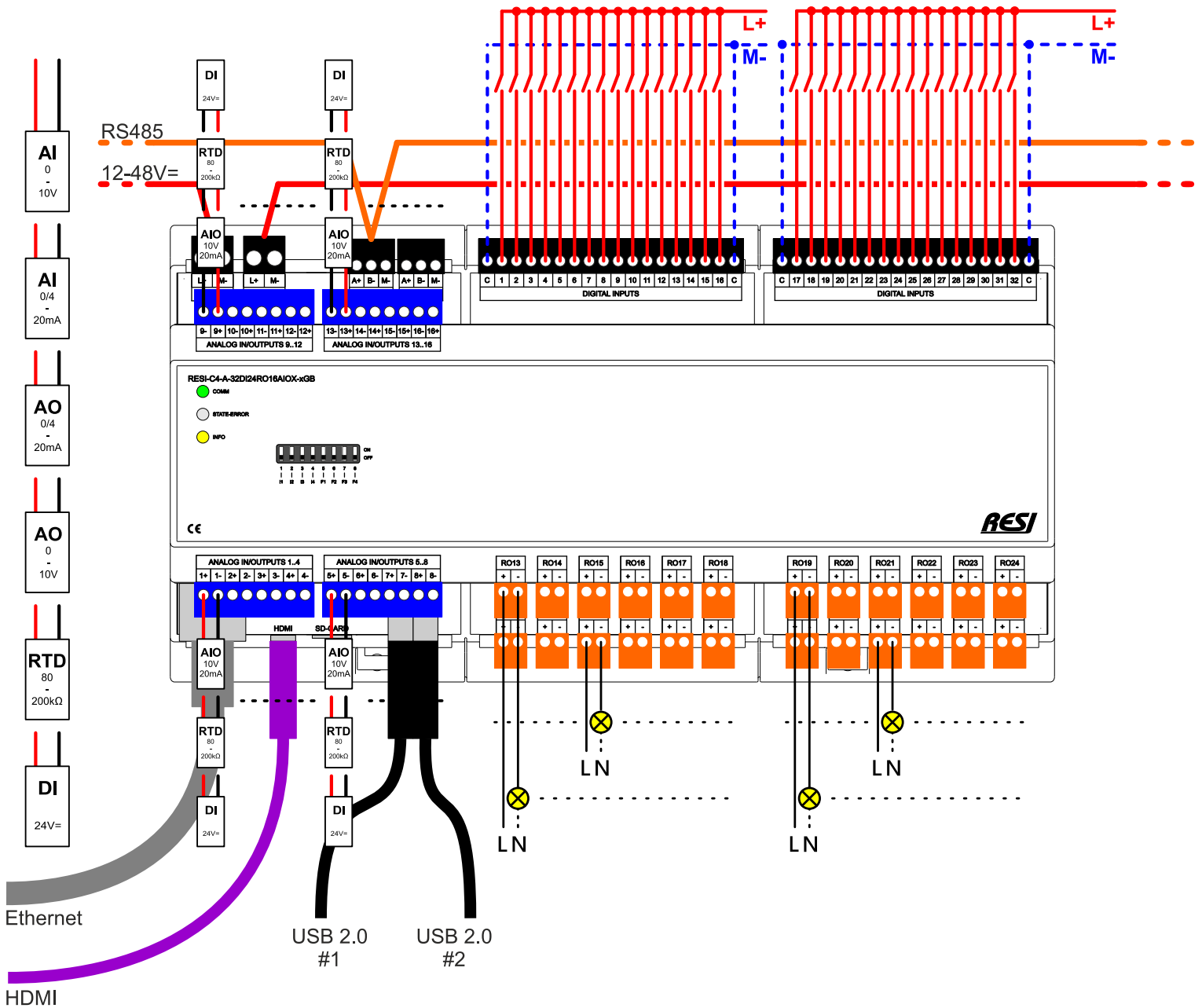
## HIGHLIGHTS

- 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  $\leq 48V=$  signals
- 24 form A relay outputs for  $\leq 30V=, \leq 250V=, \leq 6A$
- 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: 0-1M $\Omega$ , PT100, PT1000, NI1000-DIN43760 linearisation and temperature calculation in  $^{\circ}C$ ,  $^{\circ}F$  or  $^{\circ}K$
  - Digital input: 24V logic input or dry contact



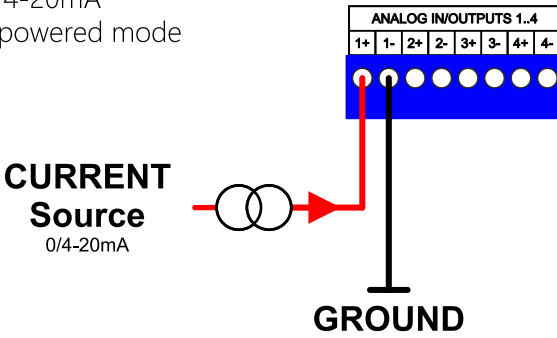
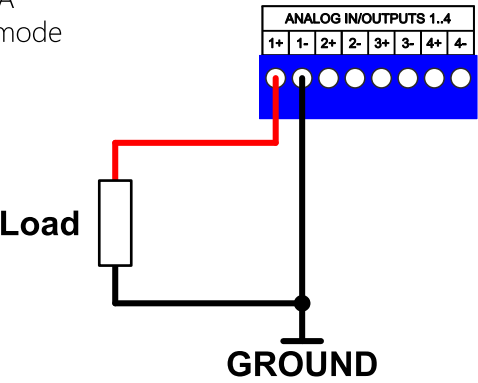
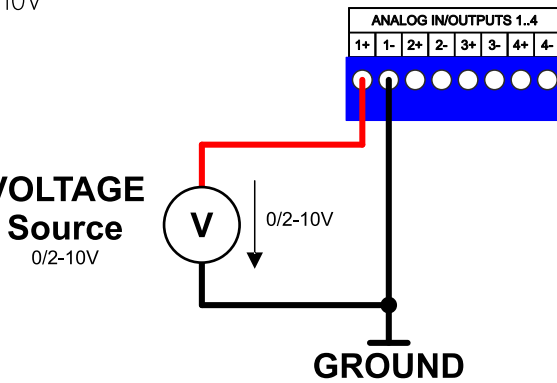
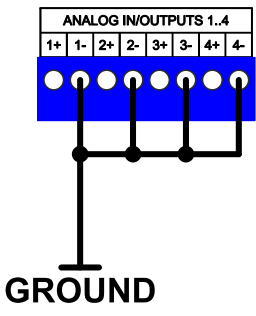
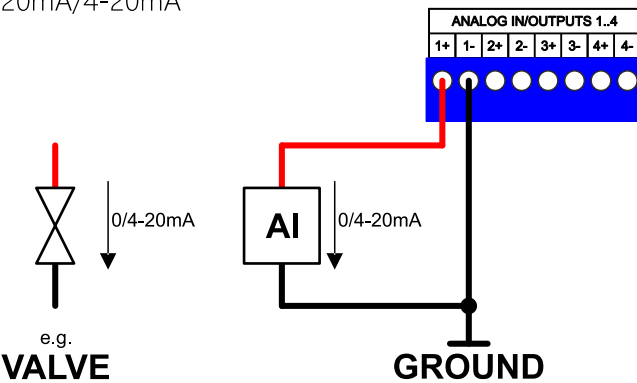
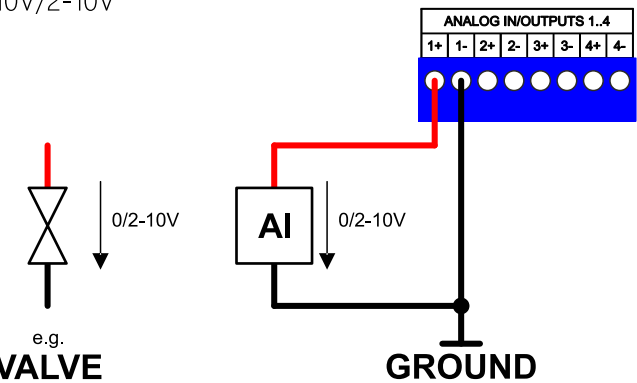
# RESI-C4-A-32DI24RO16AIOX-xGB

## Scematic



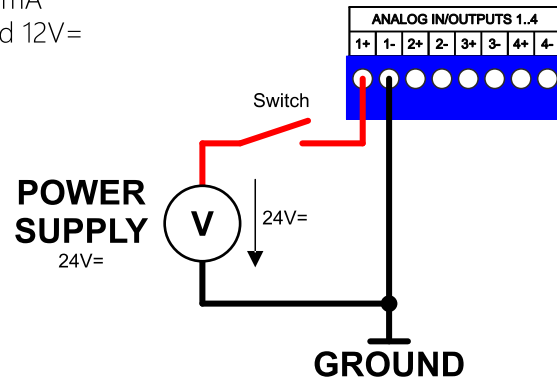
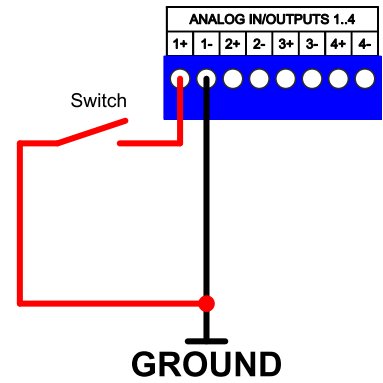
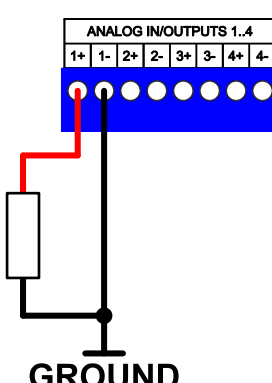
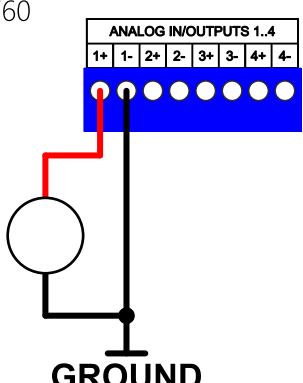
# RESI-C4-A-32DI24RO16AIOX-xGB

## Universal analog IOs - possibilities

<p><b>ANALOG INPUT</b> 0-20mA/4-20mA External powered mode</p>  <p><b>CURRENT Source</b> 0/4-20mA</p> <p><b>GROUND</b></p>	<p><b>ANALOG INPUT</b> 0-20mA/4-20mA Loop powered mode</p>  <p><b>Load</b></p> <p><b>GROUND</b></p>
<p><b>ANALOG INPUT</b> 0-10V/2-10V</p>  <p><b>VOLTAGE Source</b> 0/2-10V</p> <p><b>GROUND</b></p>	<p><b>ANALOG GROUND</b> All analog grounds are internally tied together!</p> <p>This affects to all AIOX inputs! But this analog ground is galvanically isolated from the M- of the module!</p>  <p><b>GROUND</b></p>
<p><b>ANALOG OUTPUT</b> 0-20mA/4-20mA</p>  <p>e.g. <b>VALVE</b> 0/2-10V input</p> <p><b>AI</b> 0/4-20mA</p> <p><b>GROUND</b></p>	<p><b>ANALOG OUTPUT</b> 0-10V/2-10V</p>  <p>e.g. <b>VALVE</b> 0/2-10V input</p> <p><b>AI</b> 0/2-10V</p> <p><b>GROUND</b></p>



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

<p><b>DIGITAL INPUT</b>  <math>\leq 40V</math>, 1.8mA          Threshold 12V=          Logic</p>  <p>Diagram showing a 24V DC power supply connected to a switch. The switch is connected to terminal 1+ of the ANALOG IN/OUTPUTS 1..4 block. The other terminal of the switch is connected to GROUND.</p>	<p><b>DIGITAL INPUT</b>          Loop powered          4mA</p>  <p>Diagram showing a loop-powered digital input configuration. A switch is connected to terminal 1+ of the ANALOG IN/OUTPUTS 1..4 block. The other terminal of the switch is connected to GROUND.</p>																										
<p><b>RESISTOR INPUT</b>          0-1M<math>\Omega</math></p> <table border="0"> <tr> <td>Measurement Range</td> <td>Accuracy</td> </tr> <tr> <td>0-80<math>\Omega</math></td> <td><math>\pm 0.5\% \pm 0.5\Omega</math></td> </tr> <tr> <td>80-200<math>\Omega</math></td> <td><math>\pm 0.3\%</math></td> </tr> <tr> <td>200-1k<math>\Omega</math></td> <td><math>\pm 0.2\%</math></td> </tr> <tr> <td>1k-10k<math>\Omega</math></td> <td><math>\pm 0.2\%</math></td> </tr> <tr> <td>10k-20k<math>\Omega</math></td> <td><math>\pm 0.3\%</math></td> </tr> <tr> <td>20k-100k<math>\Omega</math></td> <td><math>\pm 0.8\%</math></td> </tr> <tr> <td>100k-200k<math>\Omega</math></td> <td><math>\pm 1.0\%</math></td> </tr> <tr> <td>200k-1M<math>\Omega</math></td> <td><math>\pm 8\%</math></td> </tr> </table>  <p>Diagram showing a resistor connected between terminal 1+ of the ANALOG IN/OUTPUTS 1..4 block and GROUND.</p>	Measurement Range	Accuracy	0-80 $\Omega$	$\pm 0.5\% \pm 0.5\Omega$	80-200 $\Omega$	$\pm 0.3\%$	200-1k $\Omega$	$\pm 0.2\%$	1k-10k $\Omega$	$\pm 0.2\%$	10k-20k $\Omega$	$\pm 0.3\%$	20k-100k $\Omega$	$\pm 0.8\%$	100k-200k $\Omega$	$\pm 1.0\%$	200k-1M $\Omega$	$\pm 8\%$	<p><b>RESISTOR INPUT</b>          PT100, PT1000, NI1000-DIN43760          internal linearisation          Output: <math>^{\circ}C</math>, <math>^{\circ}F</math> or <math>^{\circ}K</math></p> <table border="0"> <tr> <td>Sensor type</td> <td>Accuracy</td> </tr> <tr> <td>PT100</td> <td><math>\pm 0.3\%</math></td> </tr> <tr> <td>PT1000</td> <td><math>\pm 0.2\%</math></td> </tr> <tr> <td>NI1000-DIN43760</td> <td><math>\pm 0.2\%</math></td> </tr> </table>  <p>Diagram showing a 2-wire RTD sensor connected between terminal 1+ of the ANALOG IN/OUTPUTS 1..4 block and GROUND.</p>	Sensor type	Accuracy	PT100	$\pm 0.3\%$	PT1000	$\pm 0.2\%$	NI1000-DIN43760	$\pm 0.2\%$
Measurement Range	Accuracy																										
0-80 $\Omega$	$\pm 0.5\% \pm 0.5\Omega$																										
80-200 $\Omega$	$\pm 0.3\%$																										
200-1k $\Omega$	$\pm 0.2\%$																										
1k-10k $\Omega$	$\pm 0.2\%$																										
10k-20k $\Omega$	$\pm 0.3\%$																										
20k-100k $\Omega$	$\pm 0.8\%$																										
100k-200k $\Omega$	$\pm 1.0\%$																										
200k-1M $\Omega$	$\pm 8\%$																										
Sensor type	Accuracy																										
PT100	$\pm 0.3\%$																										
PT1000	$\pm 0.2\%$																										
NI1000-DIN43760	$\pm 0.2\%$																										



