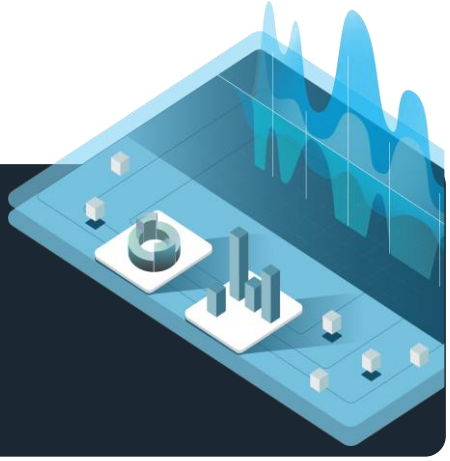
Bitpool®

enable real data

Niagara N4

MQTT Driver Documentation

July 2023



Revision History

Version	Date	Document By	Reviewed By	Status
1.0	6/07/2023	Ethan Jones		For Comments

Table of Contents

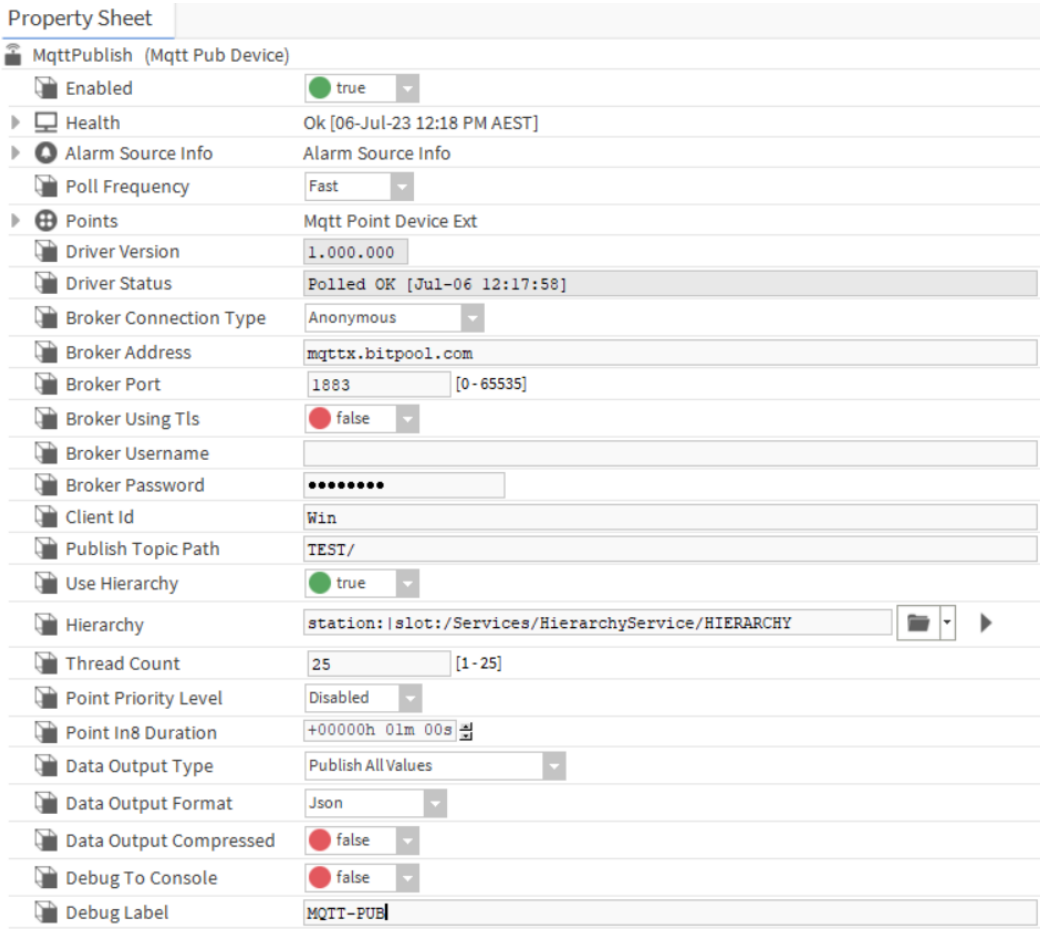
1. Introduction

The following documentation outlines the process for installing the BITPOOL N4 MQTT Device Driver on to a Niagara N4 Workstation using the standard Workbench toolset. This software module has been designed specifically to operate with Niagara N4 version framework and is not supported with older Niagara AX based systems.

