

www.vskom.de

OnRISC
IoT Manual
Edition: May 2022



Vision Systems GmbH

Tel: +49 40 528 401 0

Fax: +49 40 528 401 99

Web: www.visionsystems.de

Support: faq.visionsystems.de

The software described in this manual is furnished under a license agreement and may be used only in accordance with the terms of that agreement.

Copyright Notice

Copyright © 2009-2022 Vision Systems. All rights reserved. Reproduction without permission is prohibited.

Trademarks

VScom is a registered trademark of Vision Systems GmbH. All other trademarks and brands are property of their rightful owners.

Disclaimer

Vision Systems reserves the right to make changes and improvements to its product without providing notice.

Vision Systems provides this document “as is”, without warranty of any kind, either expressed or implied, including, but not limited to, its particular purpose. Vision Systems reserves the right to make improvements and/or changes to this manual, or to the products and/or the programs described in this manual, at any time.

Information provided in this manual is intended to be accurate and reliable. However, Vision Systems assumes no responsibility for its use, or for any infringements on the rights of third parties that may result from its use.

This product might include unintentional technical or typographical errors. Changes are periodically made to the information herein to correct such errors, and these changes are incorporated into new editions of the publication.

Contents

| | | |
|-----------|---|-----------|
| 1 | Introduction | 6 |
| 2 | IoT Related Protocols | 7 |
| 2.1 | MQTT | 7 |
| 2.1.1 | Installation | 7 |
| 2.1.2 | Example: mqtt_gpio | 8 |
| 3 | IoT Programming: Node-RED | 11 |
| 3.1 | OnRISC Devices (Baltos) | 11 |
| 3.1.1 | Installation | 11 |
| 3.1.2 | Example: Connect WLAN switch to a User LED | 12 |
| 3.1.3 | Example: GPIO Access via Modbus/TCP | 13 |
| 3.2 | Interface Converter (NetCAN Plus, NetCom Plus) | 18 |
| 3.2.1 | Enabling Node-RED Support | 18 |
| 3.2.2 | Upload Node-RED Configuration File | 18 |
| 3.2.3 | Import Node-RED Examples | 18 |
| 3.2.4 | Documentation of the Node Functions | 19 |
| 3.2.5 | Setting Device Parameters over SNMP Nodes | 20 |
| 3.2.6 | NetCAN Modes of Operation | 21 |
| 3.2.6.1 | SocketCAN | 21 |
| 3.2.6.1.1 | Preparation | 21 |
| 3.2.6.1.2 | Example of a simple TCP raw NetCAN emulation (ASCII protocol) | 22 |
| 3.2.6.1.3 | Example with MQTT broker | 23 |
| 3.2.6.2 | NetCAN Legacy | 23 |
| 3.2.6.2.1 | Preparation | 23 |
| 3.2.6.2.2 | Example with TCP Raw Port | 24 |
| 3.2.7 | NetCom Modes of Operation | 26 |
| 3.2.7.1 | Native Serial Port | 26 |
| 3.2.7.1.1 | Preparation | 26 |
| 3.2.7.1.2 | Example of a simple serial TCP raw Server | 26 |
| 3.2.7.1.3 | Example of a simple serial UDP Server and Client | 27 |

List of Figures

| | | |
|----|--|----|
| 1 | Baltos as IoT Router | 6 |
| 2 | MQTT Scheme | 7 |
| 3 | Node-RED: libonrisc Nodes | 12 |
| 4 | Node-RED: WLAN switch and LED: Initial Scheme | 13 |
| 5 | Node-RED: WLAN switch and LED: Configured Scheme | 13 |
| 6 | Node-RED: Modbus/GPIO Example Flow Chart | 14 |
| 7 | Node-RED Modbus/GPIO Example Dashboard | 15 |
| 8 | Node-RED: Modbus Write Node | 15 |
| 9 | Node-RED: Modbus Client Configuration | 16 |
| 10 | Node-RED: Modbus Read | 16 |
| 11 | Node-RED: modbus2bin Function | 17 |
| 12 | Enabling Node-RED | 18 |
| 13 | Node-RED import nodes | 19 |
| 14 | Node-RED Documentation Window | 19 |
| 15 | snmpwalk example | 20 |
| 16 | snmpget example | 20 |
| 17 | snmpset example | 20 |
| 18 | snmpset example properties | 21 |
| 19 | Disabling of CAN port | 21 |
| 20 | NetCAN emulation example | 22 |
| 21 | Function node source code | 22 |
| 22 | MQTT nodes | 23 |
| 23 | MQTT example | 23 |
| 24 | Enabling of CAN port | 24 |
| 25 | TCP example | 24 |
| 26 | Config CAN Channel code | 25 |
| 27 | Disabling of serial port | 26 |
| 28 | TCP raw serial server example | 26 |
| 29 | Serial port settings | 27 |
| 30 | UDP serial server example | 27 |
| 31 | Serial port settings | 28 |

List of Tables

1 MQTT Scheme 7

1 Introduction

This manual gives insight into IoT universe and also shows how OnRISC can be used as an IoT Router. For deeper understanding we recommend to look at this free eBook: *A Reference Guide To The Internet Of Things*¹.

IoT or the Internet of Things is a technology about connecting physical devices like sensors and actuators to the cloud in order to analyse data collected from physical devices and control the actuators. So the role of OnRISC will be to talk to sensors/actuators via its interfaces like CAN, Ethernet, serial, GPIO, WLAN etc. and provide this data to a cloud service (see Figure 1 on page 6). OnRISC device can be either a gateway between the sensors and the cloud and/or maintain controlling logic.

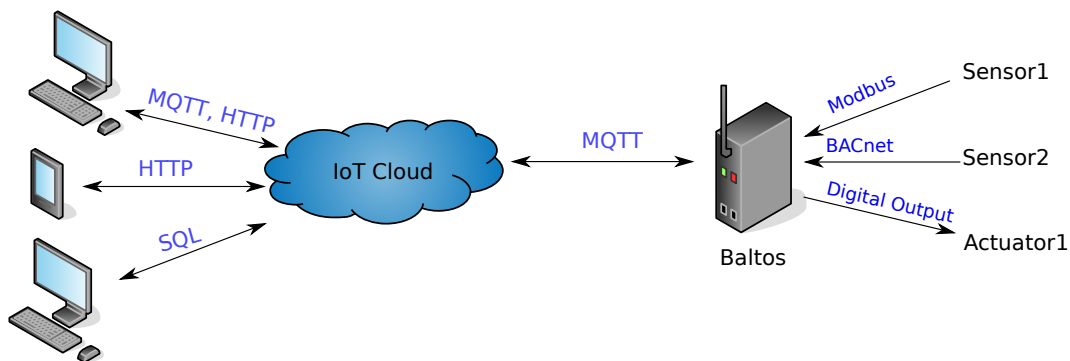


Figure 1: Baltos as IoT Router

The first section explains IoT related protocols and how you can store your data in the cloud. In the second section you'll learn how you can create your IoT applications via visual flow control framework almost without a coding effort.

