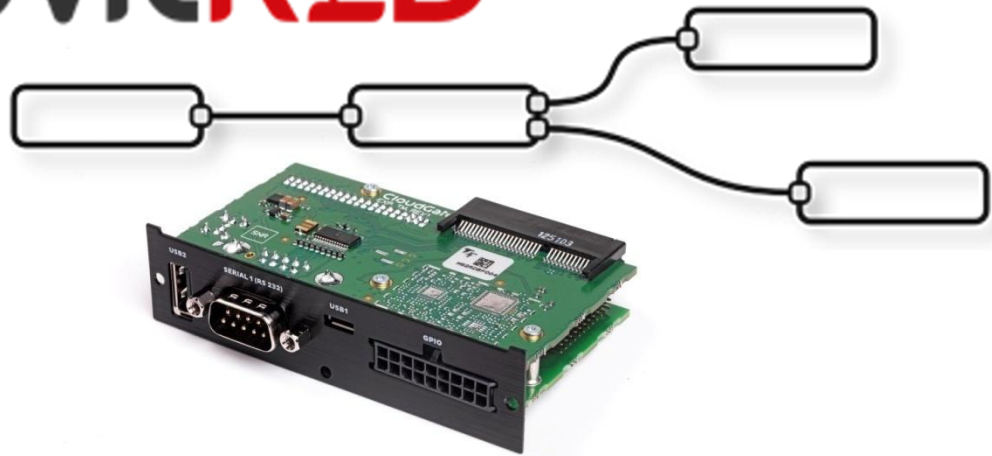




# LuvitRED



## Telematics card powered by LuvitRED

Franco Arboleda

12-Nov-15

## Table of Contents

1	Introduction .....	3
2	Using the RS232 interface from the front panel.....	5
2.1	Modifying the configuration under the Advanced Editor.....	9
2.1.1	Verifying if firewall hole is opened by LuvitRED .....	9
2.1.2	Inactivity timeout on the TCP node.....	10
3	Using the USB ports for storage.....	11
3.1	Writing data to the mass storage device.....	11
3.2	Reading data from the mass storage device. ....	14
4	Configuring the I/O interfaces.....	16
4.1	Digital outputs .....	17
4.2	Digital inputs .....	20
4.3	Analog inputs .....	23
4.3.1	Monitoring the Digital output using a GPIO query node .....	26
5	One Wire interface .....	28
6	Double SIM usage.....	32

# 1 Introduction

This document covers the configuration of the telematics card using LuvitRED.

There are currently three versions of the telematics card:

1. Telematics base board (CG1106-11957):



Figure 1: Telematics base board.

2. Telematics card with I/O expander (CG5106-11983):



Figure 2: Telematics card with I/O expander.

3. Telematics card with CAN I/O expander (CG5106-11984):



Figure 3: Telematics card with CAN I/O expander.

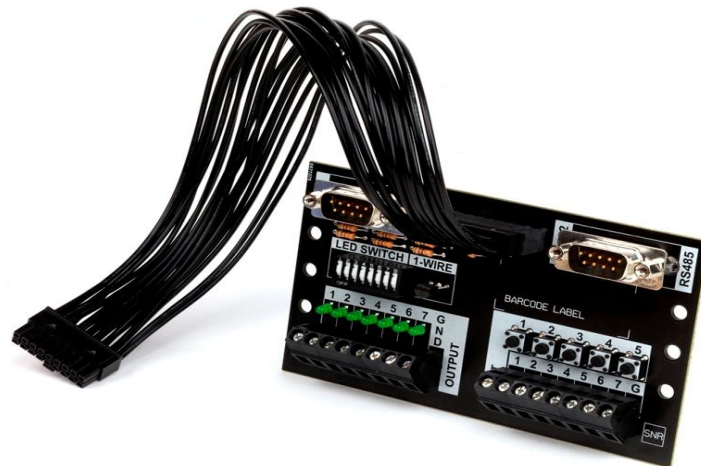
There are two differences between the Telematics card with I/O expander and the Telematics card with CAN I/O expander:

1. Two of the digital outputs of the I/O expander are used for the CAN BUS protocol on the CAN I/O expander.
2. The Auxiliary serial port available on the molex connector of the I/O expander is switchable between RS232 and RS485 (2 Wires) on the CAN I/O expander.

Visit the CloudGate Universe server (<http://cloudgate.option.com/>) for more information about these different versions of the telematics card and their configurations.

For this document, the telematics card with I/O expander is going to be used as an example, and any difference when working with the other versions will be mentioned.

The telematics breakout board (CG7101-12018) designed specifically for the telematics card with I/O and CAN I/O expanders will be used in this document:



**Figure 4: Breakout board.**

## 2 Using the RS232 interface from the front panel.

Go to the "Plugin" tab and then to the sub-tab called "Serial and GPS settings" or "LuvitRED" (the name "LuvitRED" is used from version 2.3.0 onwards):

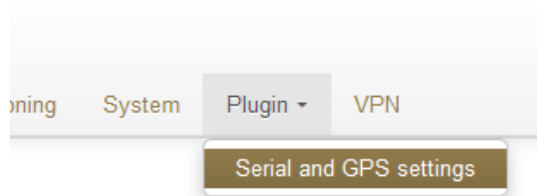


Figure 5: Plugin tab, Serial and GPS settings.

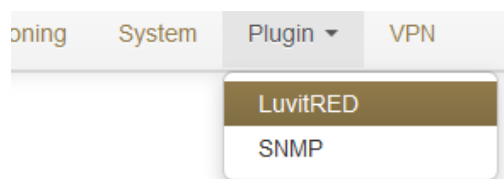


Figure 6: Plugin tab, LuvitRED.

**NOTE:** Do not focus on the SNMP (Simple Network Management Protocol) sub-tab. This tab is not going to be used on this document.

Without any configuration, the basic interface looks as follows:

 A screenshot of a configuration page titled 'Serial port to TCP local or remote server'. It contains two sections. The first section, 'Serial port to TCP local or remote server', has an 'Enable' label and two radio buttons: 'yes' (selected) and 'no'. The second section, 'GPS to TCP local or TCP/UDP remote server', also has an 'Enable' label and two radio buttons: 'yes' (selected) and 'no'. At the bottom, there are 'Save' and 'Reset' buttons on the left, and an 'Advanced Editor' button on the right.

Figure 7: Basic interface.

To configure the RS232 interface, we are going to focus on the section called "Serial port to TCP local or remote server". This section allows the configuration of one single serial port, the RS232 (`/dev/ttySP0` by default), to be accessible remotely via a local TCP server running on the CloudGate (See Figure 8) or a remote TCP server, running at another location (See Figure 9).

**NOTE:** Other, more advanced configurations, can be achieved by using the "Advanced Editor" of LuvitRED.

Serial port to TCP local or remote server

Enable
☒ yes
☐ no

Serial port settings

Baud rate

Data bits
☐ 7
☒ 8

Stop bits
☒ 1
☐ 2

Parity
☒ none
☐ even
☐ odd
☐ mark
☐ space

Flow control
☒ none
☐ XON/XOFF
☐ CTS/RTS

TCP settings

TCP server is
☒ Local
☐ Remote

Port

GPS to TCP local or TCP/UDP remote server

Enable
☐ yes
☒ no

Save
Reset
Advanced Editor

Figure 8: Serial to local TCP server.

