neo.cortec

User Guide for NeoGateway

Doc Status:	**Release**
Doc version:	**1.1**
Date:	**Feb 2017**

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

2 Introduction

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

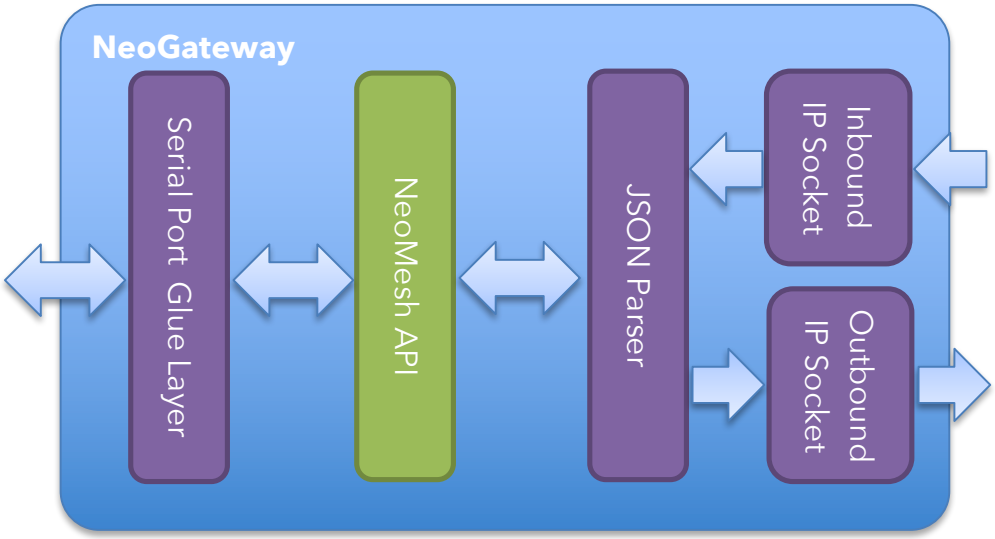
3 Abbreviations

- 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

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 a 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

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

In this folder, build the software by entering 'make' and 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 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 how the payload shall be send. Can be "acknowledged" or "unacknowledged"[1].
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

7.5.2 Send WES command 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.

7.5.3 Send WES settings to a unconfigured node

```
{"objectType":"wesResponse",
 "nodeId":16,
 "uidHex":"ffffffffff",
 "app0":"0102030405060708090a0b0c0d0e0f",
 "app1":"0102030405060708090a0b0c0d0e0f",
 }
```

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
app0 is the Generic Application settings used in Normal Mode which will be send to the node being setup.
app1 is the Generic Application settings used in Alternate Mode which will be send to the node being setup.

[1] Currently only "acknowledged" is supported

7.5.4 Send Network Command to the Gateway node

```
{"objectType":"sendNetCmd",
 "cmd":[0]}
```

where:

objectType is specifying the json object type.
cmd is the actual Network Command as an array, as the Network Command in some cases takes arguments. See Integration Manual for NCxxxx Series Modules for details.

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 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 – currently on acknowledged transmission are supported.
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

7.6.2 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.

7.6.3 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.

7.6.4 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.

7.6.5 WES Setup Request

```
{"objectType":"wesRequest",
 "gwTimestamp":"2016-11-09T16:00:00.000Z",
 "uidHex":"ffffffffff",
 "appFunction":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.
appFunction is the appFunction value property.

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 has to 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.
The interface application 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 IP Sockets. See the section 10 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, but not limited to above 0 degrees C.

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 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[2].

10.2 Preparation of Raspberry Pi

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

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 is using the UART for Bluetooth chipset, and earlier models are using it for serial console. Please refer to the Raspberry Pi website for specific information on the exact model being used.

10.2.1 Disabling Serial Console

To disable the Serial Console, perform the following steps:

First, edit the file /boot/cmdline.txt[3], and remove any reference to the serial port (ttyAMA0). An example could be:

[2] The Raspberry Pi only has 1 physical UART
[3] You need to be root to be able to write the file. Use for instance `sudo nano /boot/cmdline.txt`

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 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.

10.2.2 Disabling the Bluetooth serial interface

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.

10.3 Installing the NeoGateway software

As part of the NeoGateway zip file, there is a precompiled executable for the Raspberry Pi platform. Copy this to a desired location (eg. /home/pi/NeoGateway/).
We want to run the NeoGateway as a service[4] to make sure it runs independently, and does not require a user to login to the Raspberry Pi, and further, it will automatically be available after power up. To achieve this, copy the included NeoGW.service file to /lib/systemd/system/.

[4] A Linux service is an application (or set of applications) that runs in the background waiting to be used, or carrying out essential tasks.

The file shall be updated such that it fits to the exact system which it is being used on. For instance, the path to the NeoCortecGateway file, or path to the actual serial device.

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

```
$ sudo systemctl enable NeoGW.service
```

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

