Xompass Base Edge Module v1 Quick Guide.

Version 1.2

1. Introduction

Xompass Base Edge Module v1 is compilation of software libraries and tools based in the Microsoft's "Azure IoT Gateway" architecture, available on a Docker container based on Ubuntu.

The biggest difference with Xompass Edge Module v2 is that the latest is based on the Microsoft's "Azure IoT Edge" architecture which is a containerized, and cloud managed architecture.

This document is intended to describe how to configure the client to collect data from a device, using MODBUS RTU or TCP, and send it to the Xompass Cloud over HTTP API.

2. Software description

The client architecture is shown in figure X. All the settings required for the client to work are inside the "gw.config.json" file.

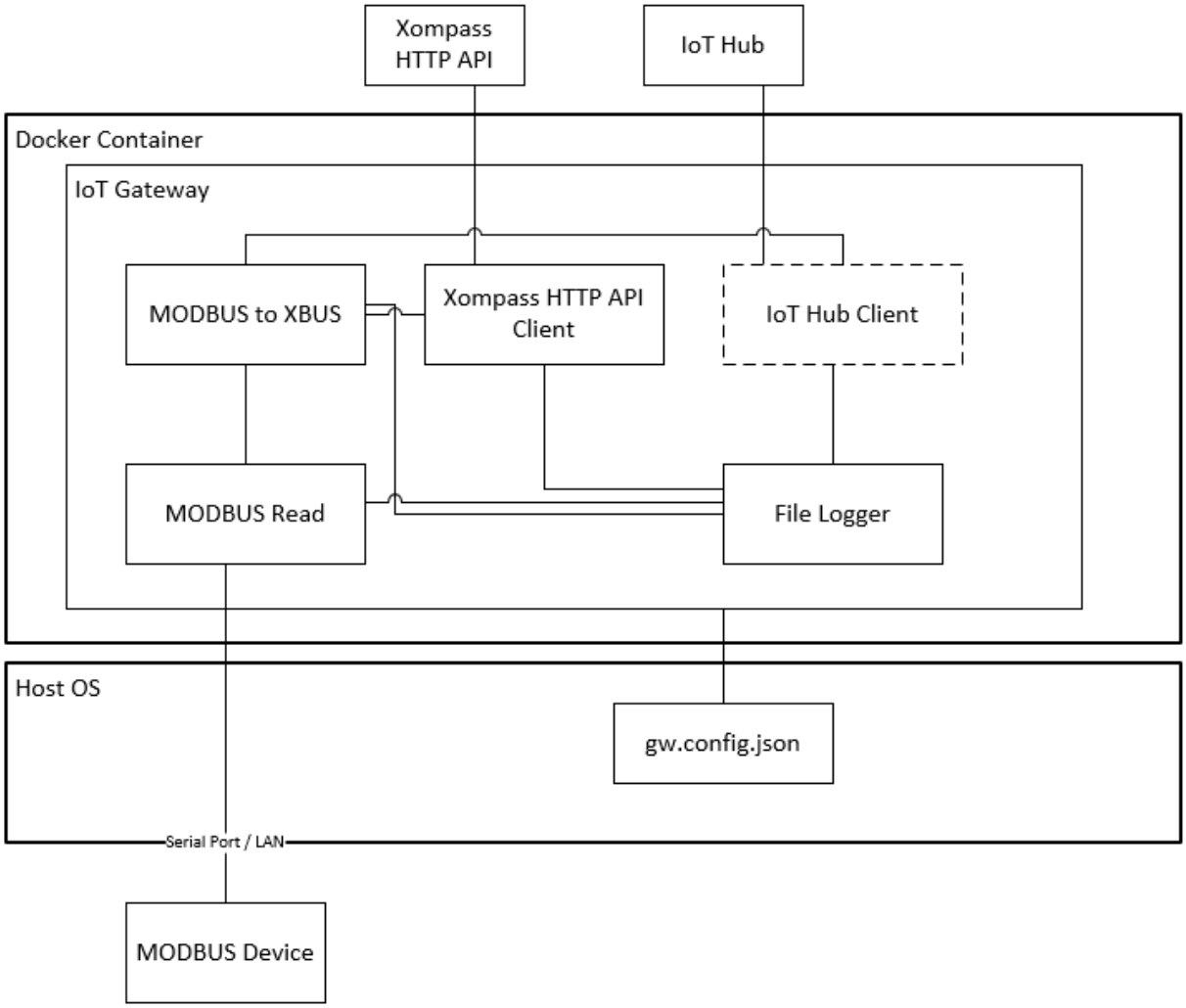

Figure 1: Xompass Edge Client v1 Architecture

Preparing your Environment

Before starting, you will need to install Docker in your system.

