neo.cortec

User Guide for NeoGateway

Doc Status:	**Release**
Doc version:	**1.3**
Date:	**June 2020**

Table of Contents

1 Document revisions

Document version	Details
1.0	Initial release
1.1	Fixed typo in path to home directory in section 10.3
1.2	Unacknowledged messages have been introduced together with some other unimplemented messages, and the document has been through a general review and update.
1.3	Overhaul of the installation steps and adding an extra control section for checking the correctness of the installation. Config files introduced.

2 Introduction

This document describes how to use the NeoGateway to interface the NeoMesh ultra low power wireless mesh network to an IP network.

3 Abbreviations

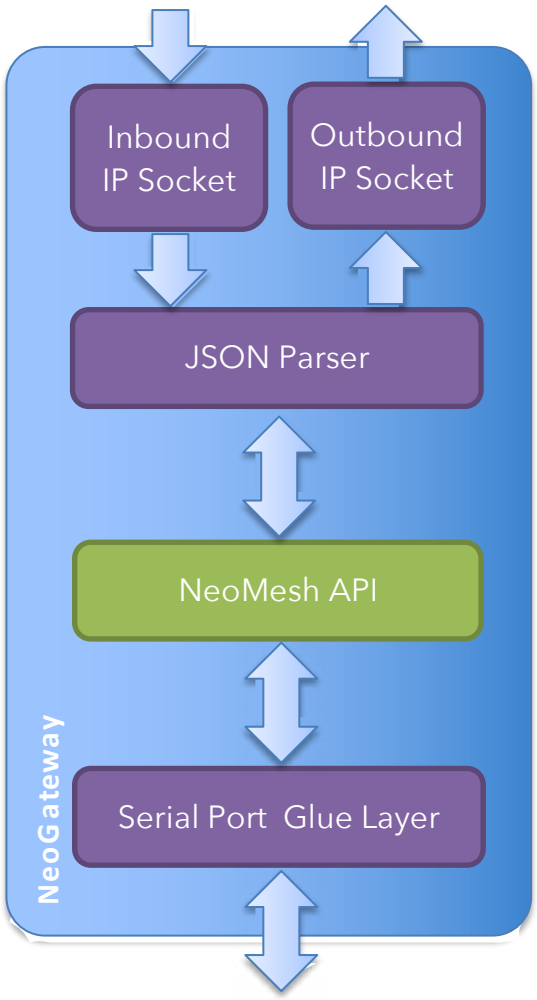
- HW – Hardware
- SW – Software
- UART - Universal Asynchronous Receiver/Transmitter
- RX – Receive
- TX – Transmit
- IP – Internet Protocol
- AAPI – Application API for the NeoCortec NeoMesh modules
- SAPI – System API for the NeoCortec NeoMesh modules
- JSON – JavaScript Object Notation

4 Definitions

(None)

5 NeoGateway Technical overview

The NeoGateway is provided as Open Source software designed to work on the Raspberry Pi platform. The software will however operate without problems on most Linux systems. The NeoGateway is based on the NeoMesh API, and is structured according to the figure below:



The architecture is such that payload data being send from a node inside the NeoMesh network to the gateway node will be delivered on the Outbound IP Socket in JSON format. Similarly, payload data being send from the IP network shall be send as a JSON formatted package to the inbound IP Socket.
This allows for a very versatile gateway design which can easily be adapted to interface with both Cloud services as well as local or remote proprietary servers.

6 Hardware integration

The NeoGateway is designed such that only limited hardware resources are required. The software is designed for the Linux OS, but can run on most embedded platforms with little adjustments required.

The minimum requirements for the hardware platform is to have a UART port and at least one GPIO (for CTS signal), alternatively a full serial port including CTS signalling. Additionally, an Ethernet interface or WLAN interface is required to connect to an IP Network.

For additional information on how to connect the NeoMesh modules to the Gateway hardware, please refer to the NeoCortec Integration Manual for NCxxxx Series Modules.

7 Using the gateway software

7.1 Prerequisites

In the following sections, it is assumed that the NeoGateway software is being used on a Linux platform. Similarly, it is assumed that the Linux installation is setup with an up-to-date GNU Compiler (GCC) and "Make" build automation tool.

7.2 Getting the software

The NeoGateway software can be downloaded from www.neocortec.com The download package contains the full source code. We recommend downloading the package to a local directory dedicated to building the gateway software.

7.3 Building the software

Note: If you intend to use the software on a Raspberry Pi platform, you DO NOT need to build the gateway software. The package you downloaded contains a prebuilt Raspberry Pi version.

To build the NeoGateway software, simply navigate to this folder:
/src/NeoCortecGateway/Release

In this folder, build the software by entering 'make' and hit enter. This will create an executable file named NeoCortecGateway.

7.4 Running the software

The NeoGateway software is executed from the terminal, and has a number of command line parameters which affect the configuration of the gateway. The syntax is as follows:

```
NeoCortecGateway [-c config-file]
NeoCortecGateway [-C key1=value1[,keyN=valueN...]]
```

where "config-file" is a text file which can contain the parameters.

If no parameters are given, the default values will be used. Note that uppercase / lowercase is important.

7.4.1 Configuration parameters

Parameter	Explanation	Possible values
uart	Specifies which serial interface to use	Any valid reference to a serial interface [Default = /dev/ttyAMA0]
speed	Specifies the baud rate on the serial interface. *NOTE*: Only B115200 is supported	B57600 B115200 (Default) B230400
ctsSource	Specifies where the CTS signal from the NCxxxx module is connected	ioctl – uses the CTS pin from a serial interface which supports hardware flow control gpio – uses a GPIO pin when the serial interface does not support hardware flow control
gpioPin	In case of using GPIO to interface the CTS signal, this parameter specifies which Pin is used	Any valid GPIO pin number
delimiter	Specifies what delimiter is being used for terminating messages on the Outbound IP Socket	\n (default)
serializer	Specifies the format used on the Inbound and Outbound IP Sockets *NOTE*: Only JSON supported	JSON (default) xml
recvPort	Specifies the port number of the Inbound IP Socket	Any valid port number [Default = 2000]
sendPort	Specifies the port number of the Outbound IP Socket	Any valid port number [Default = 2001]

ctsTimeoutSecs	Specifies the timeout period for the module to accept the message. Shall be adjusted according to the scheduled data rate of the network	[Default = 5]

7.5 Inbound IP Socket communication

In the following section each command accepted by the NeoGateway on the inbound IP Socket is described.

7.5.1 Send unacknowledged payload data to a node in the network

```
{"objectType":"sendPayload",
 "payloadType":"unacknowledged",
 "nodeId":16,
 "port":0,
 "sequenceNo":564,
 "payload":[1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19]}
```

where:

objectType is specifying the JSON object type.
payloadType specifies that the payload shall be sent "unacknowledged".
nodeId is the Node Id of the destination node as decimal (0x0010 in the example above).
port is the port on the destination node
sequenceNo is an Application sequence number.
payload is the actual payload data to send, with each byte as decimal in an array

7.5.2 Send acknowledged payload data to a node in the network

```
{"objectType":"sendPayload",
 "payloadType":"acknowledged",
 "nodeId":16,
 "port":0,
 "payload":[1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19]}
```

where:

objectType is specifying the JSON object type.
payloadType specifies that the payload shall be sent "acknowledged".
nodeId is the Node Id of the destination node as decimal (0x0010 in the example above).
port is the port on the destination node
payload is the actual payload data to send, with each byte as decimal in an array

7.5.3 Send a Node Info Request to the Gateway node

```
{"objectType":"nodeInfoRequest"}
```

where:

objectType is specifying the JSON object type.

