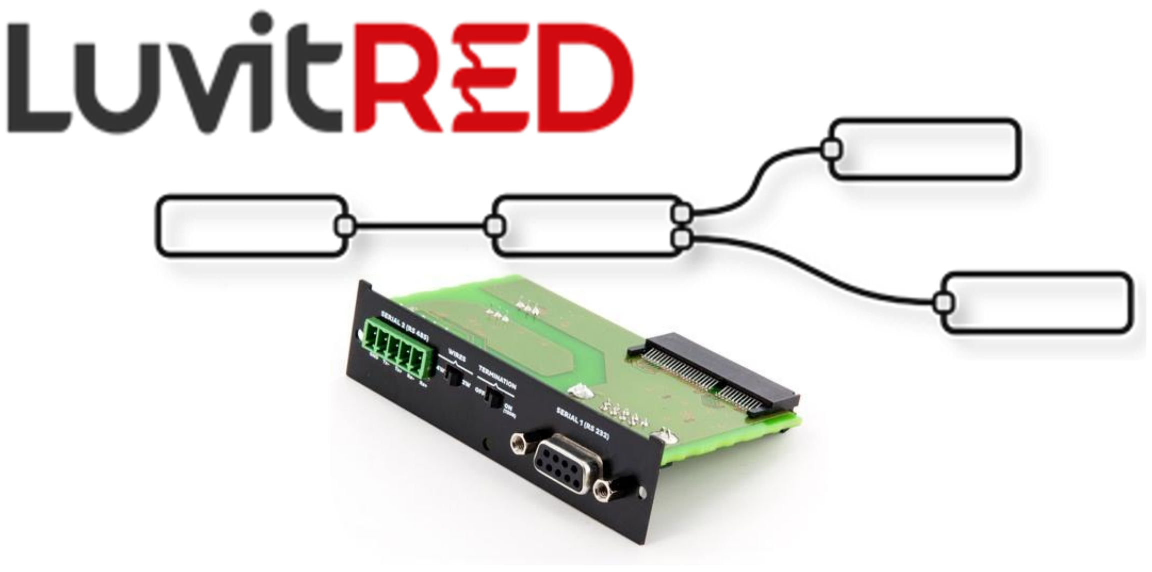


Modbus Gateway using the industrial serial card

Franco Arboleda

01-Dec-15

Table of Contents

1 Introduction

This document covers the configuration of the CloudGate with an industrial serial card for a Modbus Gateway (ModGate) application using LuvitRED.

A Modbus Gateway is basically a protocol converter from, for example Modbus TCP to Modbus RTU/ASCII. This allows a Modbus master or multiple master devices to talk with a single device concentrating information from multiple other Modbus devices behind the gateway:

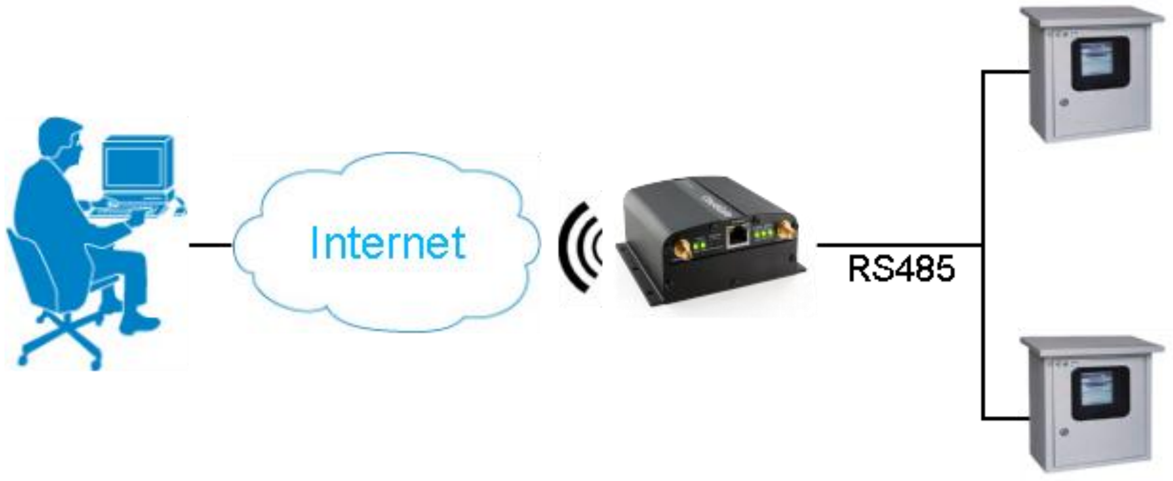


Figure 1: ModGate application.

The CloudGate can perform this action when using LuvitRED, but it can also show more information than just the information from other Modbus RTU/ASCII devices. The CloudGate is capable of using all information available to LuvitRED such as cellular information, GPS, I/O, other Modbus TCP information, etc to also report them as new modbus registers.

The industrial serial card (CG1102-11920) is shown on Figure 2 below:

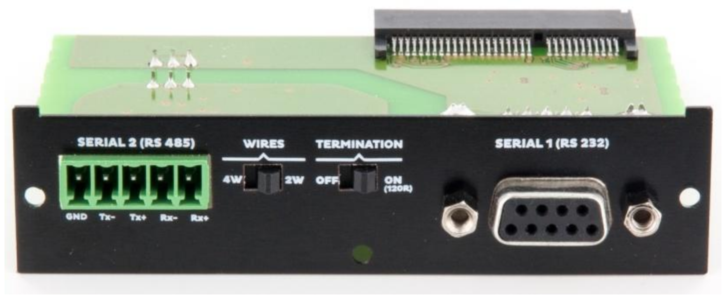


Figure 2: Industrial Serial Card

- The **RS232** interface is shown on the operating system of the CloudGate as **/dev/ttySP0**

- The **RS485** interface is shown on the operating system of the CloudGate as **/dev/ttySP4**
- The **RS485** interface also has two switchable items that need to be set correctly on the plate:
  - The first item is the amount of wires to use: 4W or 2W.
  - The second item is the termination: OFF or ON.

For this configuration and any other Modbus configuration, one needs to use the "Advanced Editor" of LuvitRED.

Notes:

1. This document assumes that the reader is familiar with the LuvitRED and the terms explained on the document called Basics_of_LuvitRED_vXXX.pdf.
2. LuvitRED version 2.5.3 or higher is required for this configuration.

2 ModGate configuration using the RS485 port.

For this configuration it is better to start from a real configuration. Let's say we have two Modbus RTU capable devices (Slave ID 1 and 2) that have some holding registers (16 bits). The specification of the devices is the following:

Slave ID	Holding Register	Contained Value
1	1	10
1	2	20
2	1	2000

Table 1: Slaves IDs and registers.

This Modbus RTU devices have a 2-Wire RS485 serial port with the following configuration:

Item	Value
Baud rate	9600
Parity	None
Data bits	8
Stop bits	1
RTS control	Disable

Table 2: Serial port configuration.

We are going to map these registers to our ModGate as follows:

ModGate ID	ModGate Register	Slave ID	Holding Register	Contained Value
1	1	1	1	10
1	2	1	2	20
1	3	2	1	2000

Table 3: MadGate register mapping .

The above table means that, instead of reading the holding register 1 from slave 2 to obtain the RPM, we want to obtain that value by requesting the holding register 3 from slave 1 (slave 1 being the Modbus gateway).

For this application we are going to use five Modbus nodes from LuvitRED: modbus in, modbus out, modbus fanout, modbus fanin and modbus request.

The "modbus in" node (See Figure 3) is the one that will be in charge of receiving the Modbus TCP requests from the master and pass it to the rest of the nodes in the flow. It is the first node in the configuration.

Figure 3: Modbus in node.

The "modbus out" node (See Figure 4) is the one that will be in charge of sending the responses back to the mater after being processed by all the rest of the nodes in the flow. It is the last node in the configuration.

Figure 4: Modbus out node.

The "modbus fanin" and "modbus fanout" nodes (See Figure 5) are the ones in charge of making a decision on where to send the incoming package (fanout) and repack the answers into a properly formatted Modbus response (fanin).

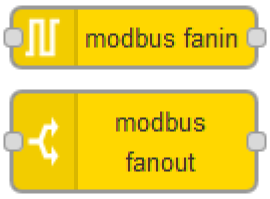


Figure 5: Modbus fanin and Modbus fanout nodes.

The "modbus request" node (See Figure 6) is the one that is actually sending the Modbus requests to the modbus RTU devices connected to the RS485 port of the CloudGate.

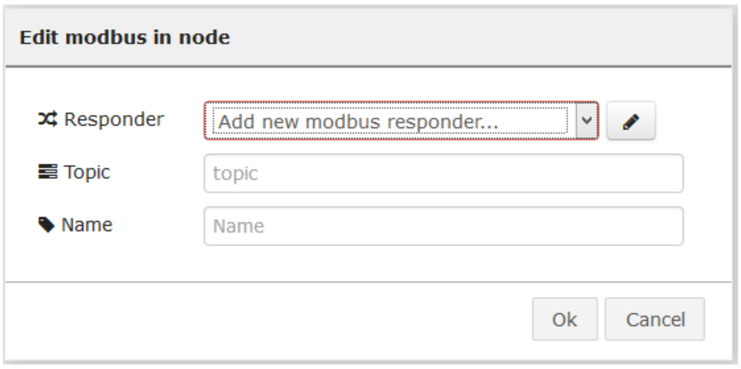
modbus request

Figure 6: Modbus request node.