The examples shown in the manual require either Buildroot or Debian installation, but the concept can be applied to other Linux distributions too.

¹<https://bridgera.com/ebook/>

2 IoT Related Protocols

There are many protocols used in IoT world: MQTT, HTTP, CoAP, AMQP etc. We will describe only MQTT protocol as it is wide spread and it is supported by the most IoT cloud providers like Amazon, IBM and Microsoft.

2.1 MQTT

As stated on the projects site² MQTT is a machine-to-machine (M2M)/"Internet of Things" connectivity protocol. It was designed as an extremely lightweight publish/subscribe messaging transport. It is useful for connections with remote locations where a small code footprint is required and/or network bandwidth is at a premium. As of version 3.1.1 MQTT became an OASIS Standard.

MQTT utilizes a "publish/subscribe" message transport model. The central part of this protocol is a MQTT broker, that receives, manages and routes messages among its nodes. Client authentication and authorization is also made by the MQTT broker. Table 1 on page 7 and Figure 2 on page 7 provide a MQTT example where three publishers send temperature sensor values (T1-T3) and three subscribers receiving only relevant values.

| Topic Name | Subscriber |
|------------|------------|
| T1 | Sub1, Sub3 |
| T2 | Sub2 |
| T3 | Sub2, Sub3 |

Table 1: MQTT Scheme

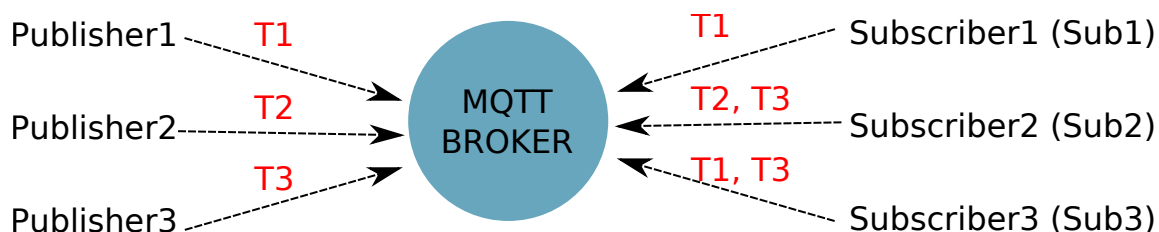


Figure 2: MQTT Scheme

2.1.1 Installation

Buildroot already provides both C and Python protocol implementations:

- BR2_PACKAGE_PAHO_MQTT_C
- BR2_PACKAGE_PYTHON_PAHO_MQTT

If you need OpenSSL support, it must be activated prior to MQTT building.

Debian installation requires following steps for C protocol implementation:

1. `apt install libssl-dev debhelper fakeroot lsb-release`

²<http://mqtt.org>

2. `cd /usr/src/`
3. `git clone https://github.com/eclipse/paho.mqtt.c.git`
4. `cd paho.mqtt.c/`
5. `mkdir build`
6. `cd build`
7. `cmake .. -DPAHO_WITH_SSL=TRUE -DPAHO_BUILD_DEB_PACKAGE=TRUE`
8. `make`
9. `cpack`
10. `dpkg -i *.deb`

For Python implementation:

1. `apt install python-pip python3-pip`
2. `pip install paho-mqtt` - for Python 2.x environment
3. `pip3 install paho-mqtt` - for Python 3.x environment

2.1.2 Example: `mqtt_gpio`

This program publishes Baltos digital inputs:

- `onrisc/gpio/input/0`
- `onrisc/gpio/input/1`
- `onrisc/gpio/input/2`
- `onrisc/gpio/input/3`

And subscribes to digital outputs:

- `onrisc/gpio/output/0`
- `onrisc/gpio/output/1`
- `onrisc/gpio/output/2`
- `onrisc/gpio/output/3`

If any output on the MQTT broker would change, Baltos would propagate this value to its digital outputs. And as soon as Baltos inputs change their state, these values will be propagated to the MQTT broker.

In order to build the C test example perform:

1. `cd /usr/src/`
2. `git clone https://github.com/visionssystemsgmbh/programming_examples.git`
3. `cd programming_examples/mqtt/c/`
4. `mkdir build`
5. `cd build/`

6. `cmake ..`

7. `make`

`mqtt_gpio` has following syntax:

```
mqtt_gpio IP address [port]
```

IP address indicates MQTT broker's IP address and *port* is an optional argument specifying broker's TCP port (default value is 1883).

`mqtt_gpio` requires following setup:

1. MQTT broker (either on Baltos or on your host)
2. connect IN0 with OUT0 using a 4,7k resistor

For our example we will use Node.js based broker Mosca³ running on a desktop PC. Install Node.js according to the project's documentation⁴. After this invoke:

1. `npm install mosca pino -g` with super user permissions
2. `mosca -v | pino`

We assume, that your host has IP address 192.168.1.170 and MQTT broker works with the standard TCP port 1883. On Baltos invoke:

```
./mqtt_gpio 192.168.1.170
```

You'll get following output showing that Baltos sent its initial digital input state:

```
Waiting for up to 10 seconds for publication of 0
on topic onrisc/gpio/input/0 for client with ClientID: Baltos
Message with token value 2 delivery confirmed
Message with delivery token 2 delivered
Waiting for up to 10 seconds for publication of 0
on topic onrisc/gpio/input/1 for client with ClientID: Baltos
Message with token value 3 delivery confirmed
Message with delivery token 3 delivered
Waiting for up to 10 seconds for publication of 0
on topic onrisc/gpio/input/2 for client with ClientID: Baltos
Message with token value 4 delivery confirmed
Message with delivery token 4 delivered
Waiting for up to 10 seconds for publication of 0
on topic onrisc/gpio/input/3 for client with ClientID: Baltos
Message with token value 5 delivery confirmed
Message with delivery token 5 delivered
```

Mosca's output shows, that client with ID "Baltos" has made a connection and subscribed to the topic "*onrisc/gpio/output/#*" i.e. all published outputs:

³<http://www.mosca.io>

⁴<https://nodejs.org/en/download/package-manager>

```
[2017-09-13T15:25:52.565Z] INFO (mosca/16765 on debian9): client connected
  client: "Baltos"
[2017-09-13T15:25:52.571Z] INFO (mosca/16765 on debian9): subscribed to topic
  topic: "onrisc/gpio/output/#"
  qos: 1
  client: "Baltos"
```

Let's toggle OUT1:

```
onrisctool -a -0x10 -b 0x10
```

In reaction to this mqtt_gpio would produce following output:

```
Waiting for up to 10 seconds for publication of 1
on topic onrisc/gpio/input/0 for client with ClientID: Baltos
Message with token value 6 delivery confirmed
Message with delivery token 6 delivered
```

3 IoT Programming: Node-RED

3.1 OnRISC Devices (Baltos)

Node-RED⁵ is a programming tool for wiring together hardware devices, APIs and online services in new and interesting ways.