7.5.4 Send Neighbor List Request to the Gateway node

```
{"objectType":"neighborListRequest"}
```

where:

objectType is specifying the JSON object type.

7.5.5 Send Network Command to the Gateway node

```
{"objectType":"networkCommand",
 "nodeId":16,
 "cmd":0,
 "payload":[1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19]}
```

where:

objectType is specifying the JSON object type.
nodeId is the Node Id of the destination node as decimal (0x0010 in the example above).
cmd is the actual network command.
payload is the actual payload data to send, with each byte as decimal in an array

7.5.6 Send WES command to the Gateway node

```
{"objectType":"wesCmd",
 "cmd":0}
```

where:

objectType is specifying the JSON object type.
cmd is the actual WES command. See Integration Manual for NCxxxx Series Modules for details.

7.5.7 Send WES Setup Response to an unconfigured node

```
{"objectType":"wesResponse",
 "nodeId":16,
 "uidHex":"fffffffff",

"appSettings":"0102030405060708090a0b0c0d0e0f101112131415161718
"}
```

where:

objectType is specifying the JSON object type.
nodeId is specifying the Node ID which the node being set up will get.
uidHex is the UID of the node which will be set up
appSettings is the 24 bytes long Generic Application settings used in Normal Mode which will be send to the node being setup.

7.5.8 Send ALT Command to the Gateway node

```
{"objectType":"altCmd",
 "cmd":0}
```

where:

objectType is specifying the JSON object type.
cmd is the actual ALT command.

7.6 Outbound IP Socket communication

When the NeoGateway is running, all AAPI messages transmitted by the NCxxxx module in the gateway are send to the outbound IP socket. Each type is described in the following sections.

7.6.1 Acknowledge for previously send data

```
{"objectType":"ack",
 "gwTimestamp":"2016-11-09T16:00:00.000Z",
 "nodeId":32}
```

where:

objectType is specifying the JSON object type.
gwTimestamp is the time of arrival at the NeoGateway encoded in ISO8601 format.
nodeId is the Node ID of the node sending the Acknowledge.

7.6.2 Non-Acknowledge for previously send data

```
{"objectType":"nack",
 "gwTimestamp":"2016-11-09T16:00:00.000Z",
 "nodeId":32}
```

where:

objectType is specifying the JSON object type.
gwTimestamp is the time of arrival at the NeoGateway encoded in ISO8601 format.
nodeId is the Node ID of the node sending the Non-Acknowledge.

7.6.3 Unacknowledged Payload Data Received

```
{"objectType":"receivedPayload",
 "gwTimestamp":"2016-11-09T16:00:00.000Z",
 "payloadType":"unacknowledged",
 ”sequenceNo”:564,
 "nodeId":32,
 "port":0,
 "packageAgeMicroSecs":750000,
 "packageAgeType":"normal",
 "payload":[1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19]}
```

where:

objectType is specifying the JSON object type.
gwTimestamp is the time of arrival at the NeoGateway for the payload encoded in ISO8601 format.
payloadType is the message type.
sequenceNo is an Application sequence number.
nodeId is the Node ID of the node sending the data
port is the port to which the data was send
packageAgeMicroSecs is the package age of the payload data in micro seconds
packageAgeType is either “normal” or “hapa”.
See “NeoCortec User Guide” document for more details on HAPA vs Normal.
payload is the actual payload data received, with each byte as decimal in an array

7.6.4 Acknowledged Payload Data Received


```
{"objectType":"receivedPayload",
 "gwTimestamp":"2016-11-09T16:00:00.000Z",
 "payloadType":"acknowledged",
 "nodeId":32,
 "port":0,
 "packageAgeMicroSecs":750000,
 "packageAgeType":"normal",
 "payload":[1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19]}
```


where:

objectType is specifying the JSON object type.
gwTimestamp is the time of arrival at the NeoGateway for the payload encoded in ISO8601 format.
payloadType is the message type.
nodeId is the Node ID of the node sending the data
port is the port to which the data was send
packageAgeMicroSecs is the package age of the payload data in micro seconds
packageAgeType is either "normal" or "hapa".
See "NeoCortec User Guide" document for more details on HAPA vs Normal.
payload is the actual payload data received, with each byte as decimal in an array

7.6.5 Node Info Reply

```
{"objectType":"nodeInfoReply",
 "gwTimestamp":"2016-11-09T16:00:00.000Z",
 “nodeId”:50,
 “uidHex”:”fffffffff”,
 “nodeType”:1}
```

where:

objectType is specifying the JSON object type.
gwTimestamp is the time status at the NeoGateway was checked, encoded in ISO8601 format.
nodeId is the Node ID of the node sending the data.
uidHex is the UID of the node sending the data.
nodeType is the type of the node sending the data:
1 = NC2400; 2= NC1000; 3= NC0400

7.6.6 Neighbour list

```
{"objectType":"neighborListReply",
 "gwTimestamp":"2016-11-09T16:00:00.000Z",
 “nodeId”:1, “RSSI”:1
 “nodeId”:2, “RSSI”:2
 “nodeId”:3, “RSSI”:3
 “nodeId”:4, “RSSI”:4
 “nodeId”:5, “RSSI”:5
 “nodeId”:6, “RSSI”:6
 “nodeId”:7, “RSSI”:7
 “nodeId”:8, “RSSI”:8
 “nodeId”:9, “RSSI”:9
 “nodeId”:10, “RSSI”:10}
```

where:

objectType is specifying the JSON object type.
gwTimestamp is the time status at the NeoGateway was checked, encoded in ISO8601 format.
nodeId is the Node ID of a neighbour node.
RSSI is the RSSI value of the neighbour node.

7.6.7 Network Command Reply

```
{"objectType":"networkCommandReply",
 "gwTimestamp":"2016-11-09T16:00:00.000Z",
 “nodeId”:50,
 “command”:0,
 "payload":[1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19]}
```

where:

objectType is specifying the JSON object type.
gwTimestamp is the time status at the NeoGateway was checked, encoded in ISO8601 format.
nodeId is the Node ID of the node sending the data.
cmd is the actual network command.
payload is the actual payload data received, with each byte as decimal in an array

7.6.8 WES Status

```
{"objectType":"wesStatus",
 "gwTimestamp":"2016-11-09T16:00:00.000Z",
 "status":0}
```

where:

objectType is specifying the JSON object type.
gwTimestamp is the time status at the NeoGateway was checked, encoded in ISO8601 format.
status is the status byte in decimal.