```
$ nc localhost 2000
```

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.

This concludes the installation of the NeoGateway on the Raspberry Pi. Next steps will be to implement an Interface Application that can send data from sensors or the like, to for instance a Cloud Platform.

10.4 Interface Application DEMO

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[5] as the programming language which is part of the Raspian distribution.
The interface application will connect the NeoMesh with Microsoft Power BI (https://powerbi.microsoft.com/) . Although Power BI strictly speaking may not be 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. See the

10.4.1 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

[5] https://nodered.org

needs to be done; 1) Create a Streaming Dataset, 2) Create a Dashboard to view the data.

10.4.1.1 Create streaming dataset

Expand the menu on the left side. Click on “Streaming datasets” to get to the section where the dataset can be setup. Then click on Add streaming set, and select API as type. Give a name to the dataset, and add these values:

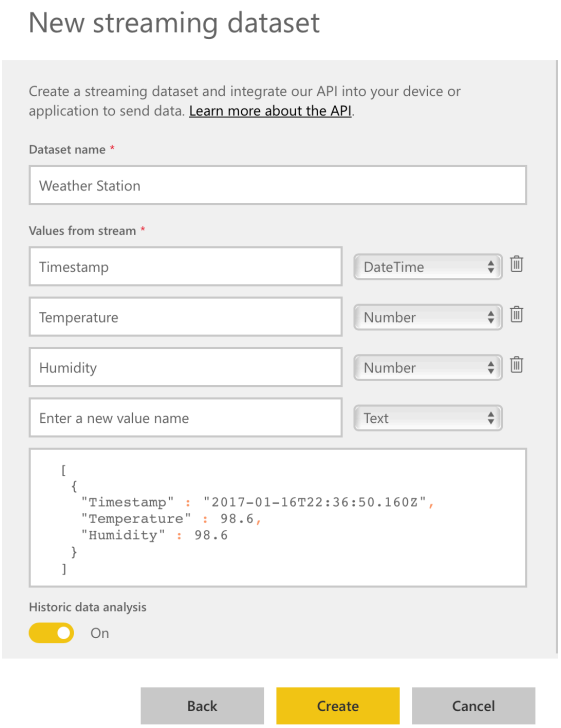


Figure 1 - Streaming dataset

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

10.4.1.2 Create Power BI Dashboard