2.1 Buildng the ModGate skeleton

Let's first configure our "modbus in" node:

1. Drag and Drop one "modbus in" node into the LuvitRED editor and double click on it (See Figure 7).



Edit modbus in node

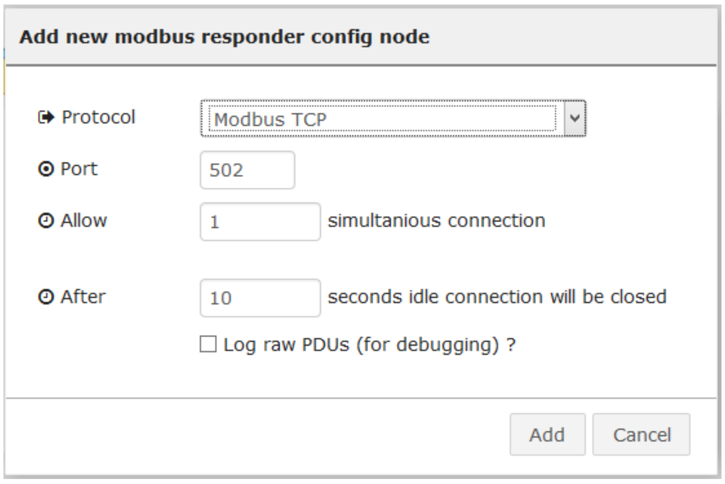
Responder: Add new modbus responder...

Topic: topic

Name: Name

Figure 7: New modbus in node configuration.

2. The first item that needs to be changed is the responder configuration. For this, click on the pencil icon to add a new modbus responder:



Add new modbus responder config node

Protocol: Modbus TCP
Port: 502
Allow: 1 simultanious connection
After: 10 seconds idle connection will be closed
☐ Log raw PDUs (for debugging) ?

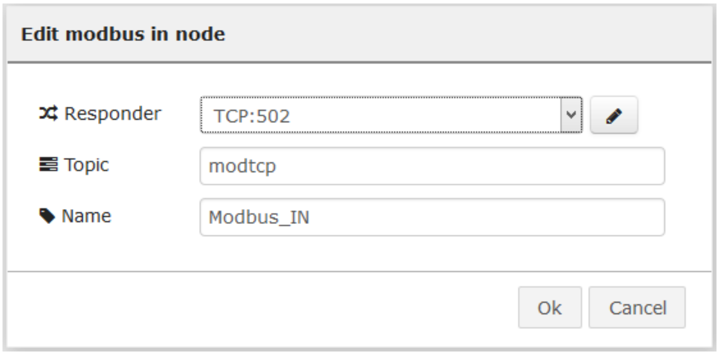
Figure 8: New responder configuration.

We are going to use the standard configuration of the modbus responder:

- Protocol: Modbus TCP.
- Port: 502.
- Allow: 1 simultaneous connection.
- After: 10 seconds idle connection will be closed.

Items 3 and 4 might be changed depending on a case by case basis as we might want to allow multiple simultaneous connections (multiple Modbus masters talking to our ModGate) or allow a higher connection timeout.

3. After adding the new responder, we just need to change the topic and name of the node as show on Figure 9:



Edit modbus in node

Responder: TCP:502
Topic: modtcp
Name: Modbus_IN

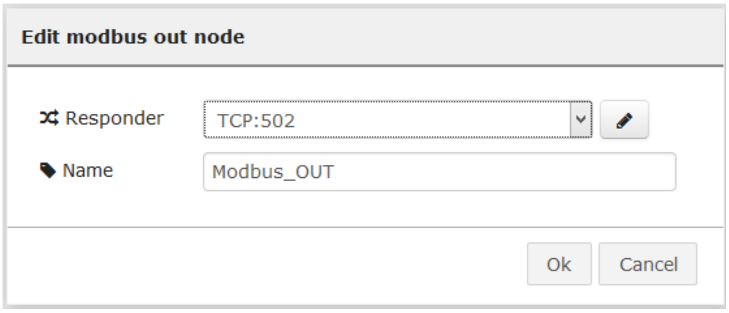
Figure 9: Modbus in node configured.

4. Click on "OK" to close the node configuration.

5. Now, let's add a "modbus out" node and edit its content:

    i. Change the "Responder" to be the same responder we just configured for the Modbus in node.
    ii. Change the Name of the node to "Modbus_OUT".

   The configuration of the "modbus out" node should look like the following:



Edit modbus out node

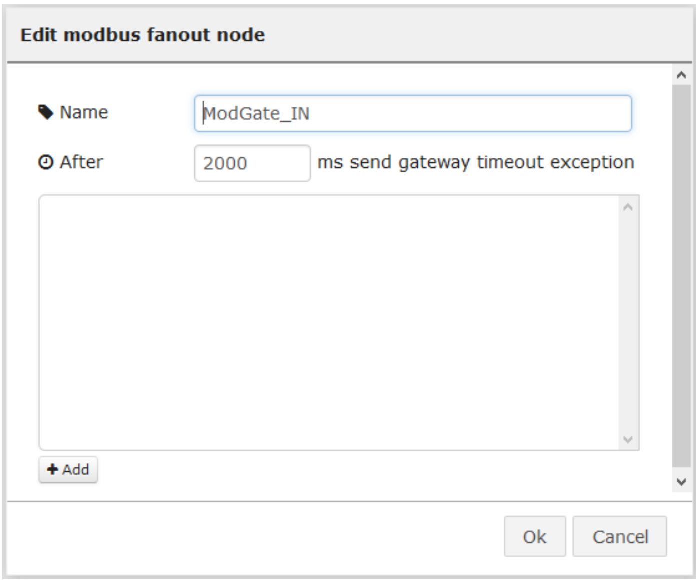
Responder: TCP:502

Name: Modbus_OUT

Figure 10: Modbus out node configured.

6. Click on "OK" to close the node configuration.

7. We need to add two more nodes to make the base skeleton of our ModGate configuration ("modbus fanout" and "modbus fanin"):
    i. Drag and drop them.
    ii. Change their names:

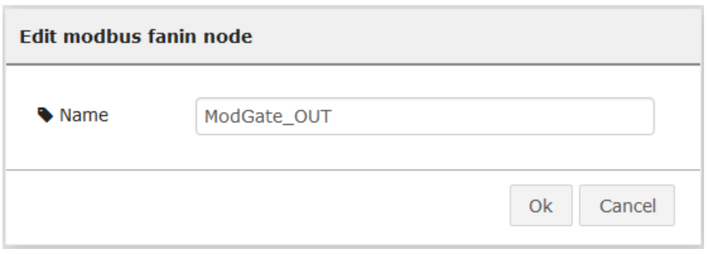


Edit modbus fanout node

Name: ModGate_IN

After 2000 ms send gateway timeout exception

Figure 11: Modbus fanout node configuration.


Edit modbus fanin node

Name: ModGate_OUT

Figure 12: Modbus fanin node configuration.

iii. Connect the four nodes as shown on Figure 13.

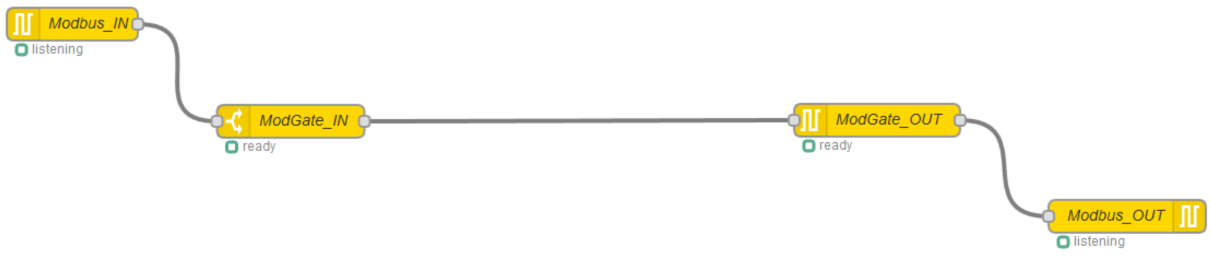
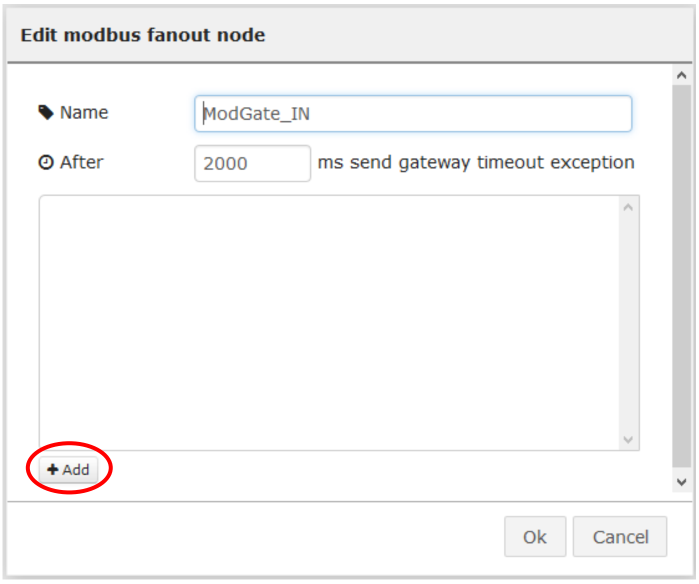


Figure 13: Modbus nodes connected.

2.2 Mapping the Modbus registers and Firewalling

In order to do the mapping explained on Table 3, we need to work on the "ModGate_IN" node (modbus fanout node).

1. Go ahead and press the "Add" button located at the bottom left of the configuration of the "ModGate_IN" node:

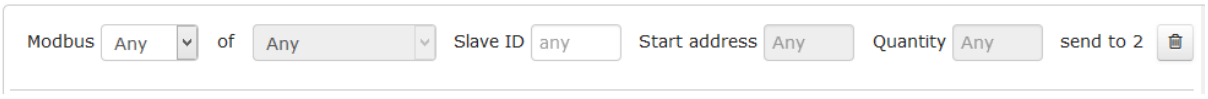

Edit modbus fanout node

Name: ModGate_IN

After: 2000 ms send gateway timeout exception

Figure 14: Add button on ModGate_IN node.