7.6.9 WES Setup Request

```
{"objectType":"wesSetupRequest",
 "gwTimestamp":"2016-11-09T16:00:00.000Z",
 "uidHex":"ffffffffff",
 "appFunctionType":0}
```

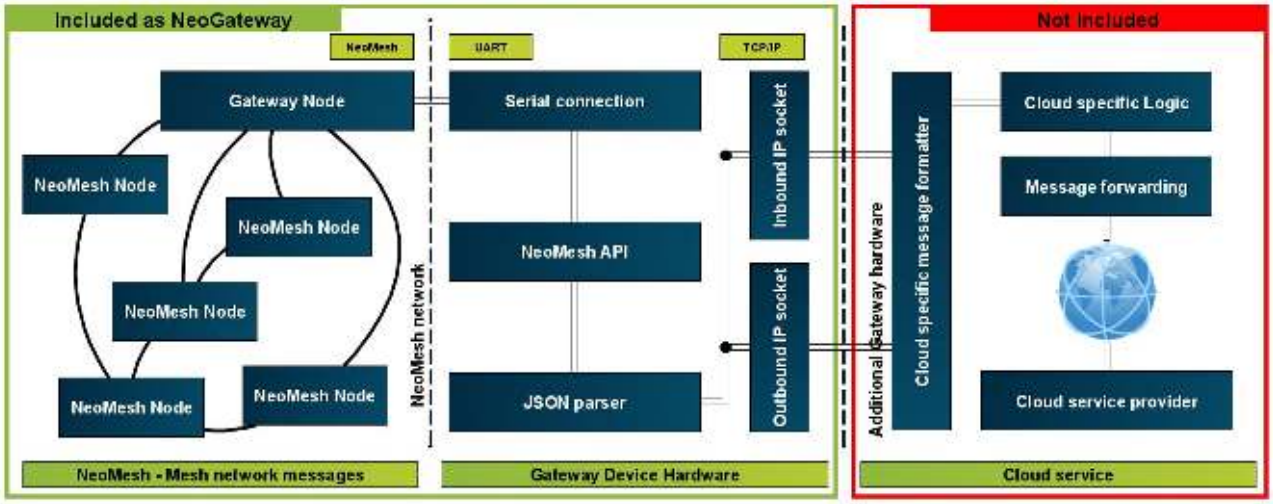
Where:

objectType is specifying the JSON object type.
gwTimestamp is the time status at the NeoGateway was checked, encoded in ISO8601 format.
uidHex is the UID of the node requesting to be configured.
appFunctionType is the type of function the nodes application performs.

8 Interfacing the NeoGateway with the Application Layer

The interface in and out of the NeoMesh network through the NeoGateway is implemented as standard IP Sockets. To interface the gateway with a given Cloud service or proprietary server, an application must be developed which on one side connects to the NeoGateway IP Sockets, and on the other implements the necessary functionality to connect with the Cloud service or proprietary server. Considering the available variety in Cloud services and the differences between used protocols and required logic, the cloud interface application does not come with the NeoGateway. We have two distinct end to end solutions for cloud services. One of them builds upon the presented IP socket based gateway code, while the other one is an ESP based monolithic solution to interface with one specific cloud service. The former is presented at the end of this User guide, for the latter please contact info@neocortec.com.

The interface application utilizing the IP sockets can be implemented in any programming language as Python, Javascript, C/C++ and others, as long as it provides the necessary methods for working with the sockets and the cloud solution. See the section 10.7 Raspberry Pi integration for an example on implementing an interface application for a cloud service.



9 Security

While many layers of security, such as challenge-response and AES encryption, ensures secure transport within a NeoMesh and all the way to the gateway, there are no security features build into the NeoGateway. To ensure the integrity of the system it is advisable to install a firewall configured to block incoming traffic. There are a number of Firewall options available for the Linux operating system, which would be a good solution.

Note: It is recommendable to implement the Interface Application on the same device as the Gateway, i.e. having the Interface Application connecting to localhost / 127.0.0.1. If

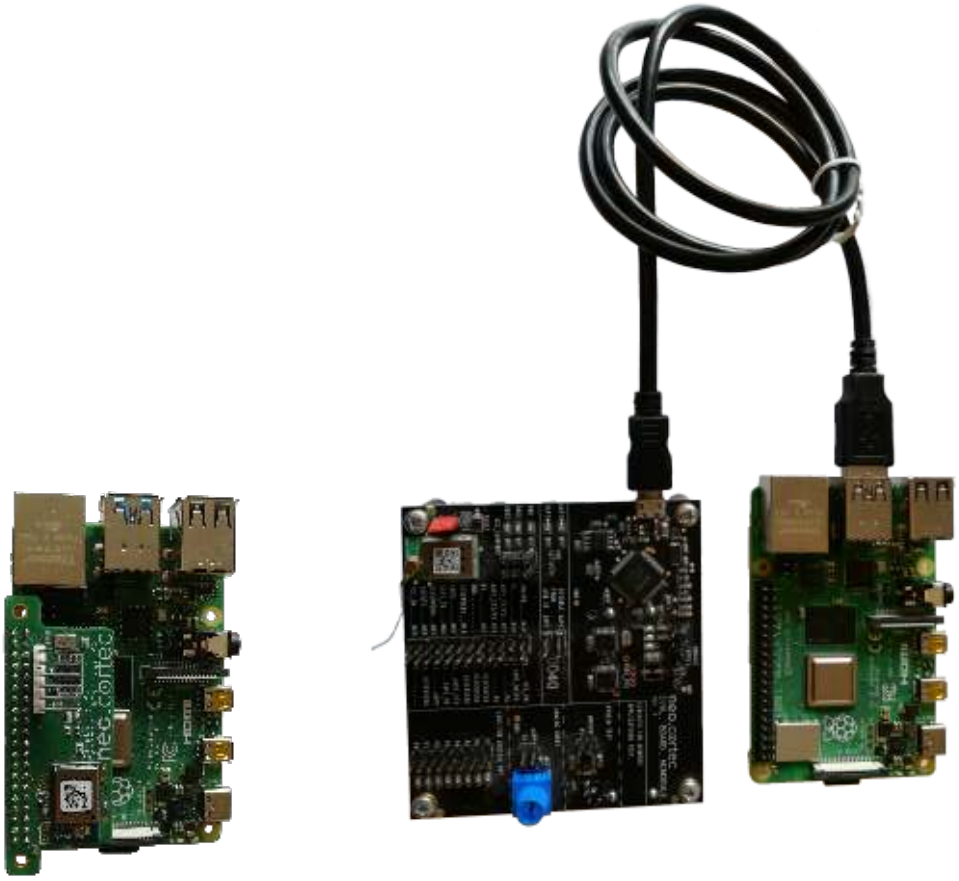
the interface application is hosted on another device, the firewall will have to be configured to only allow incoming traffic from the trusted device.

10 Raspberry Pi integration

For demonstration purposes, a Raspberry Pi Interface board is available along with a pre-compiled executable file for the Raspberry Pi.

Using the Raspberry Pi platform as a commercial gateway implementation can be done, but one has to consider if the specifications of the Raspberry Pi platform will match with those of the target product. For instance, the Raspberry Pi operating temperature range is not exactly well specified.

There are two methods for connecting a gateway node to the Raspberry Pi. We can either choose to connect an evaluation board into one of the available USB ports of the microcomputer or we can use the NeoCortec Gateway Module PiHat (Also referred as the Hardware Interface Board). The setup steps for the two configurations are different, but both uses the same NeoGateway software and both exposes the same IP sockets.



Raspberry pi with evaluation module (right) and PiHat Gateway node (left)

10.1 Hardware interface board.

The interface board is available in smaller quantities from NeoCortec. If larger quantities are required, the design files are available for customers to build their own boards.

The Hardware interface board connects the NCxxxx module to the GPIO Header of the Raspberry Pi board. This allows for the AAPI of the NCxxxx module to be connected to UART0, and the SAPI of the NCxxxx module to be connected through a Serial<->I2C converter to a virtual UART[1].

10.2 Setup steps for the different solutions

PiHat Hardware interface Board

- Download the NeoGateway software pack
- Unpack the contents of the zip file
- Disable the Bluetooth or serial console
- Connect the hardware interface board
- Install the NeoGateway software
- Run the NeoGateway software
- Test the NeoGateway software

Evaluation Board through USB

- Download the NeoGateway software pack
- Unpack the contents of the zip file
- Connect the evaluation board
- Install the NeoGateway software
- Run the NeoGateway software
- Test the NeoGateway software

10.3 Preparation of Raspberry Pi for the Hardware interface board

Note: If you use an evaluation module connected through USB, you DO NOT need this step.