You can find a sample "gw.config.json" and other resources in the Xompass-edge-base-module-public repository.

Preparing the configuration file

1. Enabling communication with Xompass HTTP API endpoint

The Xompass Edge Client v1, loads all the configuration from a local file "gw.config.json" on the Host OS, to simplify configuration updates.

The Xompass HTTP API Client has 1 critical configuration parameters:

1. Xompass Account API-KEY for authorization

The Xompass APIKEY can be found in the Data Source/API Keys section of the Xompass web application:

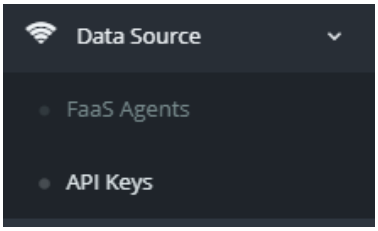

Figure 2: Xompass Application Data Source/API Keys

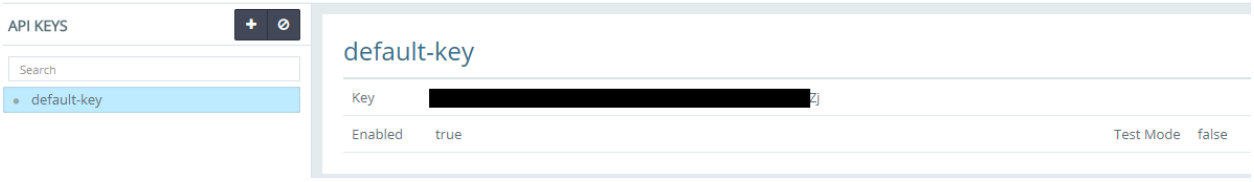

Figure 3: API Keys list and Key.

This must be added on the "gw.config.json". Inside this file, go to the "Xompass_http" module settings section.

Figure 4: Apikey configuration field on gw.config.json

2. Collecting Data from a MODBUS Device

To successfully collect data from a MODBUS device, 4 steps must be done:

1. Setup Device Information and Address
2. Configure MODBUS Read Module to read desired MODBUS Registers
3. Configure the MODBUS to XBUS module to properly parse the MODBUS Register Value

2.1. Setup Device and Address

To add a new device, you need to configure the "Modbus_read" module to communicate with it, see example in Figure 5.

Device communication setting fields description

- **serverConnectionString**: Device IP or Serial Port name. MODBUS TCP or RTU will be chosen automatically depending on this setting.
- **serverConnectionPort**: MODBUS TCP ONLY. OPTIONAL. Specifies the communication port, 502 by default.
- **interval**: Time delay, in ms, between readings (operations) will be done.
- **macAddress**: MAC Address of the device. Optional.
- **baudRate**: MODBUS RTU ONLY. Serial port setting.
- **stopBits**: MODBUS RTU ONLY. Serial port setting.
- **dataBits**: MODBUS RTU ONLY. Serial port setting.
- **parity**: MODBUS RTU ONLY. Serial port setting.
- **flowControl**: MODBUS RTU ONLY. Serial port setting.
- **deviceType**: Unique identifier, Field required by MODBUS_TO_XBUS module
- **sqliteEnabled**: 0 default.
- **operations**: List of MODBUS operation to be performed. Described further below.

2.2. Configure MODBUS Read Module to read desired MODBUS Register

To configure the MODBUS operation of a device, each operation within the device has the following fields:

- **unitId**: MODBUS ID. In MODBUS RTU is the same than the Device Address
- **functionCode**: MODBUS Function code to execute.
- **startingAddress**: MODBUS initial register address, added to the function offset.
- **length**: number of MODBUS registers to request.

