

Technical Document

Abstract MQTT Driver Guide

March 3, 2023

niagara⁴

Abstract MQTT Driver Guide

Tridium, Inc.

3951 Westerre Parkway, Suite 350
Richmond, Virginia 23233
U.S.A.

Confidentiality

The information contained in this document is confidential information of Tridium, Inc., a Delaware corporation ("Tridium"). Such information and the software described herein, is furnished under a license agreement and may be used only in accordance with that agreement.

The information contained in this document is provided solely for use by Tridium employees, licensees, and system owners; and, except as permitted under the below copyright notice, is not to be released to, or reproduced for, anyone else.

While every effort has been made to assure the accuracy of this document, Tridium is not responsible for damages of any kind, including without limitation consequential damages, arising from the application of the information contained herein. Information and specifications published here are current as of the date of this publication and are subject to change without notice. The latest product specifications can be found by contacting our corporate headquarters, Richmond, Virginia.

Trademark notice

BACnet and ASHRAE are registered trademarks of American Society of Heating, Refrigerating and Air-Conditioning Engineers. Microsoft, Excel, Internet Explorer, Windows, Windows Vista, Windows Server, and SQL Server are registered trademarks of Microsoft Corporation. Oracle and Java are registered trademarks of Oracle and/or its affiliates. Mozilla and Firefox are trademarks of the Mozilla Foundation. Echelon, LON, LonMark, LonTalk, and LonWorks are registered trademarks of Echelon Corporation. Tridium, JACE, Niagara Framework, and Sedona Framework are registered trademarks, and Workbench are trademarks of Tridium Inc. All other product names and services mentioned in this publication that are known to be trademarks, registered trademarks, or service marks are the property of their respective owners.

Copyright and patent notice

This document may be copied by parties who are authorized to distribute Tridium products in connection with distribution of those products, subject to the contracts that authorize such distribution. It may not otherwise, in whole or in part, be copied, photocopied, reproduced, translated, or reduced to any electronic medium or machine-readable form without prior written consent from Tridium, Inc.

Copyright © 2023 Tridium, Inc. All rights reserved.

The product(s) described herein may be covered by one or more U.S. or foreign patents of Tridium.

Contents

Preface.....	5
About this Guide.....	5
Document change log	5
Related Documentation.....	6
Chapter 1 Getting started	7
Requirements.....	8
Adding the network	8
Adding a device and authenticator.....	9
Setting up client certificate authentication.....	9
Discovering points.....	10
Adding points manually	12
Publishing data	12
Chapter 2 Abstract MQTT Driver with GCP Authenticator	15
Setting up Google Cloud Platform (GCP).....	15
Setting up station to connect to GCP	16
Creating device in GCP.....	16
Sending messages from Niagara to GCP	17
Sending messages GCP Console to Niagara	17
Chapter 3 Abstract MQTT Driver with Microsoft Azure.....	19
Azure SAS Tokens	19
Connecting Abstract MQTT Driver to Microsoft Azure IoT Hub	19
Setting up Azure IoT Hub on Azure portal	19
Setting up station to connect to Azure IoT Hub	21
Chapter 4 Components.....	25
abstractMqttDriver-AbstractMqttDriverNetwork.....	25
abstractMqttDriver-AbstractMqttDevice	27
Aws Jitp Mqtt Authenticator (abstractMqttDriver- AwsJitpMqttAuthenticator)	30
abstractMqttDriver-AwsMqttAuthenticator	32
abstractMqttDriver-AzureMqttSasAuthenticator	33
abstractMqttDriver-AzureSasTokenParameters	35
abstractMqttDriver-AzureIotSasToken	35
abstractMqttDriver-GenericMqttAuthenticator	36
abstractMqttDriver-GcpAuthenticator.....	38
abstractMqttDriver-GcplotParameters	39
abstractMqttDriver-TokenParameters	40
abstractMqttDriver-MqttCallbackRouter	40
abstractMqttDriver-PointCallbackHandler	41
abstractMqttDriver-MqttClientDriverPointDeviceExt.....	41
abstractMqttDriver-MqttClientDriverDeviceFolder	41
abstractMqttDriver-MqttClientDriverPointFolder	41
abstractMqttDriver-MqttBooleanObjectPublishExt.....	42