Install the latest version of the Raspian distribution which can be downloaded from https://www.raspberrypi.org/downloads/raspbian/ then follow the instructions provided on the Raspberry Pi website.

When attached to the extension header, the hardware interface board will use UART0 to connect to the application API and the System API will be accessed by a virtual UART through the pins normally configured as an I2C bus. By default, UART0 is assigned to another purpose. Depending on the version of the Raspberry Pi, it can be either Serial Console or on-board Bluetooth chipset interface. Raspberry Pi 3 model B and Raspberry Pi 4 are using the UART0 for Bluetooth chipset, and earlier models are using it for serial

[1] The Raspberry Pi only has 1 physical UART

console. Please refer to the Raspberry Pi website for specific information on the exact model being used. In order to provide full functionality for the Hardware interface board, the other services using the UART connections needs to be disabled.

If your **Raspberry Pi is generation 1 or 2**, then

Disable the Serial console, as described in section 10.3.1.

If your **Raspberry pi is generation 3 or 4**, then

Disable the Bluetooth device as described in section 10.3.2.

10.3.1 Disabling Serial Console

Note: Only perform this step your Raspberry Pi is generation 1 or 2.

To disable the Serial Console, perform the following steps:

First, edit the file /boot/cmdline.txt[2], and remove any reference to the serial port (ttyAMA0). An example could be:

Original: `dwc_otg.lpm_enable=0 console=ttyAMA0,115200 kgdboc=ttyAMA0,115200 console=tty1 root=/dev/mmcblk0p2 rootfstype=ext4 elevator=deadline rootwait`

New: `dwc_otg.lpm_enable=0 console=tty1 root=/dev/mmcblk0p2 rootfstype=ext4 elevator=deadline rootwait`

Second, edit the file /etc/inittab, and search for the following line near the end of the file:

```
T0:23:respawn:/sbin/getty -L ttyAMA0 115200 vt100
```

Delete the line, or comments it out by putting a # in the beginning.

Reboot the Raspberry Pi, and the serial port is now available for NeoGateway to use.

10.3.2 Disabling the Bluetooth serial interface

Note: Only perform this step your Raspberry Pi is generation 3 or 4.

To disable the disable Bluetooth and restore UART0 in the GPIO header, do the following:
Edit the file /boot/config.txt, and add this to the end of the file:

```
dtoverlay=pi3-disable-bt
```

To stop the Bluetooth module from using the UART, enter the following command in the terminal:

$ sudo systemctl disable hciuart

Reboot the Raspberry Pi, and the serial port is now available for NeoGateway to use.

[2] You need to be root to be able to write the file. Use for instance `sudo nano /boot/cmdline.txt`

10.4 Running the installer script for the gateway software

Note: This step is necessary for both the Hardware interface board and the evaluation board setup option. To use the precompiled executable for Raspberry Pi, you do not have to rebuild the software!3

Before you install the code, make sure that the configuration settings correspond with the way you want to use the gateway. All the settings are collected in the "gw.cfg" file. As you have seen before, you have two choices when setting up the software. You use either a Hardware interface board or an evaluation board. The differences between the config file for the two setups, you can see below:
(The settings which need to be configured differently in the two cases are highlighted with grey)

PiHat Hardware interface Board	**Evaluation Board through USB**
uart=/dev/ttyAMA0	uart=/dev/ttyUSB0
speed=B115200	speed=B115200
recvPort=2000	recvPort=2000
sendPort=2001	sendPort=2001
serializer=json	serializer=json
ctsTimeoutSecs=5	ctsTimeoutSecs=5
ctsSource=gpio	ctsSource=ioctl
gpioPin=4	gpioPin=4
delimiter=\r\n	delimiter=\r\n

After you have configured your "gw.cfg" file correctly, kindly proceed to the installation of the NeoGateway software. At this step, you should find an "install.sh" file in the "/Raspberry.Pi" folder of the downloaded NeoGateway package. Run this file with administrator privileges. Please make sure that the permissions for the install script are set correctly. For setting the permissions, navigate to the folder containing the file using a command line, then run the following command:

$ sudo chmod 777 ./install.sh

For running the install script, stay in the same folder as the install file is located, then run the command:

$ ./install.sh

[3] In more concrete terms if you wish to use the code on a Raspberry Pi, skip over section 7.3.

If your install was successful, you do not have any additional steps to complete and you can proceed to section 10.7 to test your installation.

10.5 Installing the NeoGateway software manually

Note: You only need to perform this step, if the included automatic installer script did not work correctly on your setup, or you want to double-check the correctness of the install.

As part of the NeoGateway zip file, there is a precompiled executable for the Raspberry Pi platform. Copy this to a desired location (e.g. /home/pi/NeoGateway/). To use this executable, you do not have to rebuild the software![4]

The gateway software comes with a configuration file included. This configuration file needs to be copied to the same location, as the executable. Before copying the file, modify the settings in it, according to the way you intend to set up your gateway. For the setup option, see the examples under section 10.4.

We want to run the NeoGateway as a service[5] to make sure it runs independently, and does not require a user to login to the Raspberry Pi, and further. Running the gateway as a service will automatically make it available after power up. To achieve this, copy the included NeoGW.service file to /lib/systemd/system/.

You should make sure, that the "NeoGW.service" file fits to the folder choices and folder structure you have on your platform. For instance, the path to the NeoCortecGateway file, or path to the actual serial device needs to be correct.

Please be aware, in order for the Gateway code to run, it needs to have run permissions on the machine you are using it on. You can grant run permissions by executing the command from the command line, assuming you did not change the recommended install directory:

$ sudo chmod 777 /lib/systemd/NeoCortecGateway

When the file has been adapted to use for the local system, execute the following command from the command line:

$ sudo systemctl enable NeoGW.service

Reboot the Raspberry Pi and check that the service is running by using NetCat:

$ nc localhost 2000

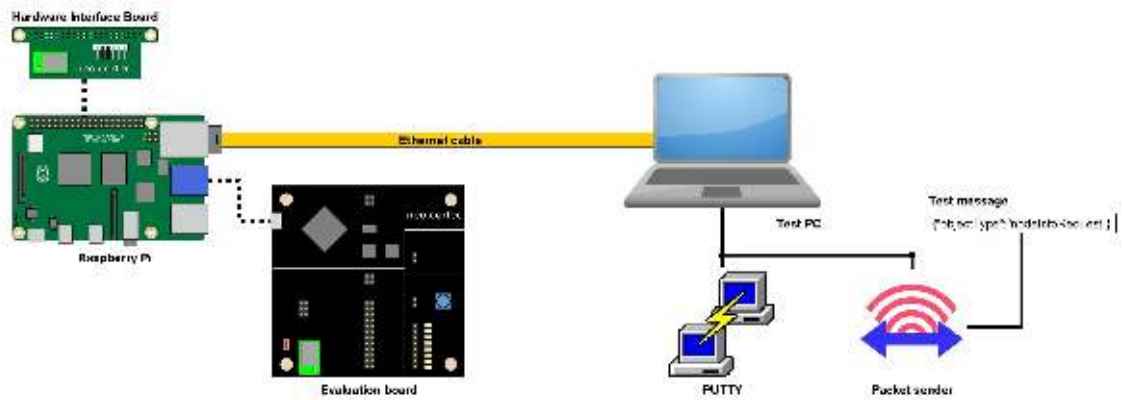
[4] In more concrete terms if you wish to use the code on a Raspberry Pi, skip over section 7.3.
[5] A Linux service is an application (or set of applications) that runs in the background waiting to be used, or carrying out essential tasks.