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

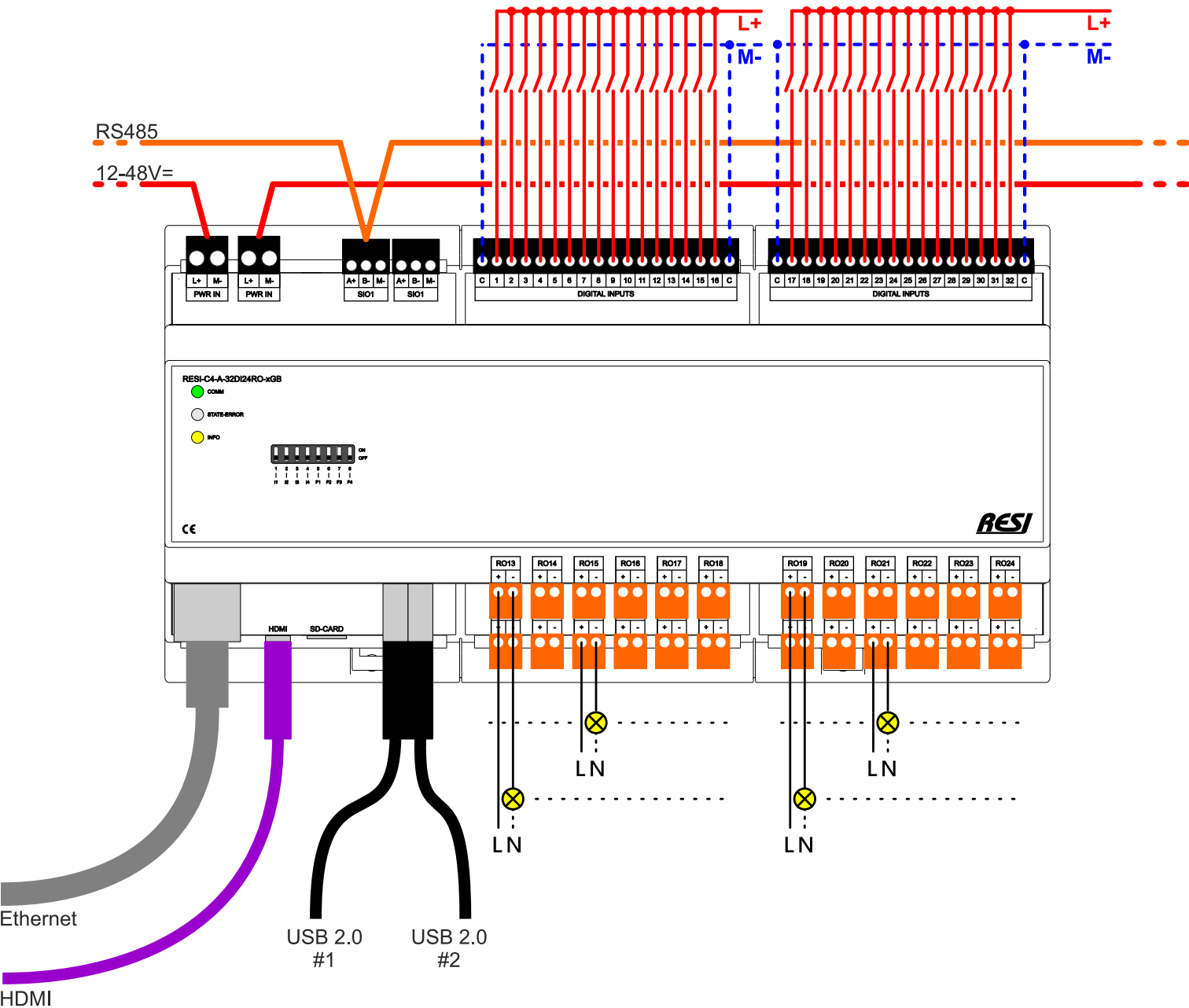
## HIGHLIGHTS

- 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  $\leq 48V=$  signals
  - 24 form A relay outputs for  $\leq 30V=$ ,  $\leq 250V=$ ,  $\leq 6A$



# RESI-C4-A-32DI24RO-xGB

## Scematic





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

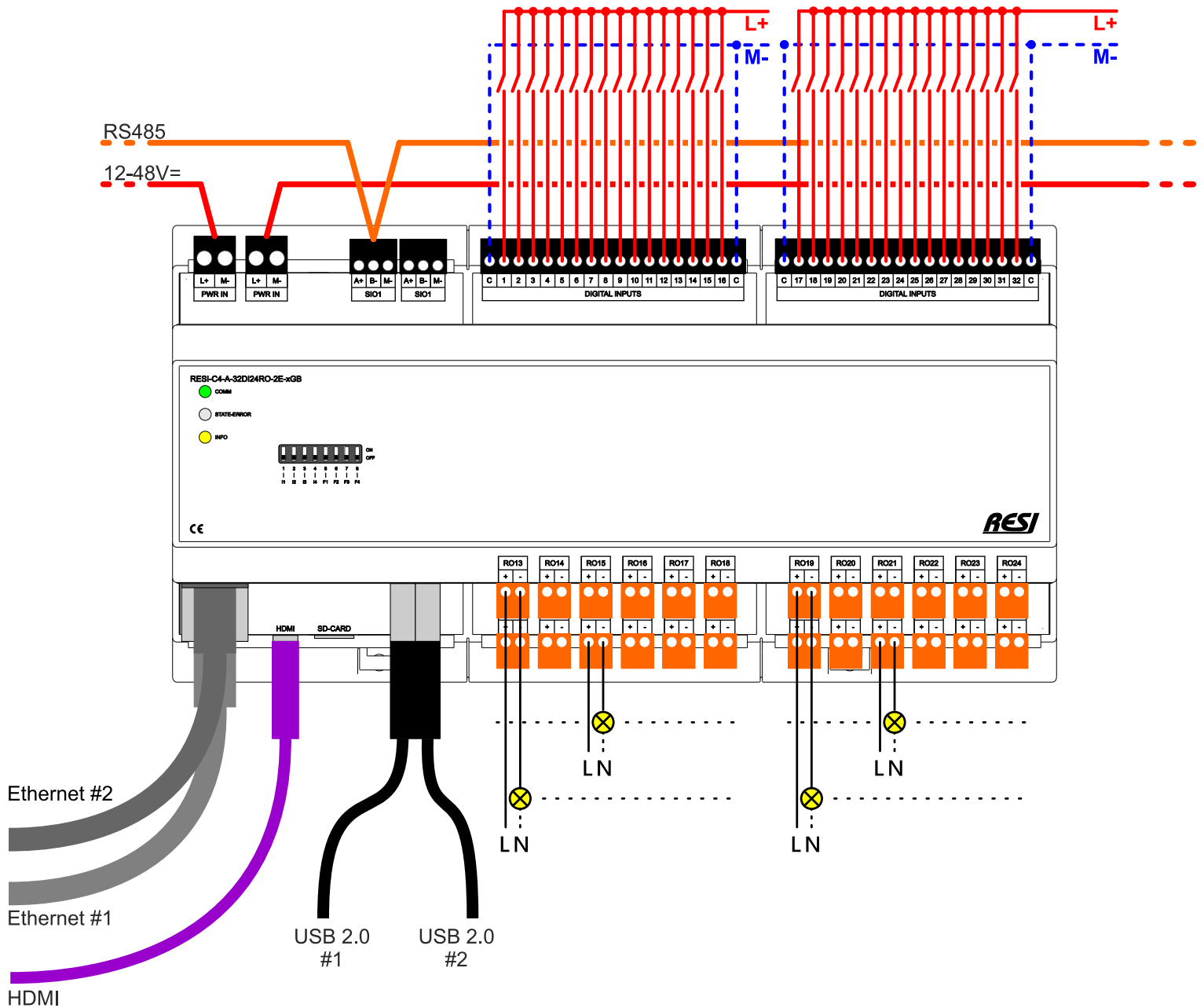
## HIGHLIGHTS

- Raspberry Compute Module 4
  - 2GB or 4GB or 8GB RAM
  - 1xSD-CARD Slot with 32GB card
  - 2xEthernet Interface for router or bridge applications
  - 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  $\leq 48V=$  signals
  - 24 form A relay outputs for  $\leq 30V=$ ,  $\leq 250V=$ ,  $\leq 6A$

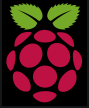


# RESI-C4-A-32DI24RO-2E-xGB

## Scematic



# Install NodeRED<sup>®</sup> components

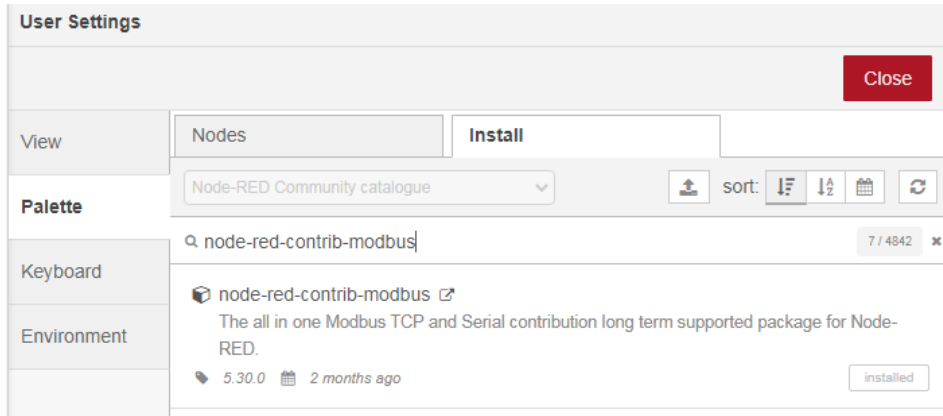


Raspberry Pi is a trademark of the Raspberry Pi Foundation.  
More information under [www.raspberrypi.org](http://www.raspberrypi.org)

# Install NodeRED<sup>®</sup> components node-red-contrib-modbus

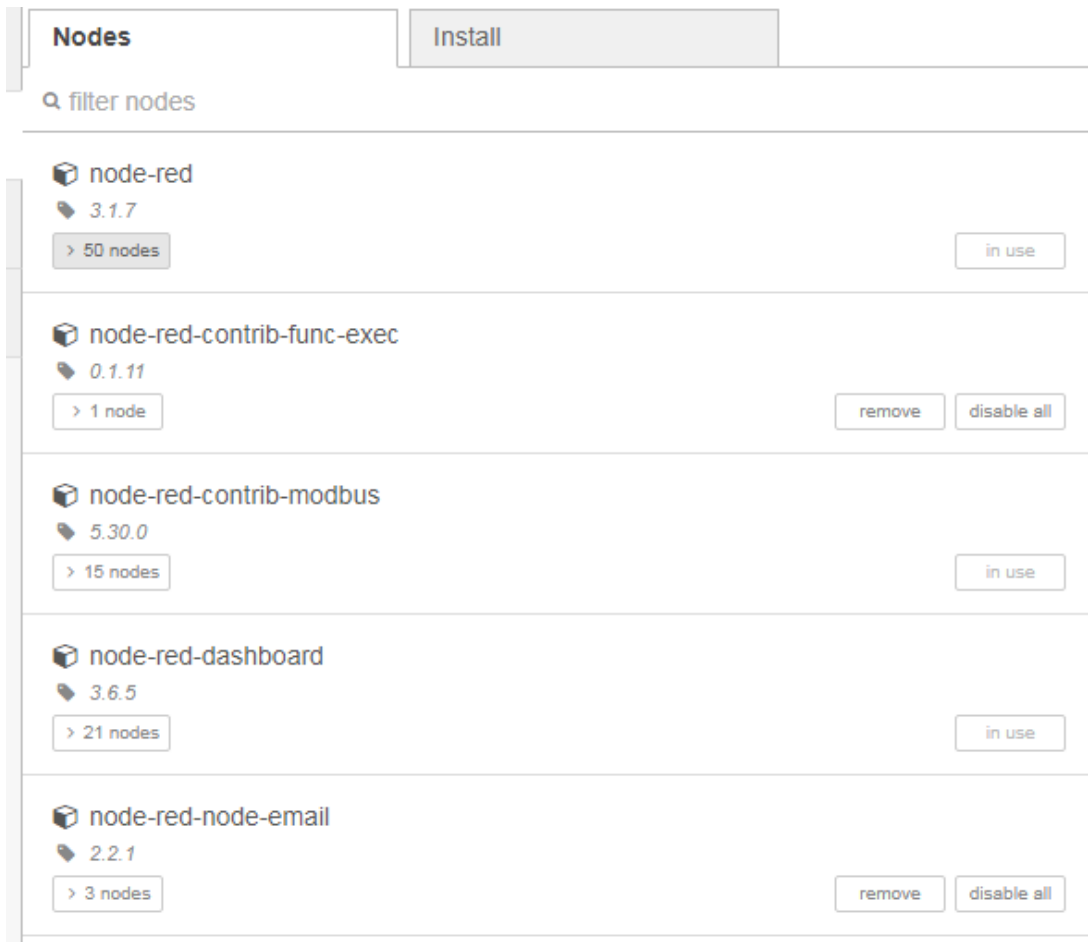
## INSTALL node-red-contrib-modbus

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

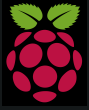


Click on Install. We have installed this component already.