It provides a browser-based editor that makes it easy to wire together flows using the wide range of nodes in the palette that can be deployed to its runtime in a single-click.

Initial Node-RED installation already provides a lot of useful functions like sending/receiving UDP/TCP packets, working with HTTP/MQTT and other protocols. The framework can also be extended via various third-party modules (see this collection⁶) and as Node-RED is written around Node.js⁷ you can access any Node.js modules via “function” node or create your own Node-RED nodes⁸.

3.1.1 Installation

In Buildroot you’ll have to activate Node.js package and provide a list of required modules:

- BR2_PACKAGE_OPENSSL
- BR2_PACKAGE_NODEJS
- BR2_PACKAGE_NODEJS_MODULES_ADDITIONAL="node-red node-red-dashboard"
- BR2_PACKAGE_NODE_RED_CONTRIB_LIBONRISC

For Debian 9 perform following steps:

1. `apt install -y apt-transport-https`
2. `echo "deb https://deb.nodesource.com/node_6.x stretch main" > /etc/apt/sources.list.d/nodesource.list`
3. `wget -q0- https://deb.nodesource.com/gpgkey/nodesource.gpg.key | apt-key add -`
4. `apt update`
5. `apt install -y nodejs`
6. `npm install -g node-red node-red-dashboard`
7. install libonrisc Node.js and Node-RED bindings as explained on respective GitHub pages⁹

⁵<https://nodered.org>

⁶<https://flows.nodered.org/>

⁷<https://nodejs.org/en/>

⁸<https://nodered.org/docs/creating-nodes/>

⁹<https://github.com/visionsystemsgmbh/libonrisc> and <https://github.com/visionsystemsgmbh/node-red-contrib-libonrisc>

3.1.2 Example: Connect WLAN switch to a User LED

This introductory example shows how to use Node-RED editor and create a simple flow connecting WLAN switch to a User LED (green LED on Baltos LED tower). This example will work only with Baltos iR5221/3220 devices.

Start Node-RED process:

```
node-red
```

We assume that Baltos has its default IP address 192.168.254.254. As soon as you can see the output shown below, you can point your browser to <http://192.168.254.254:1880/>.

```
Welcome to Node-RED
=====

15 Sep 08:53:43 - [info] Node-RED version: v0.17.5
15 Sep 08:53:43 - [info] Node.js version: v6.11.3
15 Sep 08:53:43 - [info] Linux 3.18.32 arm LE
15 Sep 08:53:47 - [info] Loading palette nodes
15 Sep 08:54:05 - [info] Dashboard version 2.4.3 started at /ui
15 Sep 08:54:07 - [info] Settings file : /root/.node-red/settings.js
15 Sep 08:54:07 - [info] User directory : /root/.node-red
15 Sep 08:54:07 - [info] Flows file : /root/.node-red/flows_onrisc.json
15 Sep 08:54:07 - [info] Server now running at http://127.0.0.1:1880/
15 Sep 08:54:07 - [info] Starting flows
15 Sep 08:54:08 - [info] Started flows
```

Scroll the Node’s panel till you find “libonrisc” section (see Figure 3 on page 12). With left mouse button pressed drag at first “onrisc wlan sw” node and then “onrisc led” node. Now find the “output” section and drag the “debug” node. Connect all three nodes as shown in Figure 4 on page 13.

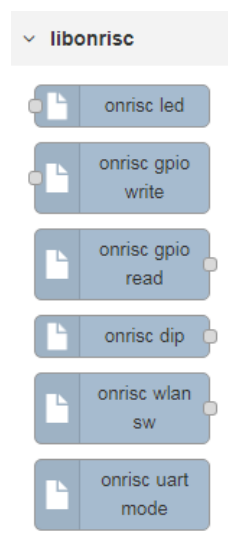


Figure 3: Node-RED: libonrisc Nodes

Perform a double-click on the “onrisc-wlan-sw” node. Specify “Name” as “WLAN switch” and “Rate” as 1000 (the value is in milliseconds). Click on “Done” to finish editing node. Now perform

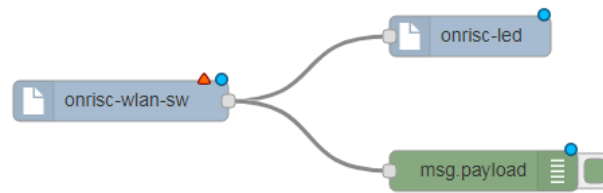


Figure 4: Node-RED: WLAN switch and LED: Initial Scheme

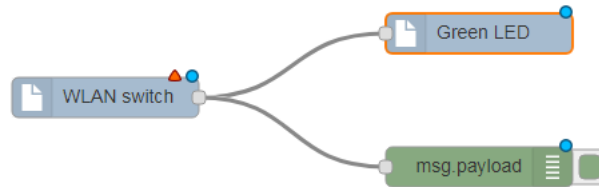


Figure 5: Node-RED: WLAN switch and LED: Configured Scheme

a double-click on the “onrisc-led” node and specify “Name” as “Green LED” and select “App” in the LED drop-down list. Click on “Done”. You’ll get following flow (see Figure 5 on page 13).

Click on “Deploy” button in the upper right corner to start the flow. Clicking on “Debug” tab in the upper right corner will show messages produced by the “debug” node. Below you can see two messages received with one second difference. This is the reading rate we configured in the “onrisc-wlan-sw” node.

```
10/10/2017, 11:21:36 AMnode: 7bb06521.99decc
msg.payload : number
0
10/10/2017, 11:21:37 AMnode: 7bb06521.99decc
msg.payload : number
0
```

Now toggle Baltos WLAN switch and you’ll see the green LED turning on and off depending on the switch position.

