Diagnostic Box configuration

V.Vani
Malavika K.V

V1.0 22.06.2022

1. Node-red Flow for IP_Scan:

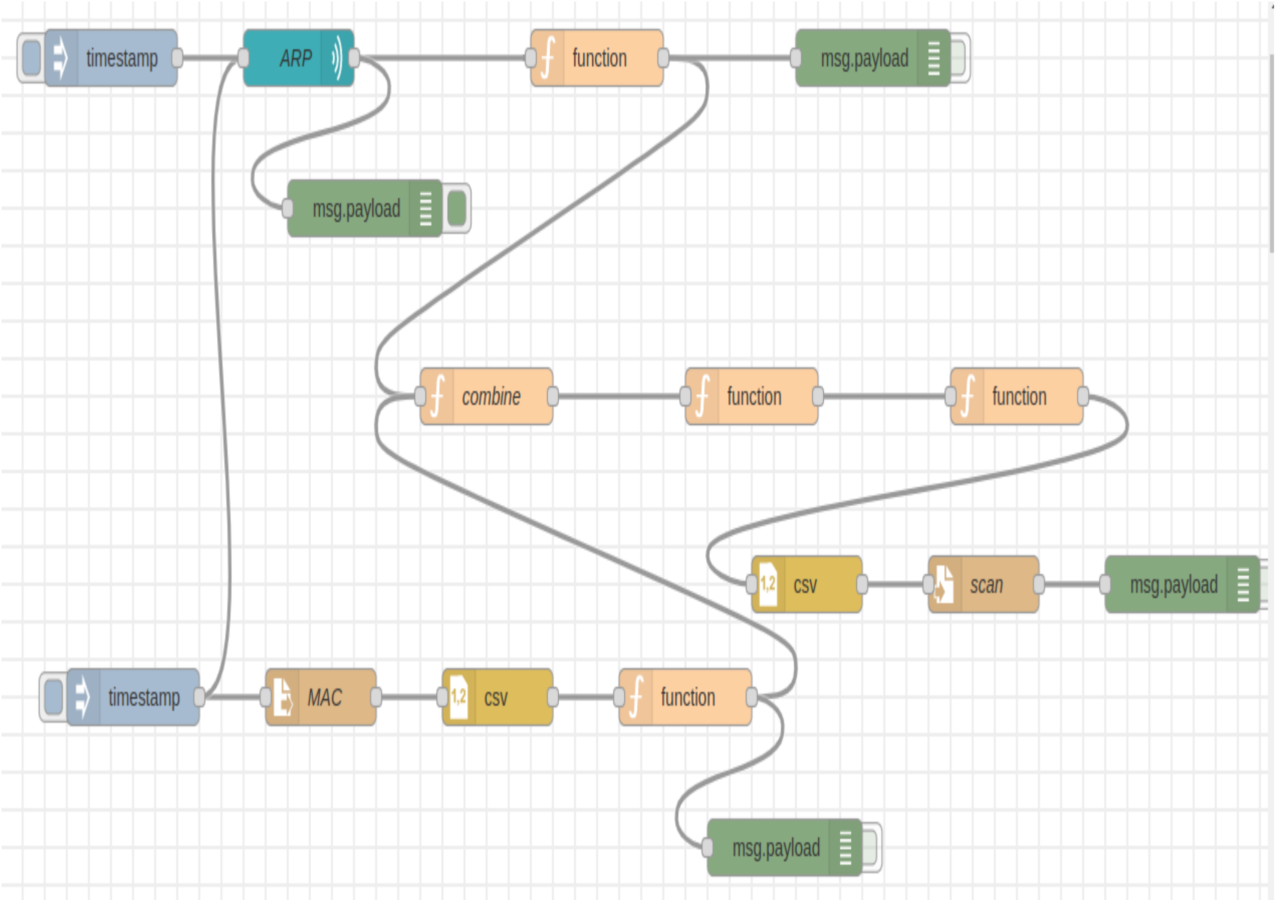
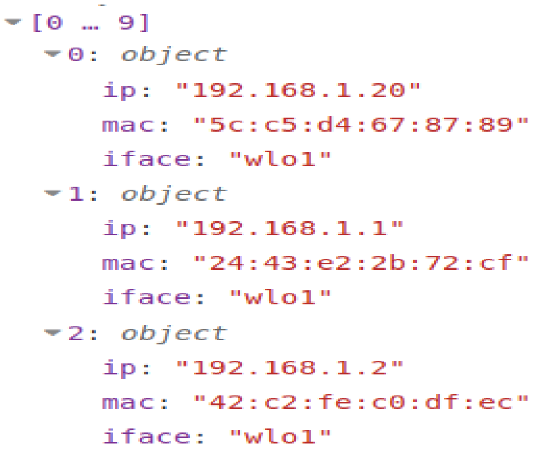


Fig – 1.1

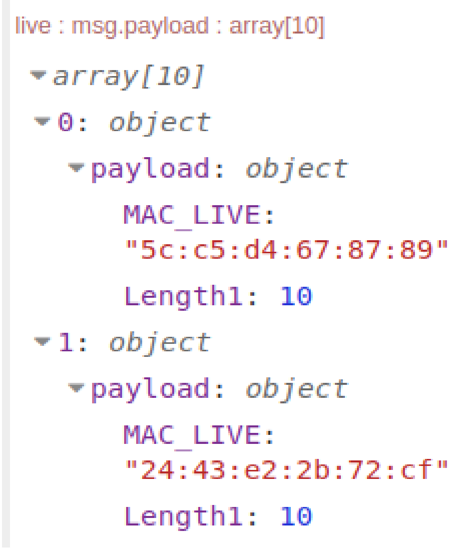
- ➢ **Arp** node is used to scan all the devices in the network and the output of node is shown in below figure.



```
▾[0 … 9]
  ▾0: object
      ip: "192.168.1.20"
      mac: "5c:c5:d4:67:87:89"
      iface: "wlo1"
  ▾1: object
      ip: "192.168.1.1"
      mac: "24:43:e2:2b:72:cf"
      iface: "wlo1"
  ▾2: object
      ip: "192.168.1.2"
      mac: "42:c2:fe:c0:df:ec"
      iface: "wlo1"
```