2. The following line will be created on the node:



Modbus Any of Any Slave ID any Start address Any Quantity Any send to 2

3. The items are organized as follows:
   i. It filters/firewalls the action that will be allowed:

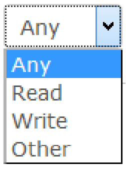


Figure 15: Action allowed on the entry.

   ii. It filters/firewalls the Type of data that will be allowed:

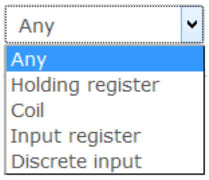


Figure 16: Type of data on the entry.

   iii. Slave ID: is the ID that is received on the Modbus TCP request. For our case, this ID is the ModGate ID.
   iv. Start address: Is the register numer requested and received on the Modbus TCP request.
   v. Quantity: Is the amount of registers that are readed.
   vi. Send to #: Is the output in which the original request is going to be forward.

The following table helps to translate Table 3 into the above configuration:

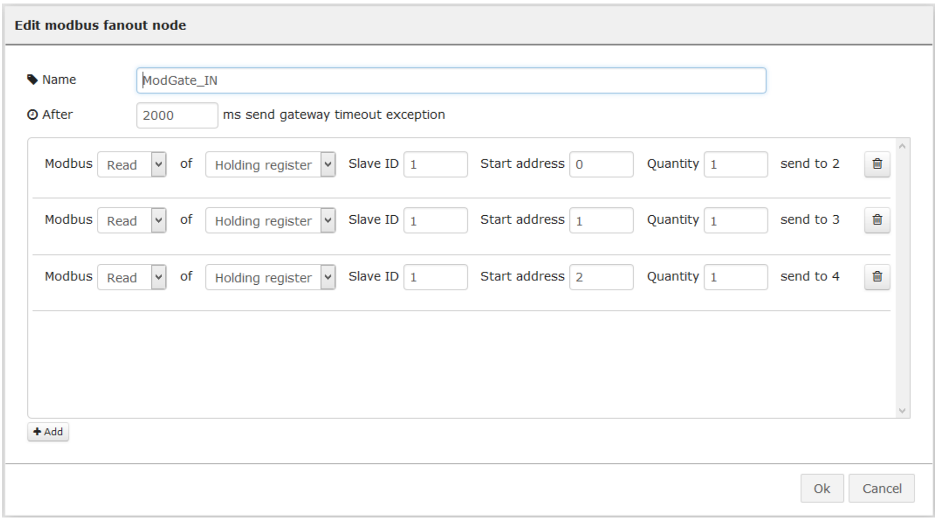
Action	Type of Data	Slave ID	Start address	Quantity
Read	Holding register	1	1	1
Read	Holding register	1	2	1
Read	Holding register	1	3	1

Table 4: Register mapping into Modbus fanout.

- Slave ID is 1, because our ModGate ID is 1.
- The Start address is basically the ModGate register numbers.
- Quantity is 1, because the values that we are getting from the physical devices are 1 holding register each.

NOTE: Keep in mind that registers actually start counting at 0 (zero), so each register number will be the number in the above table - 1.

If configured correctly, the following configuration should be in the node:


Edit modbus fanout node

Name: ModGate_IN

After: 2000 ms send gateway timeout exception

Modbus	Read	of	Holding register	Slave ID	1	Start address	0	Quantity	1	send to 2	
Modbus	Read	of	Holding register	Slave ID	1	Start address	1	Quantity	1	send to 3	
Modbus	Read	of	Holding register	Slave ID	1	Start address	2	Quantity	1	send to 4	

Figure 17: ModGate_IN mapping table.

4. Click on OK and see the "ModGate_IN" node again, it should look like this:

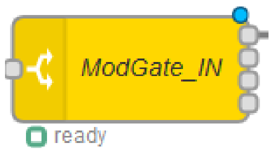


Figure 18: ModGate_IN node with multiple outputs.

This is simply because we have been adding outputs (send to #) to the node each time a new Modbus line was added into the mapping table, this actually means that each of the different requests are going to routed to the right output for processing.

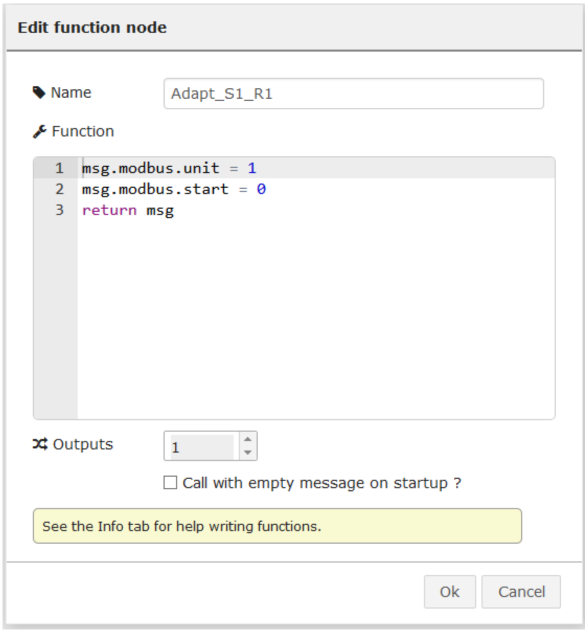
2.3 Changing the slave ID and register number

So far we are able to receive a request, we pass it to the ModGate_IN node and we route it to the right exist of the node, but we have not manipulated the content of the request to adapt it to our needs. It is still for Slave ID 1 and register 1, 2 or 3!

To do this adaptation we need to modify two items on the request:

- Slave ID: it is located in the message under ***msg.modbus.unit***
- Register numer: is located in the message under ***msg.modbus.start***

1. Let's drop a function node to do this conversion:
   i. Change the node name and include following code in the node:


Edit function node

Name: Adapt_S1_R1

Function

```
msg.modbus.unit = 1
msg.modbus.start = 0
return msg
```

Outputs: 1

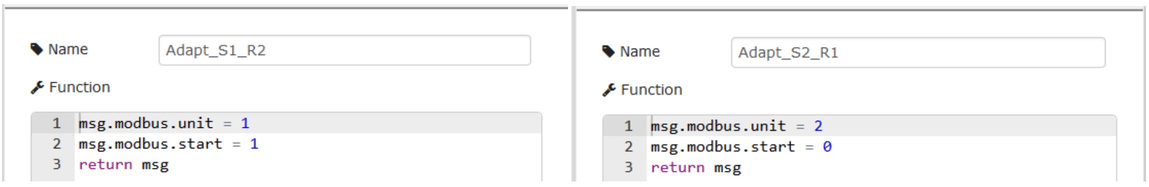
☐ Call with empty message on startup ?

See the Info tab for help writing functions.

Figure 19: Function node to Adapt Modbus request.

We are basically changing the slave ID to 1 and changing the register to be the first register (zero).

2. Add two more function nodes to change the unit and start values for the other two registers following the same idea as before:


Name: Adapt_S1_R2

Function

```
msg.modbus.unit = 1
msg.modbus.start = 1
return msg
```

Name: Adapt_S2_R1

Function

```
msg.modbus.unit = 2
msg.modbus.start = 0
return msg
```

Figure 20: New function nodes.

3. Connect the three nodes to the "ModGate_IN" node the following way:

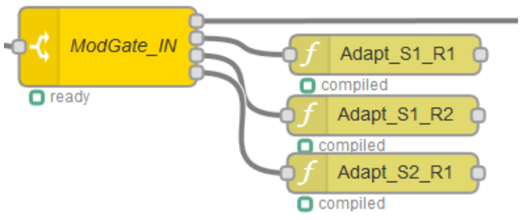
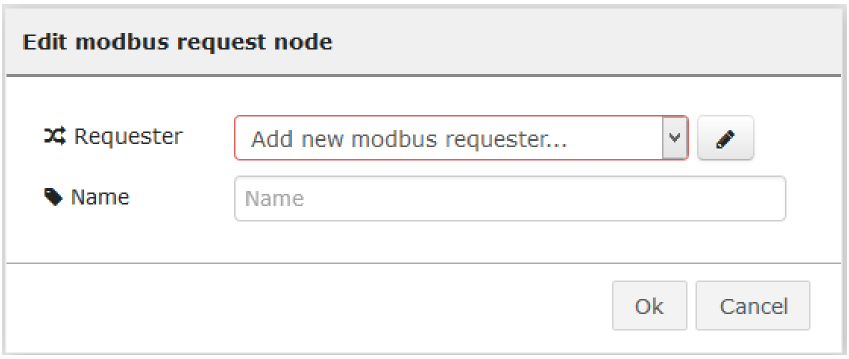


Figure 21: ModGate_IN node connected to function nodes.

2.4 Adding the requesters

So far we are still missing one of the five modbus nodes mentioned earlier in this section, but it is time to add it now:

1. Drag and Drop a "modbus requester" node into the editor.
2. Open it to edit its configuration.



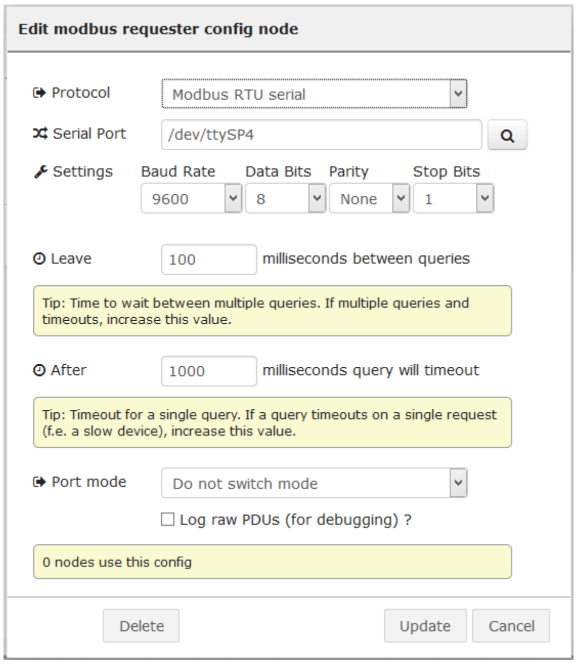
Edit modbus request node

Requester: Add new modbus requester...

