www.vscom.de

OnRISC IoT Manual

Edition: October 2017

Vision Systems GmbH
Tel: +49 40 528 401 0
Fax: +49 40 528 401 99
Web: www.visionsystems.de
Support: faq.visionsystems.de

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 Installation 11

3.2 Example: Connect WLAN switch to a User LED 12

3.3 Example: GPIO Access via Modbus/TCP 13

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

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.

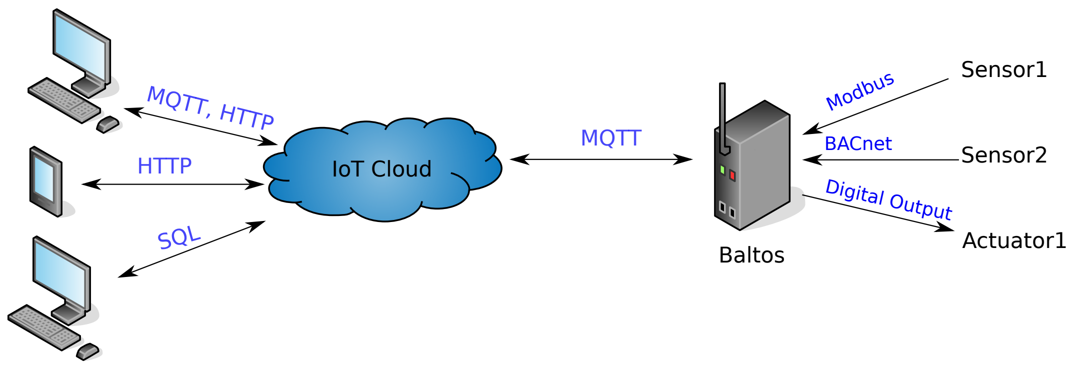


Figure 1: Baltos as IoT Router

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 site[2] 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

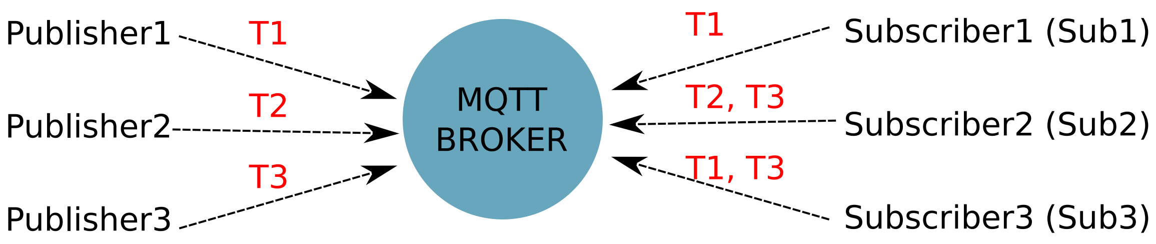


Figure 2: MQTT Scheme

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/visionsystemsgmbh/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 Mosca[3] running on a desktop PC. Install Node.js according to the project's documentation[4]. 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

Node-RED[5] 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 collection[6]) and as Node-RED is written around Node.js[7] you can access any Node.js modules via “function” node or create your own Node-RED nodes[8].

3.1 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 -qO- 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 pages[9]

[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

3.2 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.