Fig – 1.2

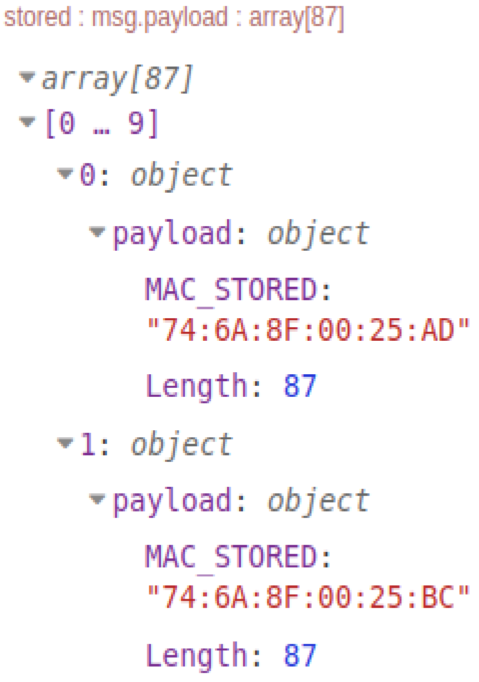
- From the output, all the live devices MAC addresses are taken and stored in MAC_LIVE array



```
live : msg.payload : array[10]

▾array[10]
 ▾0: object
   ▾payload: object
      MAC_LIVE:
      "5c:c5:d4:67:87:89"
      Length1: 10
 ▾1: object
   ▾payload: object
      MAC_LIVE:
      "24:43:e2:2b:72:cf"
      Length1: 10
```

Fig – 1.3

- A CSV file is taken, where Mac addresses of the devices present in the network are already stored as reference and taken as MAC_STORED array



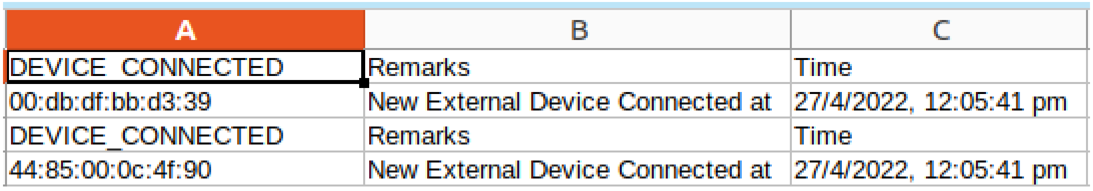
```
stored : msg.payload : array[87]

▾array[87]
 ▾[0 … 9]
   ▾0: object
     ▾payload: object
        MAC_STORED:
        "74:6A:8F:00:25:AD"
        Length: 87
   ▾1: object
     ▾payload: object
        MAC_STORED:
        "74:6A:8F:00:25:BC"
        Length: 87
```

Fig – 1.4

- ➢ Above flow will run continuously with some particular time interval and check for the count of live devices.

- ➢ If any external device gets connected to the network live count goes greater than stored

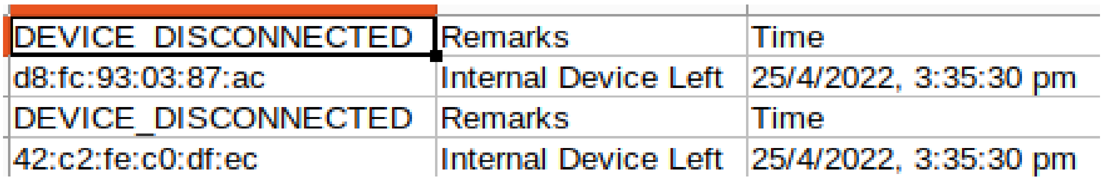
- ➢ Then it will compare both **MAC_STORED** and **MAC_LIVE** and separate that particular MAC address of the device connected newly and store in a CSV file as shown in below Fig.



A	B	C
DEVICE_CONNECTED	Remarks	Time
00:db:df:bb:d3:39	New External Device Connected at	27/4/2022, 12:05:41 pm
DEVICE_CONNECTED	Remarks	Time
44:85:00:0c:4f:90	New External Device Connected at	27/4/2022, 12:05:41 pm