Serial port to TCP local or remote server

Enable
☒ yes
☐ no

Serial port settings

Baud rate

Data bits
☐ 7
☒ 8

Stop bits
☒ 1
☐ 2

Parity
☒ none
☐ even
☐ odd
☐ mark
☐ space

Flow control
☒ none
☐ XON/XOFF
☐ CTS/RTS

TCP settings

TCP server is

Hostname

Port

GPS to TCP local or TCP/UDP remote server

Enable
☐ yes
☒ no

Figure 9: Serial to Remote TCP server.

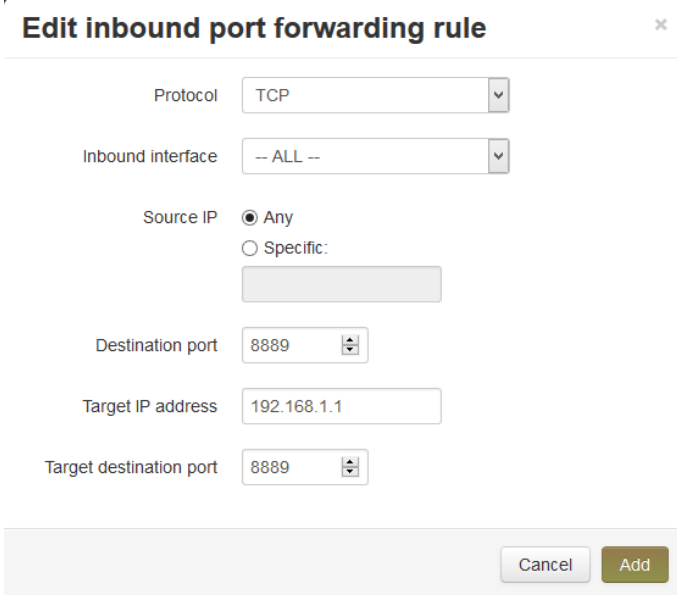
On both configurations, the settings for the CloudGate's serial interface can be found (**Serial port settings**):

- Baud rate
- Data bits
- Stops bits
- Parity
- Flow control

**NOTE:** These settings need to match the setting of the device connected to the serial interface.

In Figure 8, the CloudGate is running a local TCP server that will listen for incoming connections and forward them to the serial port. The Port number of the TCP server is, by default, **8889**, but it can be changed, as required for the application, at any moment.

If access from the WAN interface (internet) is needed, an appropriate firewall rule needs to be in place to allow the connection to the port:



**Edit inbound port forwarding rule**

Protocol: TCP

Inbound interface: -- ALL --

Source IP: ☒ Any ☐ Specific:

Destination port: 8889

Target IP address: 192.168.1.1

Target destination port: 8889

Buttons: Cancel, Add

Figure 10: Inbound port forwarding rule.

**NOTE:** All LuvitRED 2.x.x versions already opens a firewall hole to allow remote access from the WAN interface on this specific configuration. This can be verified only under the advanced editor, not on the basic interface (see section 2.1.1).

**Warning:** In Figure 9, the CloudGate will connect to a remote TCP server running on the specified port and send all the information that arrives from the device connected to the serial interface. **Be aware that this configuration may cause high data traffic.**



## 2.1 Modifying the configuration under the Advanced Editor.

After configuring the serial port under the basic interface, one can go to the Advance Editor and edit the configuration. The configuration made on the basic interface will be reflected under the Advanced editor in the following way:

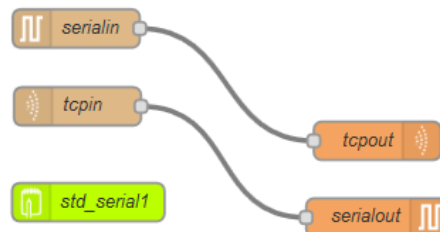


Figure 11: Same configuration under Advanced Editor.

### 2.1.1 Verifying if firewall hole is opened by LuvitRED

1. Double click on the **tcpin** node:

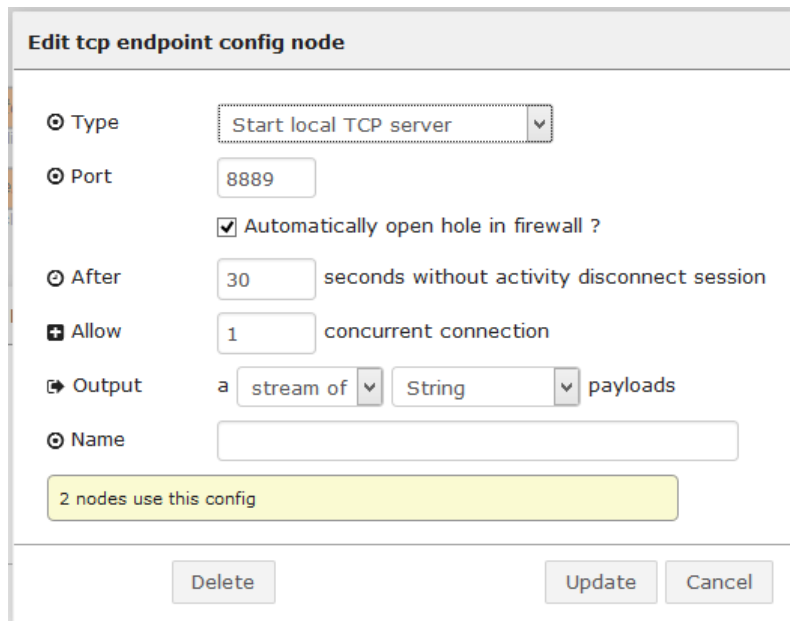
Figure 12: Tcpin node general configuration.

2. Access the "Endpoint" configuration by clicking on the pencil icon and Check if the configuration item called "Automatically open a hole in firewall?" is checked or modify it according to the needs of the configuration:

Figure 13: Endpoint configuration.

## 2.1.2 Inactivity timeout on the TCP node.

1. Open the "Endpoint" configuration as explained on section 2.1.1. Once there, add a timeout, in seconds (30 on the example below), on the "After \_\_\_ seconds without activity disconnect session" configuration item so that it closes any open connection that is not generating traffic:



**Edit tcp endpoint config node**

Type: Start local TCP server

Port: 8889

☒ Automatically open hole in firewall ?

After: 30 seconds without activity disconnect session

Allow: 1 concurrent connection

Output: a stream of String payloads

Name:

2 nodes use this config

Delete Update Cancel

Figure 14: Adding connection timeout.

### 3 Using the USB ports for storage.

From firmware version 1.46.0/2.46.0 or later, auto-mount for USB and SD mass storage devices (FAT file systems only) is supported on the CloudGate hardware.