This connects NetCat to the Outbound IP Socket of the NeoGateway, and directs the output of the gateway to the terminal. If the NeoGateway service is not running, NetCat will exit immediately. If the service is running, NetCat will not exit.

Of course this only checks, if the Outbound socket is open. This test does not tell you anything about the connection to the NeoCortec module through the UART connection. For making sure all parts of the setup are working as intended, please proceed to section 10.6 for a full test procedure.

10.6 Testing connection to NeoCortec module

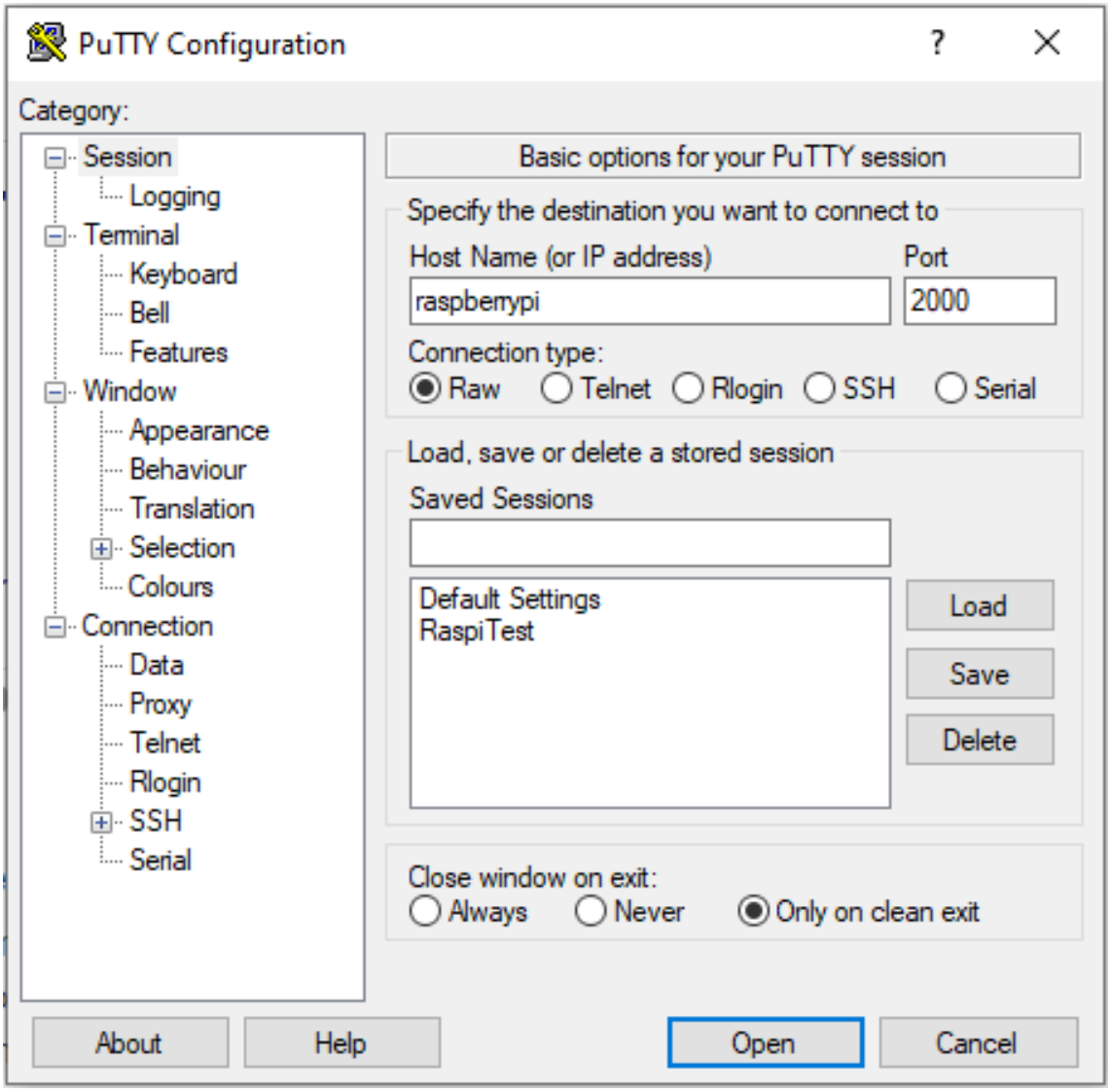
In order to verify the Gateway software is installed correctly and the UART connection to the module is working, we need to send a node info request to the send socket (2001) and receive the correct reply from the Receive socket (2000). This test procedure tests the whole connection chain from socket to module and back. It does not matter whether you have used an evaluation board or a hardware extension board, as long as the setup steps for the correct configuration have been performed. For the test you have to have EITHER the Hardware interface board OR the evaluation board. The physical setup is as follows:



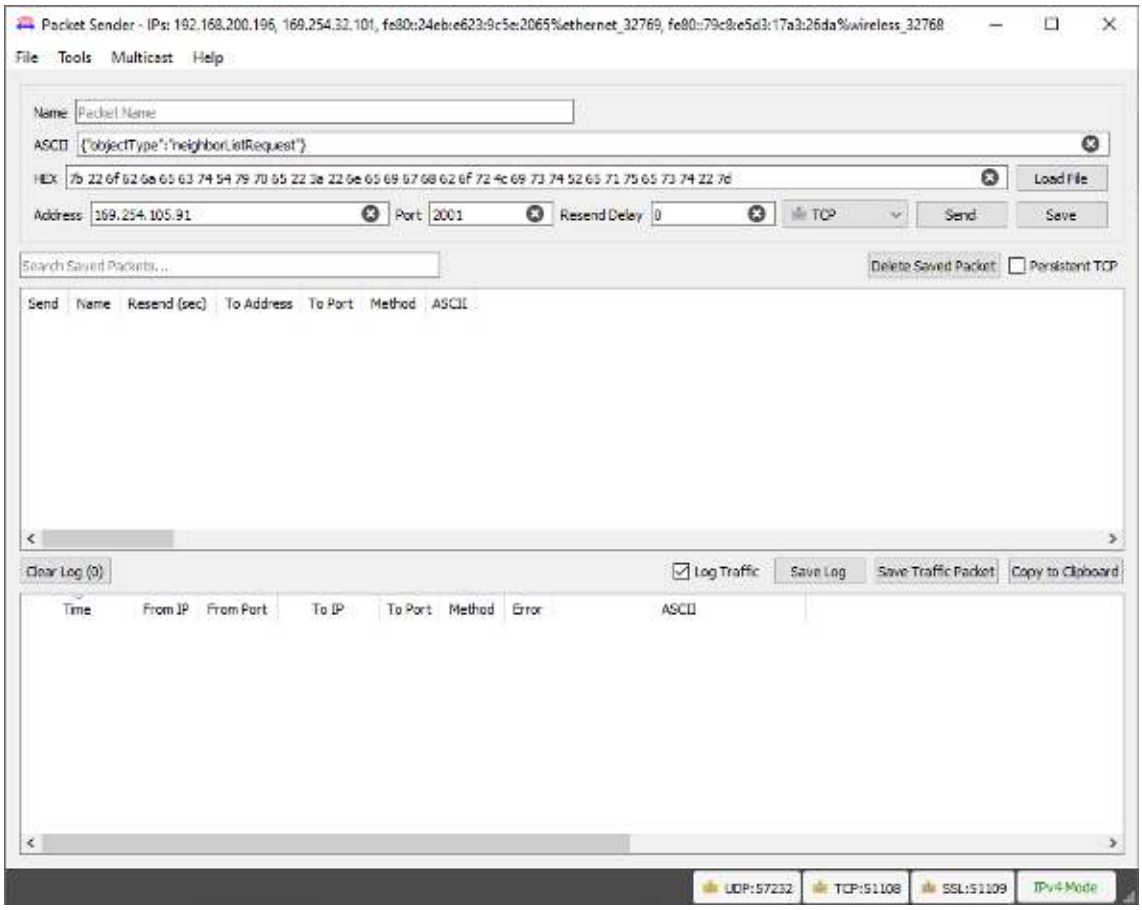
You will need a second computer connected the Pi through an Ethernet cable. On the second computer you need Putty and Packet Sender installed. Make sure the NeoCortec gateway software is running on the Pi.[6] The steps needed in order to verify the correct operation of the gateway setup are as follows:

[6] run the command "nc localhost 2000" from the RasPi's command line. If the command does not return immediately, then the socket is open.