Fig – 1.5

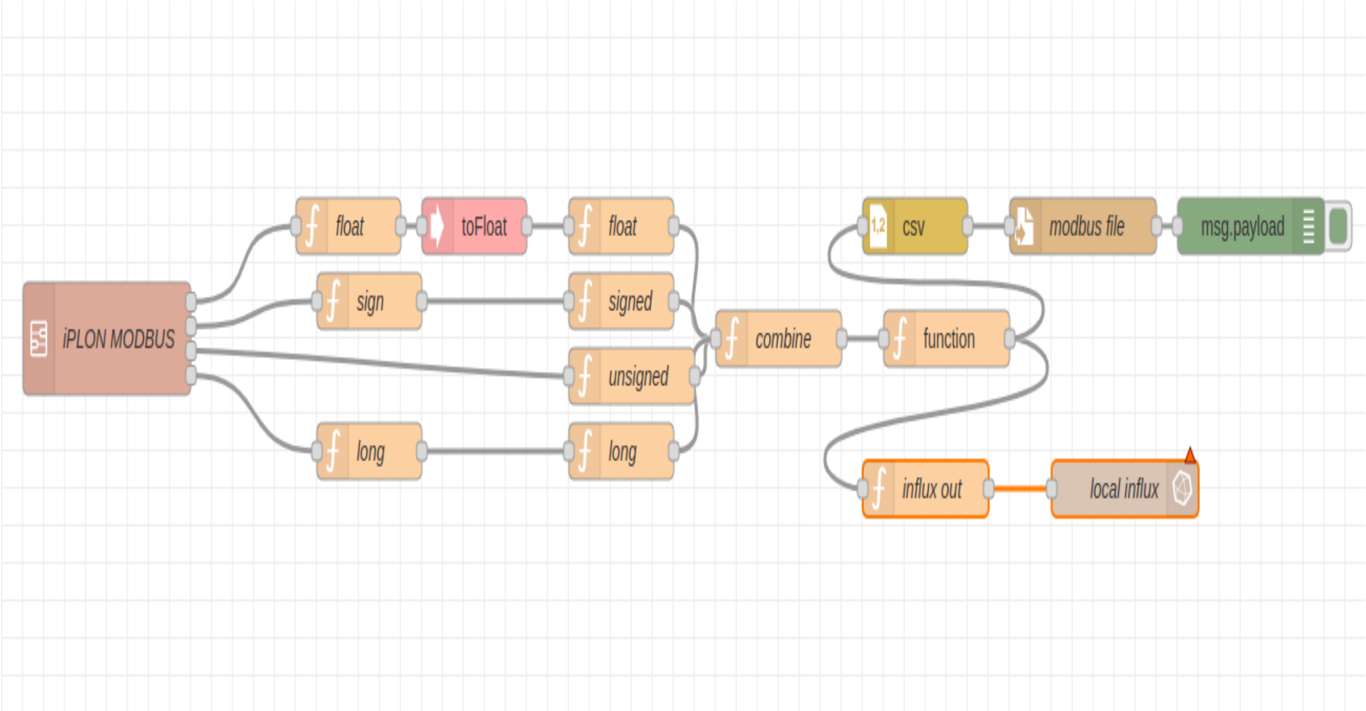
- ➢ If any device gets disconnected from the network then the count of live devices decreases and the disconnected device MAC address is saved in a CSV file as shown in below Fig.

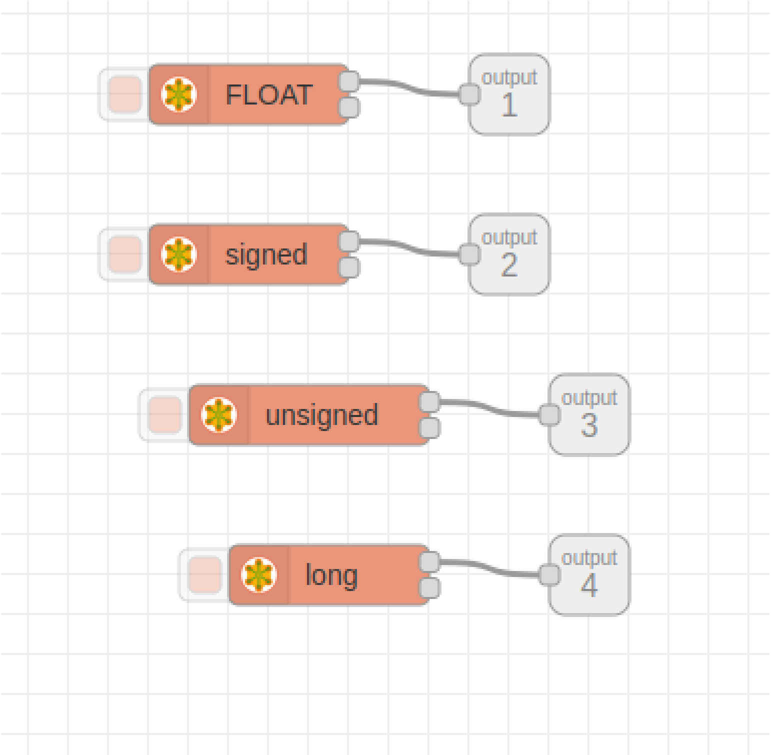


DEVICE_DISCONNECTED	Remarks	Time
d8:fc:93:03:87:ac	Internal Device Left	25/4/2022, 3:35:30 pm
DEVICE_DISCONNECTED	Remarks	Time
42:c2:fe:c0:df:ec	Internal Device Left	25/4/2022, 3:35:30 pm