Any new FAT formatted drive will be mounted under the **/mnt/** directory. Normally these drives are mounted as **sdX#**:

```
admin@cgate:~$ ls /mnt/
base_cfg  cust      cust_cfg  sda1      sdb1
admin@cgate:~$
```

Figure 15: Example of two FAT drives mounted on the Linux system (sda1 and sdb1).

In the example in Figure 15, two mounted drives are shown on the system, sda1 and sdb1. The sda1 drive is connected to the USB Type A interface while the sdb1 drive is connected on the USB OTG interface using a micro-USB to USB adapter.

Under the "Advanced editor" of LuvitRED, there are some nodes that are in charge of data storage and others for parsing data (See Figure 16):

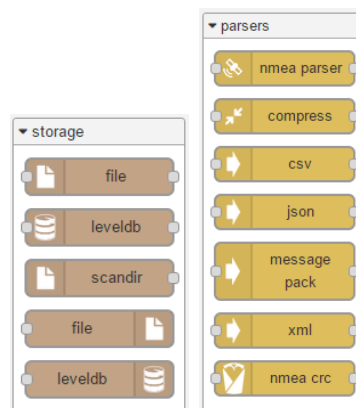


Figure 16: Storage and parsing nodes.

#### 3.1 Writing data to the mass storage device.

Let's say we want to write a file to the **sda1** drive which currently looks like this:

```
admin@cgate:~$ ls /mnt/sda1/
1_DATA  HBCD  drivers  home
CLEAN   autorun.inf  grldr    menu.lst
admin@cgate:~$
```

Figure 17: Current view of sda1 under the Linux system (shell view).

## OPTION

1. Drop a file out node:

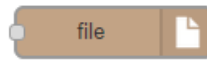
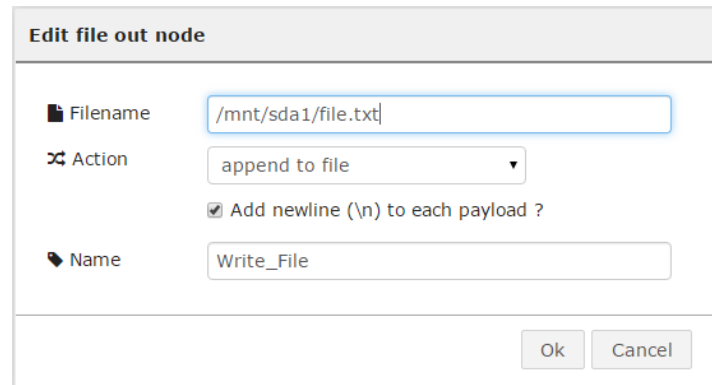


Figure 18: File out node.

2. Configure the file out node like this:
  - a. Add the filename to write using the full location: ***/mnt/sda1/file.txt***
  - b. Choose an action for the node (append to file in our case), there are three actions available:
    - i. append to file
    - ii. overwrite file
    - iii. delete file
  - c. Choose if you want to add a **newline** character at the end of every line written to the file.
  - d. Change the node's name.

A dialog box titled "Edit file out node" with a light gray header. It contains four configuration fields: "Filename" with a text input field containing "/mnt/sda1/file.txt"; "Action" with a dropdown menu showing "append to file"; a checked checkbox labeled "Add newline (\n) to each payload ?"; and "Name" with a text input field containing "Write\_File". At the bottom right, there are "Ok" and "Cancel" buttons.

Edit file out node	
Filename	/mnt/sda1/file.txt
Action	append to file
<input checked="" type="checkbox"/> Add newline (\n) to each payload ?	
Name	Write_File
Ok Cancel	

Figure 19: File out node configuration.

## OPTION

- Let's drop an inject node to send some data to the file out node. In this case the Inject node is configured to send a string "write test" every time we press the inject button:

Figure 20: Inject node configuration.

- Connect both nodes together as shown below:



Figure 21: Write flow.

- Deploy the configuration.

After pressing the inject button next to the inject node a few times (3 times in this example). The file is containing the following information when reading it on a SSH session:

```
admin@cgate:~$ ls /mnt/sda1/
1_DATA      HBCD        drivers     home
CLEAN       autorun.inf grldr       menu.lst
admin@cgate:~$ cat /mnt/sda1/file.txt
write test
write test
write test
admin@cgate:~$
```

Figure 22: File containing new information on the sda1 drive.

## 3.2 Reading data from the mass storage device.

Now that we have written information to a file in the **sda1** drive on section 3.1, we want to read it back.

1. For reading the **/mnt/sda1/file.txt** we need first to drop a file in node:



Figure 23: File in node.

2. Configure the file in node like this:
  - a. Add the filename to read using the full location: **/mnt/sda1/file.txt**
  - b. Choose a Read file action for the node (once per message in our ase), there are two actions available:
    - i. once per message
    - ii. continuously
  - c. Choose if you want to delete the file after a successful read (leave it blank for our example).
  - d. Change the node's name.

Edit file in node

Filename

/mnt/sda1/file.txt

Read file

once per message

☐ Delete file if successfully read ?

Name

Read\_File

Ok

Cancel

Figure 24: File out node configuration.

## OPTION

- Let's drop an inject node to trigger the file in node to read (this will be the message that the node is waiting for reading "once per message"). In this case the Inject node is configured with its default values, so no change on its configuration:

**Edit inject node**

Payload:

Topic:

Repeat:

☐ Fire once at start ?

Name:

**Note:** "interval between times" and "at a specific time" will use cron. See info box for details.

Ok Cancel

Figure 25: Inject node configuration.

- Let's also drop a debug node and connect the three nodes together as shown below:



Figure 26: Read flow.

- Deploy the configuration.

After pressing the inject button next to the inject node, the debug node should print the reading made by the file in node and print the result on the debug tab:

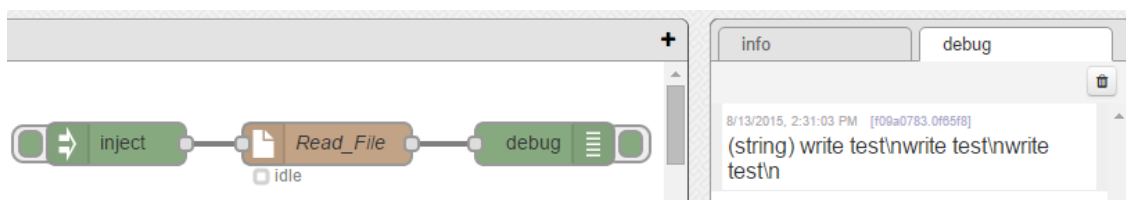


Figure 27: Debug node printing the result of reading the file.

Of course, printing the file to debug might not be something useful, but it is a good step to show that the file was correctly read. Instead of a debug node, or together with it, one could place other kind of nodes to, for example, send the file to a remote server, or make it available for a remote incoming connection.