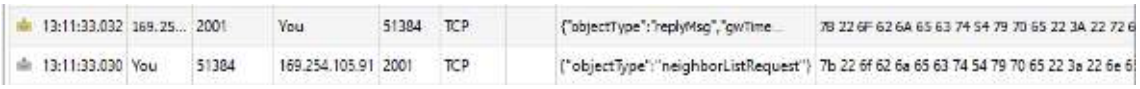
1. Connect your RaspberryPi board with the NeoCortec module to the second computer through an Ethernet cable and make sure the RasPi gets power.
2. Verify that the Gateway software is running on the Pi.
3. Initiate a connection through PUTTY. Your setup should look like the image below. The default host name for your Raspberry Pi is “raspberypi” and the port to which you need to connect is “port 2000”. Make sure you set the connection type to “raw”, then hit open.



4. After the connection is opened, start an instance of PacketSender and copy the following message into the ASCII textbox. Set the address to the IP of your Raspberry Pi and the port to "2001".



5. After you have hit send, you should see the request and the reply showing up in the bottom part of the packet sender window.



6. If you have received a reply similar to the bottom line on the picture above, your setup works. If your reply looks like the image below, you do not have connection to the NeoCortec node, and you should check the gateway setup again.

13:16:15.556	169.25...	2001	You	51405	TCP			
13:16:15.056	169.25...	2001	You	51405	TCP			
13:16:15.054	You	51405	169.254.105.91	2001	TCP		{"objectType":"neighborListRequest"}	7b 22 6f 62 6a 65 63 74 54 79 70 65 22 3a 22 6e 6

10.7 Interface Application DEMO

Note: This part of the user guide is only instructional, and does not intend to present a full, end to end solution. The goal is to show how to interface with the NeoGateway software.

The final step is to write an application which connects the NeoGateway to an application layer. In this example, we will connect the NeoGateway to a cloud service. We will be using Node-RED[7] as the programming language which can easily be installed on a Raspberry Pi.

The interface application will connect the NeoMesh with Microsoft Power BI (https://powerbi.microsoft.com/) . Strictly speaking Power BI is not a cloud platform, but a visualisation & analytics tool for various types of data, but it is easy to get started with, free to use and exemplifies well how to get data from the NeoGateway to a 3'rd party platform.

The implementation assumes that the NeoMesh nodes are configured to send temperature and humidity data. For the required configuration steps, please refer to the Quickstart guide or to the user manual.

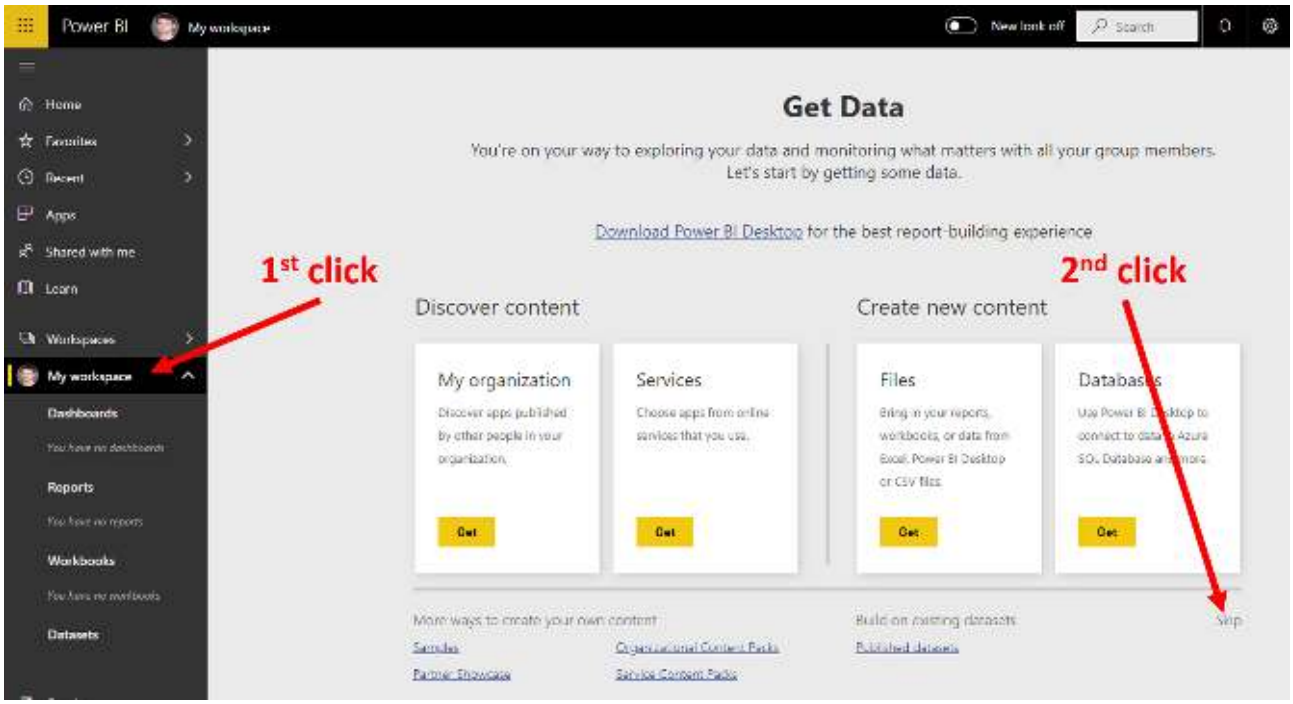
10.7.1 Setting up Power BI

If not already done, create an account at the Power BI website (https://powerbi.microsoft.com/) and sign in to the account. There are two steps which needs to be done; 1) Create a Streaming Dataset, 2) Create a Dashboard to view the data.