Name: Name

Figure 22: Modbus requester node configuration.

3. Click on the pencil button and edit the serial settings according to the specifications on Table 2 (/dev/ttySP4 = RS485):



Edit modbus requester config node

Protocol: Modbus RTU serial

Serial Port: /dev/ttySP4

Settings: Baud Rate 9600, Data Bits 8, Parity None, Stop Bits 1

Leave: 100 milliseconds between queries

Tip: Time to wait between multiple queries. If multiple queries and timeouts, increase this value.

After: 1000 milliseconds query will timeout

Tip: Timeout for a single query. If a query timeouts on a single request (f.e. a slow device), increase this value.

Port mode: Do not switch mode

☐ Log raw PDUs (for debugging) ?

0 nodes use this config

Figure 23: Serial port configuration.

4. Click on "Update".
5. Change the name of the node to "Requester_1".

6. Connect it to the first function node and to the ModGate_OUT as follows:

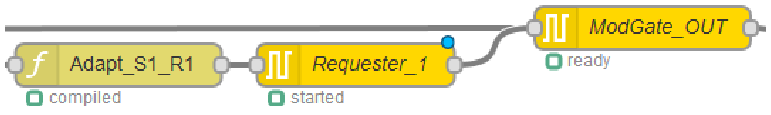


Figure 24: Requester_1 connection.

7. Do two copies of "Requester_1", change their names and connect them to the other two function nodes as follows:

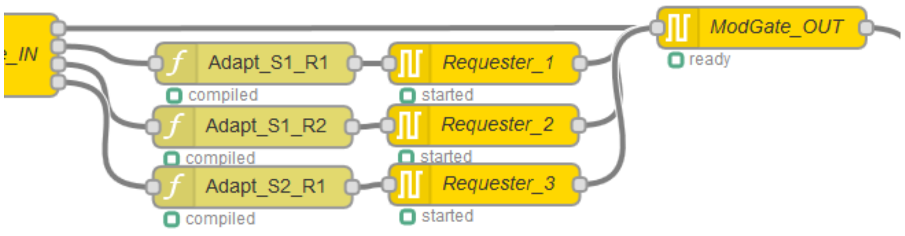


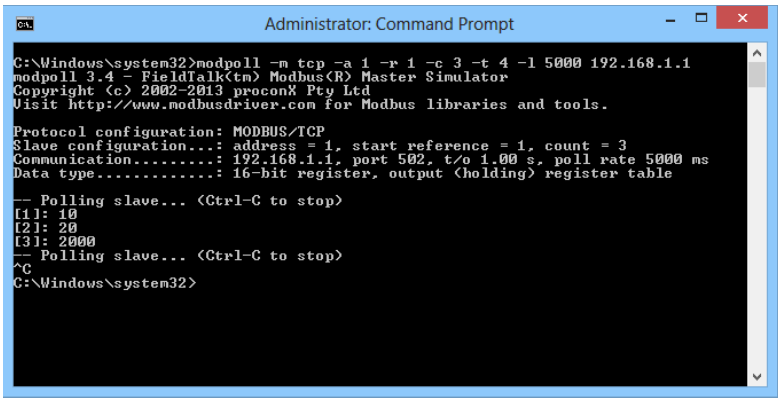
Figure 25: All requesters connected.

8. Click on "Deploy" to apply the configuration.

To test this setup we are going to use a windows tool called modpoll.exe (http://www.modbusdriver.com/modpoll.html), this tool is acting as a Modbus master to connect to ModGate. The command used is:

modpoll.exe -m tcp -a 1 -r 1 -c 3 -t 4 -l 5000 192.168.1.1

- **-m tcp**: use Modbus TCP
- **-a 1**: Connect to Slave ID 1
- **-r 1**: Start at register 1
- **-c 3**: Read 3 registers
- **-t 4**: interpret the data as 16 bit holding register values
- **-l 5000**: Poll rate to 5 seconds
- **192.168.1.1**: Local IP of the CloudGate/ModGate



```
C:\Windows\system32>modpoll -m tcp -a 1 -r 1 -c 3 -t 4 -l 5000 192.168.1.1
modpoll 3.4 - FieldTalk(tm) Modbus(R) Master Simulator
Copyright (c) 2002-2013 proconX Pty Ltd
Visit http://www.modbusdriver.com for Modbus libraries and tools.

Protocol configuration: MODBUS/TCP
Slave configuration...: address = 1, start reference = 1, count = 3
Communication.........: 192.168.1.1, port 502, t/o 1.00 s, poll rate 5000 ms
Data type.............: 16-bit register, output (holding) register table

-- Polling slave... (Ctrl-C to stop)
[1]: 10
[2]: 20
[3]: 2000
-- Polling slave... (Ctrl-C to stop)
^C
C:\Windows\system32>
```

Figure 26: Modpoll Modbus master.

3 Adding a new Modbus TCP slave

We are currently aggregating data from two different Modbus RTU serial devices, but what happens if we want to contact an extra Modbus device, but this time a Modbus TCP device?

Let say that we just added a third Modbus device to the equation and that our scenario looks like this now:

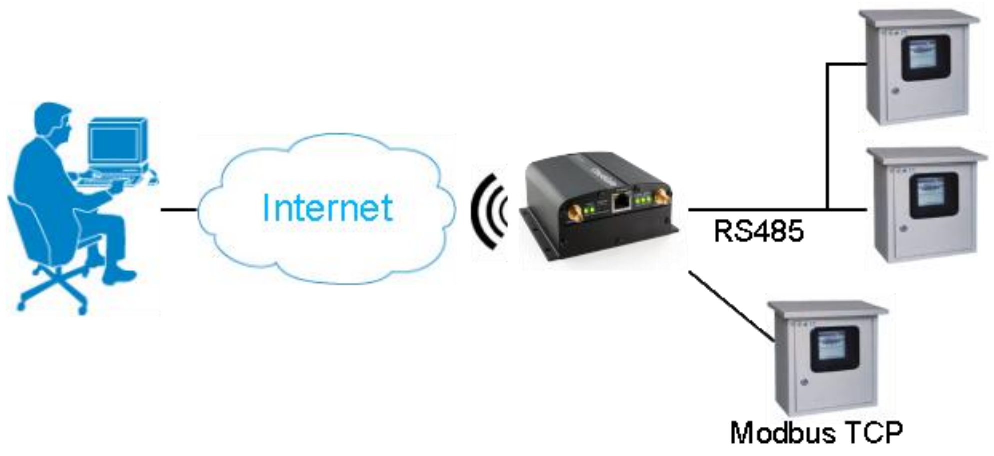


Figure 27: ModGate application - Modbus TCP added.

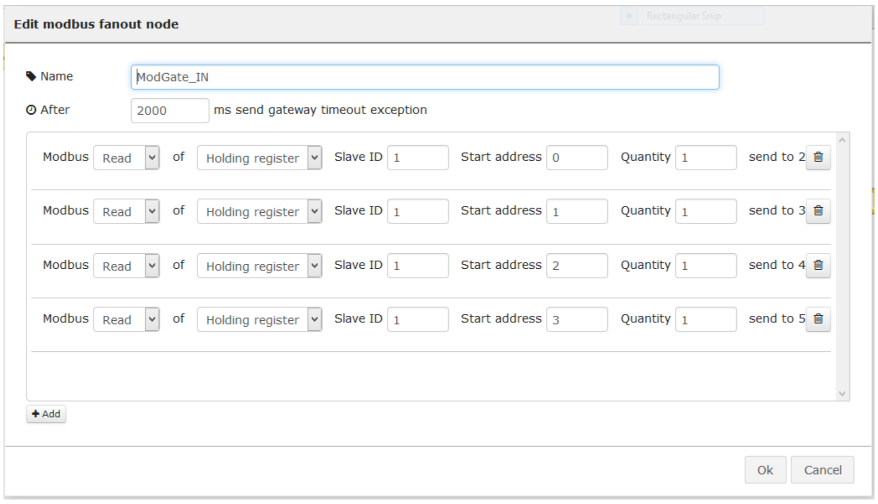
This new device is SlaveID 3 and the register we want to read is register 1 (holding register). Our mapping table will now look like this:

ModGate ID	ModGate Register	Slave ID	Holding Register	Contained Value
1	1	1	1	10
1	2	1	2	20
1	3	2	1	2000
1	4	3	1	555

Table 5: New register mapping.

These are the step to follow:

1. Add a new entry on the "ModGate_IN" node according to the values shown on the last line of the above table (This change will add an extra output on the node):



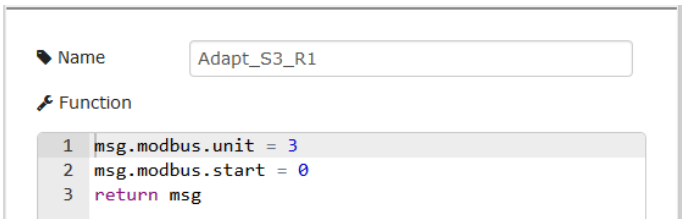
Edit modbus fanout node

Name: ModGate_IN
After: 2000 ms send gateway timeout exception

Modbus	Read	of	Holding register	Slave ID	1	Start address	0	Quantity	1	send to	2
Modbus	Read	of	Holding register	Slave ID	1	Start address	1	Quantity	1	send to	3
Modbus	Read	of	Holding register	Slave ID	1	Start address	2	Quantity	1	send to	4
Modbus	Read	of	Holding register	Slave ID	1	Start address	3	Quantity	1	send to	5

Figure 28: New entry on ModGate_IN node.