abstractMqttDriver-MqttBooleanObjectSubscribeExt	44
abstractMqttDriver-MqttNumericObjectPublishExt	46
abstractMqttDriver-MqttNumericObjectSubscribeExt	48
abstractMqttDriver-MqttStringObjectPublishExt	50
abstractMqttDriver-MqttStringObjectSubscribeExt	53
abstractMqttDriver-MqttEnumObjectPublishExt	55
abstractMqttDriver-MqttEnumObjectSubscribeExt	57
Chapter 5 Plugins	61
Abstract Mqtt Driver Device Ux Manager	61
Mqtt Client Driver Point Manager	62
Chapter 6 Windows	65
New Device Windows.....	65
Add point windows	65
Index.....	69

Preface

About this Guide

This topic contains important information about the purpose, content, context, and intended audience for this document.

Product Documentation

This document is part of the Niagara technical documentation library. Released versions of Niagara software include a complete collection of technical information that is provided in both online help and PDF format. The information in this document is written primarily for Systems Integrators. To make the most of the information in this book, readers should have some training or previous experience with Niagara software, as well as experience working with JACE network controllers.

Document Content

This document describes how to create an Abstract MQTT (MQ Telemetry Transport) client connection to the Abstract MQTT broker for the Niagara station for the purpose of publishing or subscribing the “light-weight” messaging protocol into the Niagara Framework.

Document change log

Changes to this document are listed in this topic.

March 3, 2023

- Added “abstractMqttDriver-AwsJitpMqttAuthenticator” component (as of Niagara 4.13) to “Components” chapter.

April 6, 2022

- Added chapter “Abstract MQTT Driver with Gcp Authenticator”
- Added chapter “Abstract MQTT Driver with Microsoft Azure”. Included components: “abstractMqttDriver-AzureMqttSasAuthenticator”, “abstractMqttDriver-AzureSasTokenParameters”, “abstractMqttDriver-AzureIotSasToken”.
- Updated description for the `Topic` component property.

November 19, 2020

- Added the components that support the Google Cloud Platform authenticator
- Added [Adding points manually, page 12](#).
- Added [Setting up client certificate authentication, page 9](#).
- Minor edits in other topics.

April 1, 2020

Added topic for Authenticators and updated the guide for Niagara 4.9

May 10, 2018

Added two new component topics “MqttClientDriverDeviceFolder” and “MqttClientDriverPointFolder”.

October 16, 2017

- Added information about ‘Bql Query Builder’ in “Discovering AbstractMqttPoints” topic.
- Added two new topics “Discovering AbstractMqttPoint” and “Adding Points to the database”.

- Updated the component topics and few more topics in the chapter “Setup an AbstractMqttDriverNetwork”.

July 10, 2017

Updated for initial product release.

Related Documentation

The following documents relate to the AbstractMqtt Driver.

- *Niagara Drivers Guide*
- *Niagara Platform Guide*

Chapter 1 Getting started

Topics covered in this chapter

- ◆ Requirements
- ◆ Adding the network
- ◆ Adding a device and authenticator
- ◆ Setting up client certificate authentication
- ◆ Discovering points
- ◆ Adding points manually
- ◆ Publishing data

MQTT (MQ Telemetry Transport) driver supports a lightweight messaging protocol for use on top of the TCP/IP protocol driver installation. Its publish-subscribe architecture is designed to be open and easy to implement with up to thousands of remote clients capable of being supported by a single server.

Publish and subscribe

The MQTT protocol is based on the principle of publish and subscribe. Multiple clients connect to a broker and subscribe to topics that they are interested in. Clients also connect to the broker and publish messages to topics. MQTT restricts one subscriber to one topic.

Client

The term client refers either to a publisher or a subscriber. The MQTT client can be both a publisher and a subscriber at the same time. An MQTT client is any device from a micro controller up to a full-fledged server that has an MQTT library running and is connected to an MQTT broker over any type of network. The Niagara solution for MQTT configures only the client, and not the broker. Data types supported by the client are Boolean, string, numeric and enum.