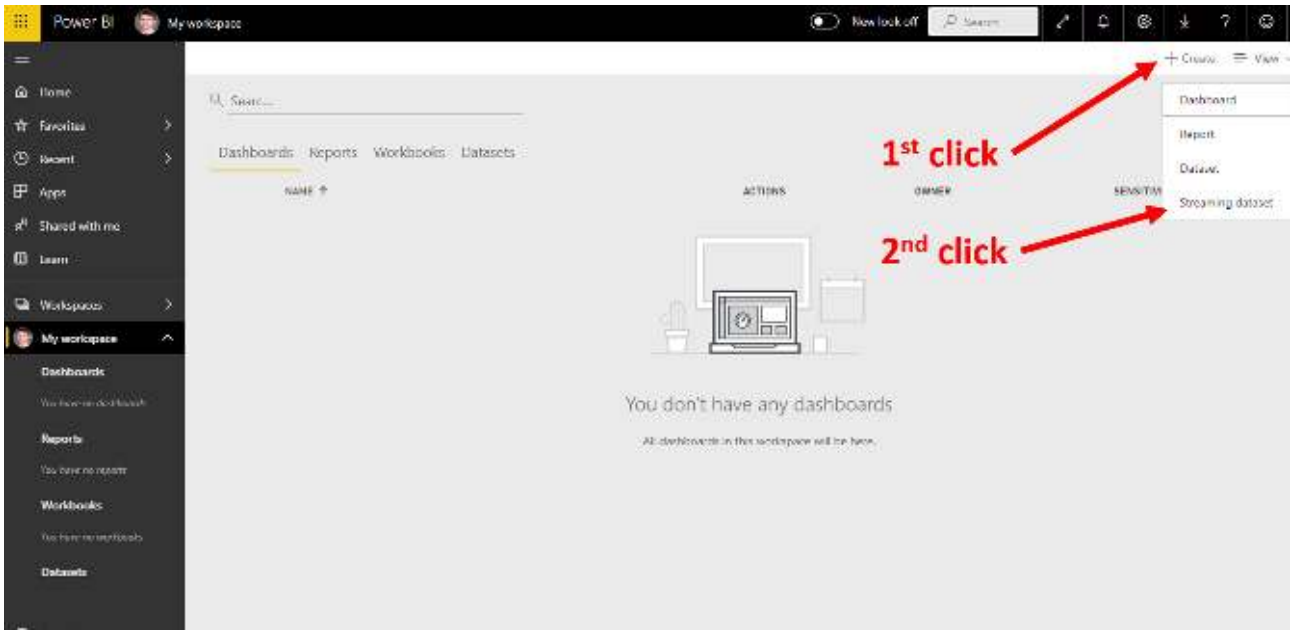
[7] https://nodered.org

10.7.1.1 Create streaming dataset

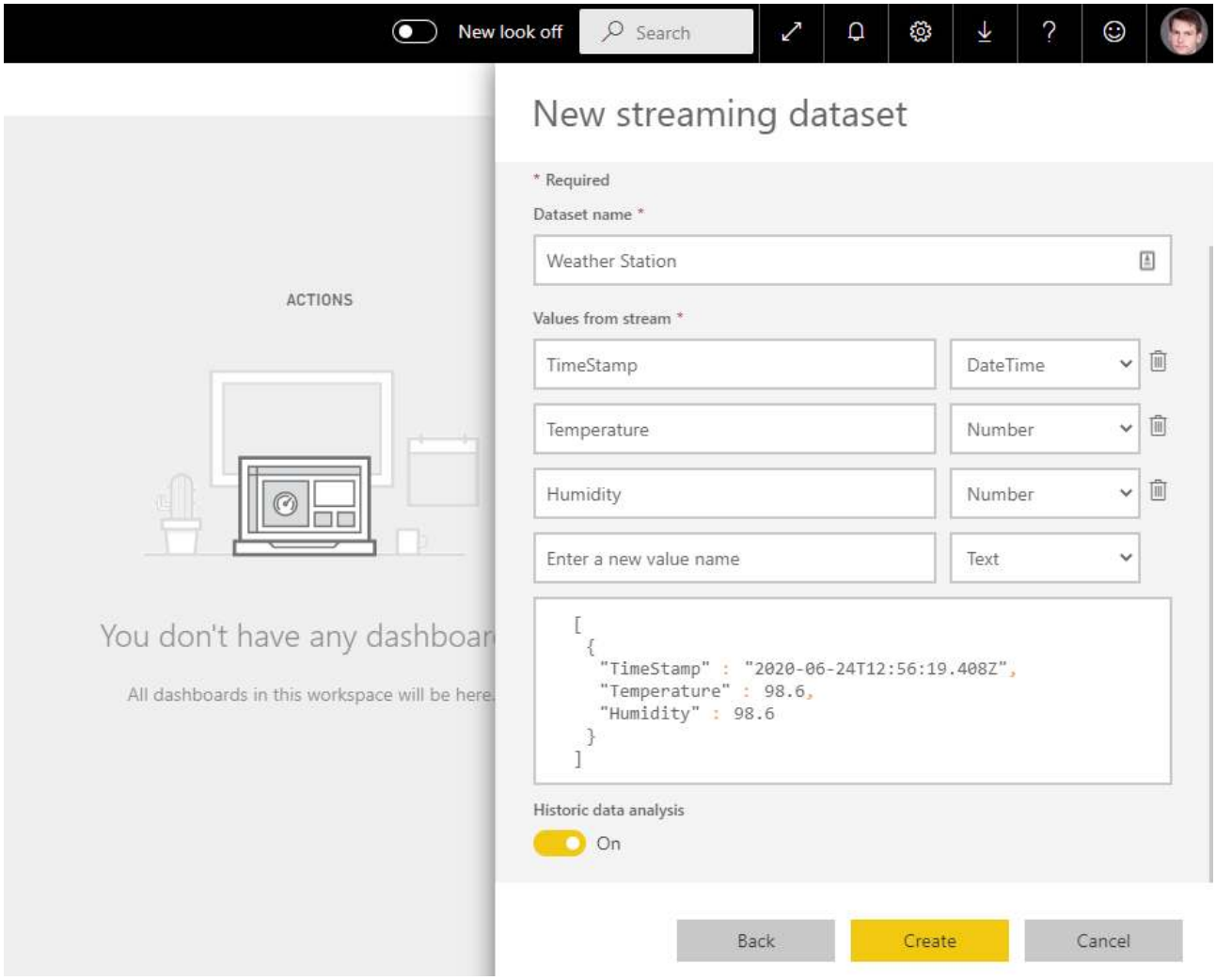
Click on the “My workspace” button then select to “skip” the prompted options.



Expand the menu under the little “+Create” button, then click on “Streaming datasets” to get to the section where the dataset can be setup.



In the pop-up menu on the right, select API as type. Give a name to the dataset, and add these values:



Click create to get the API endpoint push URL. Save the URL for later use.

10.7.1.2 Create Power BI Dashboard

Expand menu on the right side under the "+ create" once again, and click the "Dashboard" option to add a new dashboard. Give it a name. Click "Add tile" to add a chart for the temperature data. Select "CUSTOM STREAMING DATA" and click Next. Now select the dataset created in the previous section and click next. Select "Line chart" as visualization type.

Now click "Add value" under "Axis" and select Timestamp. Next click "Add value" under "Values" and select Temperature. Click Next, and then Apply. This will place a chart with the temperature data on the Dashboard.

Repeat the steps above, selecting Humidity in the final step to add a chart for the Humidity data. See the NeoCortec User Guide Document for details on how to configure the NeoMesh nodes for this.

10.7.2 Node-Red installation

This section will walk through the Interface Application in Node-Red, step by step. First you need to install NodeRed on your Linux platform, as it might not come preinstalled. If you happen to have NodeRed installed, feel free to skip this step. For a detailed installation-and-run guide, read the official getting started page

10.7.3 The code to interface with Power BI.

In order to get our data to the cloud, we need to replicate the program below.

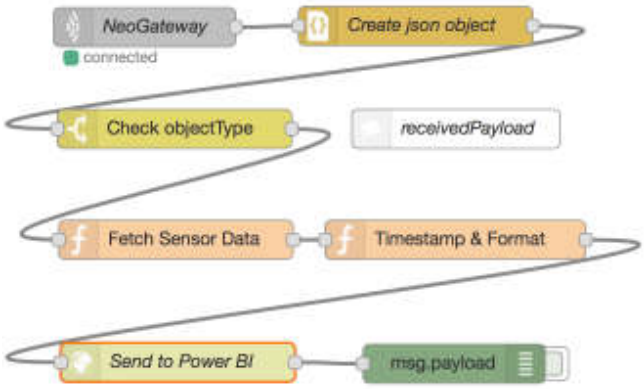


Figure 1 - Node-Red Flow

Take the nodes from the left, then drag and drop them to your "workbench". For more detailed instructions, visit the official "create your first flow" page.

