|  |
| --- |
| Modbus Gateway using the industrial serial card |
|  |





**Franco Arboleda**

**04-Nov-15**

Table of Contents

1 Introduction 3

2 ModGate configuration using the RS485 port. 5

2.1 Buildng the ModGate skeleton 6

2.2 Mapping the Modbus registers and Firewalling 9

2.3 Changing the slave ID and register number 11

2.4 Add the requesters 13

3 Adding a new Modbus TCP slave 15

# 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 : 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:

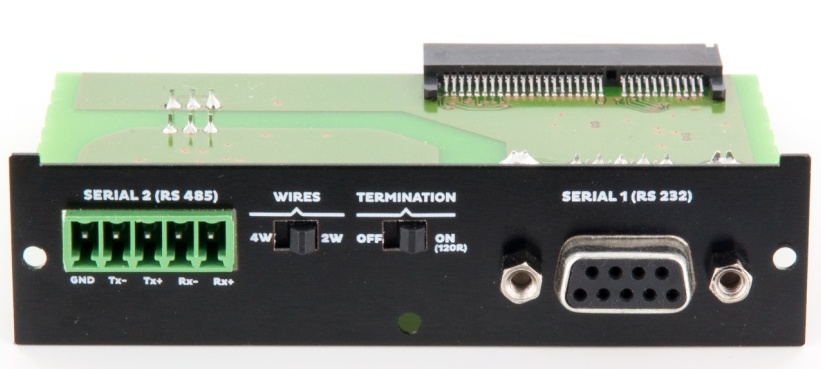


Figure : Industrial Serial Card

* 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.

**Note**: For this configuration, LuvitRED version or higher is required.

# 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 : 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 : Serial port configuration.

We are going to map these registers to out 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 : 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 ) 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 ) 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 fanout" and "modbus fanin" nodes (See Figure 5) are the ones in charge of taking a decision on where to send the incoming package (fanout) and repack the answers into a properly formatted Modbus response (fanin).

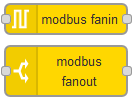


Figure : Modbus fanin and Modbus fanout nodes.

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.



Figure : Modbus request node.

## Buildng the ModGate skeleton

Let's first configure our "modbus in" node. Drag and Drop one "modbus in" node into the LuvitRED editor and double click on it (See Figure 7).

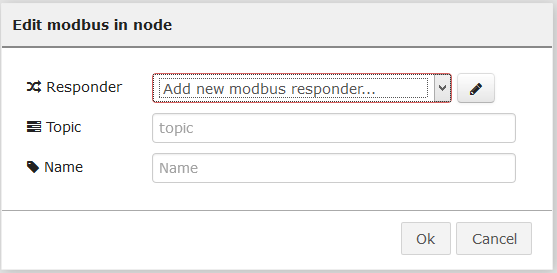


Figure : New modbus in node configuration.

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:

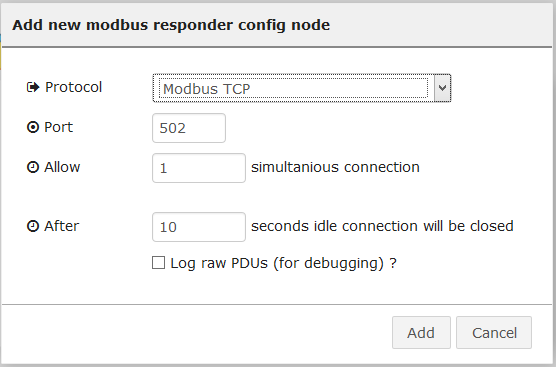


Figure : New responder configuration.

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

1. Protocol: Modbus TCP
2. Port: 502
3. Allow: 1 simultaneous connection
4. 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.

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

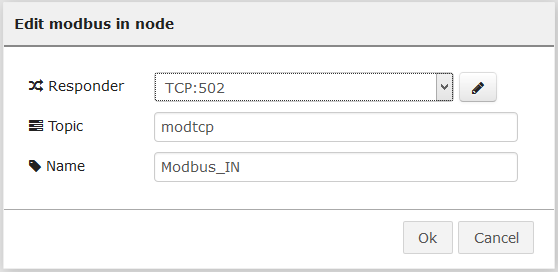


Figure : Modbus in node configured.

Click on "OK" to close the node configuration.