## 4 Configuring the I/O interfaces.

On the telematics cards, models CG5106-11983 and CG5106-11984, there are three different types of IO interfaces as explained in the documentation on the CloudGate universe:

- 5 Digital inputs (I1, I2, I3, I4 and I5)
- 2 Analog inputs (AI1 and AI2)
- 6 Digital outputs (DO1, DO2, DO3, DO4, DO6 and DO7) - 4 DOs in the case of the telematics with CAN IO expander - DO3 and DO4 are used for the CAN BUS interface.

There are three GPIO related nodes in LuvitRED, GPIO in, GPIO out and GPIO query:



Figure 28: GPIO nodes.

- The **GPIO in** node is used to read the values of either a digital input or an analog input pin.
- The **GPIO out** node is used to write a value to a Digital output pin.
- The **GPIO query** node is used to query the status of an IO pin.

The telematics breakout board has access to all those pins. For the Digital output the board also has LEDs; For the Digital inputs the board includes some buttons and for the Analog inputs there are some connectors (See Figure 29).



Figure 29: Front of the breakout board.



## 4.1 Digital outputs

Let's say that we want to turn on the LEDs of the breakout board ON. We first need to change the LED switch to the "ON" position (pressed towards the top side), when this is done, the LED number 5 is lit (This LED is actually not connected to the Digital outputs):



Figure 30: Switch on "ON" position and LED number 5 ON.

Let's start by turning on and off one single LED, such as DO1 by using two inject nodes and one GPIO out node.

1. Drop a GPIO out node and configure it the following way:

Figure 31: Default GPIO out node.

2. Add a **new** gpio out pin by pressing on the pencil icon and configure it for O1 (DO1):

Figure 32: DO1 configuration.

- Then let the rest of the configuration as it is and change the name of the node

**Edit gpio out node**

Pin: Output pin O1

Message: msg.payload

☐ Invert output ?

Name: DO1

Ok Cancel

Figure 33: Final configuration of the GPIO out node.

#### NOTES:

- The "Message" option of the configuration is determining what part of the message will be use as a trigger. In our case we will use the payload, but on more sofisticated configurations, one might need to use something else, such as **msg.payload.trigger**.
  - The "Invert output?" option of the configuration is simply inverting the meaning of the message received. If **msg.payload** is equal to 1 in our case, then the node will understand that 1 is 0.
- Drop in an Inject node and configure it the following way:
    - Change the Payload to "number"
    - Then add a "1" as the payload.
    - Leave the rest as it is and change the name of the node to "ON"

**Edit inject node**

Payload: number

1

Topic: topic

Repeat: none

☐ Fire once at start ?

Name: ON

Note: "interval between times" and "at a specific time" will use cron. See info box for details.

Ok Cancel

Figure 34: Inject node "ON" configuration.

## OPTION

- Drop a second Inject node and configure it the same way as the first inject, but in this case set the payload to "0" and change the name to "OFF":

Figure 35: Inject node "OFF" configuration.

- Let's connect the three nodes together in the following way:

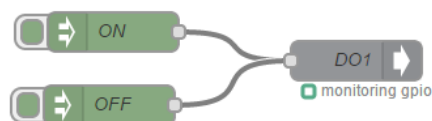


Figure 36: Final configuration.

- Deploy the new configuration and test the DO1 by pressing on the "ON" and "OFF" injects:

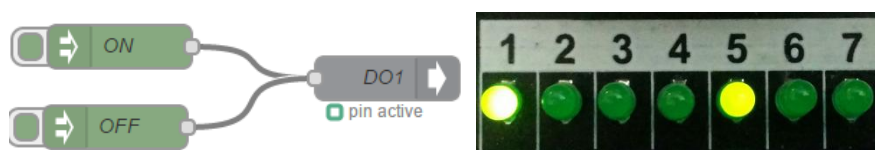


Figure 37: ON inject pressed.

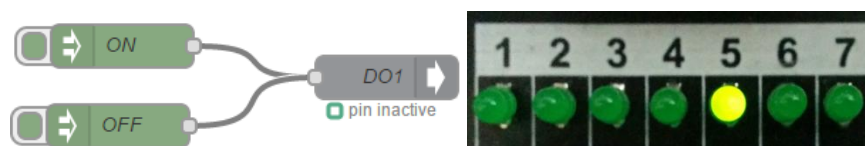


Figure 38: Off inject pressed.

## OPTION

This configuration can be reproduced for all the other LEDs simply by adding more GPIO out nodes and configure them by adding a **NEW** gpio out pin as shown on Figure 32. For this example we are connect the same inject nodes to all GPIO pins:

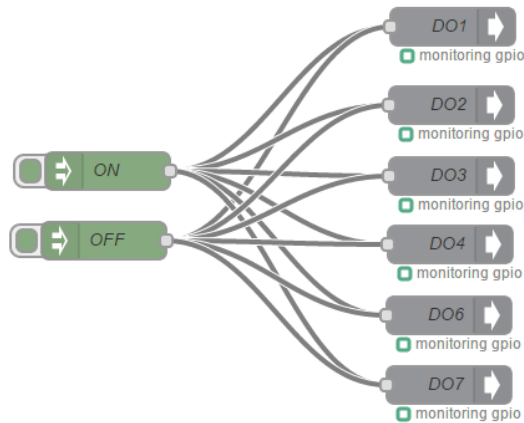


Figure 39: All Digital outputs to the same triggers.



Figure 40: All LEDs ON.

## 4.2 Digital inputs

Let's say that instead of turning the LEDs on by using an inject node we would like to do it by pressing a button (Digital Input) and turning them off pressing another button.

1. Drop a GPIO in node and configure it the following way:

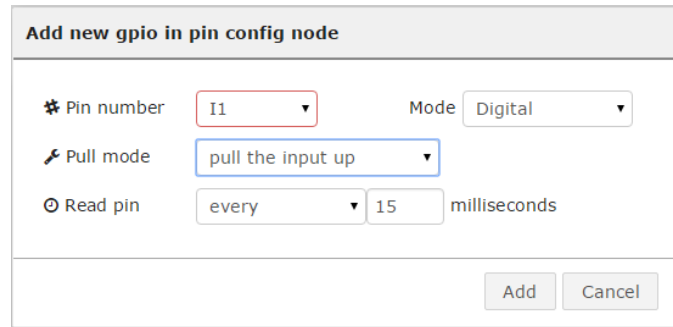
The screenshot shows the 'Edit gpio in node' configuration window. It contains the following fields and options:

- Pin:** A dropdown menu with the text 'Add new gpio in pin...' and a pencil icon.
- Message:** A dropdown menu with the text 'always sent'.
- Topic:** A text input field with the text 'topic'.
- Name:** A text input field with the text 'Name'.
- Buttons:** 'Ok' and 'Cancel' buttons at the bottom right.

Figure 41: Default GPIO in node.