2. Quick Start

Use the following procedure to get the device running with only key information.

1. Install driver.
2. Add Network and MQTT Subscribe Device from the Palette.
3. Configure Device by entering Broker Address, Port, Credentials
4. Configure desired output, including format and if using hierarchy. Save.
5. Done, now check the Points folder to see new data from the Broker.



If the above process fails to create new points, read the following documentation to assist with troubleshooting.

3. Setup

Use the following procedure to install MQTT driver on a Niagara N4 installation. Please note, this driver will not run on previous AX versions.

3.1 Driver Module Install

Before starting, make sure you have the latest version of the N4 MQTT driver.

1. Copy the 'mqtt-rt.jar' file to the Niagara modules directory on your N4 installation. Typically, this is located;

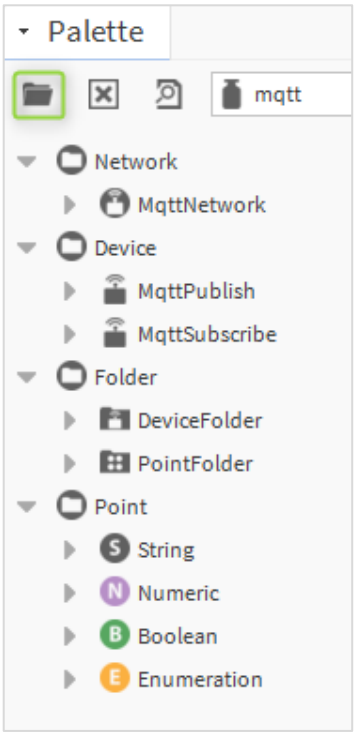
    i. C:\Niagara\Niagara-4.n.nn\modules

2. Using Niagara N4 Workbench, connect to the station's platform service and install the 'mqtt' module using the configuration page. Click the upgrade option if the module already exists.

3. With the MQTT device driver now installed, continue to the Configuration section to begin setting up the network and associated devices.

4. Configuration

Open Workbench and connect to the station, next locate the Palette (highlighted green in image) and search for module 'bMqtt'. Select and 'OK' to continue.



4.1 Network

The MQTT Network is the highest level in the folder structure and will always reside under this section in a station's hierarchy.

i. Station->Config->Drivers

Typically, device polling, monitoring and tuning policies are managed from this level. This ensures the correct behaviour of each device.

Note, multiple networks and associated devices can reside on a single station instance, however a practical implementation is dependent on the available system resources of the underlying hardware device.

4.2 Device

Both the MQTT Publish and the MQTT Subscribe device drivers are available to be placed under the network layer. Multiple instances of each device can be added; this is depended on the processing capability of the system.

4.3 Folder

Typically, the Device and Point folders are not used directly by the user. The folder structure is managed by each respective device driver a runtime.

(I) Subscribe

Once a MQTT Subscribe device driver is connected to a broker service, it creates a folder hierarchy below the 'Points' sub-folder based on the topics under the topic path. *Refer the 'Subscribe Topic Path' definition*.

(II) Publish

Since the MQTT Publish device driver is publishing topic/value pairs to a broker from another points folder on the local station, nothing will be displayed under the device's points folder. Refer the 'Publish Topic Path' and 'Publish Points Folder' definitions.

4.4 Point

Points in the Niagara framework are the key placeholders for data throughout the system. There are 4 basic types;

- **String**: Represents one or more ASCII characters. Note, the use of structured data (e.g. JSON) string variables to define a topic value is possible and discussed in more detail later in this document. *Refer the 'JSON String Handling' section.*

- **Numeric:** Represents an analog value, such as a temperature, either floating point or integer values. Numeric points will be added by the device driver if successfully converted when the point is first created. Typical string examples that can be converted are '-123.456', '888999123', '1', '0'.

- **Boolean** Represents a binary value with only two states, such as off or on. Boolean points will be added by the device driver if successfully converted when the point is first created. A 'true' or 'false' string is a typical example however '1' or '0' is considered a numeric.