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

1. Change the "Responder" to be the same responder we just configured for the Modbus in node.
2. Change the name of the.

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

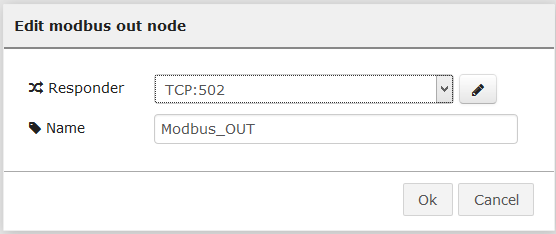


Figure 10: Modbus out node configured.

Click on "OK" to close the node configuration.

We need to drop two more nodes to make the base skeleton of our ModGate configuration: "modbus fanout" and "modbus fanin":

1. Lets drop them
2. Change their names:

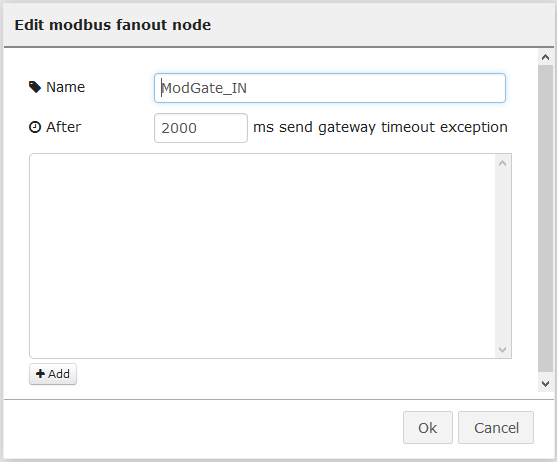


Figure : Modbus fanout node configuration.

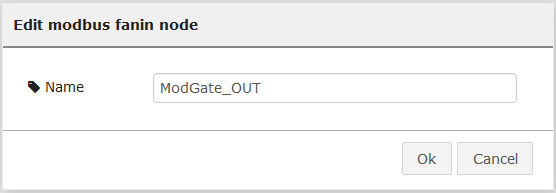


Figure : Modbus fanin node configuration.

1. Connect the four nodes as shown in Figure 13.

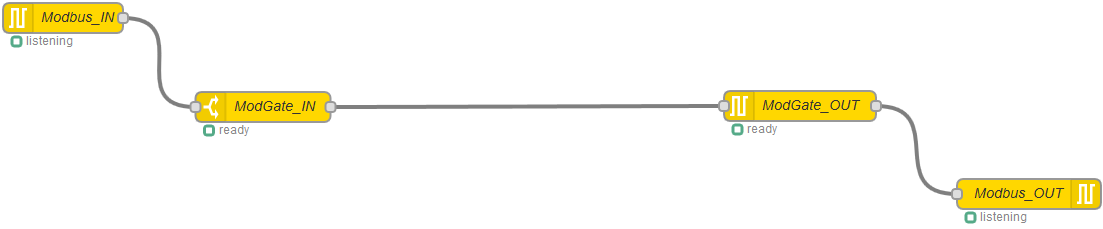


Figure : Modbus nodes connected.

## 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).

Go ahead and press the "Add" button located at the bottom left of the configuration of the "ModGate\_IN" node:

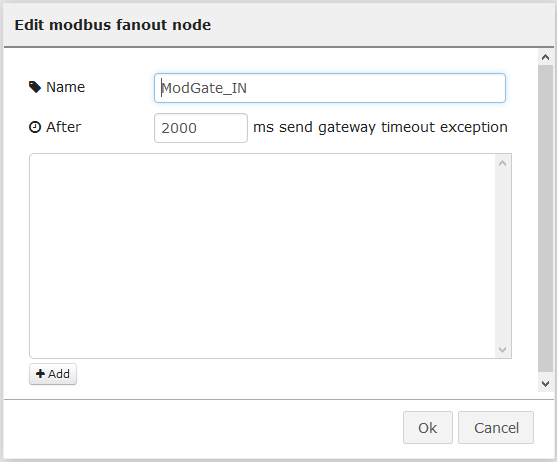


Figure : Add button on ModGate\_IN node.

The following line will be created on the node:



The items are organized as follows:

1. It filters/firewalls the action that will be allowed:

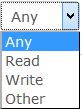


Figure : Action allowed on the entry.

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

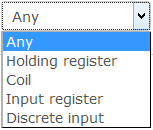


Figure : Type of data on the entry.