Fig – 1.6

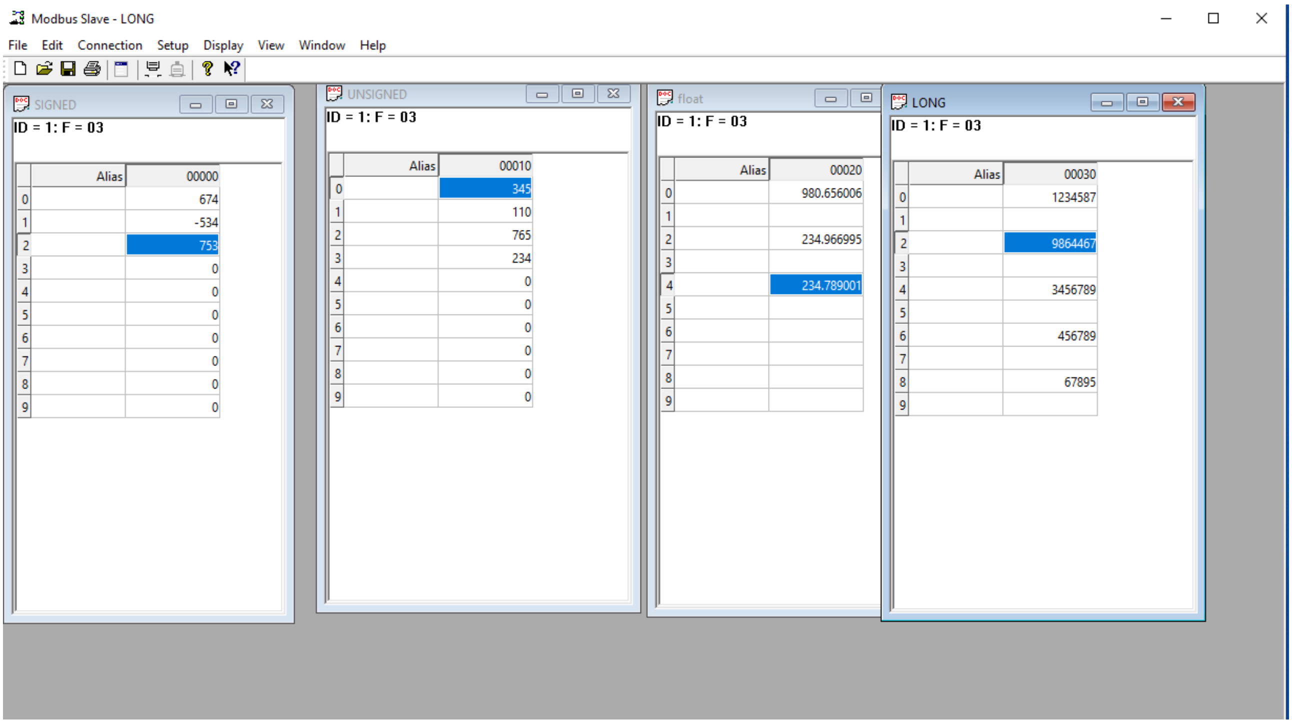
2. Modbus read work flow on Node-red





Here 4 modbus read nodes were used to read data from server for using this node first we have to install node-red-contrib-modbus node package from manage pallete.

Here i tested this flow by setting up a modbus slave in different device there feed the values in every registers as shown below.



SIGNED

ID = 1: F = 03

	Alias	00000
0		674
1		-534
2		753
3		0
4		0
5		0
6		0
7		0
8		0
9		0

UNSIGNED

ID = 1: F = 03

	Alias	00010
0		345
1		110
2		765
3		234
4		0
5		0
6		0
7		0
8		0
9		0

float

ID = 1: F = 03

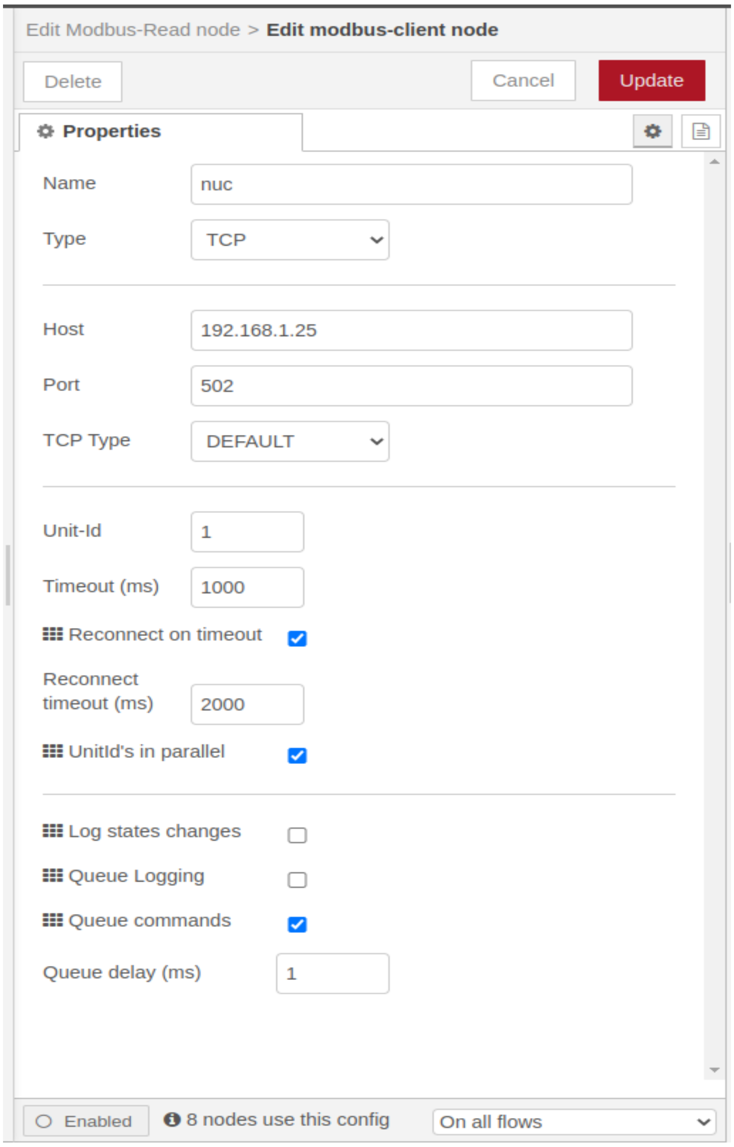
	Alias	00020
0		980.656006
1		
2		234.966995
3		
4		234.789001
5		
6		
7		
8		
9		

LONG

ID = 1: F = 03

	Alias	00030
0		1234587
1		
2		9864467
3		
4		3456789
5		
6		456789
7		
8		67895
9		

Inside all the modbus read nodes we have to give configuration like host, port ,type, unit-id as new configuration