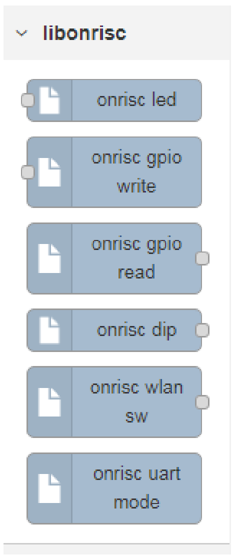


Figure 3: Node-RED: libonrisc Nodes

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

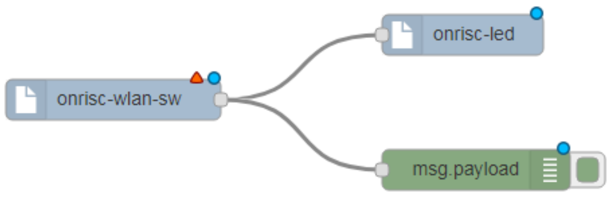


Figure 4: Node-RED: WLAN switch and LED: Initial Scheme

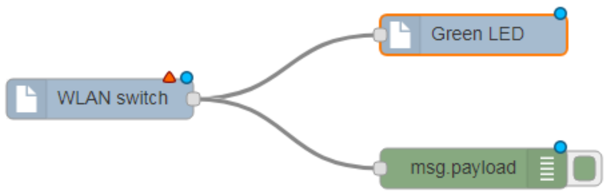


Figure 5: Node-RED: WLAN switch and LED: Configured Scheme

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.

3.3 Example: GPIO Access via Modbus/TCP

In this example you’ll learn Node-RED’s dashboard[10] feature, that allows you to create GUI. It will also show how to use Modbus[11] 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.

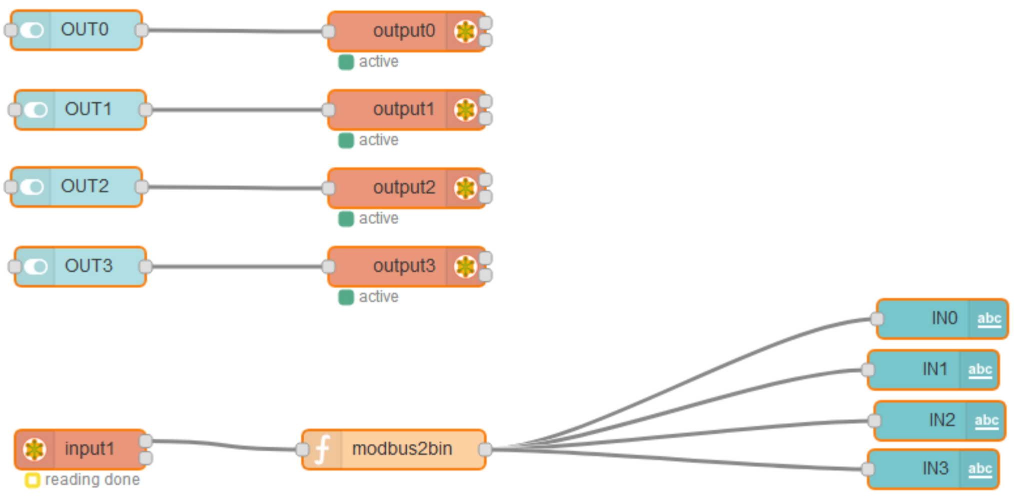
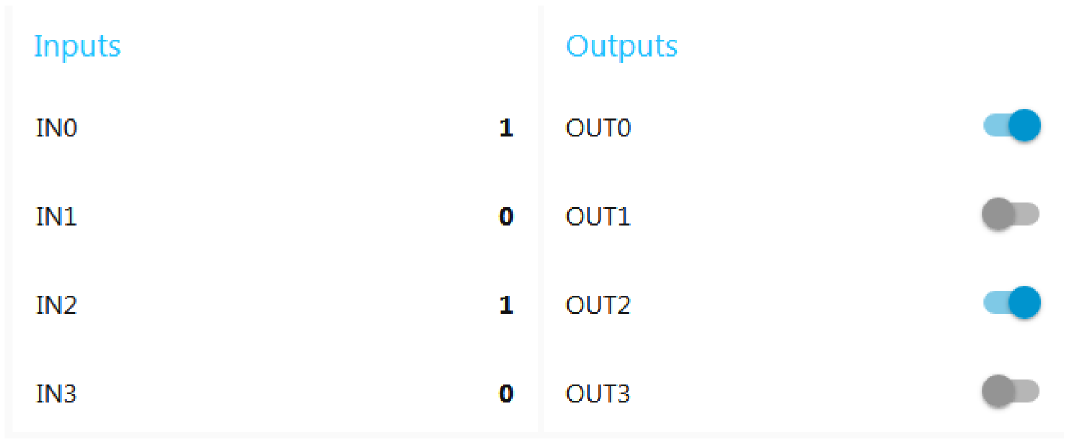
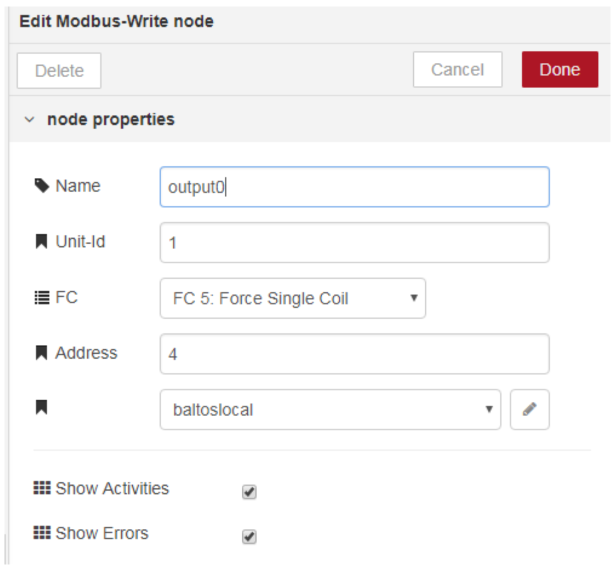