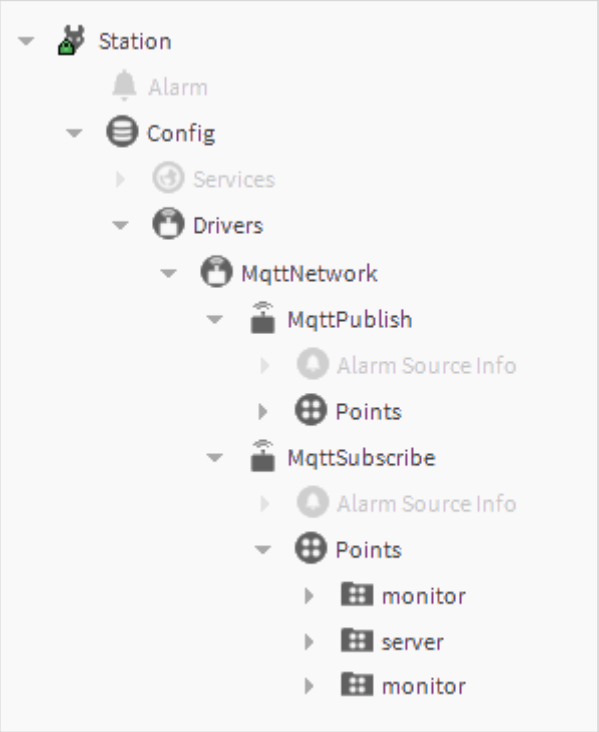
- **Enumeration**: Represents an enumerated state (more than two) e.g. off, slow, and fast. Enumerated values cannot be represented in a single unstructured string defined by the MQTT Broker, so these points will never be created by a MQTT Subscribe device driver. However, if the MQTT Publish device driver is used to publish points using a 'Data Output Format' JSON (*Refer the 'Data Output Format' definition*), then any MQTT Subscribe device driver subscribing back on these points will have the enumeration correctly set.

Like folders, these components will automatically be created as topic values are received from the broker.