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](http://www.raspberrypi.org)

# Important NodeRED<sup>®</sup> nodes explained

## BASIC PRINCIPLE OF RESI-C4 controllers

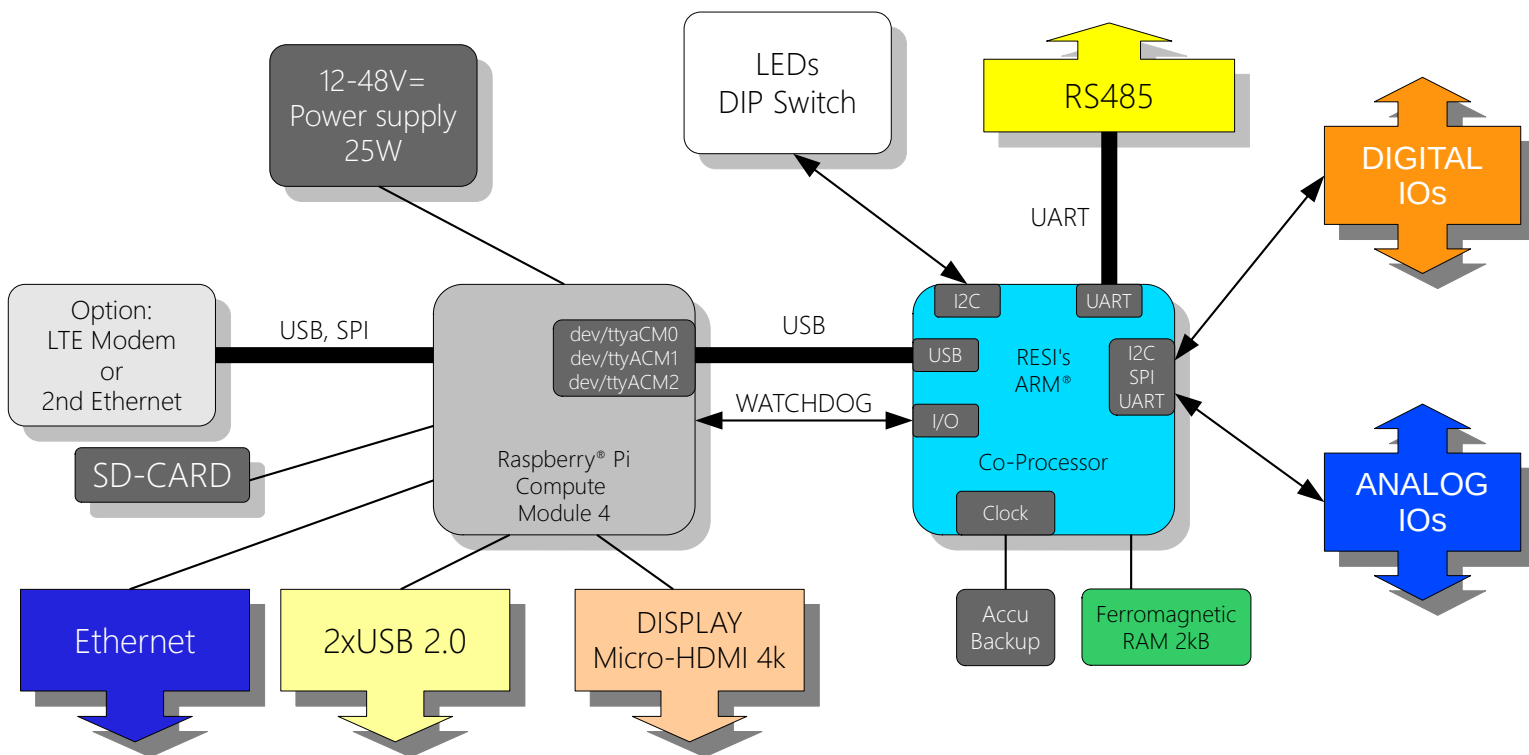
All of our RESI-C4 controller are based on the following hardware architecture. We have extended the Compute Module 4 with an ARM co-processor,, which is connected via USB to the LINUX system. All time critical actions are driven by the ARM firmware and not by LINUX. Therefore there is no real time issue 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 LINUX 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 communication

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.

**Edit modbus-client node**

Delete Cancel Update

**Properties**

**Settings** Queues Optionals

Name: C4 MODBUS/RTU

Type: Serial Expert

Serial port: /dev/ttyACM1

Serial type: RTU-BUFFERD

Baud rate: 9600

Data Bits: 8

Stop Bits: 1

Parity: None

Connection delay (ms): 1

Unit-Id: 2

Timeout (ms): 1000

Reconnect on timeout:

Reconnect timeout (ms): 2000

**Edit modbus-client node**

Delete Cancel Update

**Properties**

Settings **Queues** Optionals

UnitId's in parallel queues:  Don't use it for serial without a serial gateway.

Queue Logging:

Queue commands:

Queue delay (ms): 1

**Edit modbus-client node**

Delete Cancel Update

**Properties**

Settings Queues **Optionals**

Log states changes:

Log failures:

Show Errors:

Show Warnings:

Show Logs:



# 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 through the RS485 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!

The image displays three screenshots of the NodeRED 'Edit modbus-client node' configuration interface, showing different tabs: Settings, Queues, and Optionals.

**Settings Tab:**

- Name: C4 RS485
- Type: Serial Expert
- Serial port: /dev/ttyACM2
- Serial type: RTU
- Baud rate: 9600
- Data Bits: 8
- Stop Bits: 1
- Parity: None
- Connection delay (ms): 1
- Unit-Id: 1
- Timeout (ms): 1000
- Reconnect on timeout:
- Reconnect timeout (ms): 2000

**Queues Tab:**

- UnitId's in parallel queues:  Don't use it for serial without a serial gateway.
- Queue Logging:
- Queue commands:
- Queue delay (ms): 1

**Optionals Tab:**

- Log states changes:
- Log failures:
- Show Errors:
- Show Warnings:
- Show Logs:

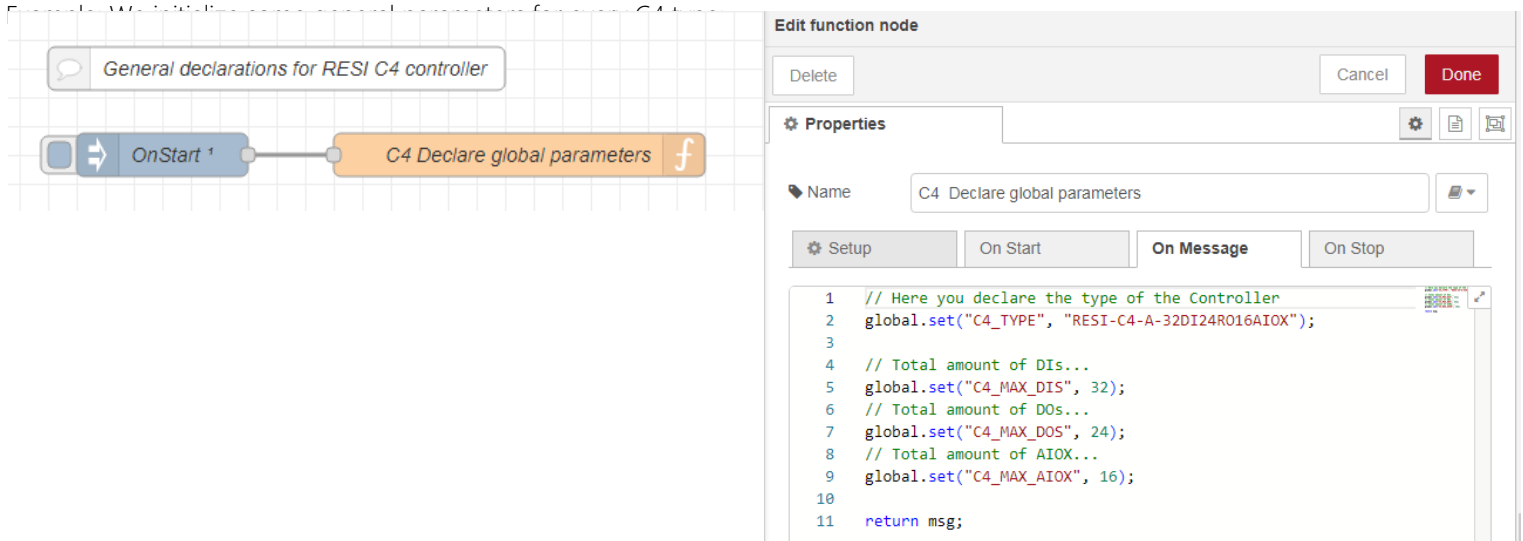




# Important NodeRED® nodes explained

## 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.

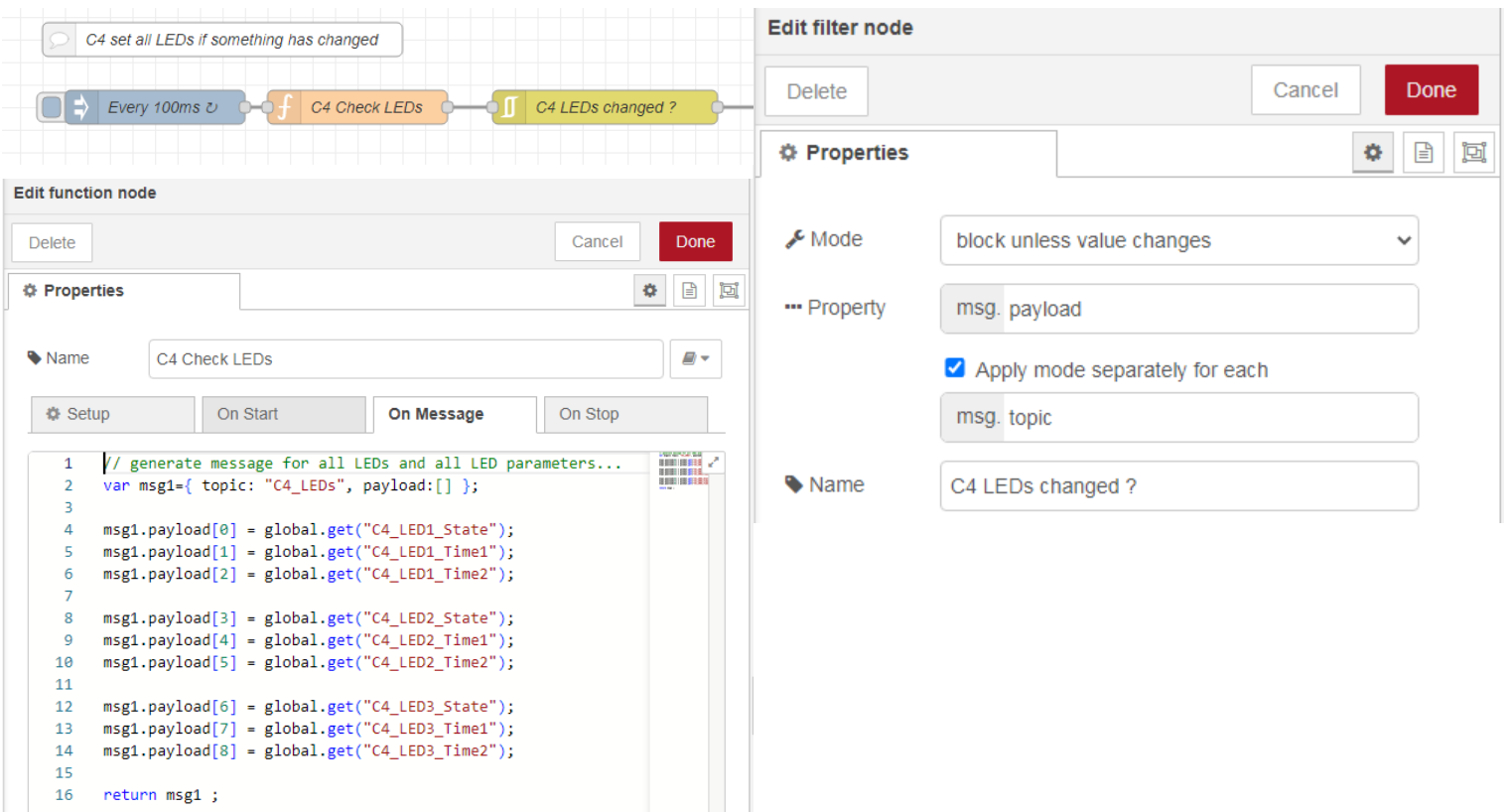


The screenshot shows a NodeRED flow with a comment "General declarations for RESI C4 controller". The flow consists of an "OnStart" node connected to a function node "C4 Declare global parameters". The "Edit function node" panel is open, showing the following code:

```
1 // Here you declare the type of the Controller
2 global.set("C4_TYPE", "RESI-C4-A-32DI24R016AIOX");
3
4 // Total amount of DIS...
5 global.set("C4_MAX_DIS", 32);
6 // Total amount of DOS...
7 global.set("C4_MAX_DOS", 24);
8 // Total amount of AIOX...
9 global.set("C4_MAX_AIOX", 16);
10
11 return msg;
```

## 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.



The screenshot shows a NodeRED flow with a comment "C4 set all LEDs if something has changed". The flow consists of an "Every 100ms" node connected to a function node "C4 Check LEDs", which is then connected to a filter node "C4 LEDs changed?". The "Edit function node" panel is open, showing the following code:

```
1 // generate message for all LEDs and all LED parameters...
2 var msg1={ topic: "C4_LEDS", payload:[] };
3
4 msg1.payload[0] = global.get("C4_LED1_State");
5 msg1.payload[1] = global.get("C4_LED1_Time1");
6 msg1.payload[2] = global.get("C4_LED1_Time2");
7
8 msg1.payload[3] = global.get("C4_LED2_State");
9 msg1.payload[4] = global.get("C4_LED2_Time1");
10 msg1.payload[5] = global.get("C4_LED2_Time2");
11
12 msg1.payload[6] = global.get("C4_LED3_State");
13 msg1.payload[7] = global.get("C4_LED3_Time1");
14 msg1.payload[8] = global.get("C4_LED3_Time2");
15
16 return msg1 ;
```

The "Edit filter node" panel is also open, showing the following configuration:

- Mode: block unless value changes
- Property: msg.payload
- Apply mode separately for each
- msg.topic
- Name: C4 LEDs changed ?