## OPTION

2. Add a new gpio in pin by pressing on the pencil icon and configure it for I1:
  - a. Set the mode to Digital (digital inputs can also work as analog inputs)
  - b. Set the Pull mode to "pull the input up"
  - c. Set the read interval to 15 ms



**Add new gpio in pin config node**

Pin number: I1 Mode: Digital

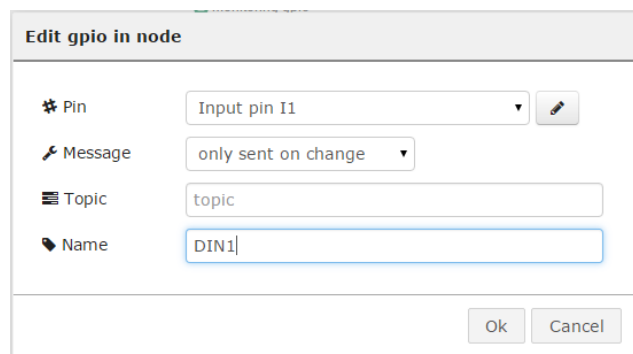
Pull mode: pull the input up

Read pin: every 15 milliseconds

Add Cancel

Figure 42: I1 configuration.

3. After adding the pin, change the Message to only sent on change and then rename the node:



**Edit gpio in node**

Pin: Input pin I1

Message: only sent on change

Topic: topic

Name: DIN1

Ok Cancel

Figure 43: Final configuration of the first GPIO in node.

## OPTION

- Drop a second GPIO in node. Be sure to **ADD** a **NEW** gpio pin instead of editing the existing one and select I2 pin instead. Configure it in the same way as the first node, changing the name to DIN2:

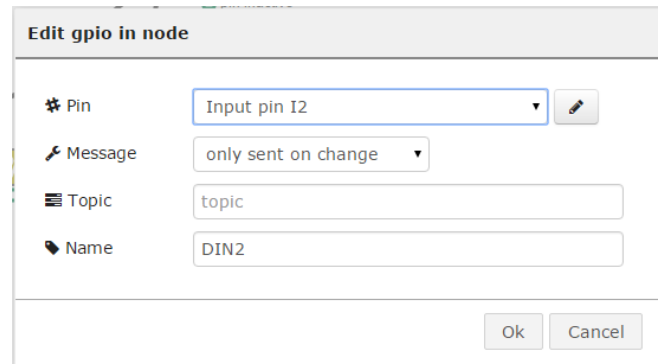


Figure 44: Final configuration of the second GPIO in node.

- Now, in order to make the second GPIO in to have a reversed action as for the first GPIO in, we need to add another node that can make such change for us. There are several options, but for this example we are going to use a "change" node:

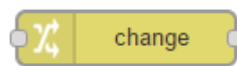


Figure 45: Change node.

- We need to configure it the following way:

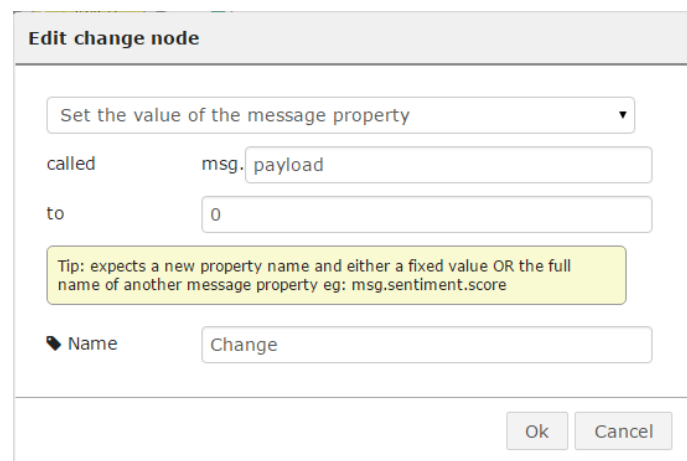


Figure 46: Change node configuration.

We are basically telling the node to change the **payload** of the message to 0 when triggered.

## OPTION

7. Connect the nodes the following way and deploy:

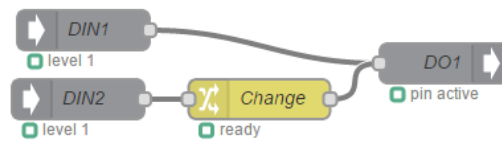


Figure 47: Final configuration using Digital inputs for one output.

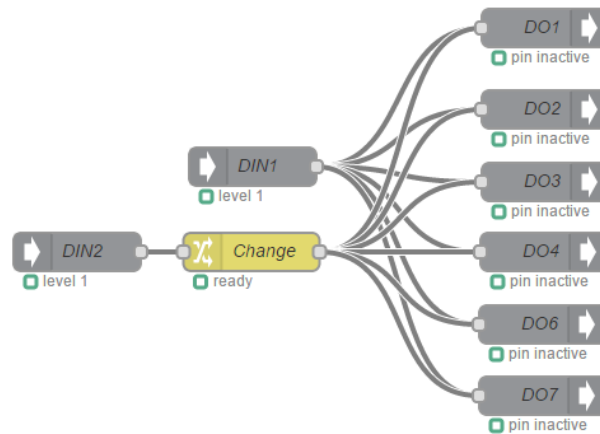


Figure 48: Final configuration using digital inputs for multiple outputs.

### 4.3 Analog inputs

For this example we are going to use the first input (I1) of the breakout board connected to the sixth input connector (A11) using a resistor (220  $\Omega$ ) the following way:

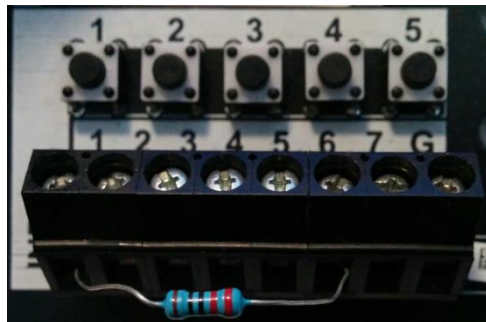
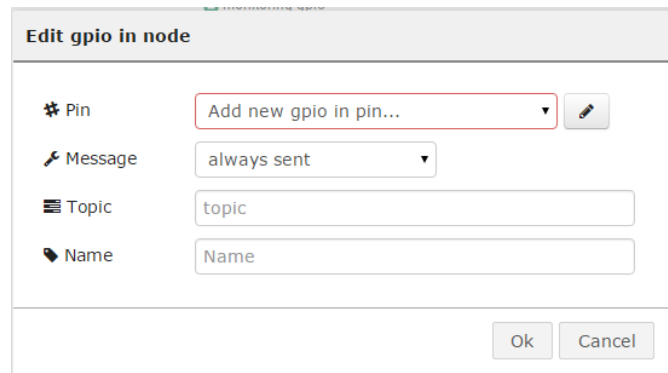


Figure 49: Resistor connected between the digital and analog inputs.


The idea behind doing this is that this setup will show a voltage variation over time on the digital input the same way a sensor will do, and when the button is pressed the voltage will go down to zero.

**NOTE:** Do not delete the GPIO in node called DIN1 created on the previous section. This node needs to be present in order to provide the necessary voltage to the PIN.

1. Drop a GPIO in node and configure it the following way:



**Edit gpio in node**

Pin: Add new gpio in pin... 