MODBUS READ Sample Config

```
{
  "name": "modbus_read",
  "loader": {
    "name": "native",
    "entrypoint": {
      "module.path": "./modules/modbus_read/win32/modbus_read.dll"
    }
  },
  "args": [
    {
      "serverConnectionString": "192.168.8.201",
      "serverConnectionPort": "502",
      "interval": "2000",
      "macAddress": "01:01:01:01:01:01",
      "baudRate": "9600",
      "stopBits": "1",
      "dataBits": "8",
      "parity": "EVEN",
      "flowControl": "NONE",
      "deviceType": "powerMeter",
      "sqliteEnabled": "0",
      "operations": [
        {
          "unitId": "1",
          "functionCode": "3",
          "startingAddress": "1",
          "length": "2"
        },
        {
          "unitId": "1",
          "functionCode": "3",
          "startingAddress": "3",
          "length": "1"
        }
      ]
    }
  ]
},
```

Figure 5: MODBUS Read sample config for MODBUS TCP device with 2 operation

2.3. Configure the MODBUS to XBUS module to properly parse the MODBUS Register Value

Once the MODBUS Read module is reading registers from the device, there is still no context to this data. To be able to map the register readings, parse and relate this to a Xompass Tag, you need to configure the MODBUS_TO_XBUS accordingly.

For each "DeviceType" configured in the MODBUS_READ module, you must provide the following mandatory fields to the "tags" array. Each element of the "tags" array, uniquely maps a set of Modbus register readings to a Xompass Tag, and provides its corresponding data type parsing information.

A configuration example is provided in Figure 8.

Every tag element must have the following fields:

- ***tag_name**: Xompass Tag Identifier
- registers: Array of registers that contains the register data. E.g: a float32 data field will be composed of two registers. The registers provided to the settings **must be in order**, for the parsing to work properly.
- ****data_type**: Data type of the Tag. This is used to know which datatype parsing will be used over the registers bytes. Datatypes available:
    - FloatBE
    - UInt16BE
    - Int16BE
- *****AID**: Xompass Asset ID.

* The Tag Id can be found in Xompass Web Application/Asset Contextualization/Asset/Tags, in the "Identifier" Column.

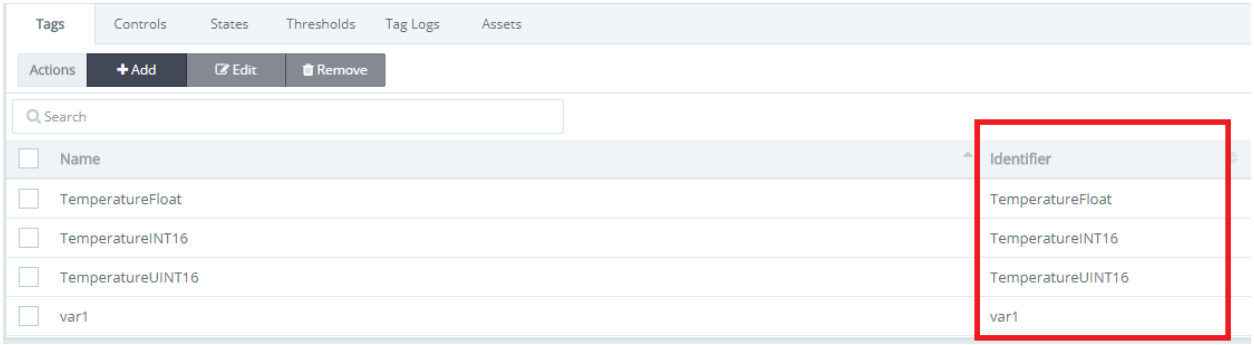

Figure 6: Xompass Tag Identifier of Asset, found in Asset Contextualization section

** If the datatype required does not exist, a new datatype can be created in the modules/modbus_to_xbus.js file (line 92) this module is in node.js to simplify source code modifications. The customized version of this file must be loaded with the option -v in the docker run command, as shown in the installation instructions. You can find the file here.