# 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!
- !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  $\leq 125$ . For coils or input bits  $\leq 2000$ .

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.

The screenshot shows a NodeRED workflow with a 'C4 LED Status' node connected to a 'C4 Update DIP+LEDx Status' node. The 'C4 LED Status' node is active (500 msec.). The settings panel for the 'C4 LED Status' node is open, showing the following configuration:

- Name: C4 LED Status
- Topic: Topic
- Unit-Id: 1
- FC: FC 3: Read Holding Registers
- Address: 65500
- Quantity: 20
- Poll Rate: 500 millisecond(s)
- Delay to activate input:
- Server: C4 MODBUS/RTU

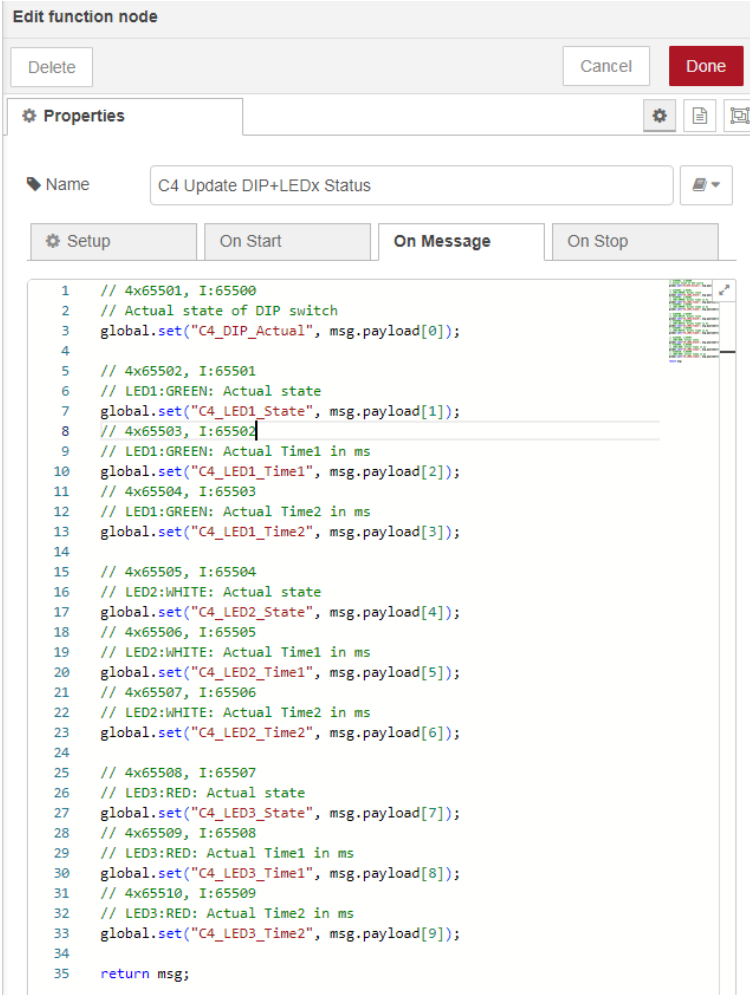
Sample from our MODBUS register documentation for the DIP switch

DIP SWITCH STATUS					
DIP SWITCH	3x65501 4x65501 !65500	????			UINT16 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.



The screenshot shows the 'Edit function node' window in NodeRED. The node is named 'C4 Update DIP+LEDx Status' and is configured for the 'On Message' trigger. The function code is as follows:

```
1 // 4x65501, I:65500
2 // Actual state of DIP switch
3 global.set("C4_DIP_Actual", msg.payload[0]);
4
5 // 4x65502, I:65501
6 // LED1:GREEN: Actual state
7 global.set("C4_LED1_State", msg.payload[1]);
8 // 4x65503, I:65502
9 // LED1:GREEN: Actual Time1 in ms
10 global.set("C4_LED1_Time1", msg.payload[2]);
11 // 4x65504, I:65503
12 // LED1:GREEN: Actual Time2 in ms
13 global.set("C4_LED1_Time2", msg.payload[3]);
14
15 // 4x65505, I:65504
16 // LED2:WHITE: Actual state
17 global.set("C4_LED2_State", msg.payload[4]);
18 // 4x65506, I:65505
19 // LED2:WHITE: Actual Time1 in ms
20 global.set("C4_LED2_Time1", msg.payload[5]);
21 // 4x65507, I:65506
22 // LED2:WHITE: Actual Time2 in ms
23 global.set("C4_LED2_Time2", msg.payload[6]);
24
25 // 4x65508, I:65507
26 // LED3:RED: Actual state
27 global.set("C4_LED3_State", msg.payload[7]);
28 // 4x65509, I:65508
29 // LED3:RED: Actual Time1 in ms
30 global.set("C4_LED3_Time1", msg.payload[8]);
31 // 4x65510, I:65509
32 // LED3:RED: Actual Time2 in ms
33 global.set("C4_LED3_Time2", msg.payload[9]);
34
35 return msg;
```



# 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:dddd 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.

The screenshot shows a NodeRED flowchart with the following nodes: 'Every 100ms' (timer), 'C4 Check LEDs' (function), 'C4 LEDs changed?' (change detector), 'C4 Set All LEDs' (function), and 'C4 Update All LEDs' (Modbus-Flex-Write node). The 'C4 Update All LEDs' node is active and has a configuration window open.

**Edit Modbus-Flex-Write node**

Settings: Name: C4 Update All LEDs, Server: C4 MODBUS/RTU, Delay to activate input:

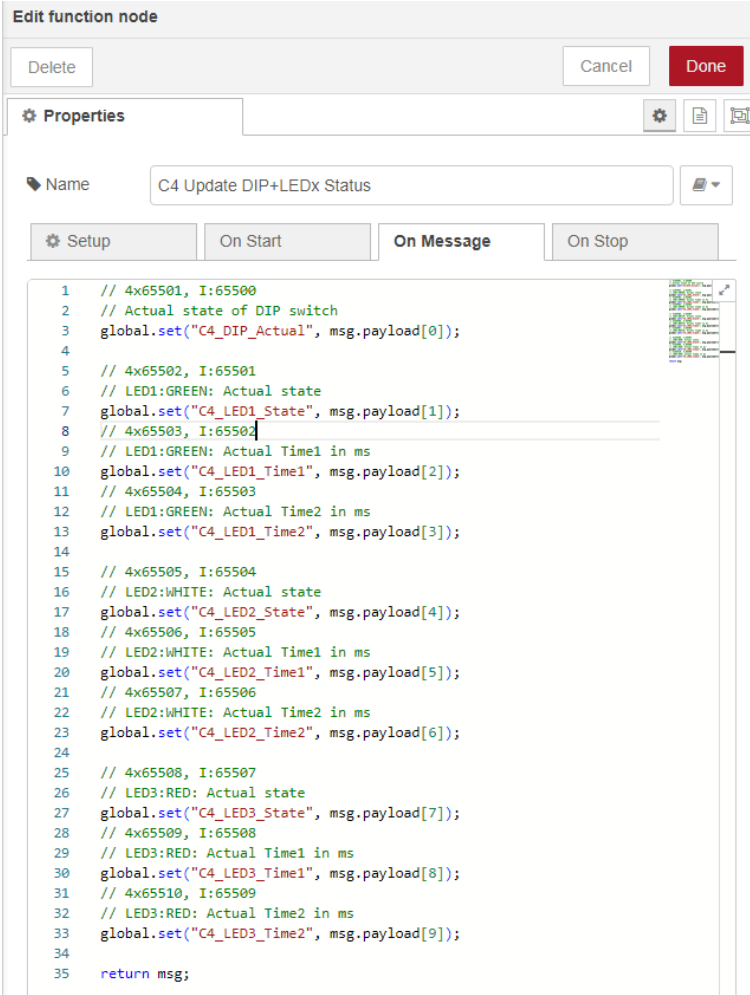
Optional payload structure:

```
34
35 msg.payload = {
36   value: [
37     C4_LEDs[0], C4_LEDs[1], C4_LEDs[2],
38     C4_LEDs[3], C4_LEDs[4], C4_LEDs[5],
39     C4_LEDs[6], C4_LEDs[7], C4_LEDs[8]
40   ],
41   // WRITE MULTIPLE REGISTERS
42   'fc': 16,
43   // INTERNAL UnitID of C4
44   'unitid': 1,
45   // START ADDRESS: 4x65502,I:65501
46   'address': 65501,
47   // 3 LEDs with 3 values each...
48   'quantity': 9
49 }
50 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.



The screenshot shows the 'Edit function node' window in NodeRED. The node is named 'C4 Update DIP+LEDx Status' and is configured for the 'On Message' event. The function code is as follows:

```
1 // 4x65501, I:65500
2 // Actual state of DIP switch
3 global.set("C4_DIP_Actual", msg.payload[0]);
4
5 // 4x65502, I:65501
6 // LED1:GREEN: Actual state
7 global.set("C4_LED1_State", msg.payload[1]);
8 // 4x65503, I:65502
9 // LED1:GREEN: Actual Time1 in ms
10 global.set("C4_LED1_Time1", msg.payload[2]);
11 // 4x65504, I:65503
12 // LED1:GREEN: Actual Time2 in ms
13 global.set("C4_LED1_Time2", msg.payload[3]);
14
15 // 4x65505, I:65504
16 // LED2:WHITE: Actual state
17 global.set("C4_LED2_State", msg.payload[4]);
18 // 4x65506, I:65505
19 // LED2:WHITE: Actual Time1 in ms
20 global.set("C4_LED2_Time1", msg.payload[5]);
21 // 4x65507, I:65506
22 // LED2:WHITE: Actual Time2 in ms
23 global.set("C4_LED2_Time2", msg.payload[6]);
24
25 // 4x65508, I:65507
26 // LED3:RED: Actual state
27 global.set("C4_LED3_State", msg.payload[7]);
28 // 4x65509, I:65508
29 // LED3:RED: Actual Time1 in ms
30 global.set("C4_LED3_Time1", msg.payload[8]);
31 // 4x65510, I:65509
32 // LED3:RED: Actual Time2 in ms
33 global.set("C4_LED3_Time2", msg.payload[9]);
34
35 return msg;
```



# Important NodeRED<sup>®</sup> 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

The image displays three screenshots of the NodeRED MQTT broker configuration node interface, showing different tabs: Connection, Security, and Messages.

**Connection Tab:**

- Name: [Text field]
- Server: 127.0.0.1
- Port: 1883
- Connect automatically
- Use TLS
- Protocol: MQTT V3.1.1
- Client ID: Leave blank for auto generated
- Keep Alive: 60
- Session:  Use clean session

**Security Tab:**

- Username: resimqtt
- Password: [Masked]

**Messages Tab:**

- Message sent on connection (birth message)**
  - Topic: Leave blank to disable birth message
  - Retain: false
  - Payload: Payload
  - QoS: 0
- Message sent before disconnecting (close message)**
  - Topic: Leave blank to disable close message
  - Retain: false
  - Payload: Payload
  - QoS: 0
- Message sent on an unexpected disconnection (will message)**
  - Topic: Leave blank to disable will message
  - Retain: false
  - Payload: Payload
  - QoS: 0



# Important NodeRED<sup>®</sup> 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_D004', ... (1 bytes))
0
Client (null) received PUBLISH (d0, q0, r0, m0, 'RESI_C4/DigitalOutputs/C4_D005', ... (1 bytes))
1
Client (null) received PUBLISH (d0, q0, r0, m0, 'RESI_C4/DigitalOutputs/C4_D006', ... (1 bytes))
0
Client (null) received PUBLISH (d0, q0, r0, m0, 'RESI_C4/DigitalOutputs/C4_D007', ... (1 bytes))
0
Client (null) received PUBLISH (d0, q0, r0, m0, 'RESI_C4/DigitalOutputs/C4_D008', ... (1 bytes))
0
Client (null) received PUBLISH (d0, q0, r0, m0, 'RESI_C4/DigitalOutputs/C4_D001', ... (1 bytes))
1
Client (null) received PUBLISH (d0, q0, r0, m0, 'RESI_C4/DigitalOutputs/C4_D002', ... (1 bytes))
0
Client (null) received PUBLISH (d0, q0, r0, m0, 'RESI_C4/DigitalOutputs/C4_D003', ... (1 bytes))
0
Client (null) received PUBLISH (d0, q0, r0, m0, 'RESI_C4/DigitalOutputs/C4_D004', ... (1 bytes))
0
Client (null) received PUBLISH (d0, q0, r0, m0, 'RESI_C4/DigitalOutputs/C4_D005', ... (1 bytes))
1
Client (null) received PUBLISH (d0, q0, r0, m0, 'RESI_C4/DigitalOutputs/C4_D006', ... (1 bytes))
0
Client (null) received PUBLISH (d0, q0, r0, m0, 'RESI_C4/DigitalOutputs/C4_D007', ... (1 bytes))
0
Client (null) received PUBLISH (d0, q0, r0, m0, 'RESI_C4/DigitalOutputs/C4_D008', ... (1 bytes))
1
```