Expand menu on the left side, and click the “+” sign next to Dashboards 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.

10.4.2 Node-Red Code

This section will walk through the Interface Application in Node-Red, step by step.

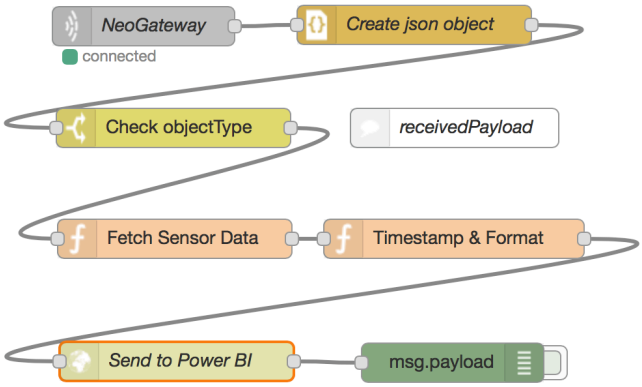


Figure 2 - Node-Red Flow

10.4.2.1 NeoGateway

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

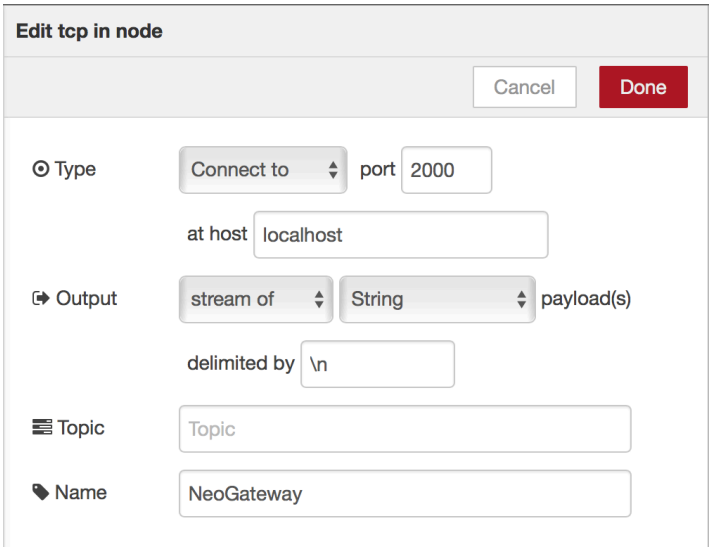


Figure 3 - TCP Input Node

10.4.2.2 Create json Object

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

10.4.2.3 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:

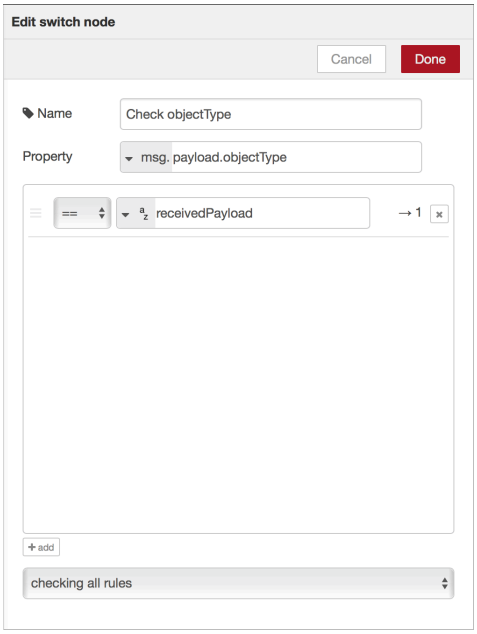


Figure 4 - Filter for objectType

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.

10.4.2.4 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:

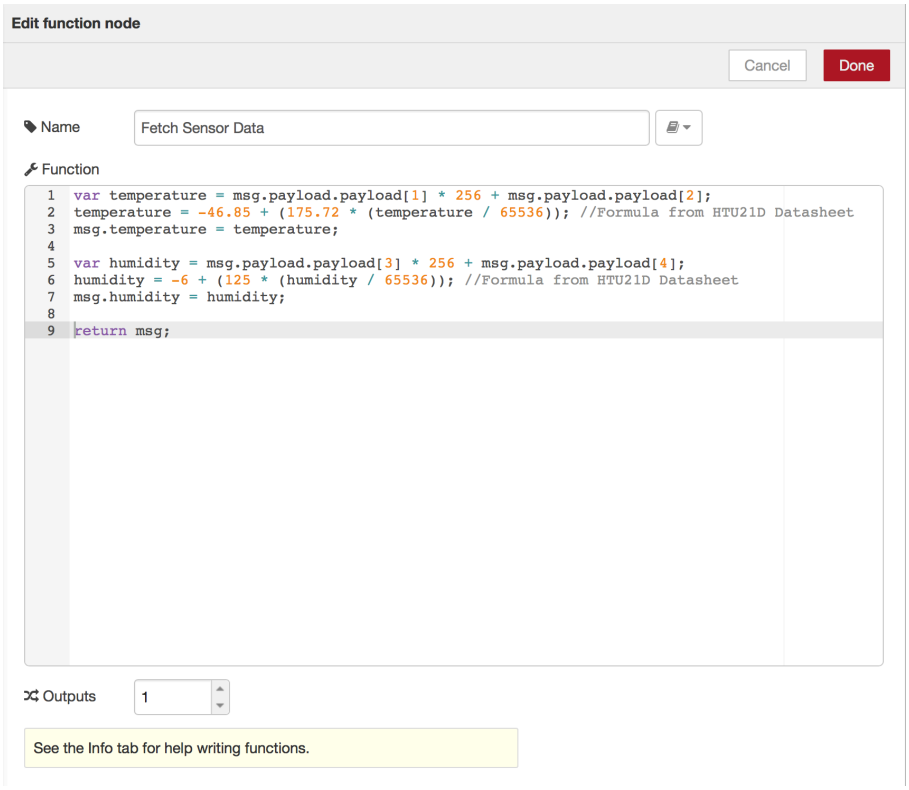


Figure 5 - Javascript code to convert raw data to sensor values

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

10.4.2.5 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:

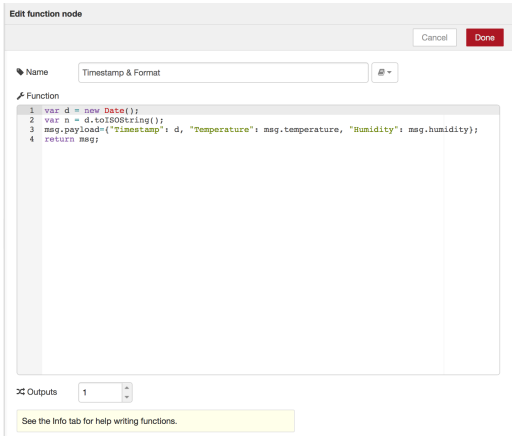


Figure 6 - Javascript code to timestamp & format data

10.4.2.6 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:

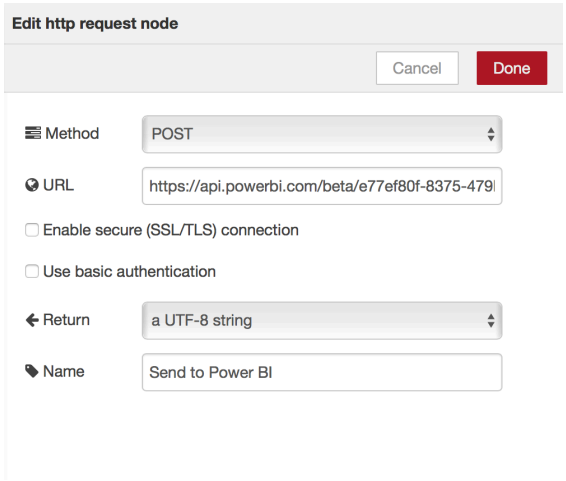


Figure 7 - HTTP POST to Power BI

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