Edit Modbus-Read node > **Edit modbus-client node**

Name: nuc

Type: TCP

Host: 192.168.1.25

Port: 502

TCP Type: DEFAULT

Unit-Id: 1

Timeout (ms): 1000

Reconnect on timeout ☑

Reconnect timeout (ms): 2000

UnitId's in parallel ☑

Log states changes □

Queue Logging □

Queue commands ☑

Queue delay (ms): 1

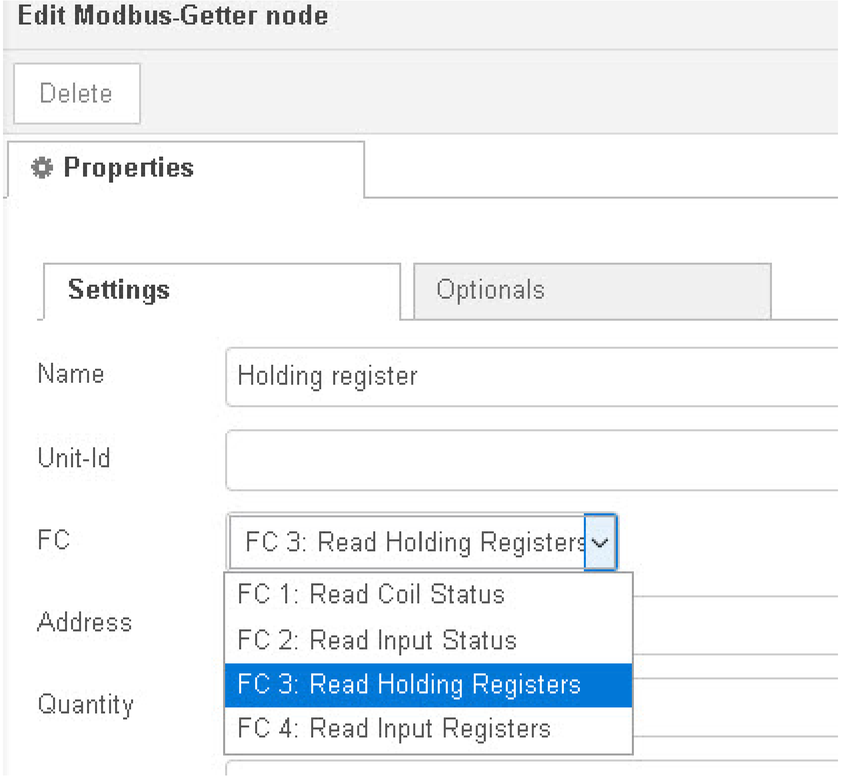
○ Enabled

8 nodes use this config

On all flows

in those 4 modbus read nodes one is for read float values and one for signed and one for unsigned values and another for long values inside all this nodes we have give the register address,quantity(how much quantity of values we need from the particular register)and also function code.

The nodes support function codes 1,2,3 and 4 as shown below:



Edit Modbus-Getter node

Name: Holding register

Unit-Id:

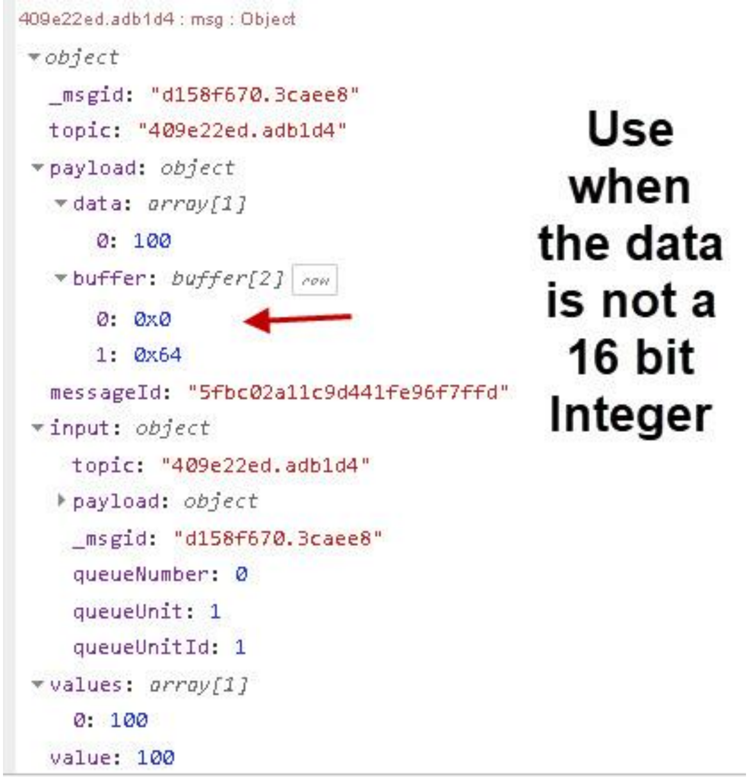
FC: FC 3: Read Holding Registers

- FC 1: Read Coil Status
- FC 2: Read Input Status
- FC 3: Read Holding Registers
- FC 4: Read Input Registers

Address

Quantity

Both nodes have two outputs which are essentially the same. I tend to use output 2.If you pass the output to a debug node and display the complete msg object you will see that you get an output like the one shown below.



```
409e22ed.adb1d4 : msg : Object
▾object
  _msgid: "d158f670.3caee8"
  topic: "409e22ed.adb1d4"
 ▾payload: object
   ▾data: array[1]
      0: 100
   ▾buffer: buffer[2] raw
      0: 0x0
      1: 0x64
  messageId: "5fbc02a11c9d441fe96f7ffd"
 ▾input: object
    topic: "409e22ed.adb1d4"
   ▸payload: object
    _msgid: "d158f670.3caee8"
    queueNumber: 0
    queueUnit: 1
    queueUnitId: 1
 ▾values: array[1]
    0: 100
  value: 100
```

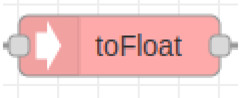
Use when the data is not a 16 bit Integer

The node assumes that the register contains a 16 bit integer and if that is the case then you can use the value field to get the data. If the data represents a float or 32 bit integer you will need to use the buffer.The syntax is msg.values and msg.payload.buffer.