2. Add a new function and adapt the name and code inside to talk to Slave ID 3 and register 1:



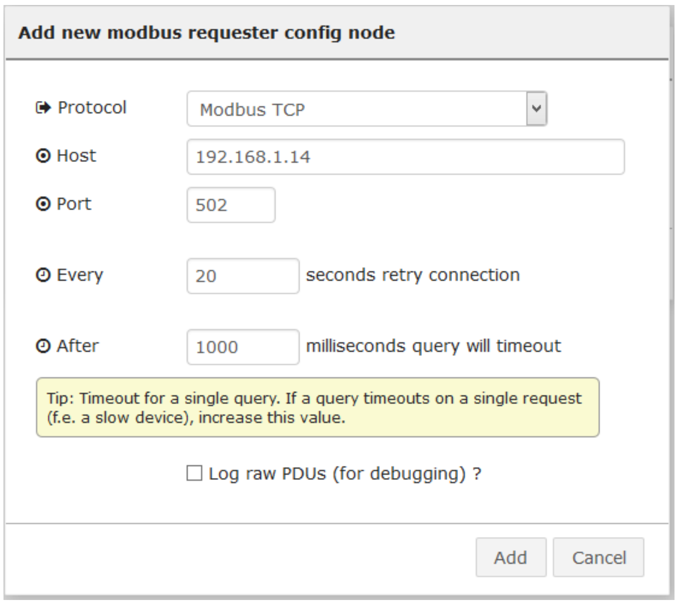
Name: Adapt_S3_R1

Function

```
msg.modbus.unit = 3
msg.modbus.start = 0
return msg
```

Figure 29: New function node.

3. Add a new requester and name it, but this time add a **new** requester and configure it for Modbus TCP. You will need to know the IP address and port (502 by default) of the Slave device. In our case the IP is 192.168.1.14:


Add new modbus requester config node

Protocol: Modbus TCP

Host: 192.168.1.14

Port: 502

Every: 20 seconds retry connection

After: 1000 milliseconds query will timeout

Tip: Timeout for a single query. If a query timeouts on a single request (f.e. a slow device), increase this value.

☐ Log raw PDUs (for debugging) ?

Figure 30: Modbus TCP requester.

4. Connect fifth output of the ModGate_IN node to the function node, then function node to the new requester node and the requester node to ModGate_OUT node and click on "Deploy":

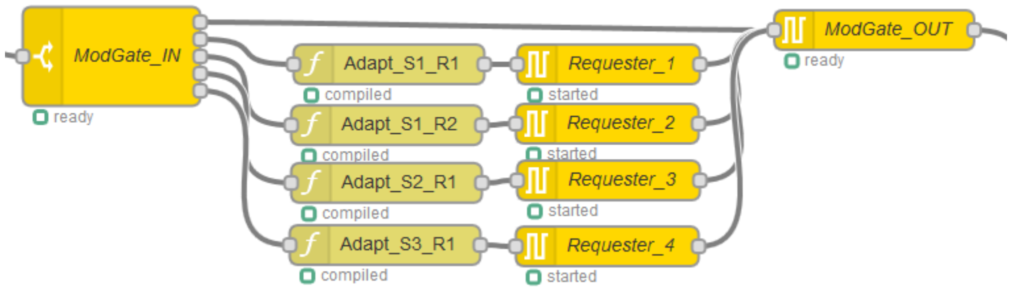


Figure 31: New nodes connected.

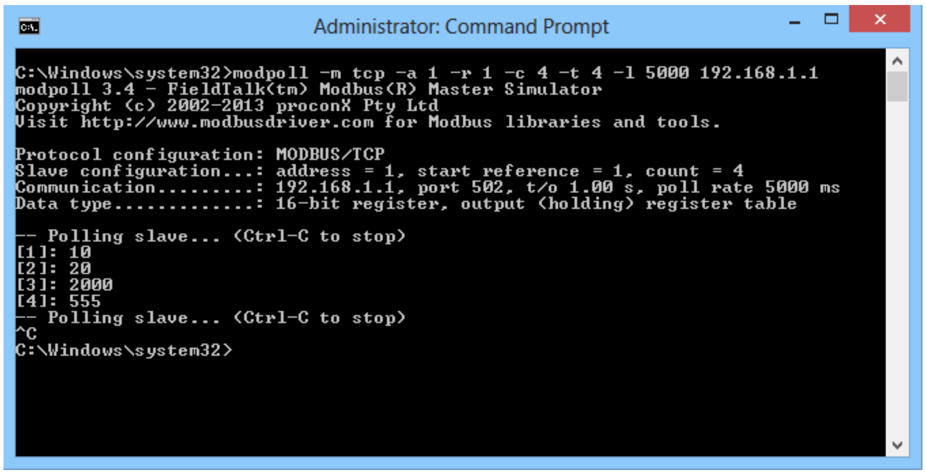
To test this setup we are going to use modpoll.exe again. The command used is:

modpoll.exe -m tcp -a 1 -r 1 -c 4 -t 4 -l 5000 192.168.1.1

We are now reading 4 registers instead of 3:

- **-m tcp**: use Modbus TCP
- **-a 1**: Connect to Slave ID 1
- **-r 1**: Start at register 1
- **-c 4**: Read **4 registers**
- **-t 4**: interpret the data as 16 bit holding register values
- **-l 5000**: Poll rate to 5 seconds

- **192.168.1.1**: Local IP of the CloudGate/ModGate



```
C:\Windows\system32>modpoll -m tcp -a 1 -r 1 -c 4 -t 4 -l 5000 192.168.1.1
modpoll 3.4 - FieldTalk(tm) Modbus(R) Master Simulator
Copyright (c) 2002-2013 proconX Pty Ltd
Visit http://www.modbusdriver.com for Modbus libraries and tools.

Protocol configuration: MODBUS/TCP
Slave configuration...: address = 1, start reference = 1, count = 4
Communication.........: 192.168.1.1, port 502, t/o 1.00 s, poll rate 5000 ms
Data type.............: 16-bit register, output (holding) register table

-- Polling slave... (Ctrl-C to stop)
[1]: 10
[2]: 20
[3]: 2000
[4]: 555
-- Polling slave... (Ctrl-C to stop)
^C
C:\Windows\system32>
```

Figure 32: Modpoll Modbus master.

4 Adding local values (16 bits)

So far we are communicating with three different Modbus devices using either Modbus RTU or Modbus TCP, but what happens if we want to add a value that is local to the CloudGate, for example the signal strength of the cellular interface?

The process is very similar as the one described above with few differences:

- No need to use a modbus request node since we are not requesting values from an external device.
- We need to use a node to obtain the signal strength: "siglevel" node

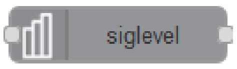


Figure 33: Siglevel node.

- We need to encode the response value properly before sending it to the "modbus fanin" node: "modbus encode" node

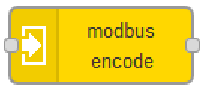


Figure 34: Modbus encode node.

First we need to take a look at the actual value we are going to add into our configuration:

- The signal strength is a negative value.
- The signal strength is a value between 0 and -113 (approximately).

Taking into account the above two items, we should be able to encode this negative value in a 16 bit register, so we can use a Modbus Holding register for this.

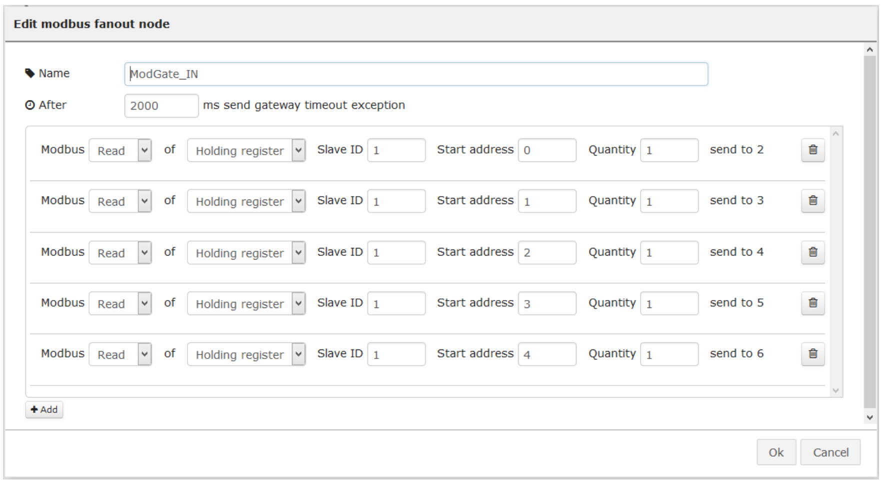
Our mapping table will now look like this:

ModGate ID	ModGate Register	Slave ID	Holding Register	Contained Value
1	1	1	1	10
1	2	1	2	20
1	3	2	1	2000
1	4	3	1	555
1	5	-	-	Signal Strength

Table 6: New register mapping.

These are the step to follow:

1. Add a new entry on the "ModGate_IN" node according to the values shown on the last line of the above table (This change will add an extra output on the node):



Edit modbus fanout node

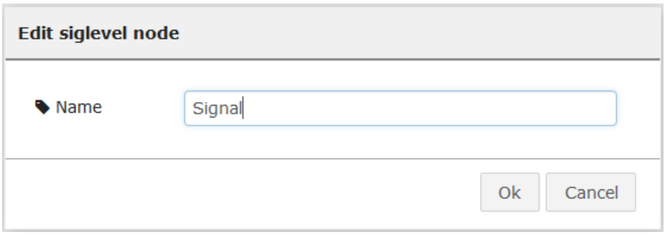
Name: ModGate_IN

After: 2000 ms send gateway timeout exception

Modbus	Read	of	Holding register	Slave ID	1	Start address	0	Quantity	1	send to 2	🗑
Modbus	Read	of	Holding register	Slave ID	1	Start address	1	Quantity	1	send to 3	🗑
Modbus	Read	of	Holding register	Slave ID	1	Start address	2	Quantity	1	send to 4	🗑
Modbus	Read	of	Holding register	Slave ID	1	Start address	3	Quantity	1	send to 5	🗑
Modbus	Read	of	Holding register	Slave ID	1	Start address	4	Quantity	1	send to 6	🗑

Figure 35: New entry on ModGate_IN node.

