nodeG5 – iotasset.json config guide

MODBUS-RTU/TCP

Firmware version: fw_nodeG5_v2.2
Guide release date: 8MAR2024

Filename :	iotasset.json
Location :	/user

1 Introduction

The file 'iotasset.json' contains the assets configuration that is read by the Modbus master program to parse specific data from Modbus slave devices. Acquired data is then inserted into local database for downstream IoT cloud clients.

2 IoTasset name definitions

Modbus IoTasset is defined by JSON objects inside JSON array "CANBUS".

Below is the list of IoTasset names that should be defined in each JSON object.

IoTasset name	**Description**
Key	Define the data tagname
IOTMODE	Define the data handling mode

Below is the list of Modbus names that must should be defined in each JSON object.

MODBUS name	**Description**
MODBUSTYPE	Define the type of Modbus communication
RTUPORT	Define the physical port used for serial communication
NODEID	Define the Modbus/RTU slave device address(ID)
IPADDR	Define the Modbus/TCP slave device IPv4 address
IPPORT	Define the Modbus/TCP slave device IP port
MODBUSFC	Define the Modbus function code
REGADDR	Define the data address of first register requested
NUMREGS	Define the number of register requested
DATATYPE	Define the data type to convert from Modbus raw data

Data type = BITS	
BITSTART	Define the position of starting bit to parse
BITCOUNT	Define the number of bits to parse

Optional names:

Data type = INTEGER, FLOAT only number=number*multiplier + adder	
SCALEMUL	Define both multiplier & adder when require scaling
SCALEADD	

POLLCYCLE	Define the multiples of Poll Period for any IoTasset

Note: Poll Period is set at web config page 'IoT-Hardware')

3 IoTasset JSON object setup information

“Key”:”value”

value	Description
string	Unique name for this data value eg temperature, voltage, pressure, rpm

“IOTMODE”:”value”

value	Description
0	Send to cloud immediately (priority data)
1	Store to local database for local IoT client processing

“MODBUSTYPE”:”value”

value	Description
RTU	Modbus/RTU standard protocol over RS-485
TCP	Modbus/TCP standard protocol over Ethernet

Note: For “Modbus/RTU over TCP”, apply Modbus/TCP protocol.

“RTUPORT”:”value”

value	Description
A	SERIAL RS-485 port (2-wire, half-duplex)
B	SERIAL RS-485 port (2-wire, half-duplex)

Note: refer to nodeG5 user manual for connector pin-out details.

“NODEID”:”value”

value	Description
1 to 247	Modbus/RTU : slave address/node ID Modbus/RTU over TCP : slave address/node ID

“IPADDR”:”value”

value	Description
n1.n2.n3.n4	Modbus/TCP : slave IPv4 address

“IPPORT”:”value”

value	Description
502	Modbus/TCP : default port 502

“MODBUSFC”:”value”

value	Description
1	Read Coil Status (FC01)
2	Read Input Status (FC02)
3	Read Holding Registers (FC03)
4	Read Input Registers (FC04)

FC=function code in Modbus protocol

“REGADDR”:”value”

value	Description
0 to 9998	Data address of first register requested (dec)

Note: Depending on device model, may require -1 offset of the register address.

“NUMREGS”:”value”

value	Description
1 to 16	Number of register requested (dec)

“BITSTART”:”value”

value	Description
15 to 0	Position of starting bit (0=right most bit) 15[MSB]....0[LSB]

“BITCOUNT”:”value”

value	Description
1 to 8	Number of bits to parse

4 Data Type definitions for standard Modbus

MODBUS FC01/02: **“DATATYPE”:”value”**
“NUMREGS”:”count”

value	count	Description
BITS	1-16	1-16 bits to unsigned integer

Note: Binary value will be converted to decimal value, eg 1110_2 will be reported as 14_{10}. The more significant bits contain the higher coil variables.

MODBUS FC03/04: **“DATATYPE”:”value”**
“NUMREGS”:”count”

value	count	Description
BITS	1	16-bit register parse to 1-8 bits (unsigned integer)
UINT8H	1	16-bit register to **8-bit unsigned integer**
UINT8L		
SINT8H		16-bit register to **8-bit signed integer**
SINT8L		
UINT16HL	1	16-bit register to **16-bit unsigned integer**
UINT16LH		
SINT16HL		16-bit register to **16-bit signed integer**
SINT16LH		
UINT32HLhl	2	16-bit register pair to **32-bit unsigned integer**
UINT32hlHL		
UINT32LHlh		
UINT32lhLH		
SINT32HLhl		16-bit register pair to **32-bit signed integer**
SINT32hlHL		
SINT32LHlh		
SINT32lhLH		