3.1.3 Example: GPIO Access via Modbus/TCP

In this example you’ll learn Node-RED’s dashboard¹⁰ feature, that allows you to create GUI. It will also show how to use Modbus¹¹ in Node-RED. We will need a modbusgpio daemon (see Section “GPIO over Modbus/TCP” in the User Manual) and hence this example will work only in Debian. Perform following actions:

1. `cd /usr/src/`
2. `git clone https://github.com/visionsystemsgmbh/programming_examples.git`
3. `cp programming_examples/node-red/gpio.json /root/.node-red/flows_onrisc.json`
4. `npm install -g --unsafe-perm node-red-contrib-modbus`

¹⁰<https://github.com/node-red/node-red-dashboard>

¹¹<https://github.com/biancode/node-red-contrib-modbus>

5. modbusgpio 502&

6. node-red

Point your browser to <http://192.168.254.254:1880/>. You'll see the flow chart as shown in Figure 6 on page 14. Now open a new tab in your browser and point it to <http://192.168.254.254:1880/ui/#/0> to see the dashboard as shown in Figure 7 on page 15. You can click on the switches to change the digital output state and if you connect INs with OUTs via 4,7k resistors, you'll see, how digital inputs get changed accordingly.

Let's look at Modbus related nodes. Double-click on the "output0" node and you'll see configuration panel as shown in Figure 8 on page 15. As one can see we write the incoming value to toggle a single coil at address 4. This is OUT0 on Baltos. "baltoslocal" describes the network connection to a Modbus/TCP server (see Figure 9 on page 16). In this case the server is on the Baltos itself so we can reach it via 127.0.0.1 address.

Double-click on the "input1" node will show "Modbus Read" node (see Figure 10 on page 16). We read 4 input status bits from address 0, i.e. IN0..IN3 on the Baltos green connector. Reading rate 1 second. This node returns an array with true or false values. In order to show single inputs we need a function node "modbus2bin", that converts the array to binary output.

The dashboard consists of one tab "Baltos Modbus GPIO Example" and two groups "Inputs" and "Outputs". The "switch" nodes toggle digital outputs and the "text" nodes show input state and are assigned to the related group.

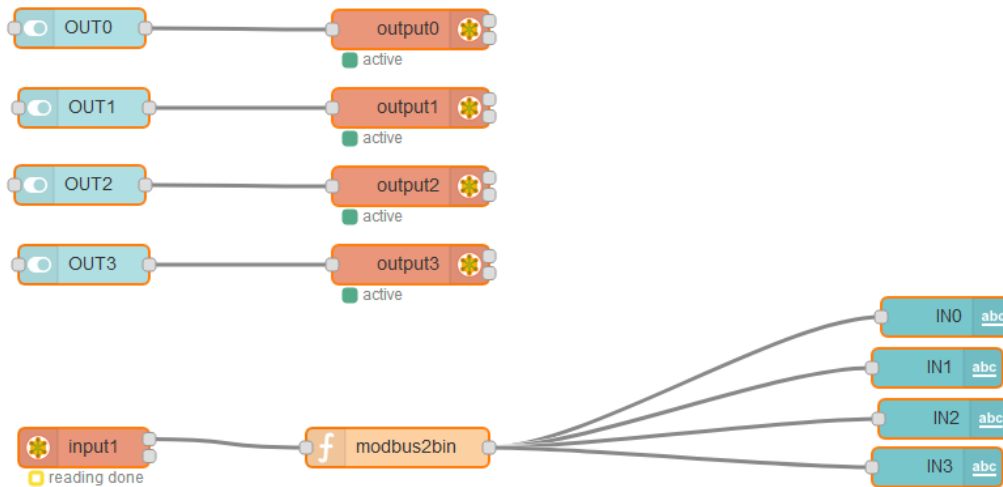


Figure 6: Node-RED: Modbus/GPIO Example Flow Chart



Figure 7: Node-RED Modbus/GPIO Example Dashboard

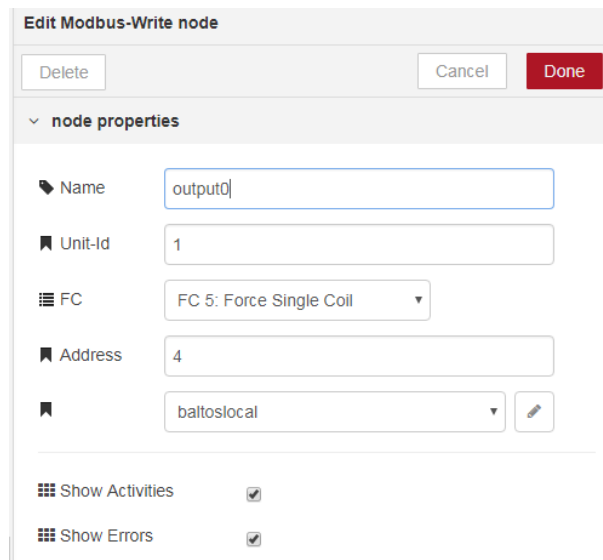


Figure 8: Node-RED: Modbus Write Node

The screenshot shows the configuration interface for a 'Modbus-Write' node in Node-RED. The title bar reads 'Modbus-Write > Edit modbus-client node'. At the top, there are three buttons: 'Delete', 'Cancel', and 'Update'. The configuration fields are as follows:

- Name:** Input field containing 'baltoslocal'.
- Type:** Dropdown menu set to 'TCP'.
- Host:** Input field containing '127.0.0.1'.
- Port:** Input field containing '502'.
- Unit-Id:** Input field containing '1'.
- Timeout (ms):** Input field containing '1000'.
- Reconnect timeout (ms):** Input field containing '2000'.
- Log states changes:** Checked checkbox.
- Queue commands:** Checked checkbox.
- Queue delay (ms):** Input field containing '1'.

Figure 9: Node-RED: Modbus Client Configuration

The screenshot shows the configuration interface for an 'Edit Modbus-Read' node in Node-RED. The title bar reads 'Edit Modbus-Read node'. At the top, there are three buttons: 'Delete', 'Cancel', and 'Done'. The configuration fields are as follows:

- Name:** Input field containing 'input1'.
- Unit-Id:** Input field containing '1'.
- FC:** Dropdown menu set to 'FC 2: Read Input Status'.
- Address:** Input field containing '0'.
- Quantity:** Input field containing '4'.
- Poll Rate:** Input field containing '1' and a dropdown menu set to 'second(s)'.
- Server:** Dropdown menu set to 'baltoslocal' with an edit icon.
- Show Activities:** Checked checkbox.
- Show Errors:** Checked checkbox.