# NodeRED® sample flows explained



Raspberry Pi is a trademark of the Raspberry Pi Foundation.  
More information under [www.raspberrypi.org](http://www.raspberrypi.org)

it's all about perfection \_\_\_\_\_

**RESI**



# 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.

General declarations for RESI C4 controller

```

1 // Here you declare the type of the Controller
2 global.set("C4_TYPE", "RESI-C4-A-32DI24RO16AIOX");
3
4 // Total amount of DIs...
5 global.set("C4_MAX_DIS", 32);
6 // Total amount of DOs...
7 global.set("C4_MAX_DOS", 24);
8 // Total amount of AIOX...
9 global.set("C4_MAX_AIOX", 16);
10
11 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.

C4 global parameters for DIP switch+LEDs

C4 DIP Switch+LEDs

C4 update actual status of DIP+LEDs

C4 LED Status (active (500 msec.))

C4 Update DIP+LEDx Status

C4 set all LEDs if something has changed

Every 100ms

C4 Check LEDs

C4 LEDs changed ?

C4 Set All LEDs

C4 Update All LEDs (active)

Try LED1:GREEN

HIT C4 LED1 to OFF

HIT C4 LED1 to ON

HIT C4 LED1 to BLINK 1s

HIT C4 LED1 to BLINK 250ms

HIT C4 LED1 to FLASH 1s/2s

HIT C4 LED1 to FLASH 250ms/1s

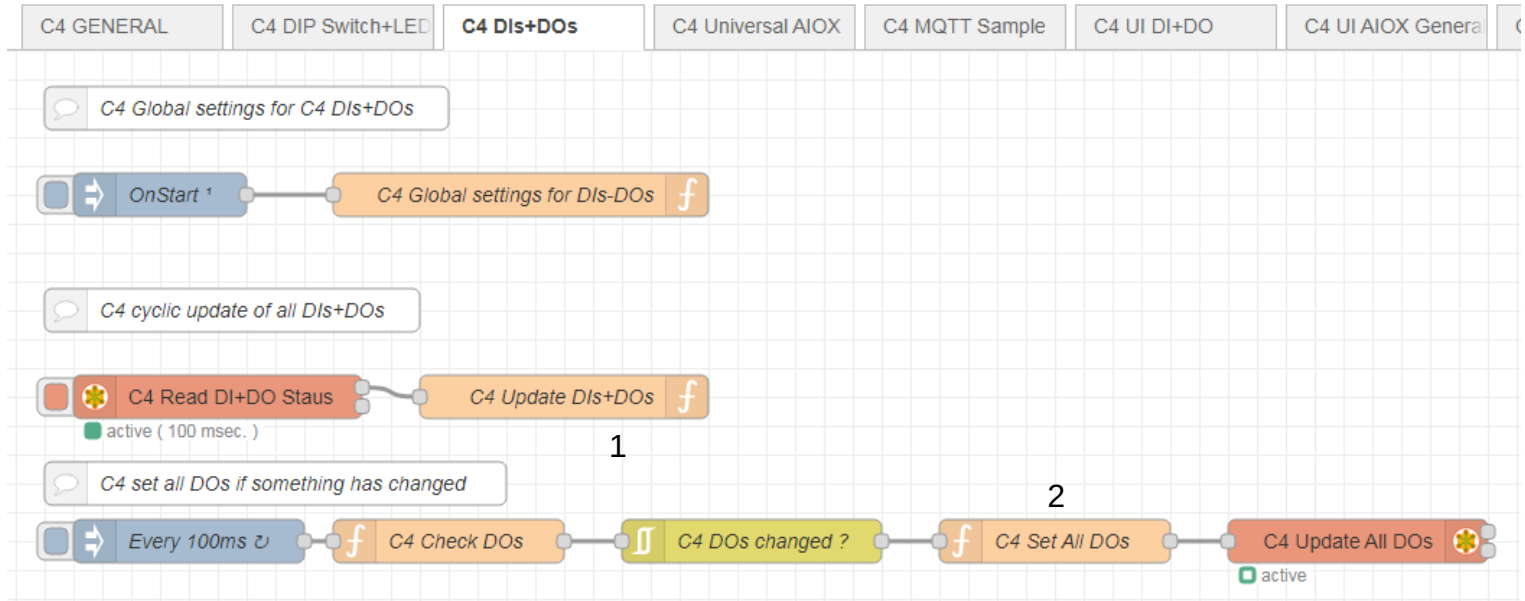


# NodeRED® flow

## 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;
2  let Word=0,Bit=0,V=0;
3
4  // Register list for C4-A-32DI24RO16AIOX
5  // 4x10001,I:10000: Change counter
6  // 4x10002,I:10001: DIGITAL INPUTS DI1-DI16
7  // 4x10003,I:10002: DIGITAL INPUTS DI17-DI32
8  // 4x10004,I:10003: DIGITAL OUTPUTS DO1-DO16
9  // 4x10005,I:10004: DIGITAL OUTPUTS DO17-DO24
10
11 // n DIS
12 for (i = 0; i < global.get("C4_MAX_DIS");i++) {
13     Word=~(i/16);
14     Bit=~(i%16);
15     // to take register 4x10002,I:10001 or 4x10003,I:10002
16     V=(msg.payload[1+Word]&(1<<Bit)) ? 1:0;
17     global.set("C4_DI"+String(i+1).padStart(2, '0'),V);
18 }
19
20 // n ROS
21 for (i = 0; i < global.get("C4_MAX_DOs"); i++) {
22     Word=~(i/16);
23     Bit=~(i%16);
24     // to take register 4x10004,I:10003 or 4x10005,I:10004
25     V=(msg.payload[3+Word]&(1<<Bit)) ? 1:0;
26     global.set("C4_DO"+String(i+1).padStart(2, '0')+"_Actual",V);
27 }
28
29 return msg ;

```

```

1  var i;
2
3  // for C4-A-32DI24RO16AIOX controller ...
4  // for C4-A-32DI24RO controller ...
5  if (global.get("C4_TYPE")=="RESI-C4-A-32DI24RO16AIOX" ||
6      global.get("C4_TYPE")=="RESI-C4-A-32DI24RO")
7  {
8      // Our controller offers 24 Relay outputs
9      // therefore we need 2 16 Bit registers
10     // 4x10004,I:10003: OUTPUTS DO1-DO16
11     // 4x10005,I:10004: OUTPUTS DO17-DO24
12     let C4_DOS=[ 0,0];
13     let Word=0,Bit=0;
14
15     // 24RO controller
16     C4_DOS[0] =C4_DOS[1]=0;
17
18     // Check for new value of C4_DOxx
19     for (i=0;i<global.get("C4_MAX_DOs");i++)
20     {
21         Word=~(i/16);
22         Bit=~(i%16);
23         if (global.get("C4_DO" +String(i+1).padStart(2, '0'))==1)
24         {
25             C4_DOS[Word] = C4_DOS[Word]|(1<<Bit);
26         }
27     }
28
29     msg.payload = {
30         'value': [ C4_DOS[0], C4_DOS[1] ],
31         // WRITE MULTIPLE HOLDING REGISTERS
32         'fc': 16,
33         // Our C4 MODBUS UnitID
34         'unitid': 1,
35         // 4x10004,I:10003: OUTPUTS DO1-DO16
36         // 4x10005,I:10004: OUTPUTS DO17-DO24
37         'address': 10003 ,
38         // for the 24 DOs or 2 registers...
39         'quantity': 2
40     }
41 }
42 else
43 {
44     msg.payload=null;
45 }
46 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 loses power.

```
1  var i,nr;
2
3  // n AIOXs ....
4
5  global.set("C4_AIOX_ISONLINE");
6
7
8  for (i=0;i<global.get("C4_MAX_AIOX");i++)
9  {
10     nr = String(i + 1).padStart(2, '0');
11     // type of AIOX: 0=Unused
12
13
14     global.set("C4_AIOX" + nr + "_TYPE", 0);
15
16     // Output value in Volt for AIOX
17     global.set("C4_AIOX" + nr + "_VO", 0);
18
19     // Input value in Volt for AIOX
20     global.set("C4_AIOX" + nr + "_VI", 0);
21
22     // Input value in mA for AIOX
23     global.set("C4_AIOX" + nr + "_CI", 0);
24
25     // Input value in Ohm for AIOX
26     global.set("C4_AIOX" + nr + "_OHM", 0);
27
28     // Input value in PT100 °C for AIOX
29     global.set("C4_AIOX" + nr + "_PT100_C", 0);
30
31     // Input value in PT1000 °C for AIOX
32     global.set("C4_AIOX" + nr + "_PT1000_C", 0);
33
34     // Input value in NI1000-DIN43760 °C for AIOX
35     global.set("C4_AIOX" + nr + "_NI1000_DIN43760_C", 0);
36
37
38
39     // possible IO types for AIOXx:
40     // =0: UNUSED
41     // =1: VOLTAGE INPUT[0-10V]
42     // =2: VOLTAGE INPUT[2-10V]
43     // =3: VOLTAGE OUTPUT[0-10V]
44     // =4: VOLTAGE OUTPUT[2-10V]
45     // =5: CURRENT INPUT LOOP POWERED[0-20mA]
46     // =6: CURRENT INPUT LOOP POWERED[4-20mA]
47     // =7: CURRENT INPUT EXTERNAL POWERED[0-20mA]
48     // =8: CURRENT INPUT EXTERNAL POWERED[4-20mA]
49     // =9: CURRENT OUTPUT[0-20mA]
50     // =10: CURRENT OUTPUT[4-20mA]
51     // =11: DIGITAL INPUT LOGIC 24V=
52     // =12: DIGITAL INPUT LOOP POWERED
53     // =13: RESISTANCE MEASUREMENT
54
55     // Configure your AIOX types here ...
56     global.set("C4_AIOX01_TYPE", 3);
57     global.set("C4_AIOX02_TYPE", 1);
58     global.set("C4_AIOX03_TYPE", 3);
59     global.set("C4_AIOX04_TYPE", 1);
60     global.set("C4_AIOX05_TYPE", 13);
61     global.set("C4_AIOX06_TYPE", 13);
62     global.set("C4_AIOX07_TYPE", 13);
63     global.set("C4_AIOX08_TYPE", 13);
64     global.set("C4_AIOX09_TYPE", 3);
65     global.set("C4_AIOX10_TYPE", 1);
66     global.set("C4_AIOX11_TYPE", 3);
67     global.set("C4_AIOX12_TYPE", 1);
68     global.set("C4_AIOX13_TYPE", 3);
69     global.set("C4_AIOX14_TYPE", 1);
70     global.set("C4_AIOX15_TYPE", 3);
71     global.set("C4_AIOX16_TYPE", 1);
72
73     return msg;
```

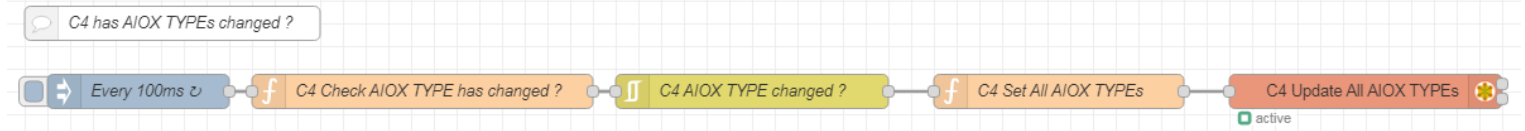


# NodeRED® flow

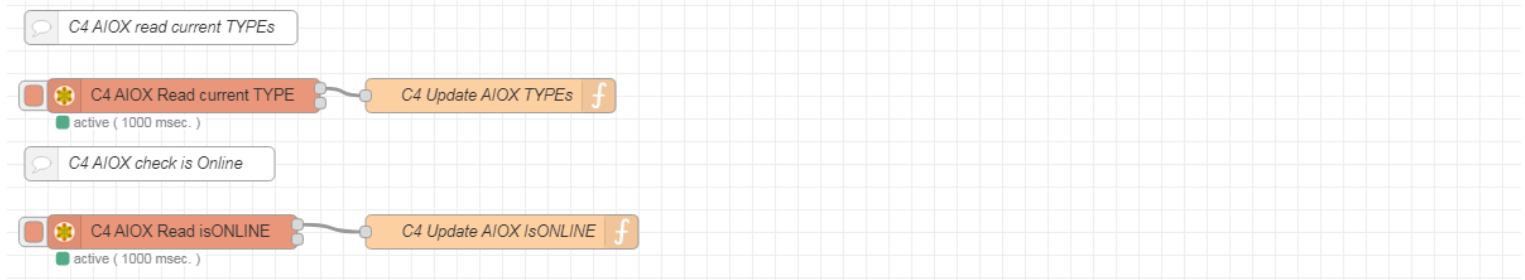
## C4-universal AIs+AOs

### C4 has AIOX TYPEs changed ?

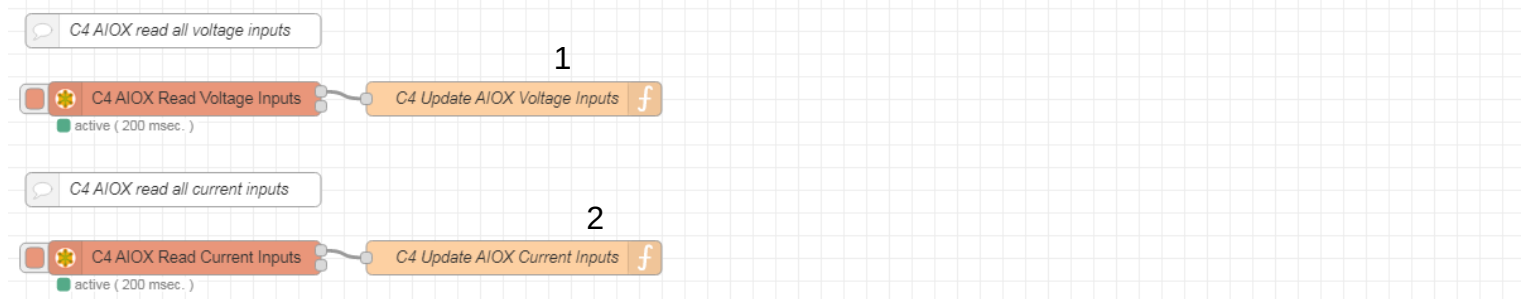
With this flow we check every 100ms if there is a new AIOX type. If yes, we update all AIOX types with one flex write to 16 holding registers.



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;
2
3  // Register list for C4-A-32DI24R016AIOX
4  // 4x40017,I:40016: VOLTAGE INPUT in VOLT*100
5  // 4x40018,I:40017: VOLTAGE INPUT in VOLT*100
6  // ...
7  // 4x40032,I:40031: VOLTAGE INPUT in VOLT*100
8
9  // n AIOX
10 for (i = 0; i < global.get("C4_MAX_AIOX");i++) {
11     let V=msg.payload[i]/100.0;
12     global.set("C4_AIOX"+String(i+1).padStart(2, '0')+"_VI",V);
13 }
14
15 return msg ;