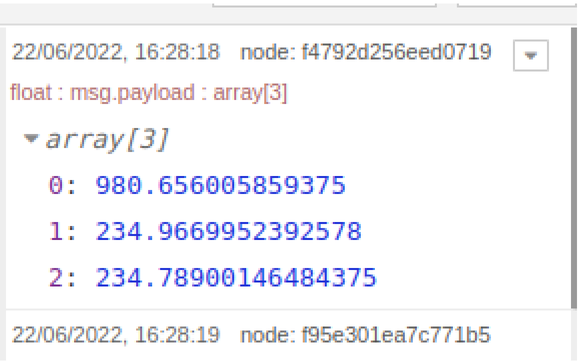
After this to integrate all the 4 modbus read as one input node i created a subflow by selecting those 4 modbus read node and > manage pallete > subflows > create subflow.and named it as IPLON MODBUS so the outputs coming from the subflow node will be as corresponding to thier allignment.

One function node is used to convert the 16bit value in float format which uses a JavaScript code which check every msg.payload <<16 and add that with next output and combine those messages in one msg.payload output.

And also using float node (node-red-contrib-float) node package to show the input values in float format.

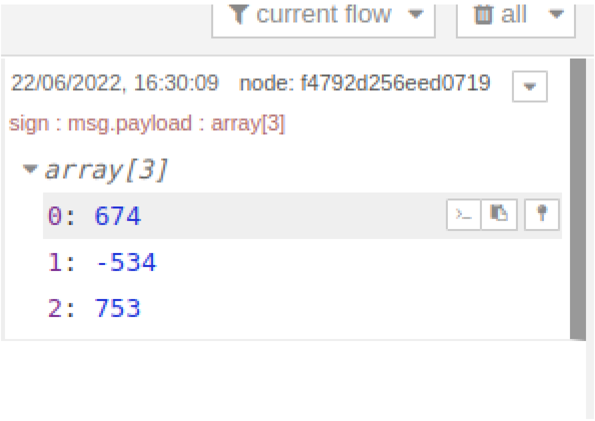


No configuration is needed inside it.



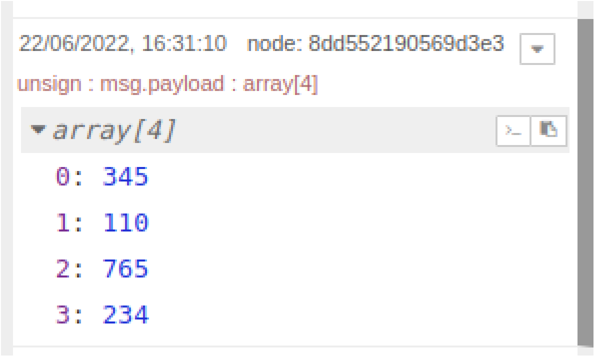
```
22/06/2022, 16:28:18   node: f4792d256eed0719
float : msg.payload : array[3]
▾array[3]
   0: 980.656005859375
   1: 234.9669952392578
   2: 234.78900146484375
22/06/2022, 16:28:19   node: f95e301ea7c771b5
```

Then for converting 16 bit sign values also we use another function node having JavaScript code which will check every msg.payload which is <<16 and set every inputs as var msg0= (msg.payload[0] << 16) >> 16; and combine those messages in one msg.payload output.



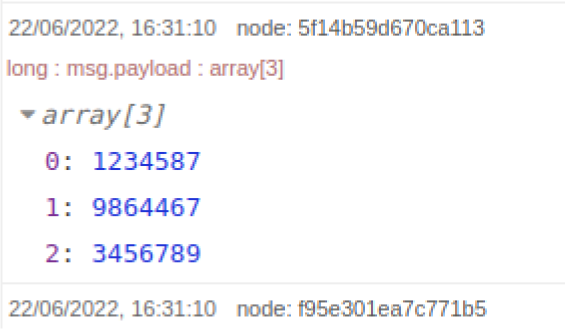
```
22/06/2022, 16:30:09   node: f4792d256eed0719
sign : msg.payload : array[3]
▾array[3]
   0: 674
   1: -534
   2: 753
```

Unsigned values can read and show as it as so that it doesn't need any function node for conversion.



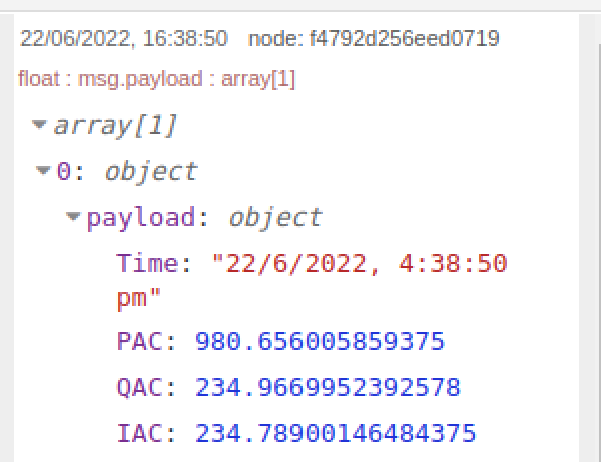
```
22/06/2022, 16:31:10   node: 8dd552190569d3e3
unsign : msg.payload : array[4]
▾array[4]
  0: 345
  1: 110
  2: 765
  3: 234
```

Then for converting long values also we use another function node having JavaScript code with eqations var msg0=(msg.payload[1]<<16)+msg.payload[0]; and combine all this equation as a single output.