Message: always sent

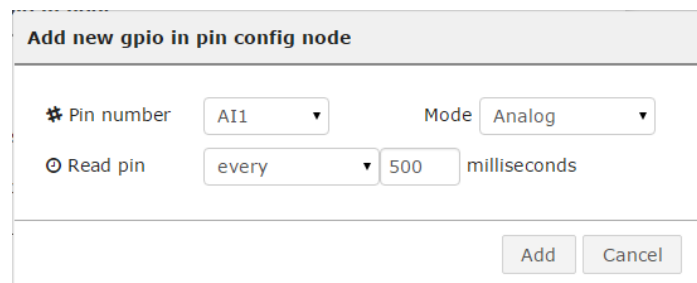
Topic: topic

Name: Name

Ok Cancel

Figure 50: Default GPIO in node.

2. Add a new gpio in pin by pressing on the pencil icon and configure it for AI1:
  - a. Set the mode to Analog (default after selecting AI1)
  - b. Set the read interval to 500 ms



**Add new gpio in pin config node**

Pin number: AI1 Mode: Analog

Read pin: every 500 milliseconds

Add Cancel

Figure 51: AI1 configuration.

3. After adding the pin, rename the node and change the "Message" to "only sent on change":



Figure 52: Final configuration of the first GPIO in node.

4. Let's say that we add a GPIO out node and configure it the same way as explained on section 4.1:

Figure 53: Digital output configured.

5. Connect both nodes together and "Deploy" the configuration:

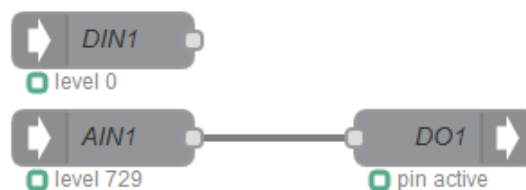


Figure 54: Final configuration for Analog input.

At this point we should see that the digital output 1 is ON all the time because the value of the Digital input is always higher than "1". The only way to turn the digital output 1 OFF is by pressing the digital input button, this is not directly related to the button itself, but because the analog input is going to have a reading of "0":

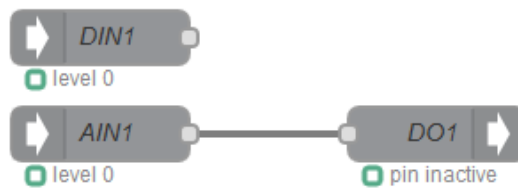


Figure 55: Analog input reading "0" after pressing the digital input.

### 4.3.1 Monitoring the Digital output using a GPIO query node

There is one GPIO node that we have not used yet, it is the GPIO query node. This node queries the status of any GPIO IN or OUT and returns the value.

Let's say that we need to monitor the status of the digital output we configured on the last section every two seconds and print it into the debug tab.

1. Drop a GPIO query node into the flow and configure it in the following way:

The screenshot shows a dialog box titled 'Edit gpio query node'. It has three input fields: 'Pin type' with a dropdown menu set to 'output', 'Pin' with a dropdown menu set to 'Output pin O1' and a small edit icon, and 'Name' with a text input field containing 'DO1\_q'. At the bottom right, there are 'Ok' and 'Cancel' buttons.

Figure 56: GPIO query node.

2. Drop an inject node that will act as a trigger for the query node and configure it as shown below:

Figure 57: Inject node acting as a trigger (2s).

- Now let's drop a debug node and connect the three nodes the following way and click on "Deploy":

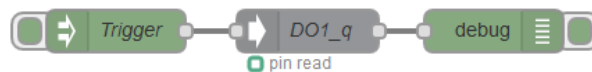


Figure 58: GPIO query flow.

- This new flow will print the status of the DO1 pin at every two seconds:

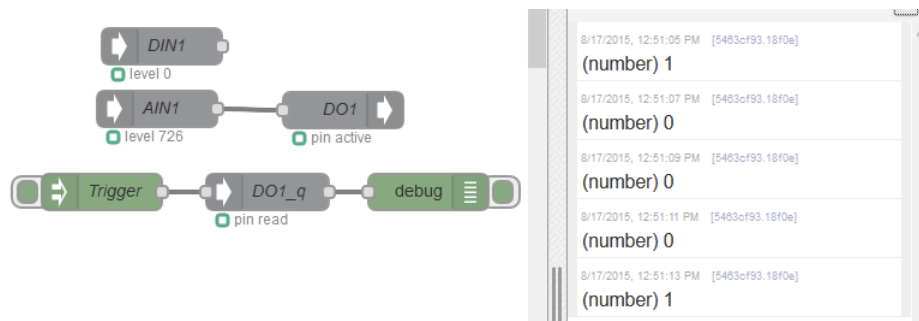


Figure 59: Output of the GPIO query node on the debug tab.

## 5 One Wire interface

On the telematics cards, models CG5106-11983 and CG5106-11984, there is a 1-Wire interface that allows connection to 1-wire enabled devices.

There are two 1-wire related nodes on LuvitRED, OWS search and OWS temp:

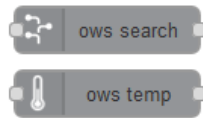


Figure 60: 1-wire nodes.

- The OWS search node is used to discover the 1-wire devices connected to the 1-wire bus.
- The OWS temp node is used to retrieve the temperature from a 1-wire temp sensor.

On the breakout board, there is one 1-wire temp sensor located right next to the LED switch:



Figure 61: 1-wire temp sensor.

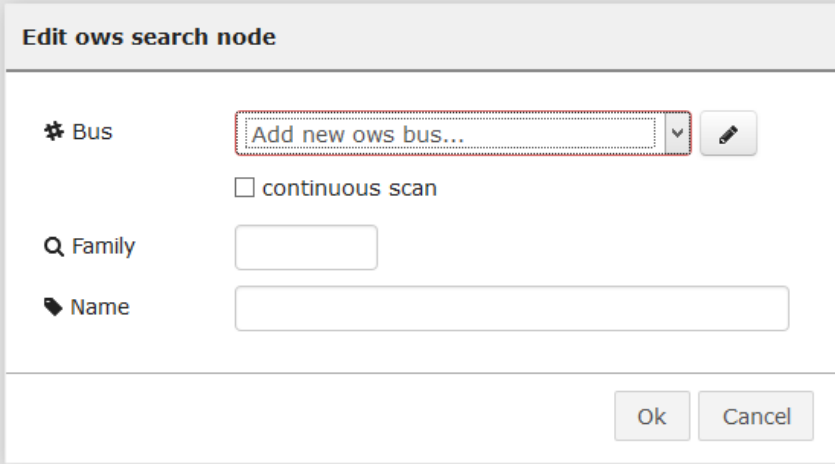
We first need to change the LED 5 switch to the "OFF" position (pressed towards the bottom side), when this is done, the LED number 5 will no longer be lit:




Figure 62: LED 5 Switch to Off position.

To get the reading of the sensor we first need to find out the serial number of it. We can do that by using the OWS search node.