2. Add a "siglevel" node and change its name:

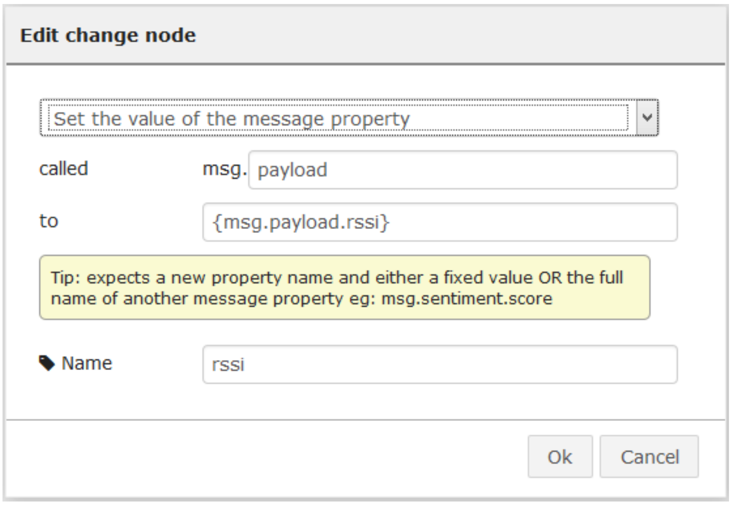


Edit siglevel node

Name: Signal

Figure 36: Siglevel node configuration.

3. Add a "change" node and configure it as shown on Figure 37:


Edit change node

Set the value of the message property

called msg. payload

to {msg.payload.rssi}

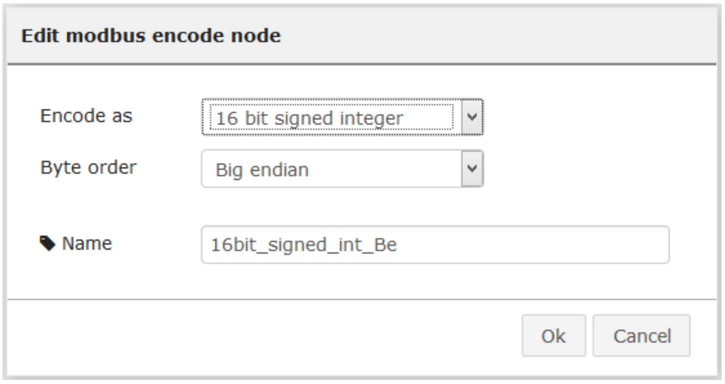
Tip: expects a new property name and either a fixed value OR the full name of another message property eg: msg.sentiment.score

Name rssi

Figure 37: Change node configuration.

- Since the siglevel node includes not only the rssi, but the ecio, we need to change the payload to only contain the rssi.
- We are setting the ***msg.payload*** to ***msg.payload.rssi***
- We are changing the name of the node.

4. Add a "modbus encode" node and configure it as shown on Figure 38:


Edit modbus encode node

Encode as 16 bit signed integer

Byte order Big endian

Name 16bit_signed_int_Be

Figure 38: Modbus encode node configuration.

- We are Encoding the value as a 16 bit **signed** integer according our evaluation made earlier over the value.
- We are keeping the Big endian Byte order.
- We are changing the name of the node.

5. Connect the nodes to the ModGate_IN and ModGate_OUT nodes as shown below and then click on "Deploy":

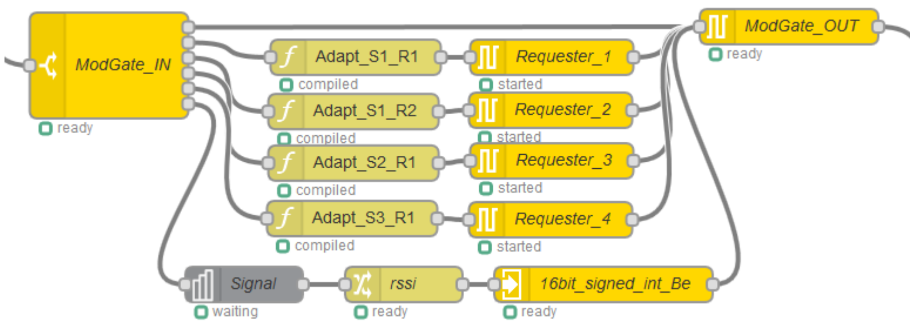


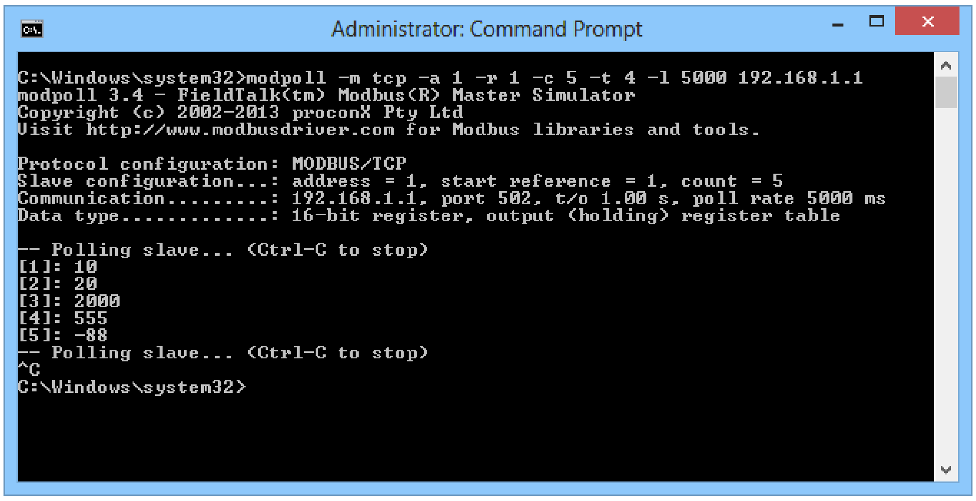
Figure 39: New nodes connected.

To test this setup we are going to use modpoll.exe again. The command used is:

modpoll.exe -m tcp -a 1 -r 1 -c 5 -t 4 -l 5000 192.168.1.1

We are now reading 5 registers instead of 4:

- **-m tcp**: use Modbus TCP
- **-a 1**: Connect to Slave ID 1
- **-r 1**: Start at register 1
- **-c 5**: Read **5 registers**
- **-t 4**: interpret the data as 16 bit holding register values
- **-l 5000**: Poll rate to 5 seconds
- **192.168.1.1**: Local IP of the CloudGate/ModGate


Administrator: Command Prompt

```
C:\Windows\system32>modpoll -m tcp -a 1 -r 1 -c 5 -t 4 -l 5000 192.168.1.1
modpoll 3.4 - FieldTalk(tm) Modbus(R) Master Simulator
Copyright (c) 2002-2013 proconX Pty Ltd
Visit http://www.modbusdriver.com for Modbus libraries and tools.

Protocol configuration: MODBUS/TCP
Slave configuration...: address = 1, start reference = 1, count = 5
Communication.........: 192.168.1.1, port 502, t/o 1.00 s, poll rate 5000 ms
Data type.............: 16-bit register, output (holding) register table

-- Polling slave... (Ctrl-C to stop)
[1]: 10
[2]: 20
[3]: 2000
[4]: 555
[5]: -88
-- Polling slave... (Ctrl-C to stop)
^C
C:\Windows\system32>
```

Figure 40: Modpoll Modbus master.

5 Adding local values (32 bits)

On section 4 we explained how to add a 16 bit value, more specifically a 16 bit signed value. In this case we want to add a value that is a signed floating point value such as the latitude, longitude and altitude.

The process is almost the same as the one explained on the previous section with few differences:

- Instead of a siglevel node, we need a "GPS" node:

Figure 41: GPS node.

- We will need to store the GPS data in a global variable in order to retrieve it upon request. There are few ways of doing this, but we are going to use a function node.
- To retrieve the data we need to use a "global retrieve" node:

Figure 42: Global retrieve node.

First we need to take a look at the actual values we are going to add into our configuration:

- The values might be negative depending on the geographical location.
- The values are floating point numbers.

Taking into account the above two items, we should be able to encode this negative value as a 32 bit value using IEEE 754 encoding, so we are going to need two Modbus Holding registers per value.

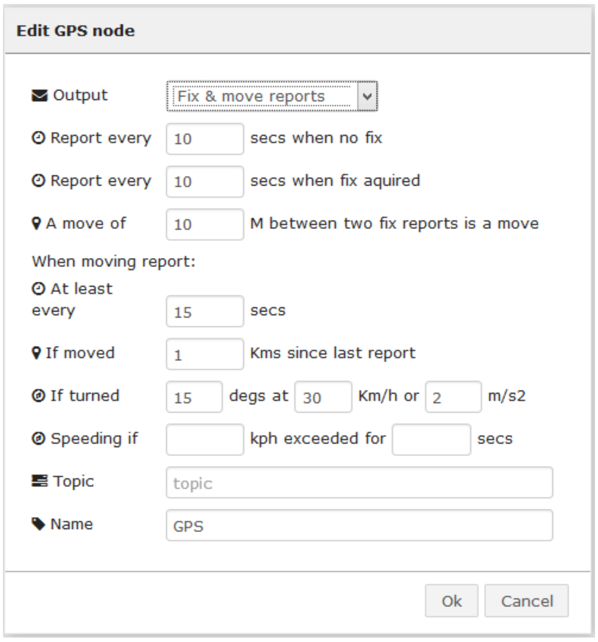
Our mapping table will now look like this:

ModGate ID	ModGate Registers	Slave ID	Holding Register	Contained Value
1	1	1	1	10
1	2	1	2	20
1	3	2	1	2000
1	4	3	1	555
1	5	-	-	Signal Strength
1	6 & 7	-	-	Latitude
1	8 & 9	-	-	Longitude
1	10 & 11	-	-	Altitude

Table 7: New register mapping.