1. Slave ID: is the ID that is received on the Modbus TCP request. For our case, this ID is the ModGate ID.
2. Start address: Is the register numer requested and received on the Modbus TCP request.
3. Quantity: Is the amount of registers that are readed.
4. 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 : 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 simply 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 all are configured correctly, the following configuration should be in the node:

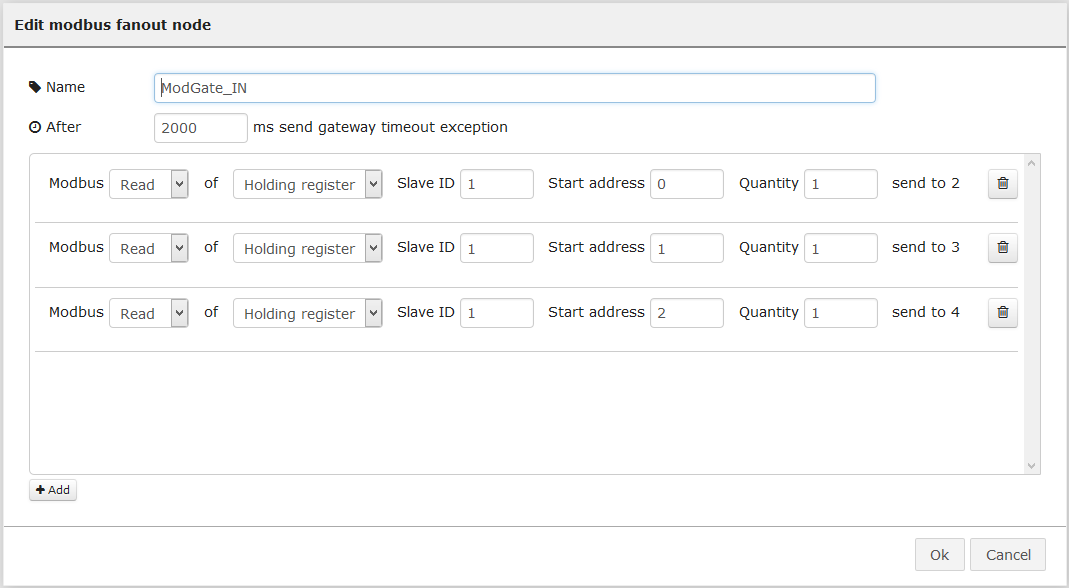


Figure : ModGate\_IN mapping table.

Click on OK and see the "ModGate\_IN" node again, it should look like this:

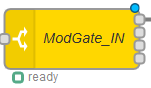


Figure : ModGate\_IN node with multiple outputs.

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.

## 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***

Let's drop a function node to do this conversion:

1. Change the node name and include following code in the node:

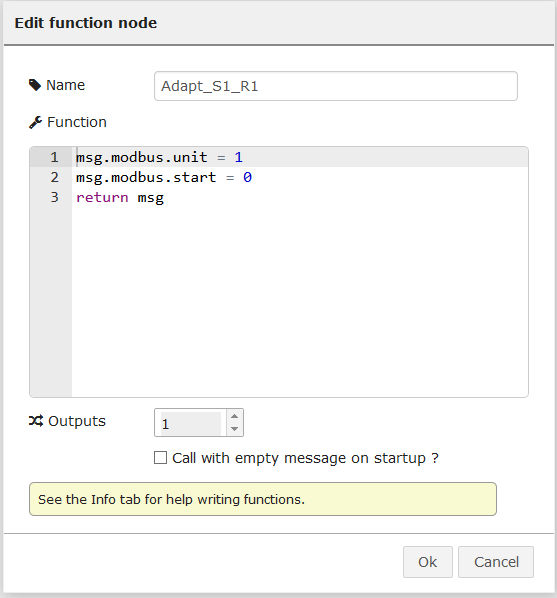


Figure : Function node to Adapt Modbus request.

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

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

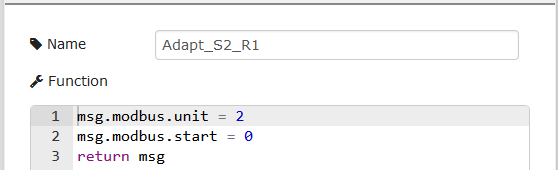
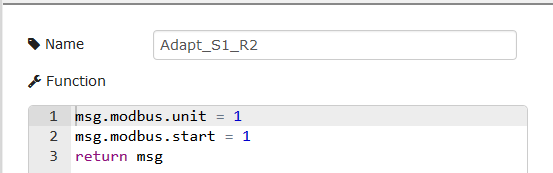


Figure : New function nodes.