Figure 10: Node-RED: Modbus Read

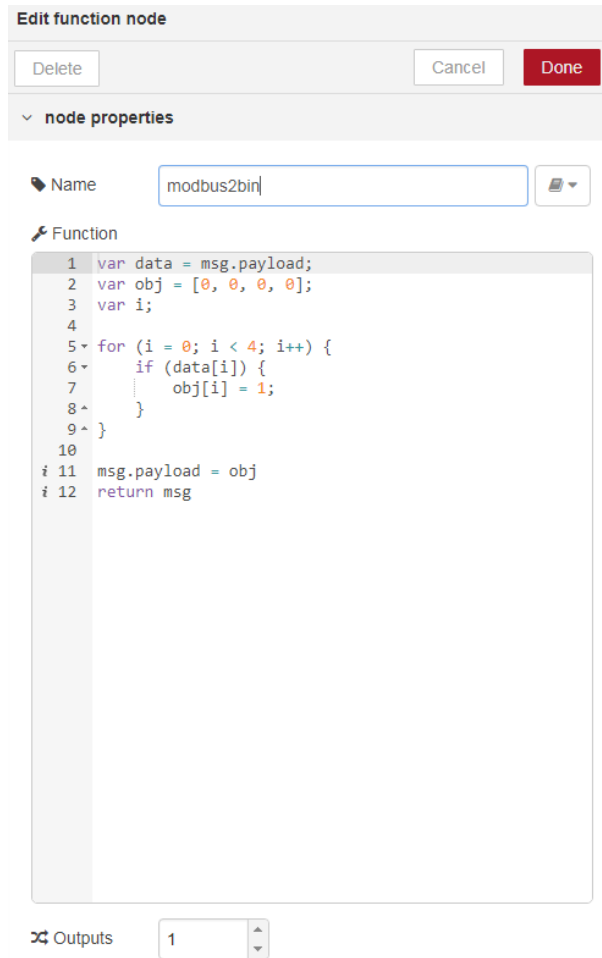


Figure 11: Node-RED: modbus2bin Function

3.2 Interface Converter (NetCAN Plus, NetCom Plus)

3.2.1 Enabling Node-RED Support

The server configuration page of the NetCAN and NetCom Plus gives you the option to enable the Node-RED support and configure the default TCP port of the service. If you do so, the converter will be restarted and the service is running. Please note, that the start of the Node-RED service takes a little bit longer than the normal NetCAN or NetCom Plus startup routines.

Node-RED

Node-RED On ▾

TCP Port

Config File:

Config File: Keine ausgewählt

[Open Worksheet in new window](#)

[Open Dashboard in new window](#)

Figure 12: Enabling Node-RED

3.2.2 Upload Node-RED Configuration File

You can download Node-RED's configuration file over the web-frontend, edit parameters of your choice and upload it back to your device (Figure 12 on page 18). Take a look at Node-RED's online documentation for all its settings ¹². One of the most important parts is the securing your device and setting a password for your dashboard ¹³. The password is a Bcrypt hash and can be generated with Node-RED's command line tools on your host or with an online generator of your choice ¹⁴.

3.2.3 Import Node-RED Examples

To import our flow examples and make use of them, you first have to download the desired ones from our GitHub repository ¹⁵. Then you must open Node-RED's worksheet over the provided link from 12. Now you can open the import dialog from the main menu and paste the example source into the text box or open the file directly (Figure 13 on page 19). And it doesn't hurt to take a look at the examples from the other device classes to build up a general understanding.

¹²<https://nodered.org/docs/user-guide/runtime/configuration>

¹³<https://nodered.org/docs/user-guide/runtime/securing-node-red>

¹⁴<https://bcrypt-generator.com/>

¹⁵https://github.com/visionsystemsgmbh/programming_examples/tree/master/CAN/node-red

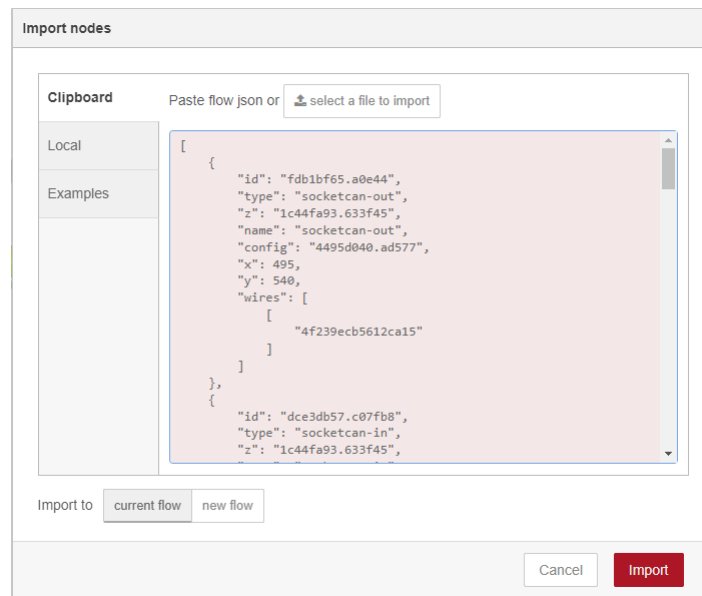


Figure 13: Node-RED import nodes

3.2.4 Documentation of the Node Functions

If you are unsure with any parameter at any point of time, you can use Node-RED's worksheet documentation feature (Figure 14 on page 19). There you'll find hints and descriptions of config node parameters and its practical meaning.