These are the step to follow:

1. First we need to store the GPS data in a global variable in order to make it available upon request:
   i. Add a new GPS node and configure it as follows:

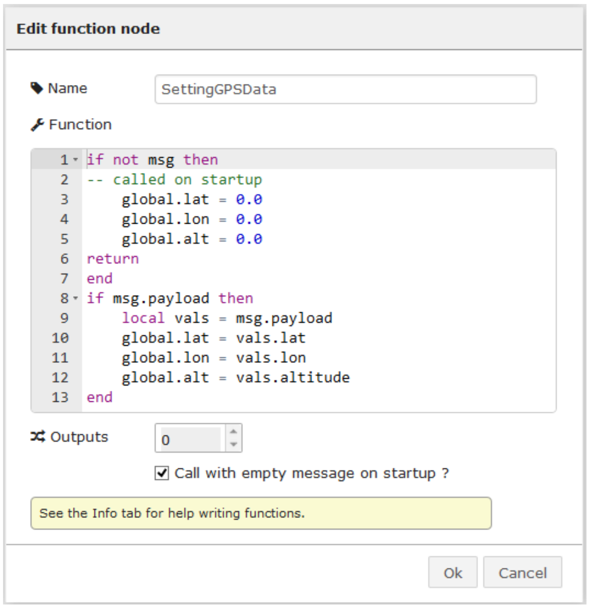


Edit GPS node

Output	Fix & move reports
Report every	10 secs when no fix
Report every	10 secs when fix aquired
A move of	10 M between two fix reports is a move
When moving report:	
At least every	15 secs
If moved	1 Kms since last report
If turned	15 degs at 30 Km/h or 2 m/s2
Speeding if	kph exceeded for secs
Topic	topic
Name	GPS

Figure 43: GPS node configuration.

   ii. Add a new function node and configure it as follows:


Edit function node

Name: SettingGPSData

Function

```
if not msg then
-- called on startup
    global.lat = 0.0
    global.lon = 0.0
    global.alt = 0.0
return
end
if msg.payload then
    local vals = msg.payload
    global.lat = vals.lat
    global.lon = vals.lon
    global.alt = vals.altitude
end
```

Outputs: 0

☑ Call with empty message on startup ?

See the Info tab for help writing functions.

Figure 44: Function node configuration.

NOTE1: The global variables are called lat, lon and alt (set to ZERO at startup)

NOTE2: Check the Call with empty message on startup checkbox!

iii. Connect both nodes together, leave them separated from the main ModGate skelethon and click on "Deploy":

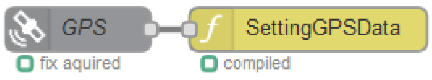
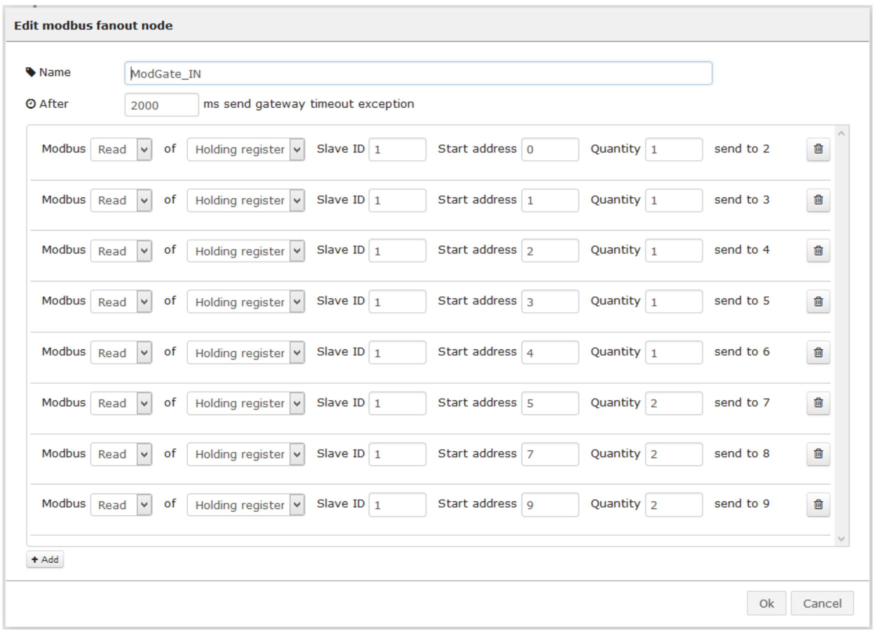


Figure 45: GPS data extraction.

2. Add three new entries on the "ModGate_IN" node according to the values shown on the last three lines of Table 7 (This change will add three extra outputs on the node):


Edit modbus fanout node

Name: ModGate_IN

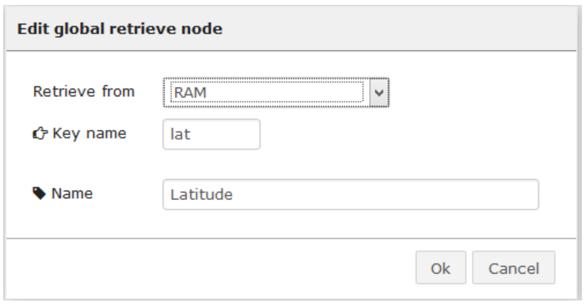
After: 2000 ms send gateway timeout exception

Modbus	Read	of	Holding register	Slave ID	1	Start address	0	Quantity	1	send to 2
Modbus	Read	of	Holding register	Slave ID	1	Start address	1	Quantity	1	send to 3
Modbus	Read	of	Holding register	Slave ID	1	Start address	2	Quantity	1	send to 4
Modbus	Read	of	Holding register	Slave ID	1	Start address	3	Quantity	1	send to 5
Modbus	Read	of	Holding register	Slave ID	1	Start address	4	Quantity	1	send to 6
Modbus	Read	of	Holding register	Slave ID	1	Start address	5	Quantity	2	send to 7
Modbus	Read	of	Holding register	Slave ID	1	Start address	7	Quantity	2	send to 8
Modbus	Read	of	Holding register	Slave ID	1	Start address	9	Quantity	2	send to 9

Figure 46: New entries on ModGate_IN node.

NOTE: Be aware that the three new entries need to have a Quantity of 2 registers and therefore the Start address needs to reflect this jump. The start addresses are 5, 7, and 9 in this case each with a Quantity of 2 instead of 1.

3. Add three "global retrieve" nodes and configure them as shown on Figure 47, Figure 48 and Figure 49:

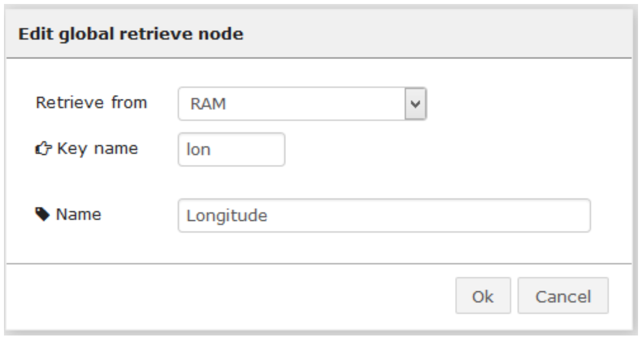

Edit global retrieve node

Retrieve from: RAM

Key name: lat

Name: Latitude

Figure 47: Global retrieve node configuration (latitude).

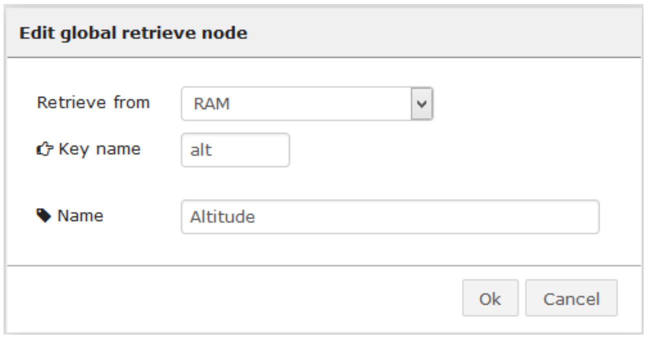

Edit global retrieve node

Retrieve from: RAM

Key name: lon

Name: Longitude

Figure 48: Global retrieve node configuration (longitude).


Edit global retrieve node

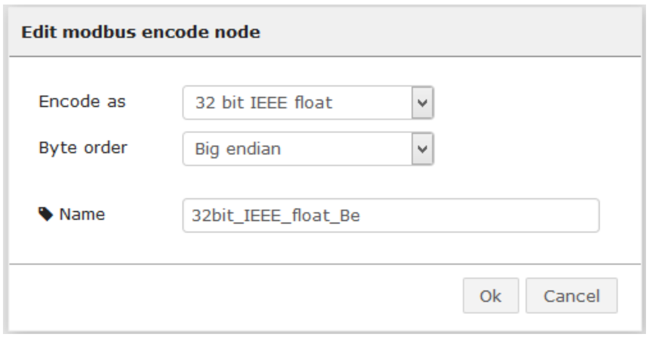
Retrieve from: RAM

Key name: alt

Name: Altitude

Figure 49: Global retrieve node configuration (altitude).

4. Add a "modbus encode" node and configure it as shown on Figure 50:


Edit modbus encode node

Encode as: 32 bit IEEE float

Byte order: Big endian

Name: 32bit_IEEE_float_Be

Figure 50: Modbus encode node configuration.

- We are Encoding the value as a 32 bit **IEEE float** according our evaluation made earlier over the value.
- We are keeping the Big endian Byte order.
- We are changing the name of the node.