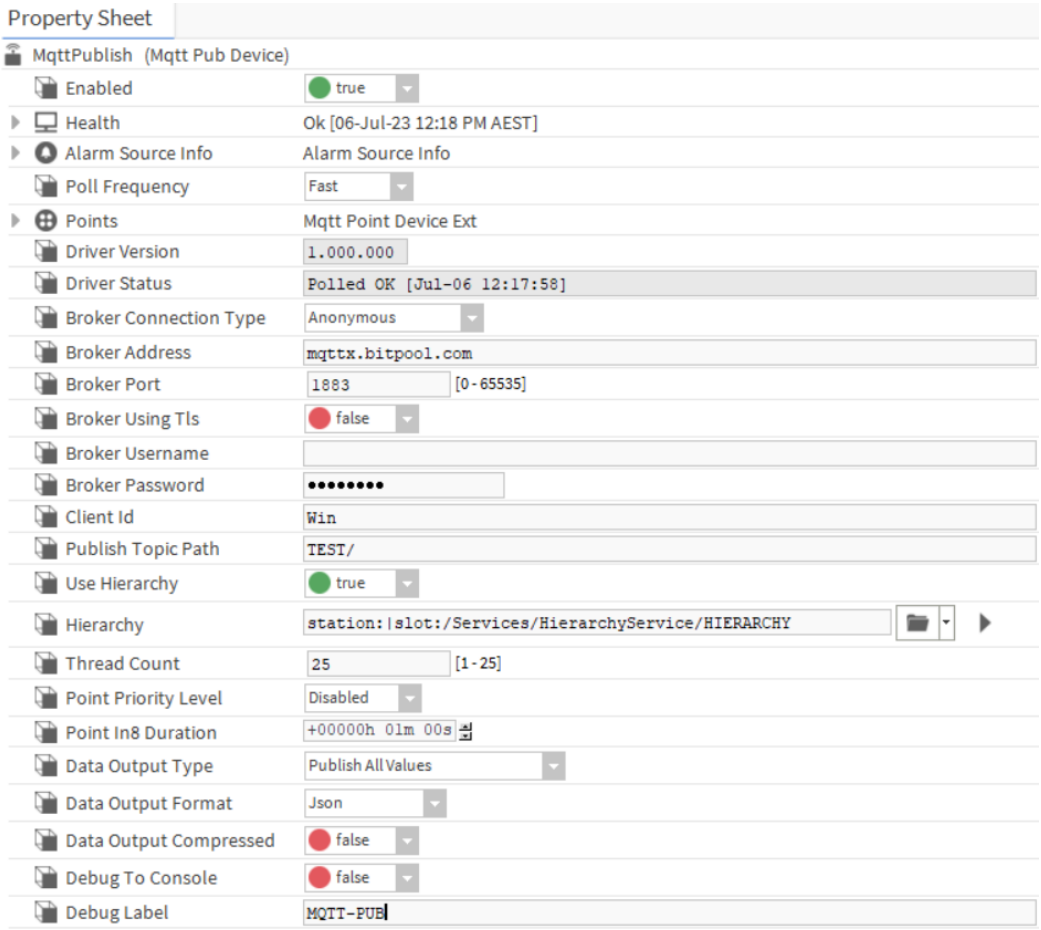
4.5 Build the Database

Drag and drop the individual components from the Palette to build the required database structure. Follow the dialog prompts for naming each component, then configure each device using the documentation in the following sections.

Note, this example shows folders below the 'Points' folder. These are only created once the Subscribe device is communicating correctly with the broker.



4.6 Publish Configuration Options



The following documentation defines the MQTT Publish device driver configuration options.

- **Driver Version:** Identifies the currently installed driver version.
- **Driver Status:** Provides a time-stamped text update of driver progress.
- **Broker Connection Type:** There are two methods for connecting to the broker.
  - Anonymous: The driver will connect to the broker service without a username or password.
  - Using Credentials: Forces the driver to send the username and password as credentials to the broker.
- **Broker Address:** The driver will connect to the broker on a valid domain name or IP address.

- **Broker Port:** The driver will connect to the broker on a valid port, *typically this is port 1883*.

- **Broker Username:** If '*Using Credentials*', the username will be sent to the broker.

- **Broker Password:** If '*Using Credentials*', the password will be sent to the broker.

- **Broker Password:** ClientID to be sent to the broker. (Brokers running 3.1 may have a limit of 23 characters, please allow an additional 4 characters for the thread identifiers)

- **Publish Topic Path:** This string prefixes the '*Publish Points Folder*' path, before being sent to the broker root topic. *Note, this string supports Niagara BFormat syntax (e.g. %displayName%).*

- **Publish Points Folder:** This is the points folder location on the station. In Niagara terms, this is called the Object Resolution Descriptor (ORD). *Note, it is important for the reader to understand the relationship that exists between Topics, ORD's and folders. Refer the 'Topics vs ORD.s' discussion. (Publish point path only valid in non-hierarchy base publish)*

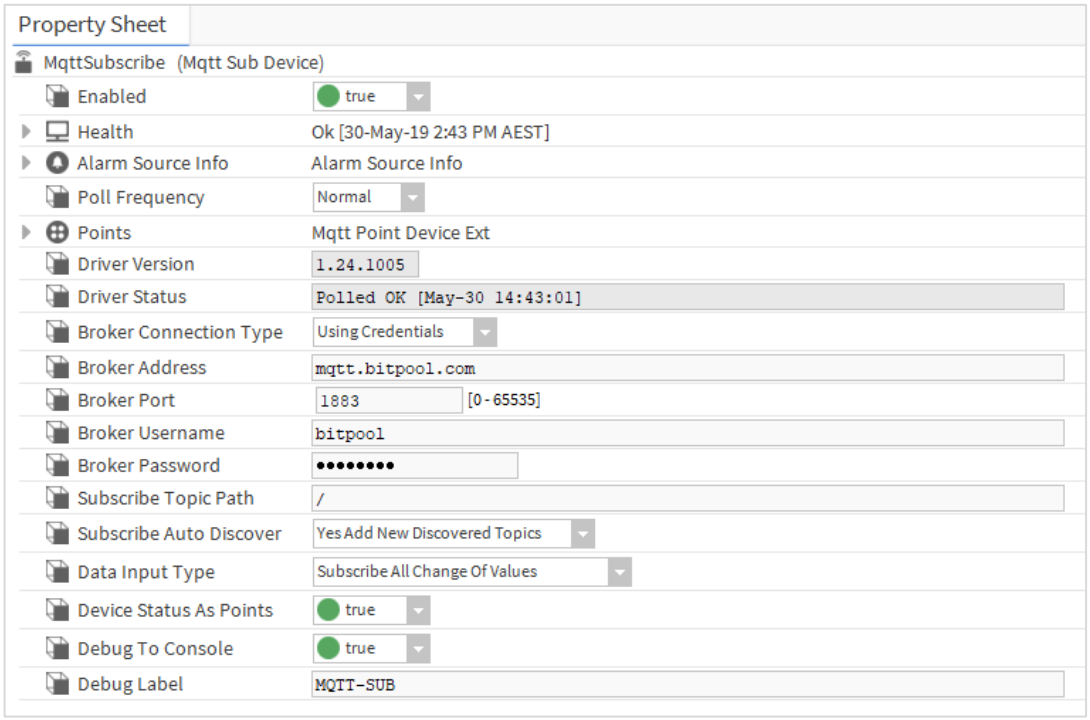
- **Data Output Type:** Output data (Niagara point data) associated with the ORD is handled before being sent to the Broker service as a topic-value pair.
    - Publish All Values: All point data is sent by the device driver.
    - Publish Only Changed Values: Only changed point data is sent by the device driver.