Broker

The broker is the central hub for messages. Depending on the actual implementation, a broker may handle many concurrently-connected MQTT clients. The broker is primarily responsible for:

- Authenticating each client
- Receiving all messages
- Filtering messages
- Deciding which subscriber is interested in each message
- Sending each message to its subscribed client

Most brokers are extensible, providing easily integrated, custom authentication, authorization and integration into back-end systems. Integration is an especially important aspect because often the broker is the component that is directly exposed to the Internet. The broker handles multiple clients and passes messages to downstream analyzing and processing systems. The Niagara The MQTT client supports a TLS connection to a broker, also using login credentials.

Devices

The MQTT driver supports these devices:

- The **DefaultMqttDevice** represents a generic MQTTNiagara client, which you can use to connect to any broker.
- The **AwsMqttDevice** connects an MQTT client to the AWS IoT Hub.
- The **GcpMqttDevice** connects the MQTT client to the Google Cloud Platform.

Authenticators

The MQTT authenticators plug in to the MQTT device according to establish a secure connection between a broker and a client device. You choose which authenticator to use for each device.

The driver provides these authenticators:

- The generic **DefaultAuthenticator** serves most devices, providing ways to connect to a broker.
- The **AwsIoTAuthenticator** makes a secure connection to the AWS IoT (Amazon Web Services Internet of Things) using the certificate authentication.
- The **GcpAuthenticator** authenticates a secure connection between a client device and a broker on a Google Cloud Platform. It uses RSA keys.

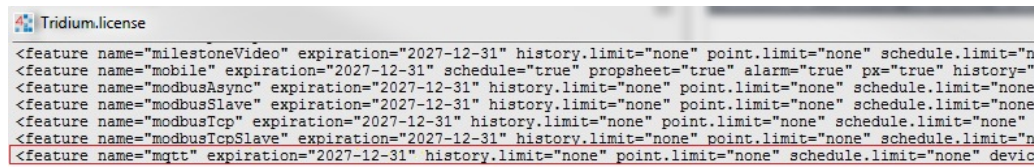
Requirements

This topic describes the licensing and software requirements for using the Niagara Abstract MQTT Driver.

License Requirements

The abstract **abstractMqttDriver** module is a licensed feature. You can check to see if your software installation is licensed for MQTT by opening the license file from the **License Manager** view. The feature name is present only if your platform is licensed for MQTT.

Figure 1 Example MQTT license with point limit attribute values set to “none”.



```

Tridium.license
<feature name="milestoneVideo" expiration="2027-12-31" history.limit="none" point.limit="none" schedule.limit="n
<feature name="mobile" expiration="2027-12-31" schedule="true" propsheet="true" alarm="true" px="true" history="
<feature name="modbusAsync" expiration="2027-12-31" history.limit="none" point.limit="none" schedule.limit="none
<feature name="modbusSlave" expiration="2027-12-31" history.limit="none" point.limit="none" schedule.limit="none
<feature name="modbusIcp" expiration="2027-12-31" history.limit="none" point.limit="none" schedule.limit="none"
<feature name="modbusIcpSlave" expiration="2027-12-31" history.limit="none" point.limit="none" schedule.limit="n
<feature name="mqtt" expiration="2027-12-31" history.limit="none" point.limit="none" schedule.limit="none" devic

```

Software Requirements

The Niagara Abstract MQTT Driver is available in all the latest versions of Niagara.

Adding the network

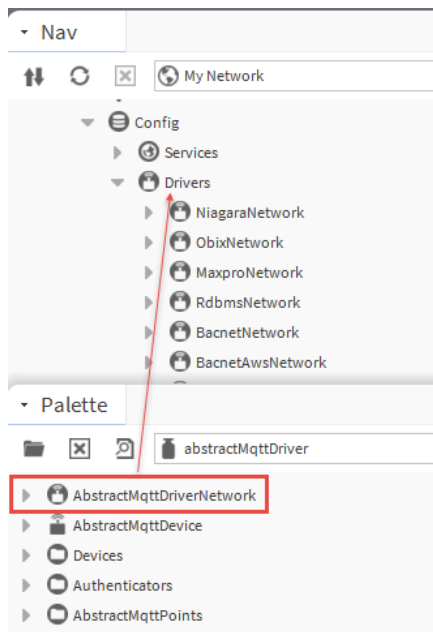
The **AbstractMqttDriverNetwork** manages all connected devices.

Prerequisites: You are working in Workbench and are connected to a controller station.