5. Connect the nodes to the ModGate_IN and ModGate_OUT nodes as shown below and then click on "Deploy":

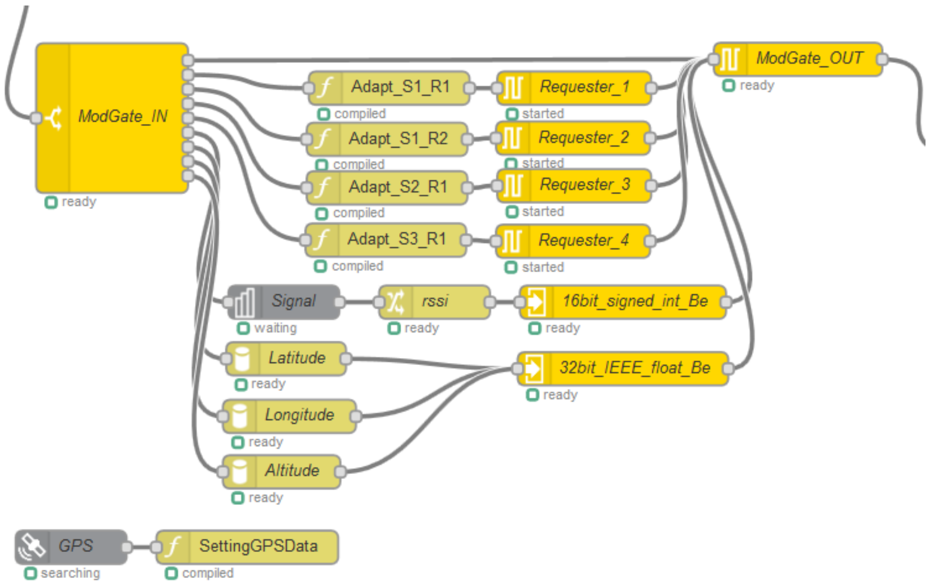


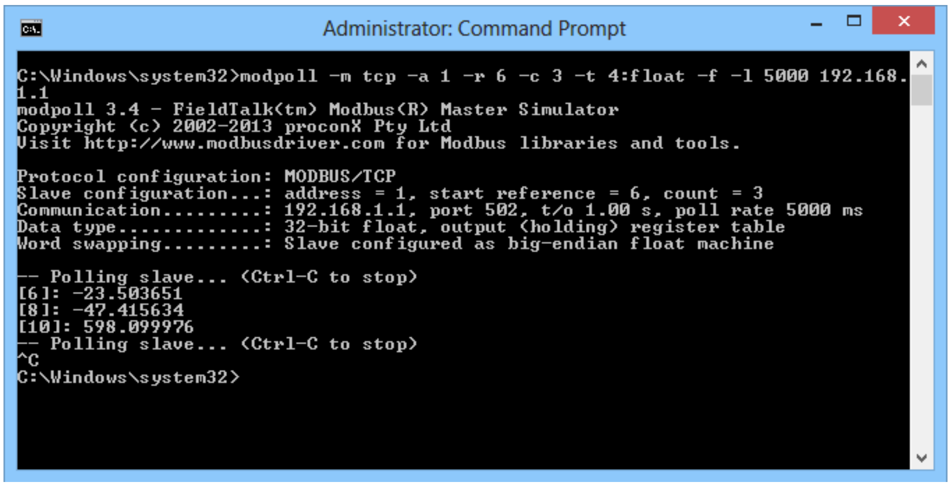
Figure 51: New nodes connected.

To test this setup we are going to use modpoll.exe again. The command used is:

modpoll.exe -m tcp -a 1 -r 6 -c 3 -t 4:float -f -l 5000 192.168.1.1

We are now reading only the new registers instead of all. This is because the modpoll.exe tool cannot retrieve values of different types (16 bit integers vs 32 bit floats) at the same time:

- **-m tcp**: use Modbus TCP
- **-a 1**: Connect to Slave ID 1
- **-r 6**: Start at register 6
- **-c 3**: Read **3 registers**
- **-t 4:float**: interpret the data as 32 bit float data type in a holding register table
- **-f:** Slave operates on big-endian 32-bit floats
- **-l 5000**: Poll rate to 5 seconds
- **192.168.1.1**: Local IP of the CloudGate/ModGate

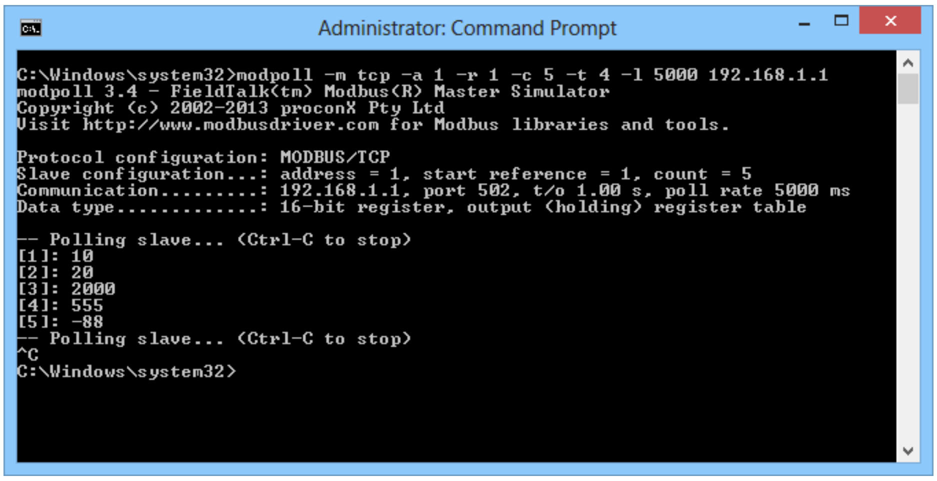


```
C:\Windows\system32>modpoll -m tcp -a 1 -r 6 -c 3 -t 4:float -f -l 5000 192.168.
1.1
modpoll 3.4 - FieldTalk(tm) Modbus(R) Master Simulator
Copyright (c) 2002-2013 proconX Pty Ltd
Visit http://www.modbusdriver.com for Modbus libraries and tools.

Protocol configuration: MODBUS/TCP
Slave configuration...: address = 1, start reference = 6, count = 3
Communication.........: 192.168.1.1, port 502, t/o 1.00 s, poll rate 5000 ms
Data type.............: 32-bit float, output (holding) register table
Word swapping.........: Slave configured as big-endian float machine

-- Polling slave... (Ctrl-C to stop)
[6]: -23.503651
[8]: -47.415634
[10]: 598.099976
-- Polling slave... (Ctrl-C to stop)
^C
C:\Windows\system32>
```

Figure 52: Modpoll Modbus master (32 bit float).

We should still be able to get the values we configured on the previous sections by using the previous modpoll.exe command:

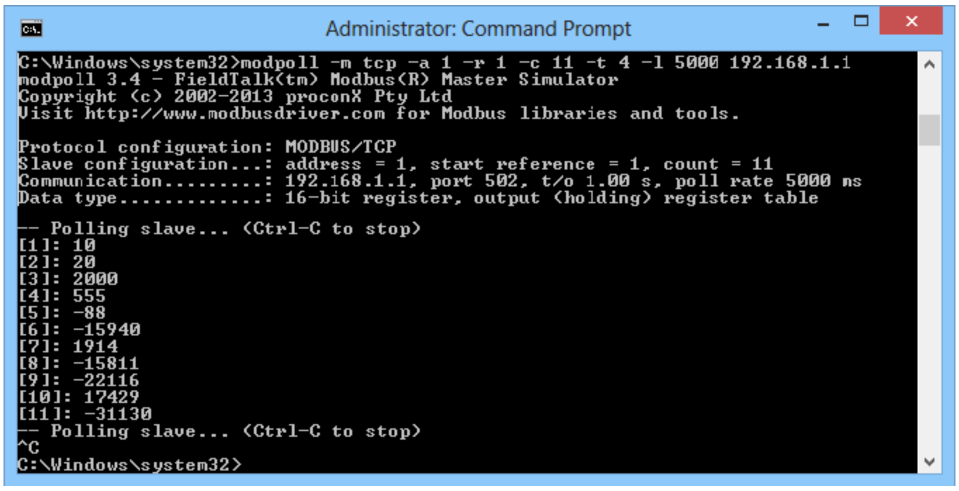
modpoll.exe -m tcp -a 1 -r 1 -c 5 -t 4 -l 5000 192.168.1.1



```
C:\Windows\system32>modpoll -m tcp -a 1 -r 1 -c 5 -t 4 -l 5000 192.168.1.1
modpoll 3.4 - FieldTalk(tm) Modbus(R) Master Simulator
Copyright (c) 2002-2013 proconX Pty Ltd
Visit http://www.modbusdriver.com for Modbus libraries and tools.

Protocol configuration: MODBUS/TCP
Slave configuration...: address = 1, start reference = 1, count = 5
Communication.........: 192.168.1.1, port 502, t/o 1.00 s, poll rate 5000 ms
Data type.............: 16-bit register, output (holding) register table

-- Polling slave... (Ctrl-C to stop)
[1]: 10
[2]: 20
[3]: 2000
[4]: 555
[5]: -88
-- Polling slave... (Ctrl-C to stop)
^C
C:\Windows\system32>
```

Figure 53: Modpoll Modbus master (16 bit integer).

If we execute the above command increasing the amount of registers to read from 5 to 11, we should get all values, but the last six values will be the IEEE encoded values:



```
C:\Windows\system32>modpoll -m tcp -a 1 -r 1 -c 11 -t 4 -1 5000 192.168.1.1
modpoll 3.4 - FieldTalk(tm) Modbus(R) Master Simulator
Copyright (c) 2002-2013 proconX Pty Ltd
Visit http://www.modbusdriver.com for Modbus libraries and tools.

Protocol configuration: MODBUS/TCP
Slave configuration...: address = 1, start reference = 1, count = 11
Communication.........: 192.168.1.1, port 502, t/o 1.00 s, poll rate 5000 ms
Data type.............: 16-bit register, output (holding) register table

-- Polling slave... (Ctrl-C to stop)
[1]: 10
[2]: 20
[3]: 2000
[4]: 555
[5]: -88
[6]: -15940
[7]: 1914
[8]: -15811
[9]: -22116
[10]: 17429
[11]: -31130
-- Polling slave... (Ctrl-C to stop)
^C
C:\Windows\system32>
```

Figure 54: Modpoll Modbus master (all values as 16 bit integers).

OPTION