- **Data Output Format:** Output data is formatted before being sent to the broker service. Either as a discrete point value string or are structured JSON string.
    - Value: All point data is sent by the device driver as a discrete text value.
    - Json: All point data is sent by the device driver as a structured JSON text payload. This is verbose representation of the point, with comprehensive metadata set.
    - Json Simple: This is simple JSON version of the point data, containing only the value and UTC date (seconds).

- **Debug to Console:** Output all driver run-time messages to the Platform Console.

- **Debug Label**: Prefix all debug messages with this string. This will assist in identifying multiple devices output messaging in the console window.

4.7 Subscribe Configuration Options



The following documentation defines the MQTT Subscribe device driver configuration options.

- **Driver Version:** Identifies the currently installed driver version.

- **Driver Status:** Provides a time-stamped text update of driver progress.

- **Broker Connection Type:**: There are two methods for connecting to the broker.
    - Anonymous: The driver will connect to the broker service without a username or password.
    - Using Credentials: Forces the driver to send the username and password as credentials to the broker.

- **Broker Address:** The driver will connect to the broker on a valid domain name or IP address.

- **Broker Port:** The driver will connect to the broker on valid port, *typically this is port 1883*.

- **Broker Username:** If '*Using Credentials*, the username will be sent to the broker.

- **Broker Password:** If '*Using Credentials*', the password will be sent to the broker.

- **Subscribe Topic Path:** This string identifies the broker topic this device will accept messages from. *Note, this string supports Niagara BFormat syntax (e.g. %displayName%).*

- **Data Input Type:** Input data associated with a broker topic is handled before received into the system.
    - Subscribe All Change Of Values: All topic data is imported when detected by the device driver.
    - Subscribe Only Last Change Of Value: Only the last (most recent) topic's data is received by the system.

- **Device Status as Points:** If enabled, the folder '*mqttDeviceStatus*' is added below the '*Points*' folder and provides a list of points to identify the driver's current status.

- **Debug to Console:** Output all driver run-time messages to the Platform Console.

- **Debug Label:** Prefix all debug messages with this string. This will assist in identifying multiple devices output messaging in the console window.

4.8 Topics vs Ord's

In order to join a Niagara system with a MQTT broker, a relationship must exist between a broker topic and the Niagara ORD (Object Resolution Descriptor). This relationship is managed by the device driver.

The MQTT protocol defines a structured hierarchical system to define topics (similar to a standard file-system path, of folders and files).

i. /house/room/light

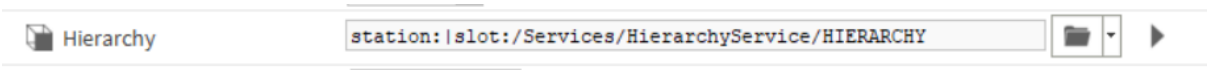
The Niagara system also defines a structured hierarchical protocol for point location using an ORD.