In the following sections we will move from node to node from top left to bottom right.

10.7.3.1 NeoGateway

The first box is a TCP Input Node. It connects to the Outbound socket of the NeoGateway. It is configured like this:

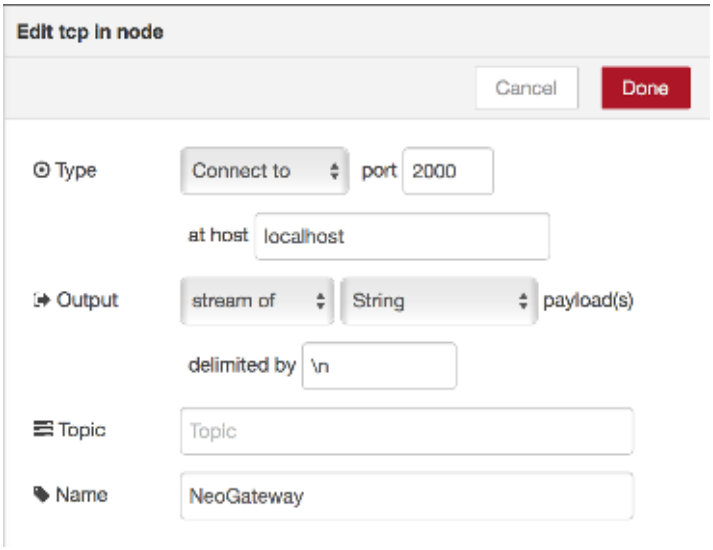


Figure 2 - TCP Input Node

10.7.3.2 Create JSON object

This node converts the received data from the NeoGateway into a true JSON object in Node-Red. No configuration is required.

10.7.3.3 Check objectType

This node looks at the “objectType” field of the JSON message, and filters out any other object types than “receviedPayload”. It is configured like this:

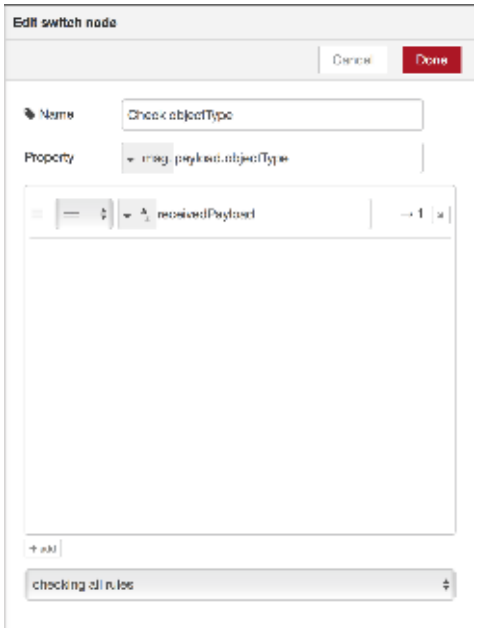


Figure 3 - Filter for objectType

Note: In a real implementation of an Interface Application, there shall be handlers for all object types, but in this case it is simplified to only handle receivedPayload. Make sure you do not mistype the filter criteria.

10.7.3.4 Fetch Sensor Data

This node is a Function Node, which contains Javascript Code which converts the raw JSON data to temperature and humidity data. It is configured like this:

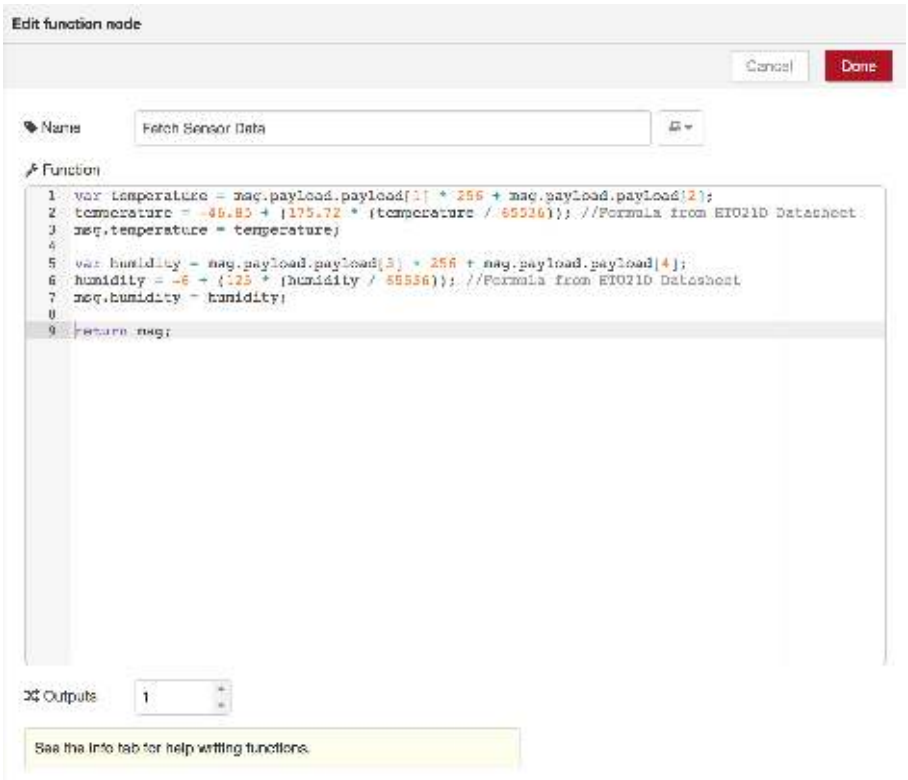


Figure 4 - Javascript code to convert raw data to sensor values

More details on how the payload data is formatted can be found in the NeoCortec User Guide Document.

10.7.3.5 Timestamp & Format

This is a Function Node, which contains Javascript Code that timestamps the data and converts into a format expected by Power BI. It is configured like this:

Figure 5 - Javascript code to timestamp & format data

10.7.3.6 Send to Power BI

This is a HTTP Request node which in this case connects to the API endpoint of Power BI streaming dataset created when setting up Power BI. It is configured like this:

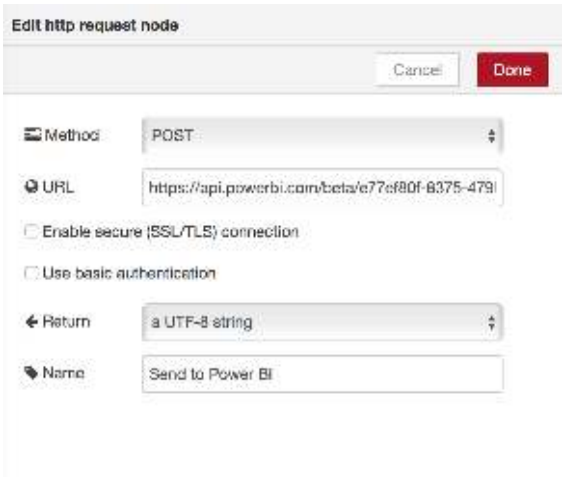


Figure 6 - HTTP POST to Power BI

The actual URL shall be the one created when setting up the Power BI streaming dataset.