```

```

1  |var i;
2
3  // Register list for C4-A-32DI24R016AIOX
4  // 4x40097,I:40096: CURRENT INPUT in MILLIAMPERE*100
5  // 4x40098,I:40097: CURRENT INPUT in MILLIAMPERE*100
6  // ...
7  // 4x40111,I:40110: CURRENT INPUT in MILLIAMPERE*100
8
9  // n AIOX
10 for (i = 0; i < global.get("C4_MAX_AIOX");i++) {
11     let V=msg.payload[i]/100.0;
12     global.set("C4_AIOX"+String(i+1).padStart(2, '0')+"_CI",V);
13 }
14
15 return msg ;

```

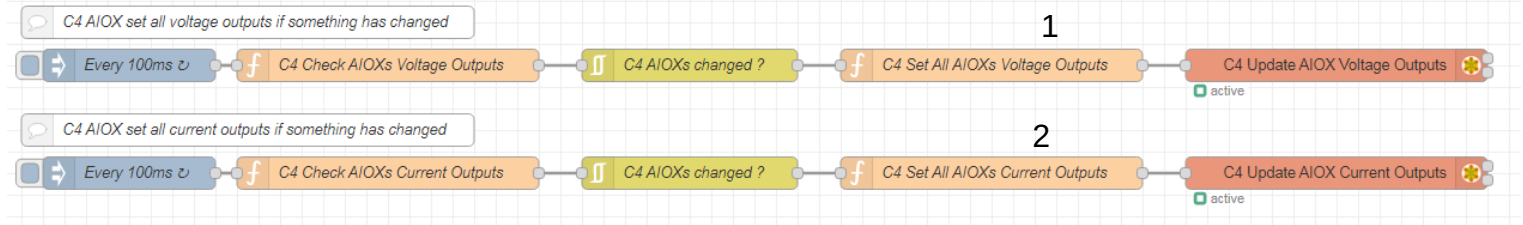


# NodeRED® flow

## 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

```

1 var i;
2 let V;
3 // for C4-A-32DI24R016AIOX controller ...
4 // for C4-A-32DI24R0 controller ...
5 if (global.get("C4_TYPE")=="RESI-C4-A-32DI24R016AIOX")
6 {
7     // 4x40049,I:40048: VOLTAGE OUTPUT AIOX1 VOLTS*100
8     // 4x40050,I:40049: VOLTAGE OUTPUT AIOX2 VOLTS*100
9     // ...
10    // 4x40064,I:40063: VOLTAGE OUTPUT AIOX16 VOLTS*100
11
12    msg.payload = {
13        'value': [ ],
14        // WRITE MULTIPLE HOLDING REGISTERS
15        'fc': 16,
16        // Our C4 MODBUS UnitID
17        'unitid': 1,
18        'address': 40048,
19        // for the 16 AIOX...
20        'quantity': 16
21    }
22
23    for (i=0;i<global.get("C4_MAX_AIOX");i++)
24    {
25        V=Math.trunc(global.get("C4_AIOX" +String(i+1).padStart(2, '0')+"_VO")*100);
26        msg.payload.value.push(V);
27    }
28 }
29 else
30 {
31     msg.payload=null;
32 }
33 return msg;

```

2

```

1 var i;
2 let V;
3 // for C4-A-32DI24R016AIOX controller ...
4 // for C4-A-32DI24R0 controller ...
5 if (global.get("C4_TYPE")=="RESI-C4-A-32DI24R016AIOX")
6 {
7     // 4x40129,I:40128: CURRENT OUTPUT AIOX1 MILLIAMPERE*100
8     // 4x40130,I:40129: CURRENT OUTPUT AIOX1 MILLIAMPERE*100
9     // ...
10    // 4x40144,I:40143: CURRENT OUTPUT AIOX1 MILLIAMPERE*100
11
12    msg.payload = {
13        'value': [ ],
14        // WRITE MULTIPLE HOLDING REGISTERS
15        'fc': 16,
16        // Our C4 MODBUS UnitID
17        'unitid': 1,
18        'address': 40128,
19        // for the 16 AIOX...
20        'quantity': 16
21    }
22
23    for (i=0;i<global.get("C4_MAX_AIOX");i++)
24    {
25        V=Math.trunc(global.get("C4_AIOX" +String(i+1).padStart(2, '0')+"_CO")*100);
26        msg.payload.value.push(V);
27    }
28 }
29 else
30 {
31     msg.payload=null;
32 }
33 return msg;

```



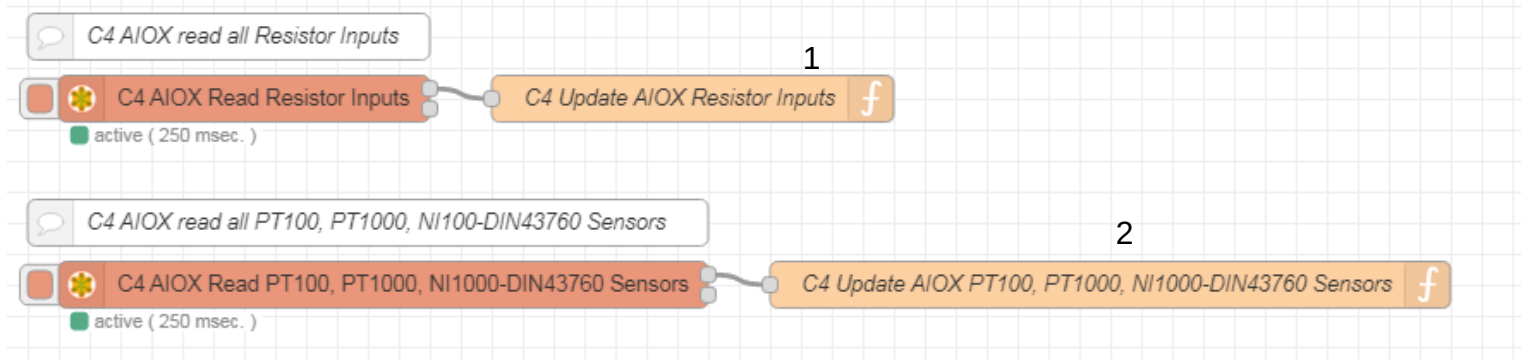
# NodeRED® flow

## 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\_AIOxx\_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\_AIOxx\_PT100\_C**, the PT1000 sensor values to **C4\_AIOxx\_PT1000\_C** and the NI1000-DIN43760 sensor values to **C4\_AIOxx\_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;
2
3  // Register list for C4-A-32DI24R016AIOX
4  // 4x41501,I:41500: RESISTOR INPUT in OHM*100
5  // 4x41502,I:41501: RESISTOR INPUT in OHM*100
6  // ...
7  // 4x41531,I:41530: RESISTOR INPUT in OHM*100
8
9  // n AIOX
10 for (i = 0; i < global.get("C4_MAX_AIOX");i++) {
11     let V=(msg.payload[i*2+1]+(msg.payload[i*2]<<16))/100.0;
12     global.set("C4_AIOX"+String(i+1).padStart(2, '0')+"_OHM",V);
13 }
14
15 return msg ;

```

1

```

1  var i;
2  let V,sint16;
3
4  // Register list for C4-A-32DI24R016AIOX
5  // 4x41049,I:41048: PT100 INPUT 1 in °C*100
6  // 4x41050,I:41049: PT100 INPUT 2 in °C*100
7  // ...
8  // 4x41064,I:41063: PT100 INPUT 16 in °C*100
9
10 // 4x41065,I:41064: PT1000 INPUT 1 in °C*100
11 // 4x41066,I:41065: PT1000 INPUT 2 in °C*100
12 // ...
13 // 4x41080,I:41079: PT1000 INPUT 16 in °C*100
14
15 // 4x41081,I:41080: NI1000-DIN43760 INPUT 1 in °C*100
16 // 4x41082,I:41081: NI1000-DIN43760 INPUT 2 in °C*100
17 // ...
18 // 4x41096,I:41095: NI1000-DIN43760 INPUT 16 in °C*100
19
20 // n AIOX
21 for (i = 0; i < global.get("C4_MAX_AIOX");i++) {
22     // PT100...
23
24     if (msg.payload[i]>32767) sint16=msg.payload[i]-65536;
25     else sint16=msg.payload[i];
26     V=sint16/100.0;
27     global.set("C4_AIOX"+String(i+1).padStart(2, '0')+"_PT100_C",V);
28
29     // PT1000...
30     if (msg.payload[i+16]>32767) sint16=msg.payload[i+16]-65536;
31     else sint16=msg.payload[i+16];
32     V=sint16/100.0;
33     global.set("C4_AIOX"+String(i+1).padStart(2, '0')+"_PT1000_C",V);
34
35     // NI1000-DIN43760...
36     if (msg.payload[i+32]>32767) sint16=msg.payload[i+32]-65536;
37     else sint16=msg.payload[i+32];
38     V=sint16/100.0;
39     global.set("C4_AIOX"+String(i+1).padStart(2, '0')+"_NI1000_DIN43760_C",V);
40 }
41
42 return msg ;

```

2



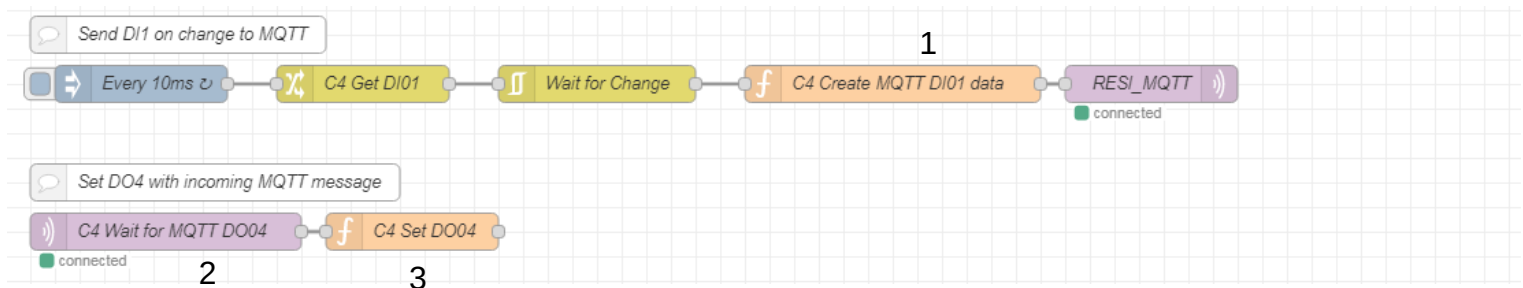
# NodeRED® flow

## MQTT messages for DI1-DOs

### MQTT messages

The next flows demonstrate how to 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 sent 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 DI1+DOs will check the change and update via MODBUS the outputs.



```
1 var msg1={
2   topic:"RESI_C4/DigitalInputs/C4_DI01",
3   payload: msg.payload
4 }
5
6 return msg1;
```

The image shows the configuration panel for an MQTT node named "C4 Wait for MQTT DO04". The "Server" is set to "127.0.0.1:1883". The "Action" is "Subscribe to single topic". The "Topic" is "RESI\_C4/DigitalOutputs/C4\_DO04". The "QoS" is set to "2". The "Output" is "auto-detect (parsed JSON object, string or buf)". The "Name" is "C4 Wait for MQTT DO04".