ii. /points/house/room/light

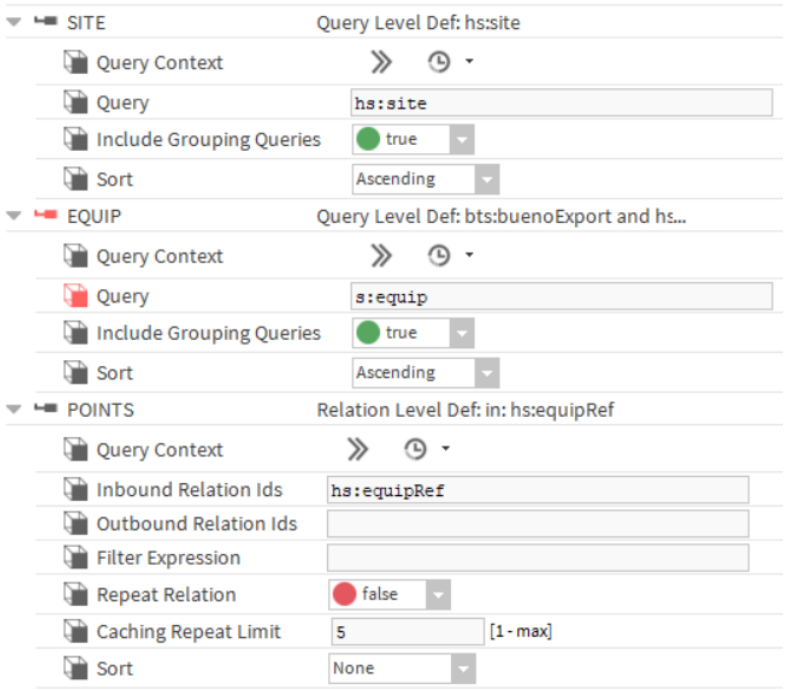
To join these two paths together, a map is created as new topics arrive. The map facilitates the movement of data between each side; either publishing ORD point data to the broker topic or receiving broker topic values back to the subscribed ORD points.

4.9 Use Hierarchy

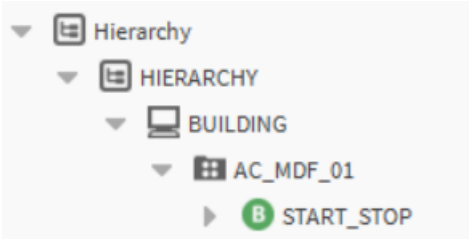
If the "Use Hierarchy" configuration option is enabled, the Topic path deviates from the default Ord based structure. The Ord is replaced with the resolved Hierarchy Path from the Niagara Hierarchy Service.



This Hierarchy resolves into a simple hierarchy structure, featuring a base level using the *siteRef*, then the *equipTag* and then the *equipRef.*



This will build out a hierarchy showing the "Site, Equipment, then Points"



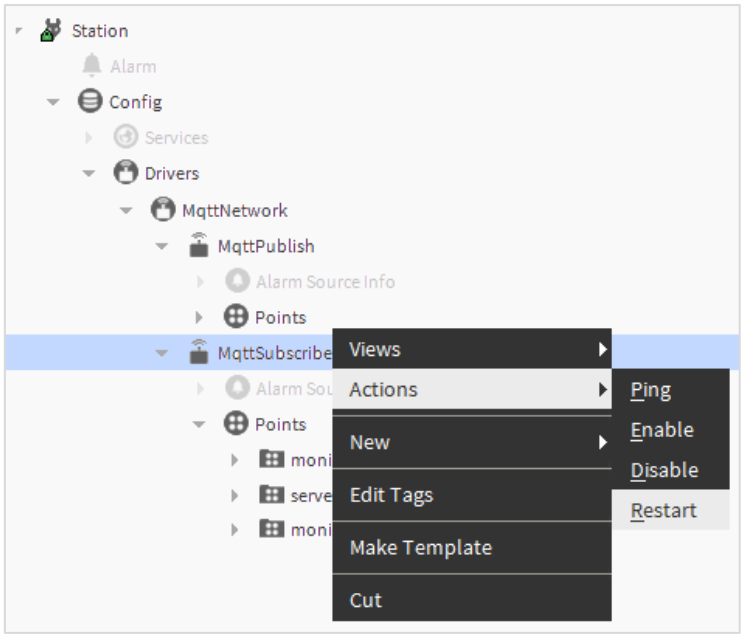
This will replace the relevant topic with the same structure, resulting in a topic of: *TEST/BUILDING/AC_MDF_01/START_STOP*