*** The Xompass Asset ID can be found on the URL of Xompass Web Application/Asset Contextualization/Asset on the URL

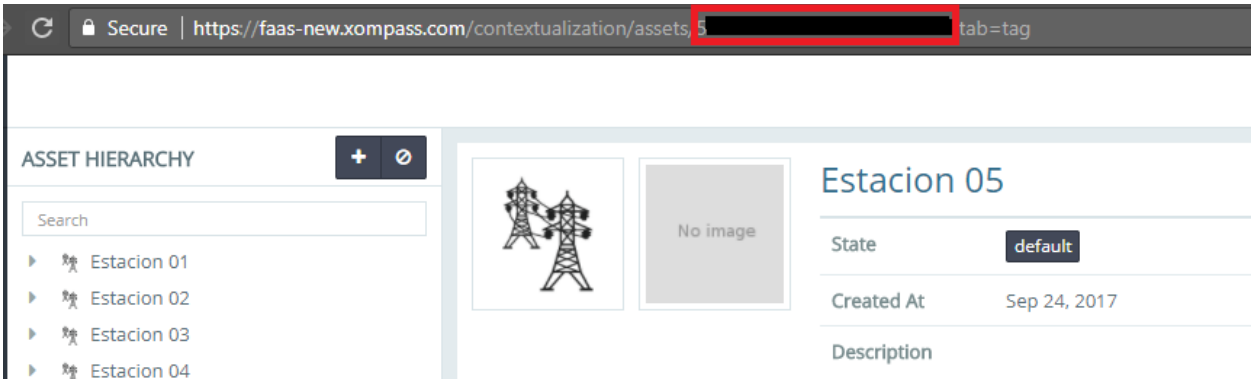

Figure 7: Asset ID on the Asset URL in Asset Contextualization section.

MODBUS_TO_XBUS Module Sample config

```
{
  "name": "modbus_to_xbus",
  "loader": {
    "name": "node",
    "entrypoint": {
      "main.path": "./modules/modbus_to_xbus.js"
    }
  },
  "args": {
    "powerMeter": {
      "tags": [
        {
          "tag_name": "TemperatureFloat",
          "registers": [40001, 40002],
          "data_type": "FloatBE",
          "AID": "5a154ffada26ae1dcfa78ce8"
        },
        {
          "tag_name": "TemperatureUINT16",
          "registers": [40003],
          "data_type": "UInt16BE",
          "AID": "5a154ffada26ae1dcfa78ce8"
        },
        {
          "tag_name": "TemperatureINT16",
          "registers": [40003],
          "data_type": "Int16BE",
          "AID": "5a154ffada26ae1dcfa78ce8"
        }
      ]
    }
  }
}
```

Figure 8: Sample MODBUS_TO_XBUS settings for "powerMeter" device, mapping 4 registers to 3 Xompass Tags with different data type parsing modes.

3. Install the software a local service or daemon

For this step, you must have Docker installed in your platform (Linux/MacOS/Win32). If you want to enable xompass-edge-base-module to start when your system starts, make sure Docker does.

Instructions

After installing docker, enable the docker service to always enable.

```
sudo systemctl enable docker
```

1 line command exmaple:

```
docker run -d --name <container-name> --restart always -v <absolute-path-to-gw.config.json>:/module/gw.config.json xompass/edge-base-module:<arch>
```

The above command will keep running the container <container-name> in the background and will restart every time Docker starts.

Note: This will automatically download the Docker image if it does not already exist in your system.

Parameters

- `<container-name>`: Name of the container running in Docker. This is useful for future reference.
- `<absolute-path-to-gw.config.json>`: Self-explanatory. If this argument is not passed it will use the bundled configuration file (which, for the moment, will not work in Windows systems).
- `-d`: Runs container as a daemon.
- `--restart always`: Starts the container with Docker.
- <arch>: architecture, currently supported tags: **latest-linux-64** and **latest-arm**

When using modified modbus_to_xbus.js file 1 line command

```
docker run -d --name <container-name> --restart always -v <absolute-path-to-gw.config.json>:/module/gw.config.json –v <absolute-path-to-modbus_to_xbus.js>:/module/modules/modbus_to_xbus.js xompass/edge-base-module:<arch>
```

Remove container (disable service)

```
docker rm <container-name or id>
```

Restart container

```
docker restart <container-name or id>
```

Monitor container logs

```
docker logs –f <container-name or id>
```

4. Changelog

Version	Date	Author	Detail
1.0	12-Feb-18	Andres Ulloa	Document Created
1.1	13-Feb-18	Rodolfo Castillo M.	Add service setup instructions as Docker containers
1.2	13-Feb-18	Andres Ulloa	Removed obsolete instructions.