Figure 6: Node-RED: Modbus/GPIO Example Flow Chart



Inputs		Outputs	
IN0	1	OUT0	●
IN1	0	OUT1	○
IN2	1	OUT2	●
IN3	0	OUT3	○

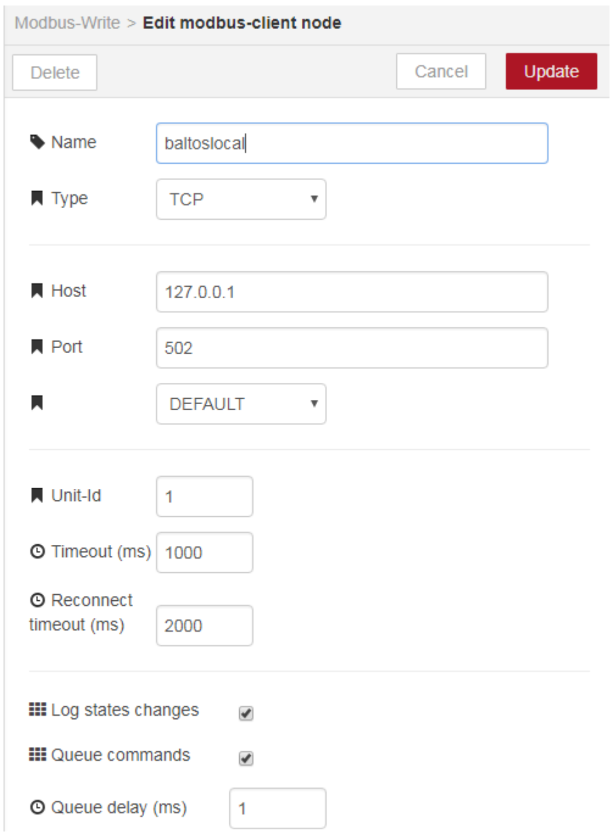
Figure 7: Node-RED Modbus/GPIO Example Dashboard



Edit Modbus-Write node

Name	output0
Unit-Id	1
FC	FC 5: Force Single Coil
Address	4
	baltoslocal
Show Activities	✓
Show Errors	✓