4.10 Device Restart

The driver can be enabled / disabled using the device configuration page,

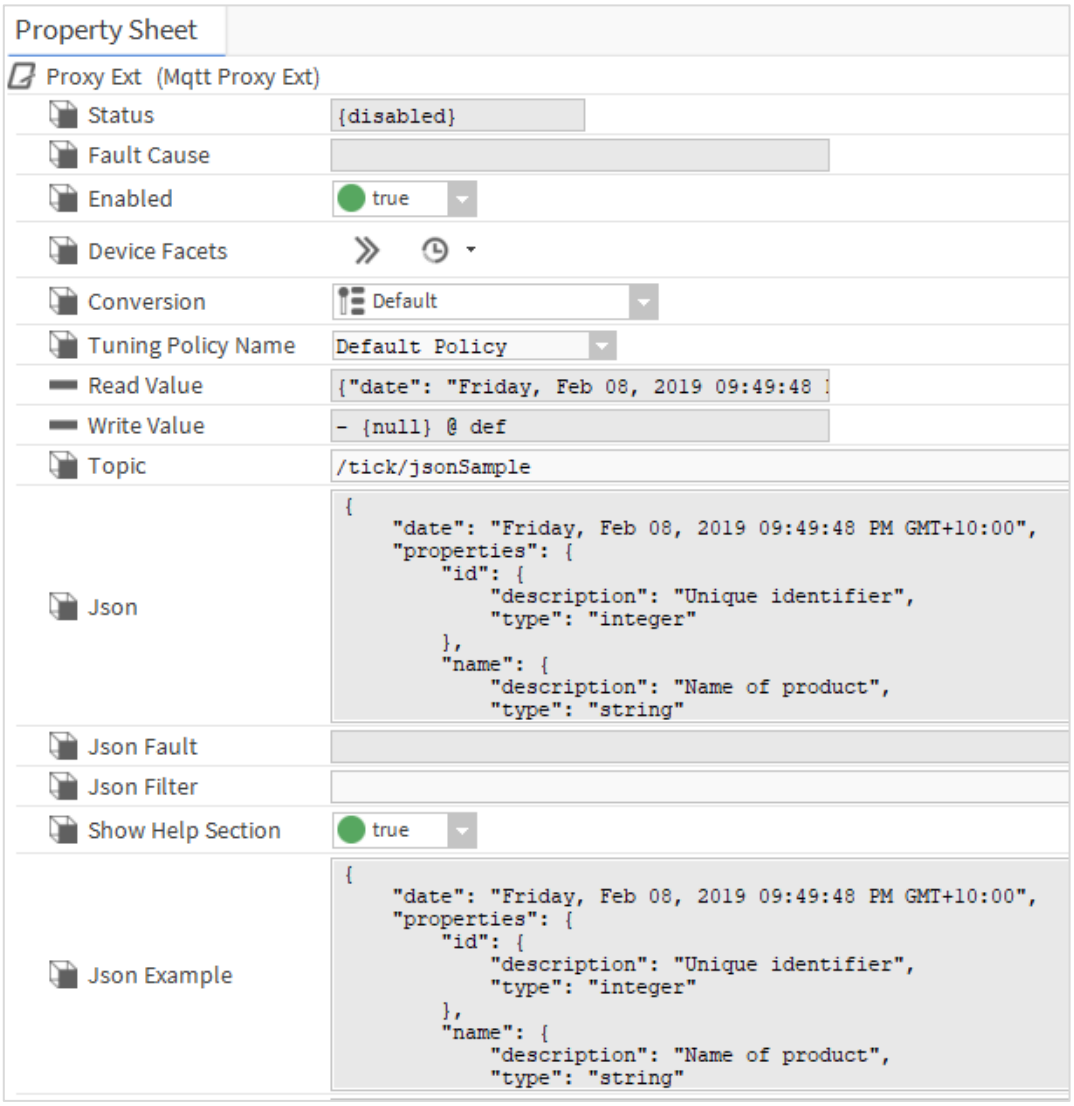
i. Station->Config->Drivers->MqttNetwork->MqttSubDevice

It can also be restarted using the right click context menu as shown here.



4.11 JSON Point Strings

If a received topic value contains a structured JSON string, the device driver creates a string point with additional properties.



These properties are included to allow discrete control over the point's output value and are shown in this image.

- **JSON:** A large scrollable text field is included in the string configuration page, to better visualise the full JSON structure.
- **JSON Fault:** This field is used to output error messages when a user enters an invalid filter into the '*JSON Filter*' property. This is a good way to check that the filter expression is correctly formatted.

- **JSON Filter:** This allows the user to enter a string expression which can specifically target individual element or elements of the JSON object. *Refer the 'JSON Filtering' section.*

- **Show Help Section:** Show quick filter 'How-to' examples. Select 'True', then Save and Refresh buttons to see more help information.