Note: U=unsigned, S=signed, HLhl=big endian, lhLH=little endian
hlHL=word-little-endian, byte-big-endian
Lhlh=word-big-endian, byte-little-endian

MODBUS FC03/04: **“DATATYPE”:”value”**
“NUMREGS”:”count”

value	count	Description
STRING16	8	Set of eight 16-bit registers to **16 ASCII characters** (e.g. abcdefghijklmnop)
STRING16R		Set of eight 16-bit registers to **16 ASCII characters**, byte swapped (e.g. badcfehgjilknmpo)
STRING8	4	Set of four 16-bit registers to **8 ASCII characters** (e.g. abcdefgh)
STRING8R		Set of four 16-bit registers to **8 ASCII characters**, byte swapped (e.g. badcfehg)

MODBUS FC03/04: **“DATATYPE”:”value”**
“NUMREGS”:”count”

value	count	Description
FLOAT32ABCD	2	16-bit register pair to IEEE-754 single precision floating point number. Byte orientation=ABCD,DCBA,BADC,CDAB AB=word data from first 16-bit register CD=word data from second 16-bit register
FLOAT32DCBA		
FLOAT32BADC		
FLOAT32CDAB		

5 Example for IOT asset configuration


```
{
  "MODBUS":[
    {
      "Key":"M1_Current",
      "IOTMODE":"1",
      "MODBUSTYPE":"RTU",
      "RTUPORT":"A",
      "NODEID":"1",
      "MODBUSFC":"3",
      "REGADDR":"500",
      "NUMREGS":"2",
      "DATATYPE":"FLOAT32ABCD",
      "SCALEMUL":"0.1",
      "SCALEADD":"0"
    },
    {
      "Key":"Flow_Switch",
      "IOTMODE":"1",
      "MODBUSTYPE":"RTU",
      "RTUPORT":"A",
      "NODEID":"5",
      "MODBUSFC":"2",
      "REGADDR":"180",
      "NUMREGS":"3",
      "DATATYPE":"BITS"
    },
    {
      "Key":"INV_Status",
      "IOTMODE":"1",
      "MODBUSTYPE":"RTU",
      "RTUPORT":"A",
      "NODEID":"7",
      "MODBUSFC":"3",
      "REGADDR":"250",
      "NUMREGS":"1",
      "DATATYPE":"BITS",
      “BITSTART”:”8”,
      “BITCOUNT”:”2”
    },
    {
      "Key":"Batt_Voltage",
      "IOTMODE":"1",
      "MODBUSTYPE":"RTU",
      "RTUPORT":"A",
      "NODEID":"11",
      "MODBUSFC":"3",
      "REGADDR":"390",
      "NUMREGS":"1",
      "DATATYPE":"UINT16HL",
      "SCALEMUL":"0.5",
      "SCALEADD":"0.1",
      "POLLCYCLE":"5"
    },
```

```
    {
      "Key":"Batt_Temperature",
      "IOTMODE":"1",
      "MODBUSTYPE":"RTU",
      "RTUPORT":"A",
      "NODEID":"11",
      "MODBUSFC":"3",
      "REGADDR":"350",
      "NUMREGS":"2",
      "DATATYPE":"UINT32HLhl",
      "POLLCYCLE":"10"
    },
    {
      "Key":"M2_Current",
      "IOTMODE":"1",
      "MODBUSTYPE":"TCP",
      "IPADDR":"192.168.1.130",
      "IPPORT":"502",
      "NODEID":"11",
      "MODBUSFC":"3",
      "REGADDR":"1999",
      "NUMREGS":"2",
      "DATATYPE":"FLOAT32ABCD"
    },
    {
      "Key":"M2_ActivePower",
      "IOTMODE":"1",
      "MODBUSTYPE":"TCP",
      "IPADDR":"192.168.1.130",
      "IPPORT":"502",
      "NODEID":"21",
      "MODBUSFC":"3",
      "REGADDR":"2999",
      "NUMREGS":"2",
      "DATATYPE":"FLOAT32ABCD"
    }
  ]
}
```


6 Methods to upload ‘iotasset.json’ file to nodeG5

-Upload the iotasset.json file from your computer using the ‘Upload iotasset.json’ button in the ‘IoT Hardware’ tab.

-Put the iotasset.json file in /user folder of USB drive.
Plug the USB drive into any USB-A port and click the ‘Upload to nodeG5’ button in the ‘Management’ tab.

-Use SCP/Putty console or WinSCP.

<EOF>