```
1 if (msg.payload == 0 || msg.payload == 1) {
2   global.set("C4_DO04", msg.payload);
3 }
4
5 return msg;
```

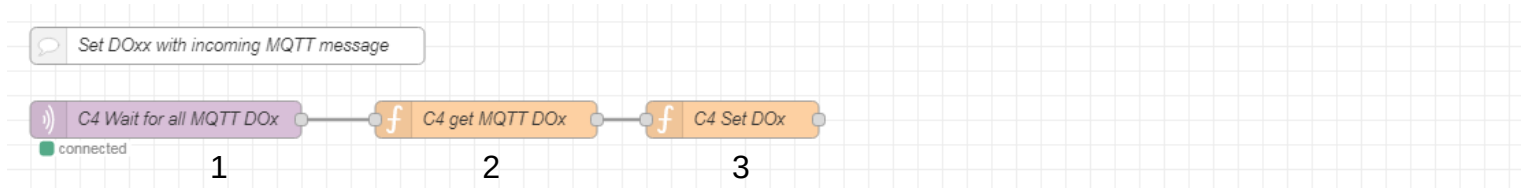


# NodeRED® flow

## 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 if the topic has the string format DOxx. If yes, the payload is checked for 0 or 1 to update the global variable 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.



### Edit mqtt in node

Delete 1 Cancel Done

**Properties**

Server: 127.0.0.1:1883

Action: Subscribe to single topic

Topic: RESI\_C4/DigitalOutputs/#

QoS: 2

Output: auto-detect (parsed JSON object, string or buf)

Name: C4 Wait for all MQTT DOx

```
1 |if (msg.topic.startsWith("RESI_C4/DigitalOutputs/"))
2 |{
3 |   let DOx=msg.topic.split("/").pop();      2
4 |   msg.topic=DOx;                          2
5 |}
6 |return msg;

1 |let ok=false;
2 |
3 |if (msg.topic.startsWith="C4_DO")          3
4 |{
5 |   let DOnr=msg.topic.substring(5,100);
6 |   if (Number.isInteger(parseInt(DOnr)))
7 |   {
8 |     let DOx=parseInt(DOnr,10);
9 |     if (DOx>=1 && DOx<global.get("C4_MAX_DOS"))
10 |    {
11 |      | ok=true;
12 |    }
13 |    if ((msg.payload == 0 || msg.payload == 1) && ok)
14 |    {
15 |      var DOxNAME="C4_DO"+String(DOx).padStart(2, '0');
16 |      global.set(DOxNAME, msg.payload);
17 |    }
18 |  }
19 |}
20 |return msg;
```



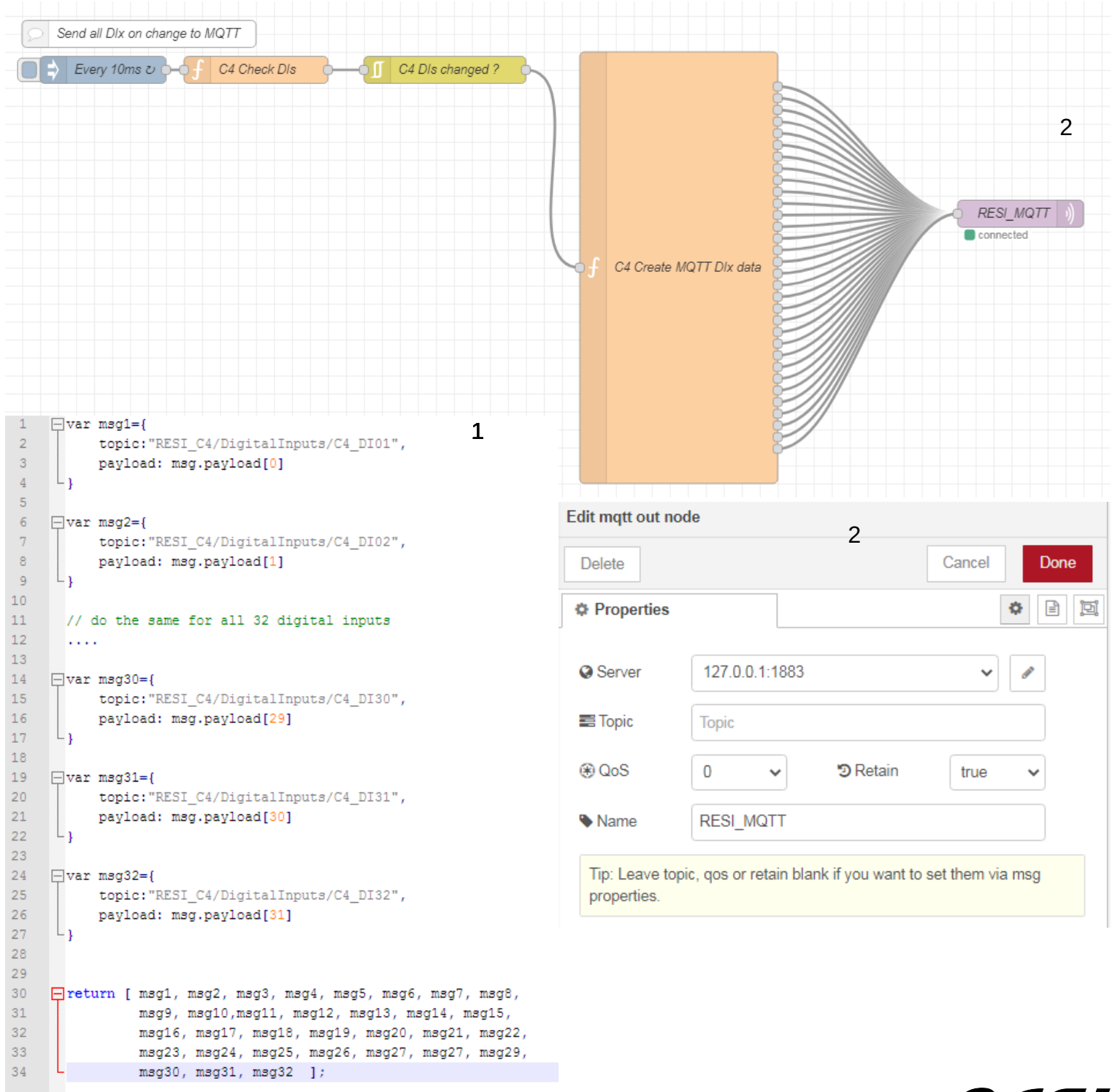


# NodeRED® flow

## 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 knot we prepare a new message named msg1 to msg32. The topic 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.

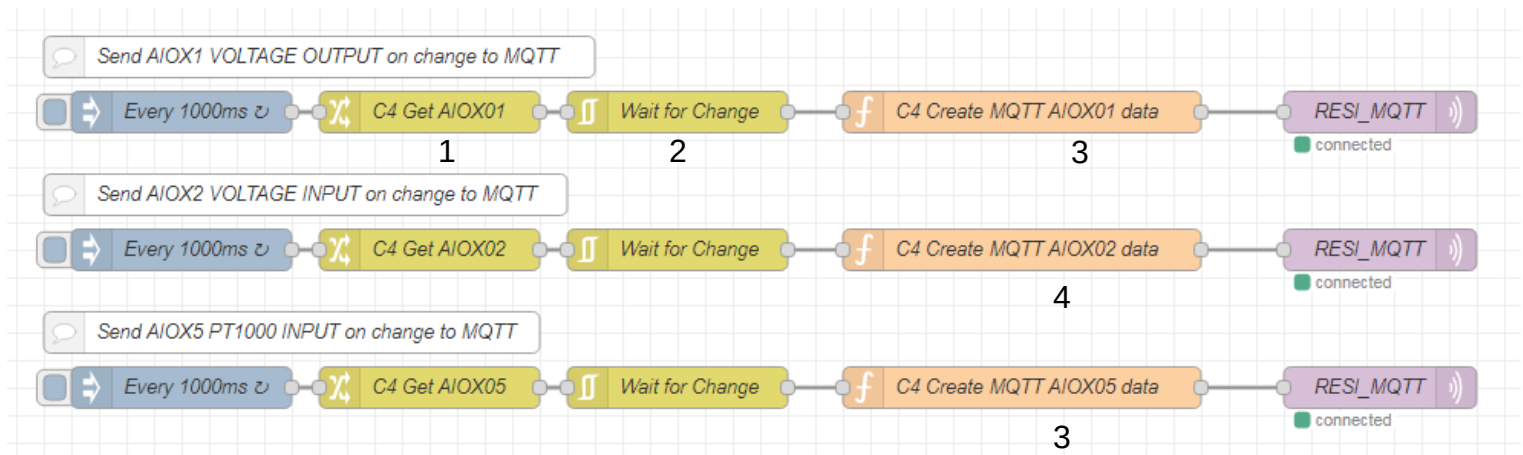


# NodeRED® flow

## 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.



**Edit change node**

Delete 1 Cancel Done

Properties

Name C4 Get AIOX01

Rules

Set msg.payload to the value global.C4\_AIOX01\_VO

Deep copy value

**Edit filter node**

Delete 2 Cancel Done

Properties

Mode block unless value change is greater or equal

0.01 compared to last valid output value

Property msg.payload

Apply mode separately for each

msg.topic

Name Wait for Change

```
1 var msg1={
2   topic:"RESI_C4/AnalogInOutputs/VoltageOutputs/C4_V001",
3   payload: msg.payload
4 }
5
6 return msg1;
```

3

```
1 var msg1={
2   topic:"RESI_C4/AnalogInOutputs/VoltageInputs/C4_VI01",
3   payload: msg.payload
4 }
5
6 return msg1;
```

4

```
1 var msg1={
2   topic:"RESI_C4/AnalogInOutputs/PT1000Inputs/C4_PT1000_C_05",
3   payload: msg.payload
4 }
5
6 return msg1;
```

5



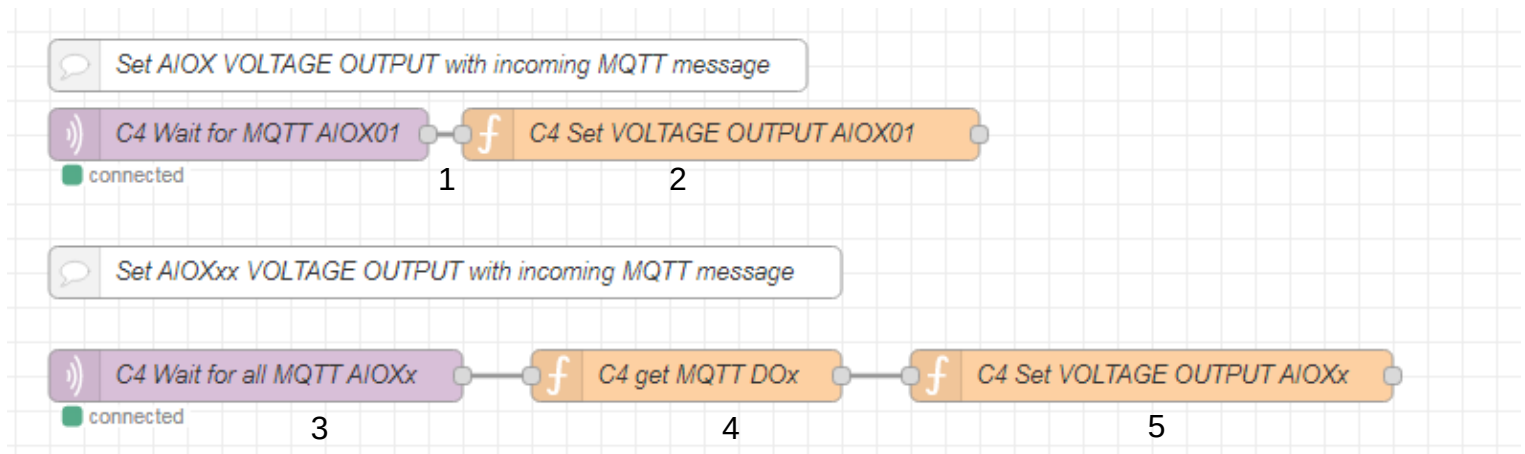
# NodeRED® flow

## 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 retrieve 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.



Edit mqtt in node

Delete 1 Cancel Done

---

Properties

Server: 127.0.0.1:1883

Action: Subscribe to single topic

Topic: RESI\_C4/AnalogInOutputs/VoltageOutputs/C4\_VO01

QoS: 2

Output: auto-detect (parsed JSON object, string or buffer)

Name: C4 Wait for MQTT AIOX01

```

1  if (msg.payload >= 0 || msg.payload <= 10.00) {
2  |   global.set("C4_AIOX01_VO", msg.payload);
3  }
4
5  return msg;|
    
```

Edit mqtt in node

Delete 3 Cancel Done

---

Properties

Server: 127.0.0.1:1883

Action: Subscribe to single topic

Topic: RESI\_C4/AnalogInOutputs/VoltageOutputs/#

QoS: 2

Output: auto-detect (parsed JSON object, string or buffer)

Name: C4 Wait for all MQTT AIOXx

```

1  if (msg.topic.startsWith("RESI_C4/AnalogInOutputs/VoltageOutputs/"))
2  {
3  |   let VOx=msg.topic.split("/").pop();           4
4  |   msg.topic=VOx;
5  }
6  return msg;|