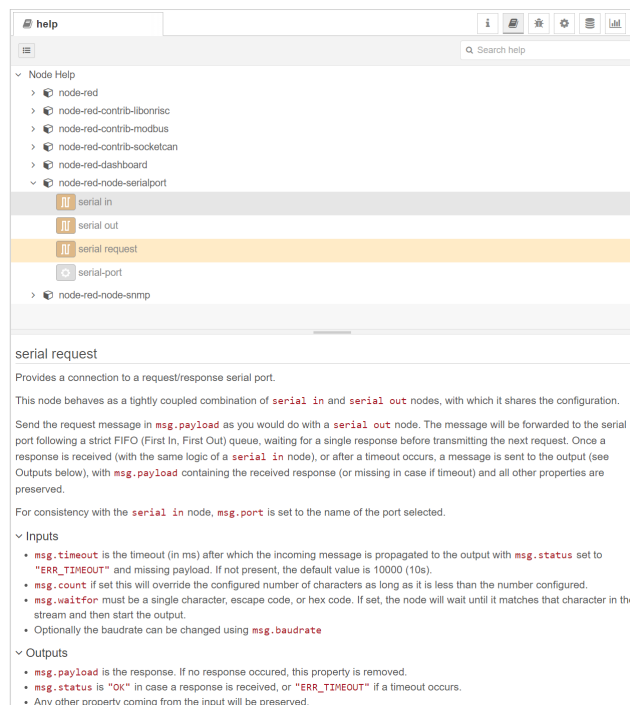


Figure 14: Node-RED Documentation Window

3.2.5 Setting Device Parameters over SNMP Nodes

All NetCAN and NetCom Plus device settings can be configured with SNMP ¹⁶. We have console tools¹⁷ to get/set SNMP settings and the related MIB files are part of the firmware archive¹⁸ (see the `snmp` folder inside the ZIP file). You can also omit the MIB files and use the numerical OIDs with Node-RED's SNMP nodes directly. To get the desired one, you can take a look at the MIB files and read them out or call `snmpwalk` for a sub-tree (Figure 15 on page 20).

```
c:\Tools\SNMP>snmpwalk -c public -v1 192.168.1.89 VSCOM-MIB::vsCom
VSCOM-MIB::vsnciSerialNumber.0 = Gauge32: 210108110
VSCOM-MIB::vsnciSwVersion.0 = Gauge32: 262144
VSCOM-MIB::vsnciHwVersion.0 = Gauge32: 262146
VSCOM-MIB::vsnciConfigPort.0 = INTEGER: 23
VSCOM-MIB::vsnciIpAddress.0 = IpAddress: 192.168.1.89
VSCOM-MIB::vsnciIpNetMask.0 = IpAddress: 255.255.255.0
VSCOM-MIB::vsnciBcastAddress.0 = IpAddress: 192.168.1.255
VSCOM-MIB::vsnciIPUseDHCP.0 = INTEGER: 1
VSCOM-MIB::vsnciInternalControl.0 = INTEGER: nop(0)
VSCOM-MIB::vsnciPrdType.0 = STRING: "Plus 111"
VSCOM-MIB::vsnciGateway.0 = IpAddress: 192.168.1.1
VSCOM-MIB::vsnciDNSServer.0 = IpAddress: 0.0.0.0
```

Figure 15: snmpwalk example

Next, you can run `snmpget` with the “-On” switch for a single parameter returning its full OID (Figure 16 on page 20).

```
c:\Tools\SNMP>snmpget -c public -v1 -On 192.168.1.89 VSCOM-MIB::vsnciIpAddress.0
.1.3.6.1.4.1.12695.1.6.0 = IpAddress: 192.168.1.89
```

Figure 16: snmpget example

Now insert a `snmpset` node into your flow and set its properties to the desired values (Figure 18 on page 21). You must take care, that the community name is set to `root` or if you'd set a password for your device, the community name must be the plain text password. And also note that every parameter change must be saved into the flash with the object “`vsnciInternalControl(10)`” and the value “`commit-flash(2)`”.

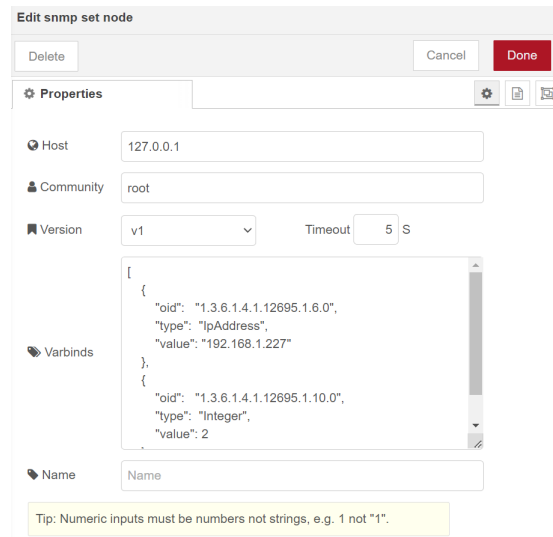


Figure 17: snmpset example

¹⁶ https://en.wikipedia.org/wiki/Simple_Network_Management_Protocol

¹⁷ <https://www.vsc.com/de/download/multiio/winXP/tools/netcom-tools.zip>

¹⁸ <https://www.vsc.com/de/download/multiio/others/driver/>



```

[
  {
    "oid": "1.3.6.1.4.1.12695.1.6.0",
    "type": "IpAddress",
    "value": "192.168.1.227"
  },
  {
    "oid": "1.3.6.1.4.1.12695.1.10.0",
    "type": "Integer",
    "value": 2
  }
]

```

Tip: Numeric inputs must be numbers not strings, e.g. 1 not "1".

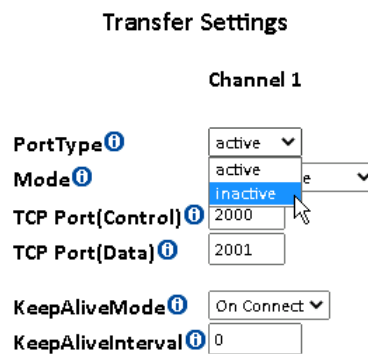
Figure 18: snmpset example properties

3.2.6 NetCAN Modes of Operation

3.2.6.1 SocketCAN

3.2.6.1.1 Preparation

The SocketCAN node is based on the `node-red-contrib-socketcan`¹⁹ module. In order to be able to use the module, you must disable the CAN port in the channel configuration of the NetCAN's Plus web front-end (Figure 19 on page 21).



Transfer Settings

Channel 1

PortType **active**

Mode **inactive**

TCP Port(Control) 2000

TCP Port(Data) 2001

KeepAliveMode On Connect

KeepAliveInterval 0

Figure 19: Disabling of CAN port

If you do so, you can switch to Node-RED's worksheet view. Therefore visit the web front-end on the default port 1880 (e.g. `http://<IP address>:1880`).

¹⁹<https://github.com/grodansparadis/node-red-contrib-socketcan>

3.2.6.1.2 Example of a simple TCP raw NetCAN emulation (ASCII protocol)

We create a basic function set of the original NetCAN firmware function with this example (Figure 20 on page 22). You should download the sample flow from our GitHub repository and import it in Node-RED²⁰.

