V1

MODBUS MQTT Plug-In

26-02-2024

BRIAN RODRIGUEZ || DUURZAMEBOUWKEET B.V.

Table of contents

Introduction

Introduction

This document will describe the function and behavior of the MODBUS plug-in. The MODBUS plug-in will be used to format incoming and outgoing data from and to MODBUS devices. This plug-in will send and receive data through the MQTT server

The plugin needs to be written in python and compiling will be done using PyInstaller or Nuitka the following should be provided:

- Document describing the working of the code
- Source code/files with comment included in the code
- List of all the packages installed in the venv and their version
- ARM compiled binary
    - Compiled as an all-in-one binary
    - E.g PyInstaller command: *pyinstaller --onefile --windowed main.py*
- The plugin is going to run as a systemd service.
    - No need to supply an systemd .service file
    - Developer needs run the plug-in as an service and test if everything works as it should

Running the plug-in

The plug-in will run as a systemd service. The following arguments will be used:

- systemctl start %plugin_name%
- systemctl stop %plugin_name%
- systemctl restart %plugin_name%

When the plug-in is running and is stopped and/or restarted there should be no leftovers of the previous plugin process.

Initialization

At the start (boot/load) of this plug-in the initialization phase will run and provide the information needed for the plug-in to function as intended.

Plug-in configuration

During the initialization phase the plug-in will read the JSON data in MQTT topic **config/cabinet**. This topic will hold the configuration data for all the plug-ins and modules. The MODBUS Plug-in will get its configuration parameters from the plugin.modbus object.

Key	Type	Purpose
config_update_interval	Integer	Seconds to (re)check config as described in Plug-in configuration
device_update_interval	Integer	Default interval in seconds for device polling
device_path	String	Hardware address for the RS485 device

Optional configuration parameters

In the configuration data as described above there are a few key-value pairs that are optional.

If the key-value pairs aren't present in the configuration that's read from the MQTT server, the plugin takes on default values.

If the key-value pairs are present in the configuration that's read from the MQTT server, then these will overrule the default values as described before.

Key	Type	Default	Purpose
mqtt_server	String	"127.0.0.1"	Provide the address of the MQTT server
mqtt_user	String	"mqttuser"	MQTT authentication username
mqtt_pass	String	"6cac070009ba15169fe80c5d1106a90f"	MQTT authentication password. This will be a MD5 hash.

Get list of devices

For this plug-in to work properly, it will have to know what MODBUS devices are available on the bus. Therefore in the topic **config/cabinet** under the object devicelist. This will mostly contain the information of data that have to be read from an slave.

Key	Type	Purpose
slaveX	Object	Friendly name of slave device
slaveX.id	Integer	Modbus slave id
slaveX.datapoints	Object	The list of objects with datapoints of the given MODBUS
slaveX.datapoints.X	Object	Datapoint in modbus slave device
slaveX.datapoints.X.fc	Integer	MODBUS functioncode (e.g. 5 for write single coil, 3 for Multiple Holding Registers etc.)
slaveX.datapoints.X.address	Integer	Address (e.g. 0 for coil 0)

Example of MODBUS plug-in configuration

```
{
    "plugin": {
        "modbus": {
            "config_update_interval": 5,
            "device_update_interval": 1,
            "poll_timeout": "Optional value in seconds for overruling the default 30s polling timeout",
            "device_path": "/dev/ttyusb0",
            "mqtt": {
                "mqtt_server": "This is an optional param",
                "mqtt_user": "This is an optional param",
                "mqtt_pass": "this is an optional param"
            },
            "devicelist": {
                "slave0": {
                    "id": 0,
                    "datapoints": {
                        "relay_0": {
                            "polling_interval": "Optional value for datapoint polling in seconds",
                            "friendly_name": "this is the friendly name",
                            "fc": 5,
                            "address": 0
                        },
                        "relay_1": {
                            "friendly_name": "this is the friendly name",
                            "fc": 5,
                            "address": 1
                        }
                    }
                },
                "slave1": {
                    "id": 1,
                    "datapoints": {
                        "measurement1": {
                            "fc": 3,
                            "address": 258
                        },
                        "measurement2": {
                            "fc": 3,
                            "address": 259
                        }
                    }
                }
```

Figure 1: Example of plugin config as provided in config/cabinet

Persistency/fallback

During the initialization the plug-in will save the configuration and device list persistent so it's no longer depending on the data in the object as described above. If for some reason the configuration cannot be read from the MQTT topic, the plug-in will have a fallback configuration from the previous initialization/configuration update.