4.12 JSON Filtering

When a topic value is received in JSON format, a string point is created with additional properties. These properties allow for filtering of the JSON structure to a discrete output value. This new resultant value is then used as the new output of the string point.

Identified in the '*JSON Point Strings*' section, an input field allows the user to enter a filter expression. In its basic form, the expression represents a pathway through the JSON object.

For example;

If the user looks to find the 'color' of an 'apple' using this JSON object, the following expression would be entered.

ii. fruit.apple.color

The result of 'red' would now replace the string property output value.

4.13 Advanced Filtering Duplicates

By now duplicating the string point (whilst keeping the 'Topic' property the same), the user can now build up a list of individual string points which representing an individual discrete element of the JSON block.

This is a useful technique for deconstructing complex data structures in to individual components for later analysis or data manipulation.

4.14 Debug Console

This feature can be enabled using the device configuration page and is configurable per device.

i. Station->Config->Drivers->MqttNetwork->MqttPubDevice

ii. Station->Config->Drivers->MqttNetwork->MqttSubDevice

An additional option to prefix each message with a user defined string is also configurable. Shown here is a typical example of time-stamped messages sent to the Platform Console screen.

```
[May-30 14:42:48][MQTT-PUB]
[May-30 14:42:48][MQTT-SUB] MQTT SUBSCRIBE DRIVER [1.24.1005]
[May-30 14:42:48][MQTT-SUB] Debugging to console enabled
[May-30 14:42:51][MQTT-SUB] Device disabled
[May-30 14:42:53][MQTT-PUB] Polling <PUB>
[May-30 14:42:53][MQTT-PUB]  Published Broker Topic: <mqtt.bitpool.com/>
[May-30 14:42:53][MQTT-PUB]   Source ORD: <station:|slot:/Drivers>
[May-30 14:42:53][MQTT-PUB]   All Points: 36
[May-30 14:42:53][MQTT-PUB]   Changed Points: 36
[May-30 14:42:53][MQTT-PUB]   Output Type:  <Send ALL values>
[May-30 14:42:53][MQTT-PUB]    Published: 36 <mqtt.bitpool.com><OutputFormat=VALUE>
[May-30 14:42:53][MQTT-PUB] Polled OK
[May-30 14:42:53][MQTT-PUB]
[May-30 14:42:58][MQTT-PUB] Polling <PUB>
[May-30 14:42:58][MQTT-PUB]  Published Broker Topic: <mqtt.bitpool.com/>
[May-30 14:42:58][MQTT-PUB]   Source ORD: <station:|slot:/Drivers>
[May-30 14:42:58][MQTT-PUB]   All Points: 36
[May-30 14:42:58][MQTT-PUB]   Changed Points: 36
[May-30 14:42:58][MQTT-PUB]   Output Type:  <Send ALL values>
[May-30 14:42:58][MQTT-PUB]    Published: 36 <mqtt.bitpool.com><OutputFormat=VALUE>
[May-30 14:42:58][MQTT-PUB] Polled OK
[May-30 14:42:58][MQTT-PUB]
[May-30 14:43:00][MQTT-SUB] MQTT configuration has changed, restarting connection
[May-30 14:43:00][MQTT-SUB] MQTT SUBSCRIBE DRIVER [1.24.1005]
[May-30 14:43:00][MQTT-SUB] Configuring MQTT client
[May-30 14:43:00][MQTT-SUB]  Connecting to Broker: <mqtt.bitpool.com>
[May-30 14:43:00][MQTT-SUB]  Using credentials for user: bitpool
[May-30 14:43:00][MQTT-SUB]   Connected OK
[May-30 14:43:00][MQTT-SUB]    Using Topic Source: </>
[May-30 14:43:00][MQTT-SUB]
```

5. Further Reading

1. Public MQTT Brokers
    a. https://github.com/mqtt/mqtt.github.io/wiki/public_brokers

2. Understanding the MQTT Protocol
    a. https://mosquitto.org/

    b. http://mqtt.org/

    c. https://docs.oasis-open.org/mqtt/mqtt/v3.1.1/mqtt-v3.1.1.html

    d. https://www.cloudmqtt.com/docs/index.html

    e. http://www.steves-internet-guide.com/

3. JSON Filtering
    a. https://github.com/json-path/JsonPath