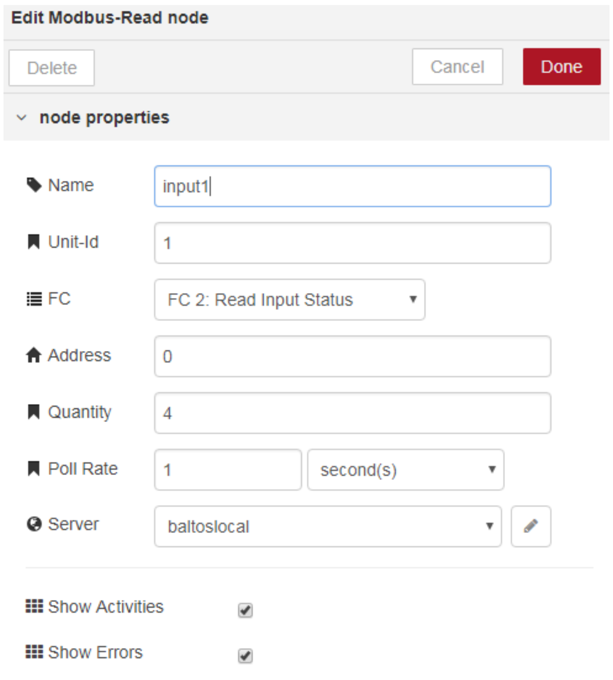
Figure 8: Node-RED: Modbus Write Node


Modbus-Write > **Edit modbus-client node**

Delete | Cancel | Update

Name	baltoslocal
Type	TCP
Host	127.0.0.1
Port	502
	DEFAULT
Unit-Id	1
Timeout (ms)	1000
Reconnect timeout (ms)	2000
Log states changes	✓
Queue commands	✓
Queue delay (ms)	1

Figure 9: Node-RED: Modbus Client Configuration

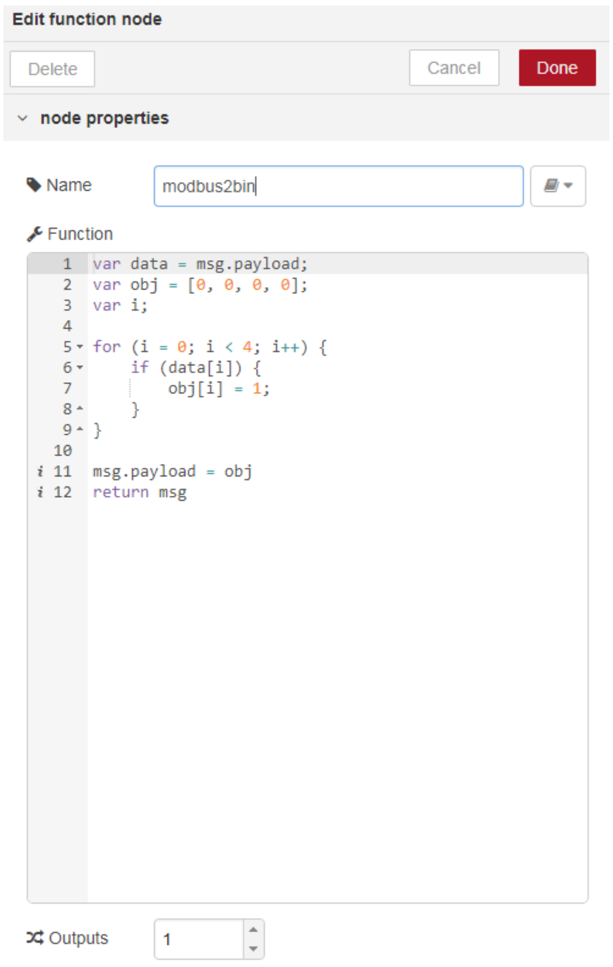

Edit Modbus-Read node

Delete | Cancel | Done

node properties

Name	input1
Unit-Id	1
FC	FC 2: Read Input Status
Address	0
Quantity	4
Poll Rate	1 second(s)
Server	baltoslocal
Show Activities	✓
Show Errors	✓

Figure 10: Node-RED: Modbus Read



Name: modbus2bin

```
var data = msg.payload;
var obj = [0, 0, 0, 0];
var i;

for (i = 0; i < 4; i++) {
    if (data[i]) {
        obj[i] = 1;
    }
}

msg.payload = obj
return msg
```

Outputs: 1

Figure 11: Node-RED: modbus2bin Function