1. Connect the three nodes to the "ModGate\_IN" node the following way:

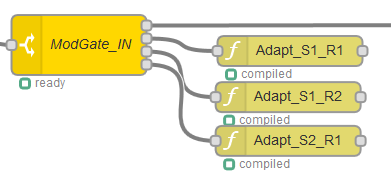


Figure : ModGate\_IN node connected to function nodes.

## Add 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

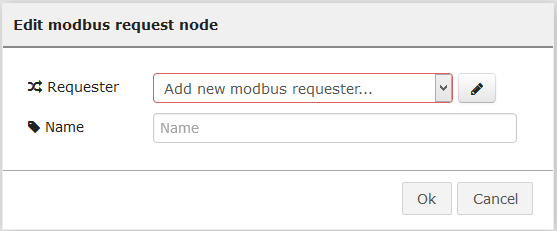


Figure : Modbus requester node configuration.

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

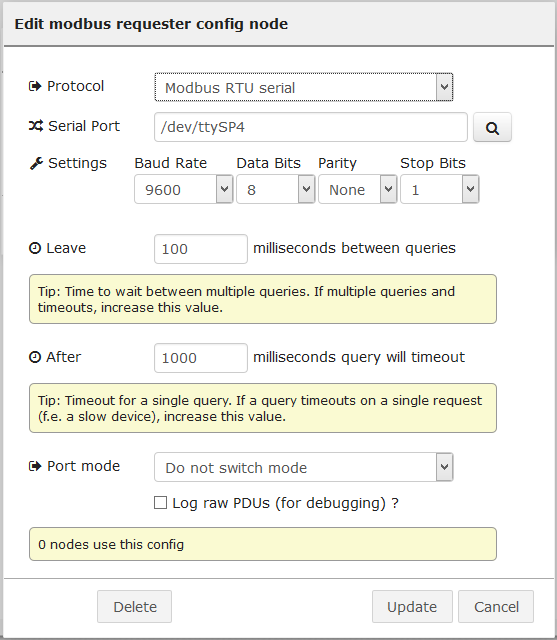


Figure : Serial port configuration.

1. Click on "Update".
2. Change the name of the node to "Requester\_1"
3. Connect it to the first function node and to the ModGate\_OUT as follows:



Figure : Requester\_1 connection.

1. Do two copies of "Requester\_1", change their names and connect them to the other two function nodes as follows:

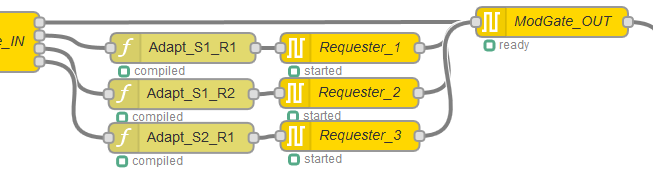


Figure : All requesters connected.

Hi 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

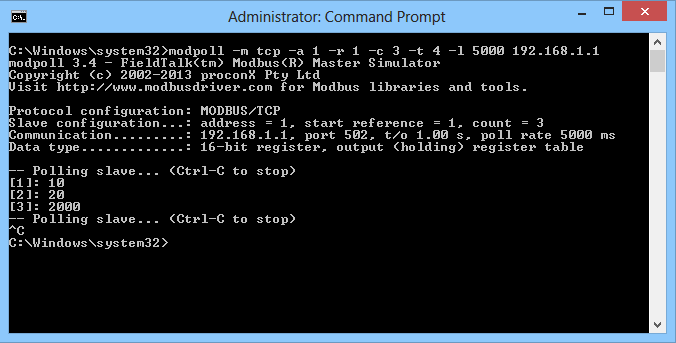


Figure : Modpoll Modbus master.

# Adding a new Modbus TCP slave

We are now 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:



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 : 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):