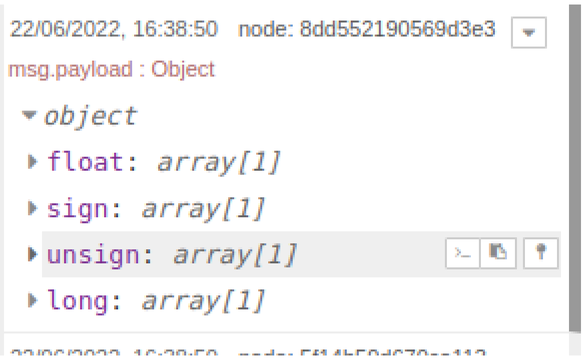
```
22/06/2022, 16:31:10   node: 5f14b59d670ca113
long : msg.payload : array[3]
▾array[3]
  0: 1234587
  1: 9864467
  2: 3456789
22/06/2022, 16:31:10   node: f95e301ea7c771b5
```

To set up output in a structured format we use function node which take input and give headers to each outputs give a structre to output.Here for all 4 outputs i used function node which has same format only header naming will differ.



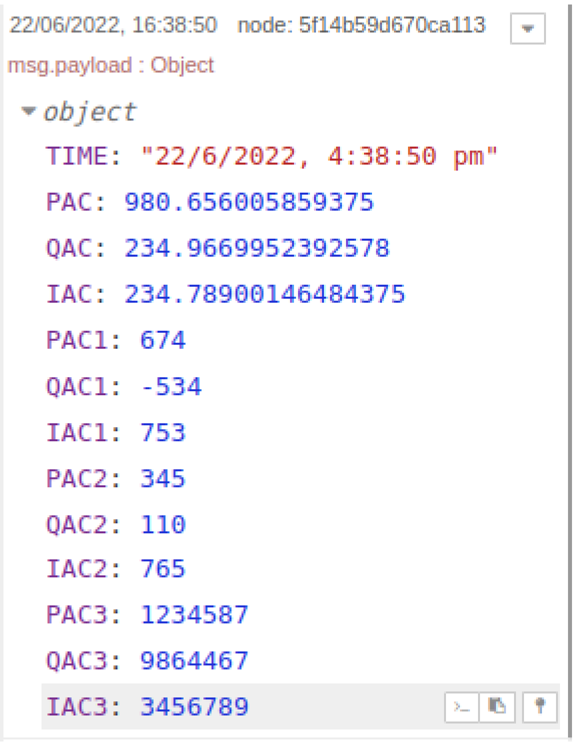
```
22/06/2022, 16:38:50   node: f4792d256eed0719
float : msg.payload : array[1]
▾array[1]
 ▾0: object
   ▾payload: object
      Time: "22/6/2022, 4:38:50
      pm"
      PAC: 980.656005859375
      QAC: 234.9669952392578
      IAC: 234.78900146484375
```

To merge all the 4 function output as a single output we have to combine those with msg.topic every inputs should have different topic names and merge those with a function node.



```
22/06/2022, 16:38:50   node: 8dd552190569d3e3
msg.payload : Object
▾object
 ▸float: array[1]
 ▸sign: array[1]
 ▸unsign: array[1]
 ▸long: array[1]
```

After this give header to all output and give strutured output with the use of function node.

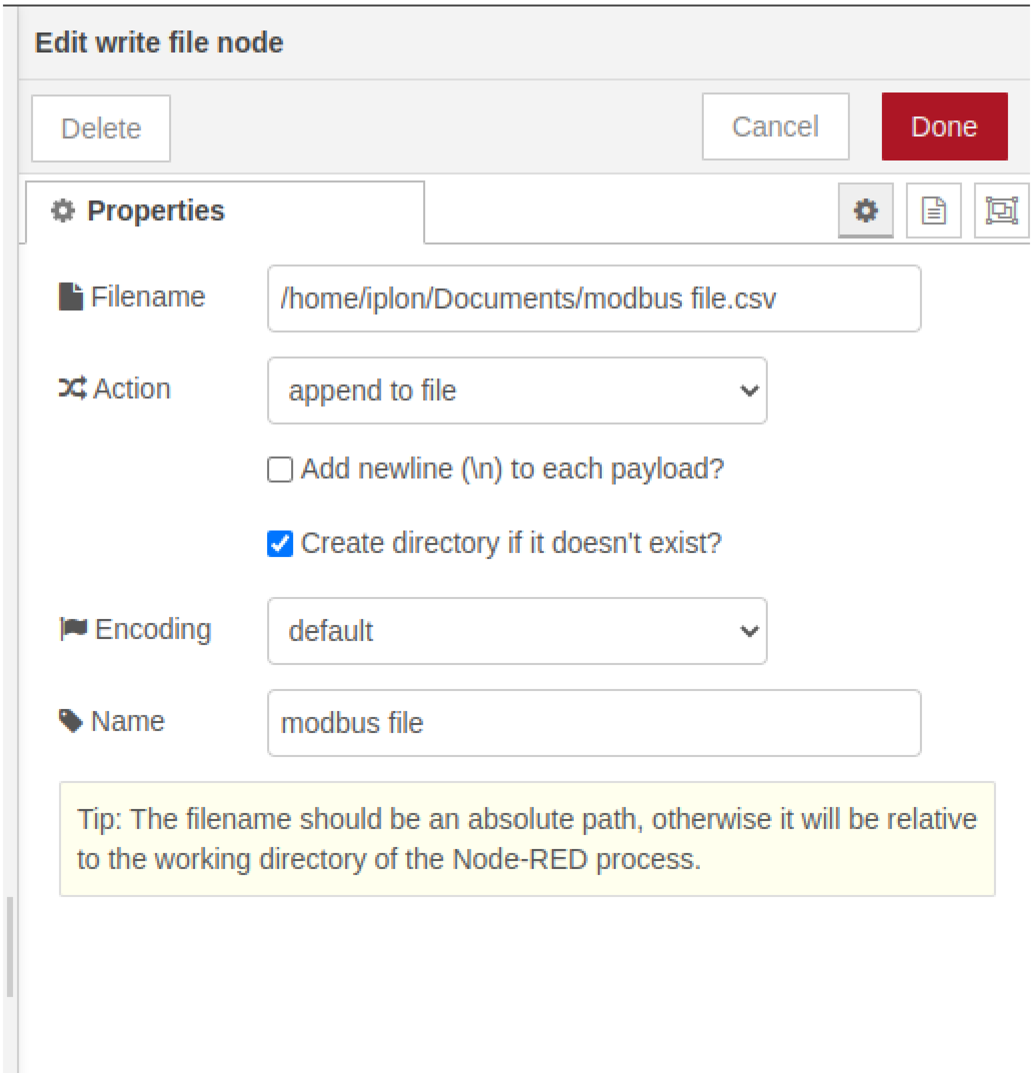


```
22/06/2022, 16:38:50   node: 5f14b59d670ca113
msg.payload : Object
▾object
  TIME: "22/6/2022, 4:38:50 pm"
  PAC: 980.656005859375
  QAC: 234.9669952392578
  IAC: 234.78900146484375
  PAC1: 674
  QAC1: -534
  IAC1: 753
  PAC2: 345
  QAC2: 110
  IAC2: 765
  PAC3: 1234587
  QAC3: 9864467
  IAC3: 3456789
```