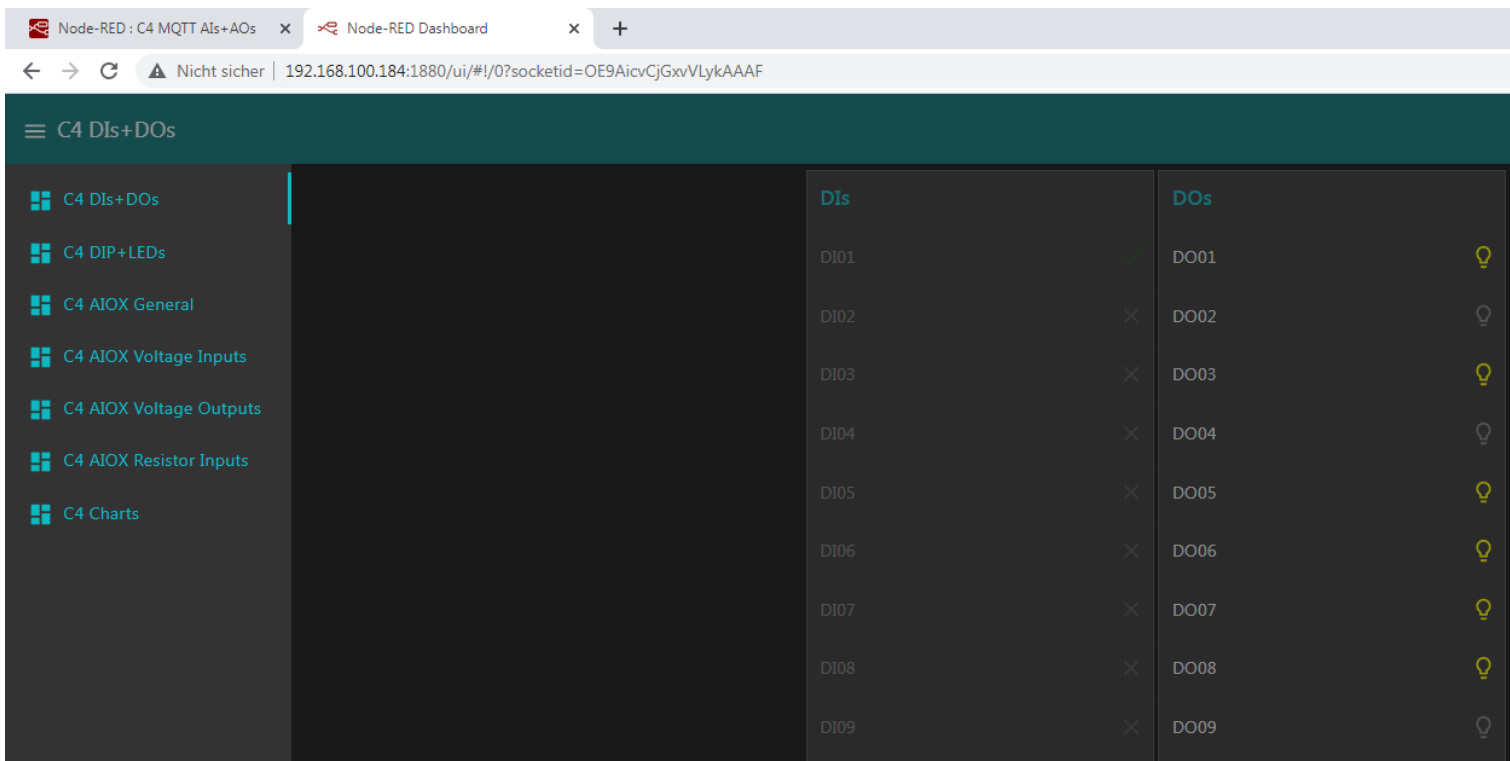
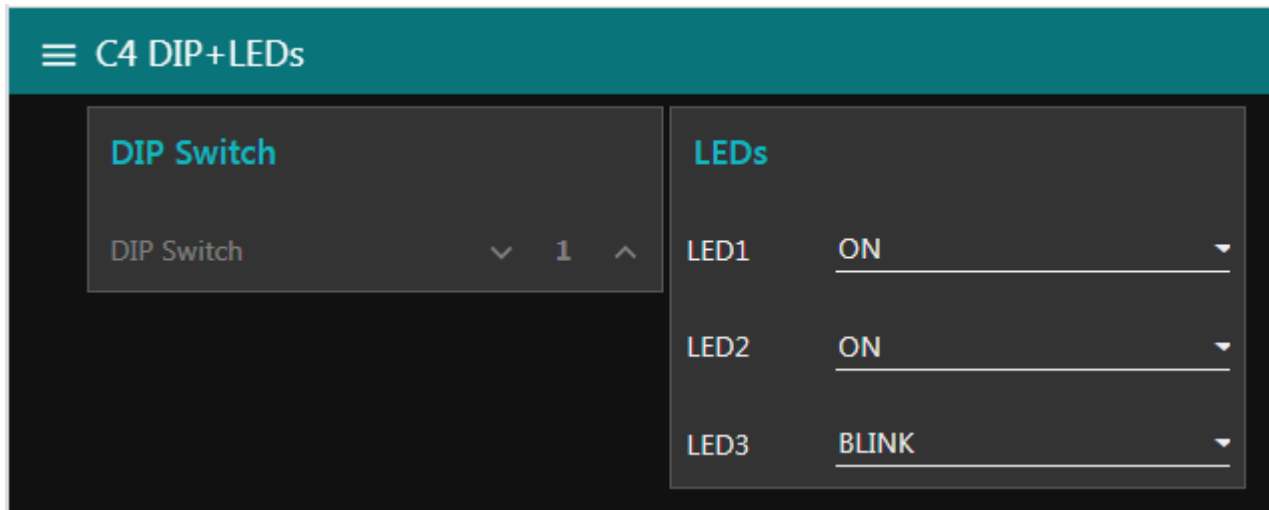
2  let value=0.;
3
4  node.log(msg.topic);           5
5  if (msg.topic.startsWith("C4_VO")
6  {
7  |   let VOnr=msg.topic.substring(5,100);
8  |   if (Number.isInteger(parseInt(VOnr,10)))
9  |   {
10 |     let VOx=parseInt(VOnr,10);
11 |     if (VOx>=1 && VOx<global.get("C4_MAX_AIOX"))
12 |     {
13 |       | ok=true;
14 |     }
15 |   }
16 |   if (!isNaN(parseFloat(msg.payload)))
17 |   {
18 |     value=parseFloat(msg.payload);
19 |   }
20 |   else
21 |   {
22 |     | ok=false;
23 |   }
24 |   if ((value >= 0 || value <= 10.0) && ok)
25 |   {
26 |     | var VOxNAME="C4_AIOX"+String(VOx).padStart(2, '0')+"_VO";
27 |     | global.set(VOxNAME, value);
28 |   }
29 |   }
30 | }
31 return msg;|
    
```



# NodeRED® flow dashboard UI

nodered dashboard UI

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



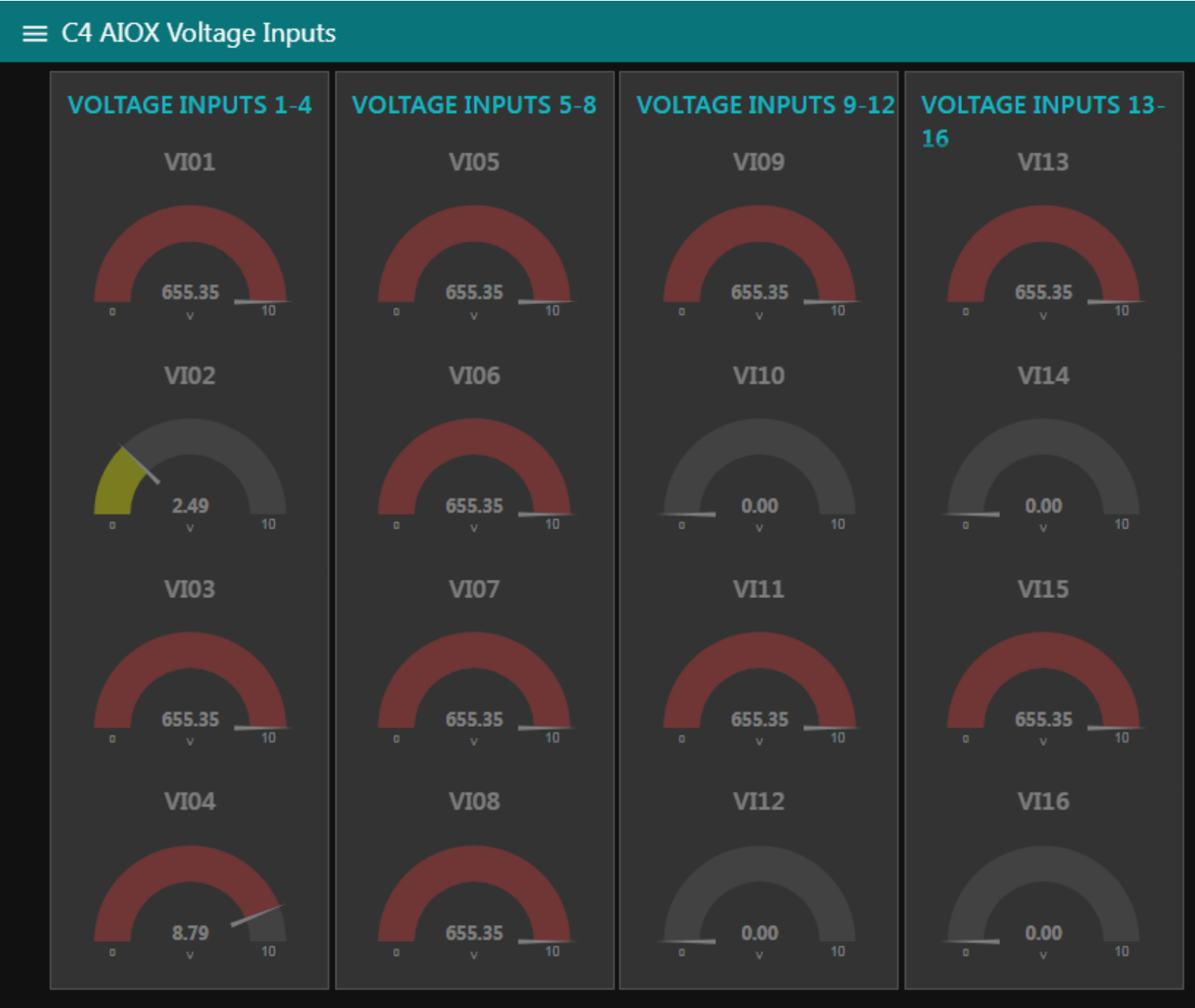
# NodeRED® flow dashboard UI

☰ C4 AIOX General

General info	AIOX 1-8 TYPE	AIOX 9-16 TYPE
AIOX IS ONLINE ✓	AIOX01 TYPE <u>VO[0-10V]</u> ▼	AIOX09 TYPE <u>VO[0-10V]</u> ▼
	AIOX02 TYPE <u>VI[0-10V]</u> ▼	AIOX10 TYPE <u>VI[0-10V]</u> ▼
	AIOX03 TYPE <u>VO[0-10V]</u> ▼	AIOX11 TYPE <u>VO[0-10V]</u> ▼
	AIOX04 TYPE <u>VI[0-10V]</u> ▼	AIOX12 TYPE <u>VI[0-10V]</u> ▼
	AIOX05 TYPE <u>RESISTANCE</u> ▼	AIOX13 TYPE <u>VO[0-10V]</u> ▼
	AIOX06 TYPE <u>RESISTANCE</u> ▼	AIOX14 TYPE <u>VI[0-10V]</u> ▼
	AIOX07 TYPE <u>RESISTANCE</u> ▼	AIOX15 TYPE <u>VO[0-10V]</u> ▼
	AIOX08 TYPE <u>RESISTANCE</u> ▼	AIOX16 TYPE <u>VI[0-10V]</u> ▼



# NodeRED® flow dashboard UI



# NodeRED® flow dashboard UI

☰ C4 AIOX Voltage Outputs

VOLTAGE OUTPUTS 1-8			VOLTAGE OUTPUTS 9-16		
VO01	▼ 2.50 ▲		VO09	▼ 0.00 ▲	
VO02	▼ 0.00 ▲		VO10	▼ 0.00 ▲	
VO03	▼ 8.80 ▲		VO11	▼ 0.00 ▲	
VO04	▼ 0.00 ▲		VO12	▼ 0.00 ▲	
VO05	▼ 0.00 ▲		VO13	▼ 0.00 ▲	
VO06	▼ 0.00 ▲		VO14	▼ 0.00 ▲	
VO07	▼ 0.00 ▲		VO15	▼ 0.00 ▲	
VO08	▼ 0.00 ▲		VO16	▼ 0.00 ▲	



# NodeRED® flow dashboard UI

## ☰ C4 AIOX Resistor Inputs

### RESISTORS

Sensor	OHM	PT100[°C]	PT1000[°C]	NI1000-DIN43760[°C]
AIOX01	-0.01	-327.68	-327.68	-327.68
AIOX02	-0.01	-327.68	-327.68	-327.68
AIOX03	-0.01	-327.68	-327.68	-327.68
AIOX04	-0.01	-327.68	-327.68	-327.68
AIOX05	1094.75	-327.67	24.32	16.93
AIOX06	222408.16	-327.67	-327.67	-327.67
AIOX07	221314.77	-327.67	-327.67	-327.67
AIOX08	220231.99	-327.67	-327.67	-327.67
AIOX09	-0.01	-327.68	-327.68	-327.68
AIOX10	-0.01	-327.68	-327.68	-327.68
AIOX11	-0.01	-327.68	-327.68	-327.68
AIOX12	-0.01	-327.68	-327.68	-327.68
AIOX13	-0.01	-327.68	-327.68	-327.68
AIOX14	-0.01	-327.68	-327.68	-327.68
AIOX15	-0.01	-327.68	-327.68	-327.68
AIOX16	-0.01	-327.68	-327.68	-327.68





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