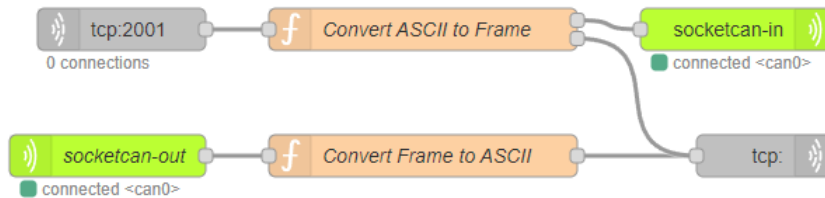


Figure 20: NetCAN emulation example

The flow contains two function nodes which convert the ASCII frames to Node-RED's SocketCAN structure and vice versa. As you can see, its quite simple to add your own processing of the relevant data (Figure 21 on page 22).

```

Edit function node
Delete Cancel Done
Properties
Name: Convert ASCII to Frame
Setup On Start On Message On Stop
1- function hexToBytes(hex) {
2   for (var bytes = [], c = 0; c < hex.length; c += 2)
3     bytes.push(parseInt(hex.substr(c, 2), 16));
4   return bytes;
5- }
6
7   cmd = msg.payload;
8
9   if (cmd.charAt(0).toUpperCase() === 'T')
10- {
11     can = { canfd: false, rtr: false };
12
13     if (cmd.charAt(0) === 'T') // extended frame
14- {
15       can.ext = true;
16       can.canid = Number('0x' + cmd.substr(1, 9));
17       can.dlc = Number('0x' + cmd.substr(9, 10));
18       can.data = hexToBytes(cmd.substr(10));
19       answer = 'Z\r\n';
20- }
21     else // standard frame
22- {
23       can.ext = false;
24       can.canid = Number('0x' + cmd.substr(1, 4));
25       can.dlc = Number('0x' + cmd.substr(4, 5));
26       can.data = hexToBytes(cmd.substr(5));
27       answer = 'z\r\n';
28- }
29
30     msg = { payload: can };
31     msg2 = { payload: answer };
32     return [msg, msg2]
33- }
34   else if (cmd.toUpperCase() === 'V')
35     answer = 'V1010\r\n';
36   else if (cmd.toUpperCase() === 'N')
37     answer = 'N12345678\r\n';
38   else if (cmd.toUpperCase() === 'F')
39     answer = 'F00\r\n';
40   else if (cmd.toUpperCase() === 'O' || cmd.toUpperCase() === 'C' ||
41     cmd.charAt(0).toUpperCase() === 'Z' ||
42     cmd.charAt(0).toUpperCase() === 'D' ||
43     cmd.charAt(0).toUpperCase() === 'S' ||
44     cmd.charAt(0).toUpperCase() === 'W')
45     answer = '\r\n';
46   else
47     answer = '\b\r\n'; // error status
48
49   msg.payload = answer;
50   return [null, msg];

```

Figure 21: Function node source code

²⁰https://github.com/visionsystemsgmbh/programming_examples/tree/master/CAN/node-red/flow_netcan_emulation_tcp.json

3.2.6.1.3 Example with MQTT broker

For this example you can use your own MQTT broker as shown in the section 2.1 or a free one as the broker.emqx.io ²¹. You can find the associated nodes (Figure 22 on page 23) in the network section of Node-RED's worksheet.

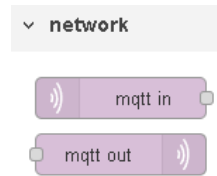


Figure 22: MQTT nodes

Then you can draw a simple flow on the worksheet (Figure 23 on page 23).

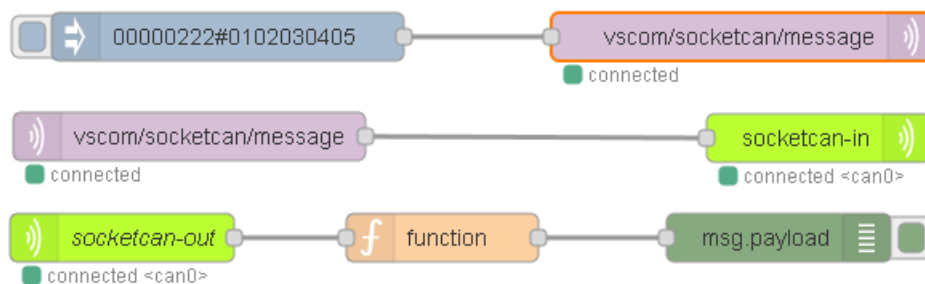


Figure 23: MQTT example

This example sends out a text formatted frame to the MQTT broker which will be received from the client on the other side and directed to the connected SocketCAN node that it sends out to the physical CAN port. You can download and import this flow (Figure 23 on page 23) from our GitHub example repository ²².

3.2.6.2 NetCAN Legacy

3.2.6.2.1 Preparation

To use the legacy mode of the NetCAN Plus which works with the built-in functions of its firmware, you should leave all port type settings enabled and set your desired parameters (Figure 24 on page 24).

²¹<https://www.emqx.io/>

²²https://github.com/visionsystemsgmbh/programming_examples/tree/master/CAN/node-red/flow_mqtt_broker.json

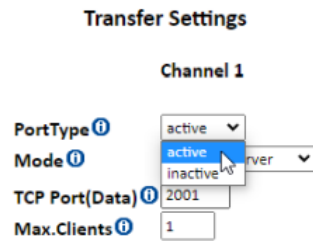


Figure 24: Enabling of CAN port

3.2.6.2.2 Example with TCP Raw Port

You can communicate with the NetCAN's Plus built-in functions locally over TCP for example (Figure 25 on page 24). The “Config CAN Channel” injection node runs once and configures the initial parameters of the CAN channel. You could inject a test frame with the “Inject Test Frame” node which send it out to the TCP request node. This one also retrieves the incoming frames and status on the CAN port and prints it out to the debug node.

This example could also be downloaded from our repository ²³ and imported into a Node-RED flow.

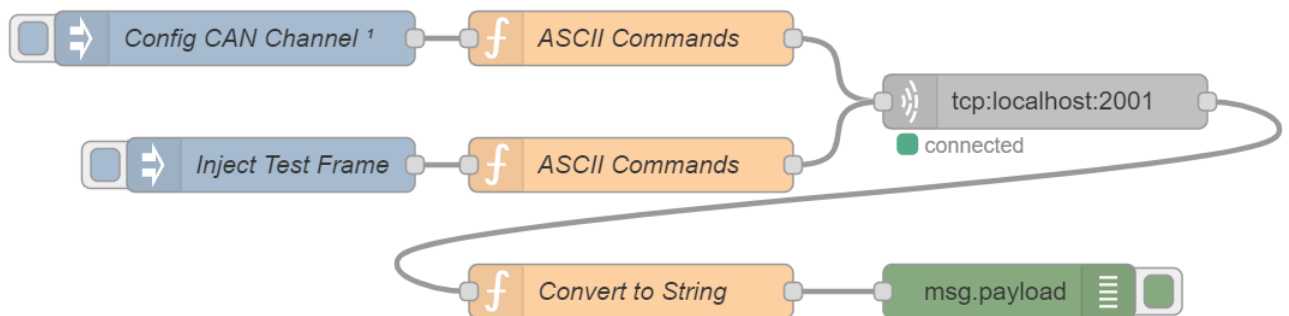


Figure 25: TCP example

Figure 26 on page 25 shows you the initialization code of the function node triggered once from "Config CAN Channel".