This persistent data will have to be stored in such a way that when the plug-in boots/loads completely and isn't able to read the config on the MQTT server, it will have its local copy from the previous initialization/configuration update. The persistent data has to be a JSON formatted file in the same directory as the plug-in binary named modbus-config-cache.conf

Functions

Switching coils

The MODBUS plug-in will read the data from the MQTT topic **data/modbus/request**. This will contain an array of objects. The objects will hold all the data needed for the MODBUS plug-in. The reading of this topic should be instant/real-time.

Key	Type	Purpose
id	Integer	Slave ID
Fc	Integer	Function code
Address	Int	Address (e.g. number of coil or register)
Value	Int	Value to write to the register

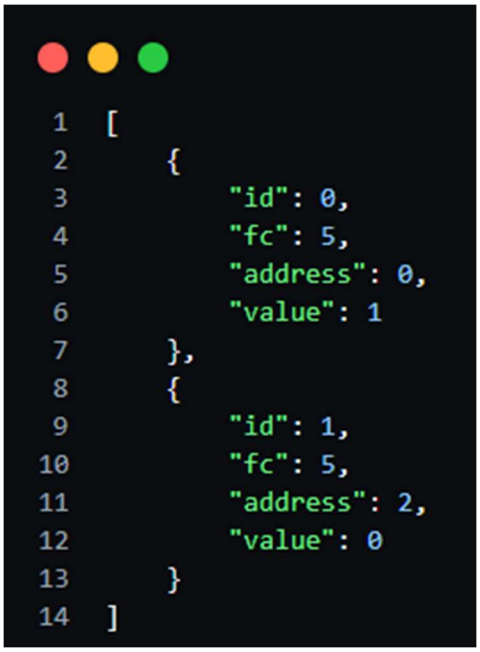

Figure 2: Example of array found in system/modbus/request topic

When the plug-in reads an object in the array, and sends it to the corresponding MODBUS slave, it will have to remove the object from the array. E.g.: *figure 2: When the plugin switched on coil 0 of slave 0, there will only be 1 object left in the array. When the plugin also switched off coil 2 of slave 1, the array should be empty.*

Reading data

In the configuration the devicelist provides a list of datapoints that have to be read at an interval. The default polling interval is the value from the key-value **device_update_interval**, which is also found in the configuration. An optional key-value pair **polling_interval** can be provided to overrule the default polling interval.

The data should be published under the topic data/modbus/response as follows:

Key	**Type**	**Purpose**
friendly_name	String	Friendly name of datapoint
value	-	Value response of read datapoint
polling_interval	Integer	Optional polling interval in seconds

Monitoring the state of slaves and datapoints

When the plug-in wrote a datapoint as described above, it will have to "remember" the datapoint.

Example with data from figure 2:
*When the plug-in switched the coils of the slave devices, it should remember the state the coil or datapoint should be in. The **device_update_interval** that the plug-in got from config file (chapter plug-in configuration) will be used as an poll interval for this datapoint. When the plug-in send an ON command to a coil and the polled value isn't on the plug-in should retry to send the ON command. This will be repeated for a maximum of 3 times before the plug-in outputs an error. The error is resolved when the actual state is equal to the preferred state.*

Timeout

A time-out should be implemented. When no data is received within the timeout an error should be generated. The default timeout is 30s but this value can be overruled by the optional **poll_timeout** key-value pair in the configuration.

Error output

When an error occurs this will have to be sent to the topic system/error/modbus.
The error should contain the following:

Key	**Type**	**Purpose**
friendly_name	String	If available, otherwise empty string
id	Integer	Slave ID
Fc	Integer	Function code
Address	Int	Address (e.g. number of coil or register)
Description	String	e.g. "Could not write to coil" or "timeout"
preferred_state	-	Value that the datapoint should have
actual_state	-	Value that the datapoint currently is in

Summary

MQTT Topic overview

MQTT Topic	**Method**	**Purpose**
config/cabinet	GET	Get plugin configuration
data/modbus/request	GET	Data to write to modbus
data/modbus/response	SET	Data read from modbus
System/error/modbus	SET	Data read from modbus