1. Drop a OWS search node into the editor and configure it as follows:



**Edit ows search node**

⚙️ Bus  

☐ continuous scan

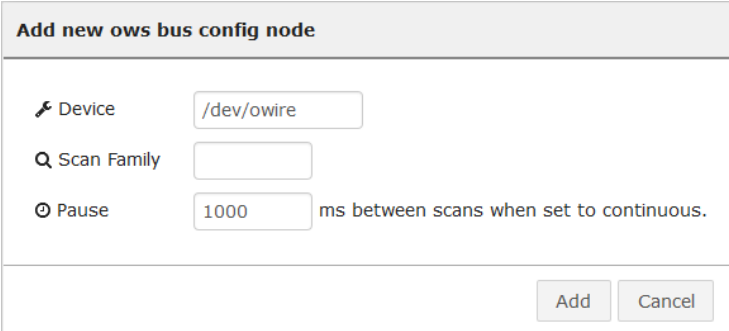
🔍 Family

🏷️ Name

Ok Cancel

Figure 63: OWS search node defaults.

2. Add a new OWS bus by clicking on the pencil and configure it as follows (default configuration):



**Add new ows bus config node**

🔧 Device

🔍 Scan Family

⌚ Pause  ms between scans when set to continuous.

Add Cancel

Figure 64: Adding a 1-wire bus.

3. Click on "Add" and then change the name of the OWS search node and click on "OK":

Figure 65: OWS search configured.

4. Connect an inject and a debug node to the OWS search node as show below and then "Deploy":

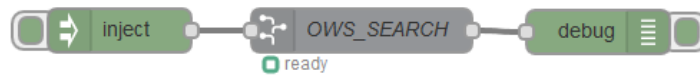


Figure 66: OWS search configuration.

When clicking on the inject node, the debug node will print the serial number of the 1-wire sensor connected (It can take a moment to print the serial number):



Figure 67: Serial number of the 1-wire sensor on the Breakout board.

**NOTE:** Make sure the LED Switch number 8 is pressed on the ON position (towards the number) as can be seen on Figure 61.

5. We can now drop a OWS temp node into the flow and configure it as follows:

Figure 68: OWS temp node configuration.

- a. The "Bus" should be already configure as the OWS temp node will get the bus we configured for the OWS search node.
- b. Keep the "Message" as it is.

- c. Change the name of the node and click on "OK".
2. Connect the input of this node to the output of the OWS search node and drop a new debug node to be connected on the output of the OWS temp node and then click on "Deploy":

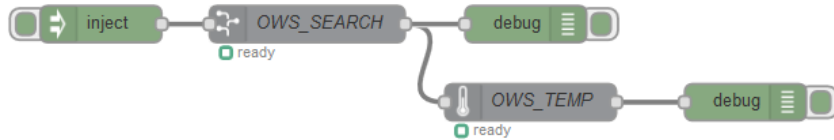


Figure 69: OWS temp node connected to the flow.

When clicking on the inject node, the debug node will print the serial number of the 1-wire sensor connected (It can take a moment to print the serial number) and then it will print the same serial number followed by the temperature value:

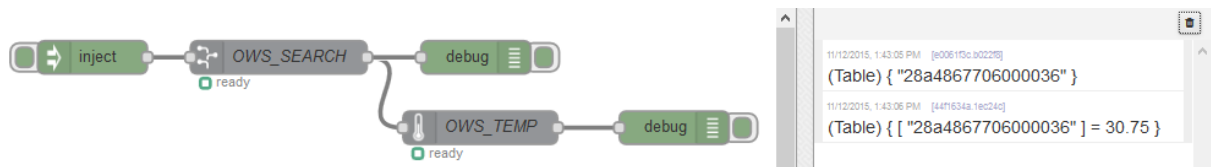


Figure 70: Serial and temperature value of the 1-wire sensor on the Breakout board.

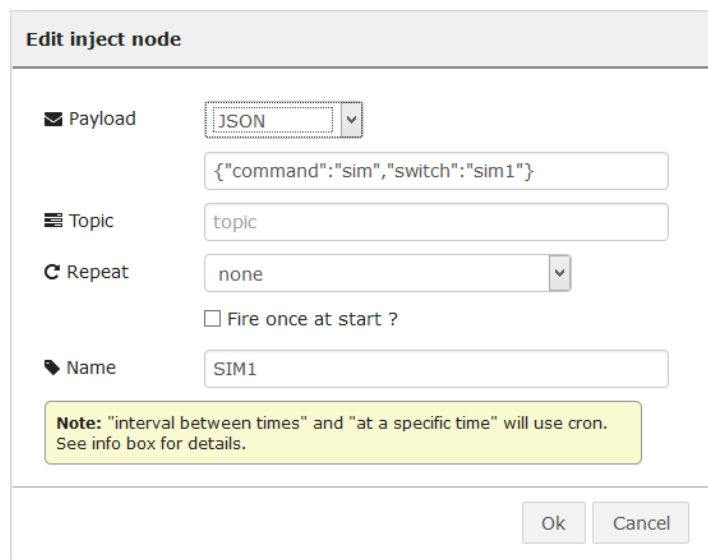
**NOTE:** The value of the sensor shown on Figure 70 is in degrees Celsius. Depending on the specific sensor on the breakout board or the sensor connected to the 1-wire bus, the value might be presented in degrees Fahrenheit. Please, refer to the manual of the specific sensor for more information.





For the following example, we are only going to use the switch methods "**sim1**" and "**sim2**". We could use two inject nodes to trigger the SIM switch:

1. Drag and drop an inject node, change the payload to JSON, change its name to SIM1 and set the value of the payload to `{"command":"sim","switch":"sim1"}`:



**Edit inject node**

✉ Payload: JSON

📄 Topic:

🔄 Repeat: none  
☐ Fire once at start ?

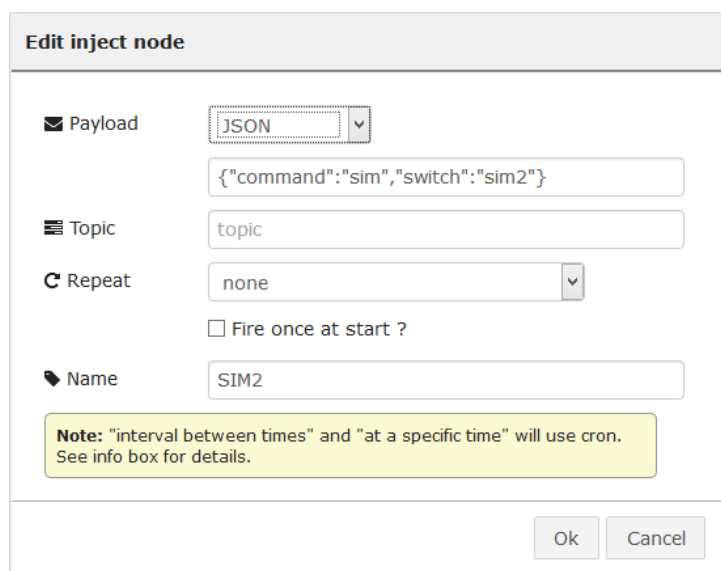
📌 Name:

**Note:** "interval between times" and "at a specific time" will use cron. See info box for details.

Ok Cancel

Figure 73: Inject node, sim switch command sim1.