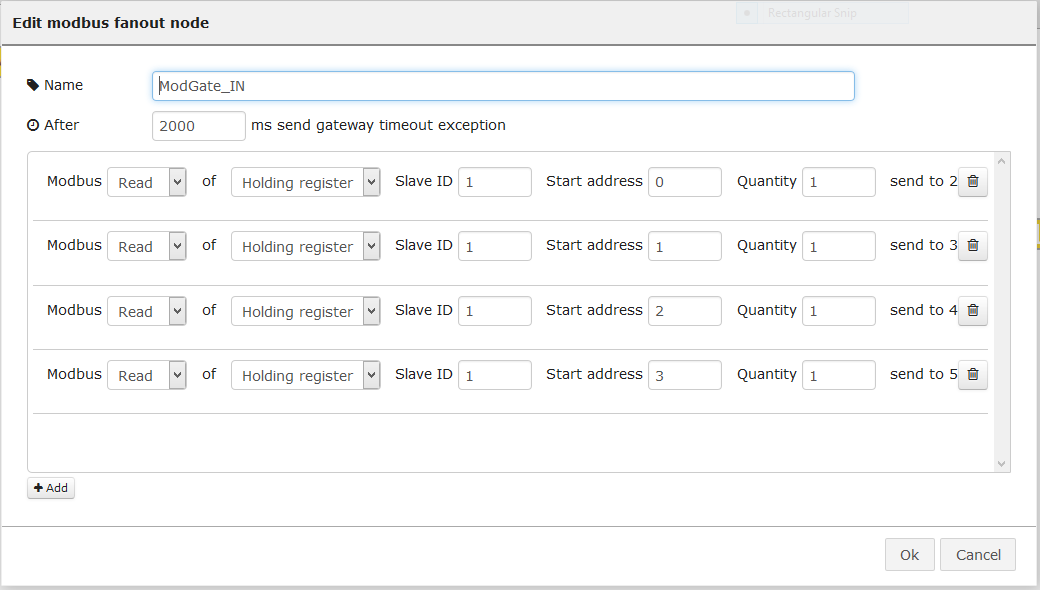


Table : New entry on ModGate\_IN node.

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

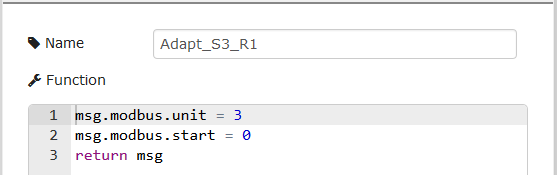


Figure : New function node.

1. 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:

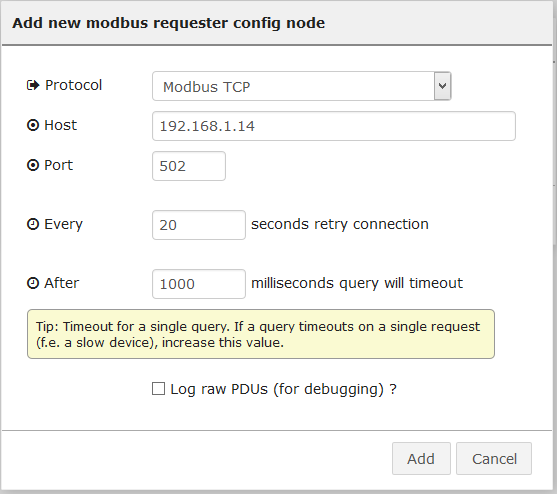


Figure : Modbus TCP requester.

1. 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":

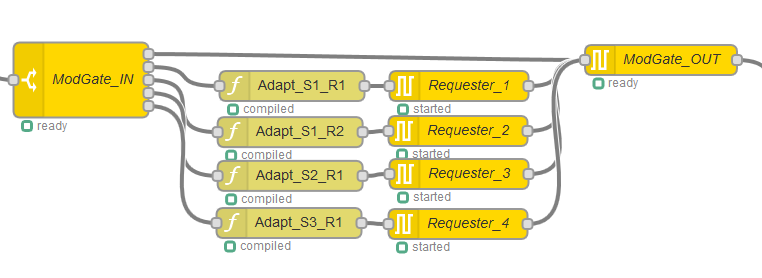


Figure : New nodes connected.

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

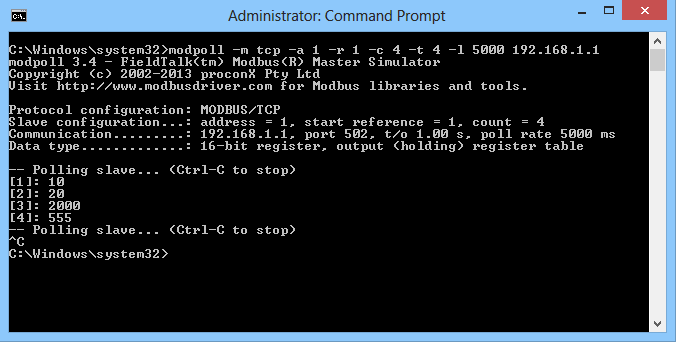


Figure : Modpoll Modbus master.