From this output is going to to flow one to create csv file and another one for influx out.

Write a to a csv file using a csv node to pass output as message per row and setting header format.

Then a write file node to feed the path where to save the file.



Edit write file node

Filename: /home/iplon/Documents/modbus file.csv

Action: append to file

☐ Add newline (\n) to each payload?

☑ Create directory if it doesn't exist?

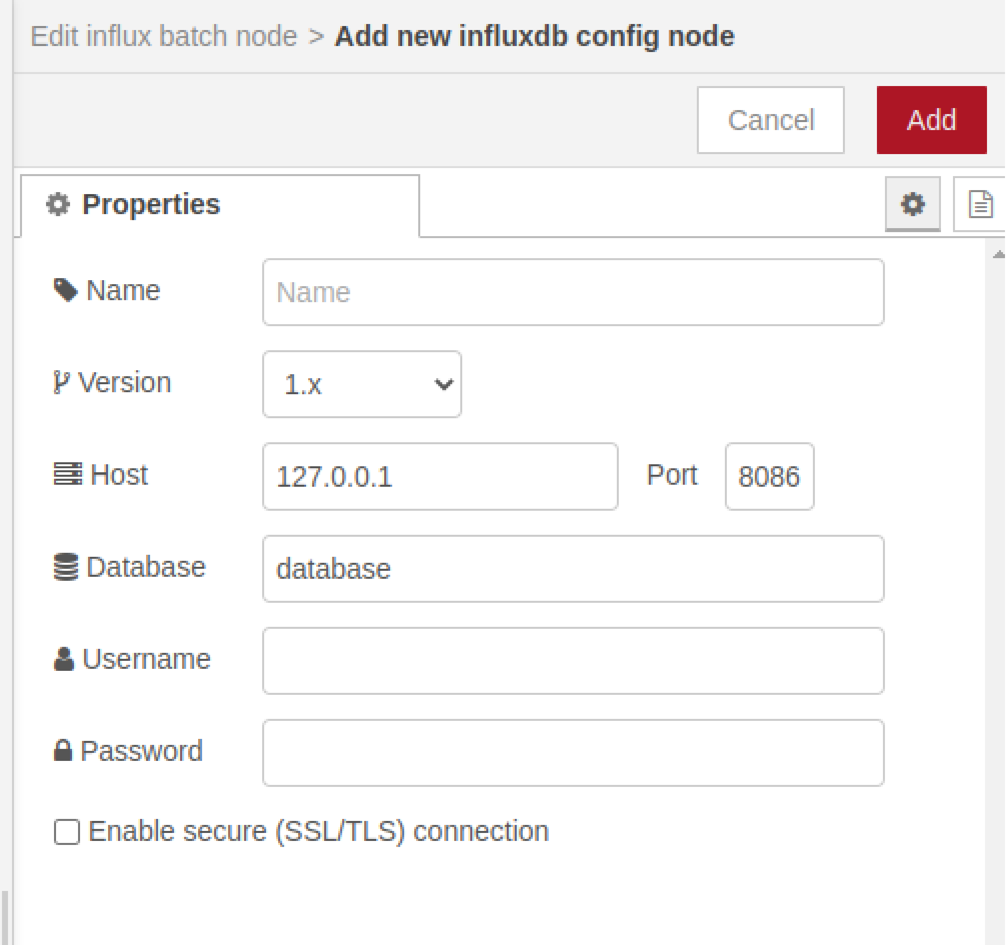
Encoding: default

Name: modbus file

Tip: The filename should be an absolute path, otherwise it will be relative to the working directory of the Node-RED process.

One more function node s using to give measurement, field,tag,timestamp before we injecting this data to influx DB

influxDB batch node (node-red-contrib-influxdb) used here to out this data to influx database for this we have to give some configuration details to this node.



Edit influx batch node > **Add new influxdb config node**

Name: Name

Version: 1.x

Host: 127.0.0.1 Port: 8086

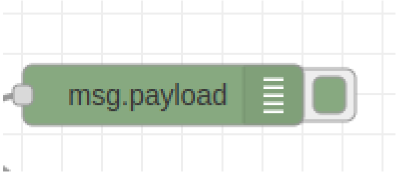
Database: database

Username:

Password:

☐ Enable secure (SSL/TLS) connection

Use debug nodes to take out output from each nodes.



msg.payload