²³https://github.com/visionsystemsgmbh/programming_examples/tree/master/CAN/node-red/flow_legacy_tcp.json

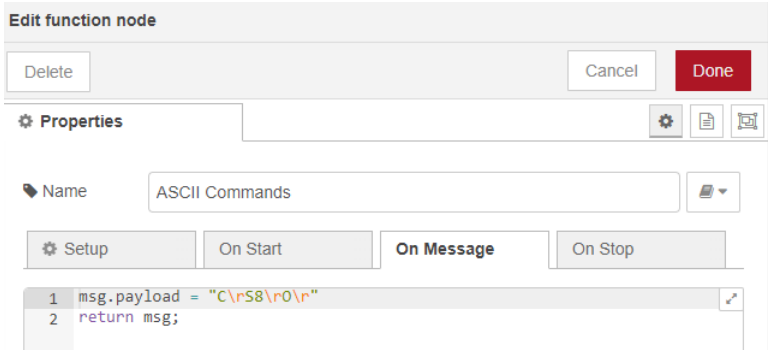


Figure 26: Config CAN Channel code

3.2.7 NetCom Modes of Operation

3.2.7.1 Native Serial Port

3.2.7.1.1 Preparation

The native serial port node is based on the `node-red-node-serialport`²⁴ module. In order to be able to use the module, you must disable the serial port in the serial config page of the NetCom's Plus web front-end (Figure 27 on page 26).

Serial Settings

| | |
|---------------------------|---------------|
| | Port 1 |
| PortType (current) | rs232 |
| Baud Base | 3000000 |
| PortType ⓘ | rs232 ▼ |
| Baudrate ⓘ | rs232 |
| Manual ⓘ | inactive |
| FlowType ⓘ | 115200 |
| DataBit ⓘ | None ▼ |
| | 8 ▼ |

Figure 27: Disabling of serial port

If you do so, you can switch to Node-RED's worksheet view. Therefore visit the web front-end on the default port 1880 (e.g. `http://<IP address>:1880`).

3.2.7.1.2 Example of a simple serial TCP raw Server

This example creates a simple server which listens on the configured TCP port and routes the data to the serial port and vice versa (Figure 28 on page 26). This example is also available from our Git repository²⁵.

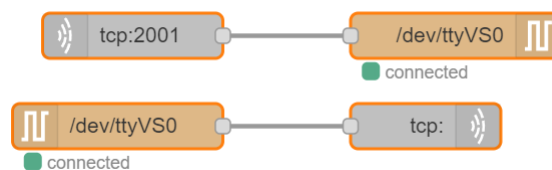


Figure 28: TCP raw serial server example

You must configure your desired serial port parameters in the node settings tab (Figure 29 on page 27). The serial ports device files of the NetCom Plus devices are assigned in the form of `"/dev/ttyVS[0-16]"`.

²⁴<https://github.com/node-red/node-red-nodes/tree/master/io/serialport>

²⁵https://github.com/visionsystemsgmbh/programming_examples/tree/master/COM-ports/node-red/flow_tcp_serial_server.json

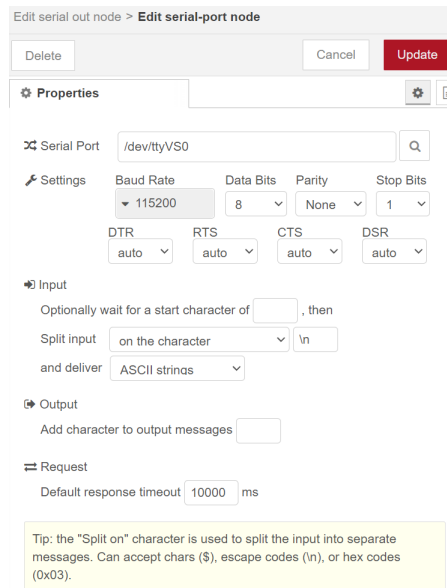


Figure 29: Serial port settings

3.2.7.1.3 Example of a simple serial UDP Server and Client

The example shows a simple UDP client and server flow which transfers binary data between the serial port (Figure 30 on page 27). You'll find the example in our Git repository ²⁶.

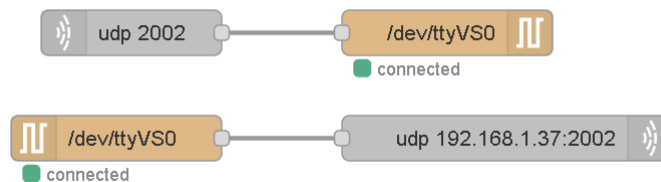


Figure 30: UDP serial server example

Figure 31 on page 28 shows the parameters of the serial in node which transfers the incoming data after a timeout of 10 ms to UDP out node and its configured destination.

²⁶https://github.com/visionsystemsgmbh/programming_examples/tree/master/COM-ports/node-red/flow_udp_serial_client_server.json

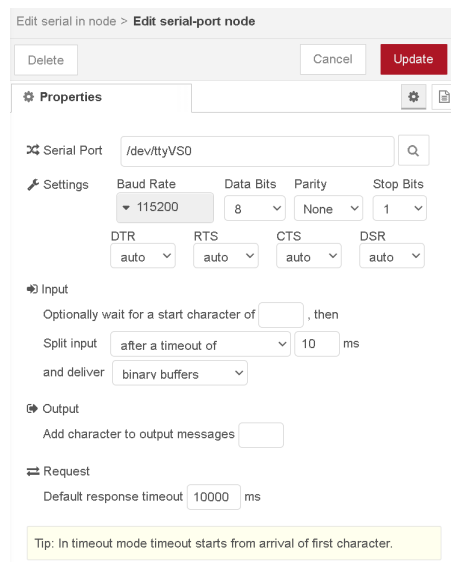


Figure 31: Serial port settings