2. Drag and drop a second inject node, change its payload to JSON, change its name to SIM2 and set the value to `{"command":"sim","switch":"sim2"}`:



**Edit inject node**

✉ Payload: JSON

📄 Topic:

🔄 Repeat: none  
☐ Fire once at start ?

📌 Name:

**Note:** "interval between times" and "at a specific time" will use cron. See info box for details.

Ok Cancel

Figure 74: Inject node, sim switch command sim2.

## OPTION

The internal cellular interface is rebooted after a switch command is sent to the WAN control, this is to allow the modem to read the new SIM card and start the registration to the new mobile network. Some feature of the CloudGate will not be available during this rebooting process.

The WAN monitor node now contains two new items on its WWAN output:

- **sim.current**: Current SIM being used by the CloudGate.
- **sim.mode**: SIM card selected for switch.

When these two values are equal, it means that the CloudGate is already using the correct (Selected) SIM card.

The WAN monitor node also has a new input that allows for requesting the status of one of its topics (e.g. WWAN - WAN monitor has two topics, one called WAN and the other called WWAN).

3. Let's drop an inject node (changing its payload to string and set it to WWAN), a debug node and a WAN monitor node and connect them together the following way:

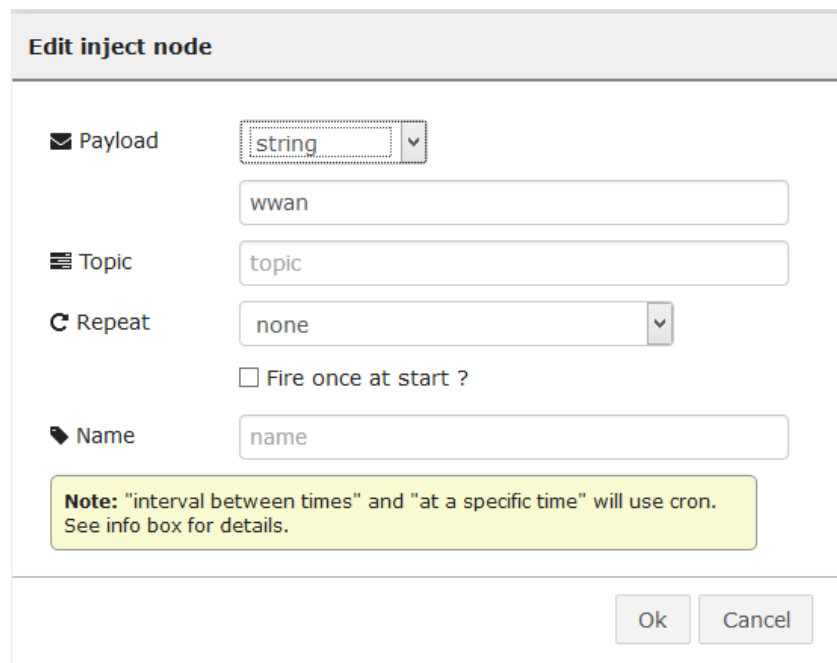


Figure 75: Inject node configuration.

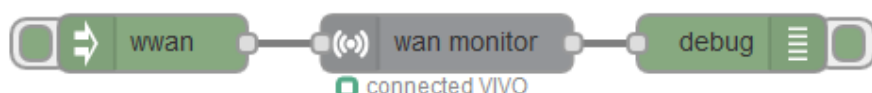


Figure 76: Wan monitor node connection.

## OPTION

- Let's also connect the two inject nodes we created before to a WAN control node as shown below:



Figure 77: SIM1 and SIM2 inject nodes connected to Wan control node.

- And now click on "Deploy".

The WAN monitor node should tell us which SIM is currently used and which SIM is selected for the switch:

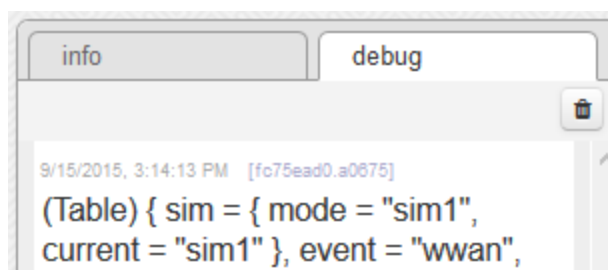


Figure 78: Debug output from Wan monitor node showing "sim1" on both mode and current.

Let's switch the SIM to **sim2** by pressing on the SIM2 inject node and see the output of the WAN monitor node:

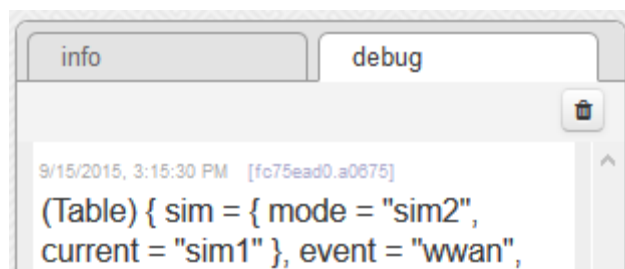


Figure 79: Sim mode showing "sim2".

## OPTION

Wait for the modem to reboot and then see the output of the WAN monitor node:

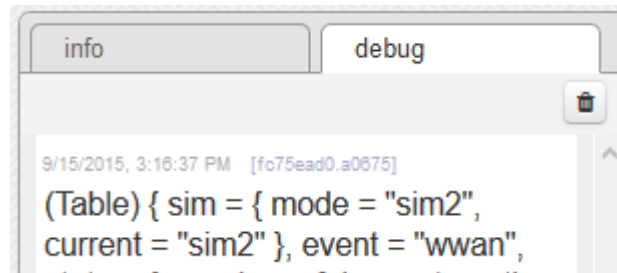


Figure 80: Both SIM mode and current are now set to "sim2".

This output means that after selecting the second SIM card and the cellular modem being rebooted, the CloudGate is starting to use the second SIM on the telematics card.

Another way of checking if the SIM card is actually switched is by checking the IMSI value (unique per SIM card). This value can be obtained from the **sysinfo** node by triggering it:



Figure 81: IMSI of the first SIM card.

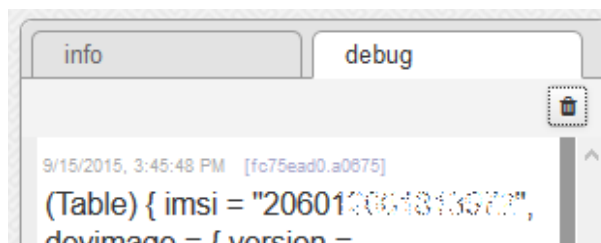


Figure 82: IMSI of the second SIM card.

○ ○  
○ ○ P T I O N  
○ ○