Step 1 Open the station and expand **Config→Drivers**.

Step 2 Open the **abstractMqttDriver** palette.

Step 3 Drag the **AbstractMqttDriverNetwork** to the **Drivers** node in the Nav tree.



The **Name** window opens.

- Step 4 Accept the default name or give your instance of the **AbstractMqttDriverNetwork** a more meaningful name, as desired, and click **OK**.

Adding a device and authenticator

A device can be any equipment. The authenticator provides secure communication between an MQTT broker and an MQTT client in the device.

Prerequisites: The network is installed and the abstractMqttDriver palette is open.

- Step 1 Open the station and expand **Config→Drivers**.

- Step 2 Do one of the following:

- Drag the device component from the palette to the network node in the Nav tree, name the device and click **OK**.
- Double-click the network node, click **New**, select the **Abstract Mqtt Device** from the drop-down list, click **OK**, name the device and click **OK**.

You may use the preconfigured devices present under the **Devices** folder. They are already configured with an authenticator.

- Step 3 To configure the device, right-click it in the Nav tree and click **Views→AX Property Sheet**.

- Step 4 To add an authenticator, drag it from the palette to the device node under the network.

- Step 5 To configure the authenticator, right-click it in the Nav tree and click **Views→AX Property Sheet**.

Setting up client certificate authentication

AWS and GCP devices use client certificate authentication to encrypt communication and authenticate the broker to the client. AWS IoT requires the MQTT client to register with AWS IoT, which generates a client certificate. GCP devices use RSA keys. After acquiring your client certificate, this procedure documents how to import it into the **User Key Store**.

Prerequisites: You are working in Workbench running on a PC and are connected to a controller station.

Step 1 Using a secure channel, download the certificate from the AWS IoT portal or Google Cloud Platform to your PC.

CAUTION: Always share certificates over a secure channel.

Step 2 Working in the station, expand **Config**→**Services**→**PlatformServices** and double-click **CertManagerService**.

The **Certificate Management** view opens.

This same view is available by expanding **Platform** and double-clicking **Certificate Management**.

Step 3 At the bottom of the view, click the **Import** button and navigate to where you saved the certificate.

Step 4 After selecting the certificate, click **Open**.

The next step associates this certificate with the device.

Step 5 Expand **Config**→**Drivers**→**AbstractMqttDriverNetwork**.

Step 6 Do one of the following:

- If you are using the AWS IoT cloud, expand **AwsMqttDevice** and double-click **authenticator**.
- If you are using the Google Cloud Platform, expand **GcpMqttDevice**→**authenticator** and double-click **Token Parameters**.

Step 7 To select the client certificate, use the **Certificate Alias** drop-down list.

You are ready to connect your client to the broker.

Discovering points

You can add points individually or discover local points under each device. Discovery includes writing a BQL (Baja Query Language) expression to locate the points in the station.

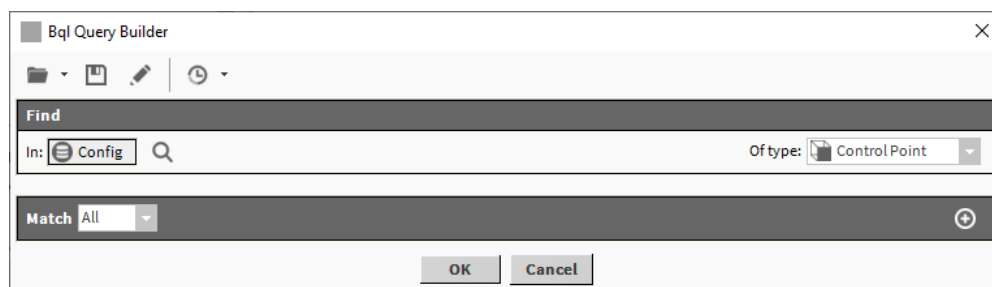
Step 1 In the Nav tree, expand the **Config**→**Drivers**→**AbstractMqttDriverNetwork** and expand the device.

Step 2 To view the point manager, right-click the **Points** node and click **Views**→**Mqtt Client Driver Point Manager** or double click the **Points** node.

The **Mqtt Client Driver Point Manager** opens.

Step 3 Click the **Discover** button.

The **Bql Query Builder** window opens.

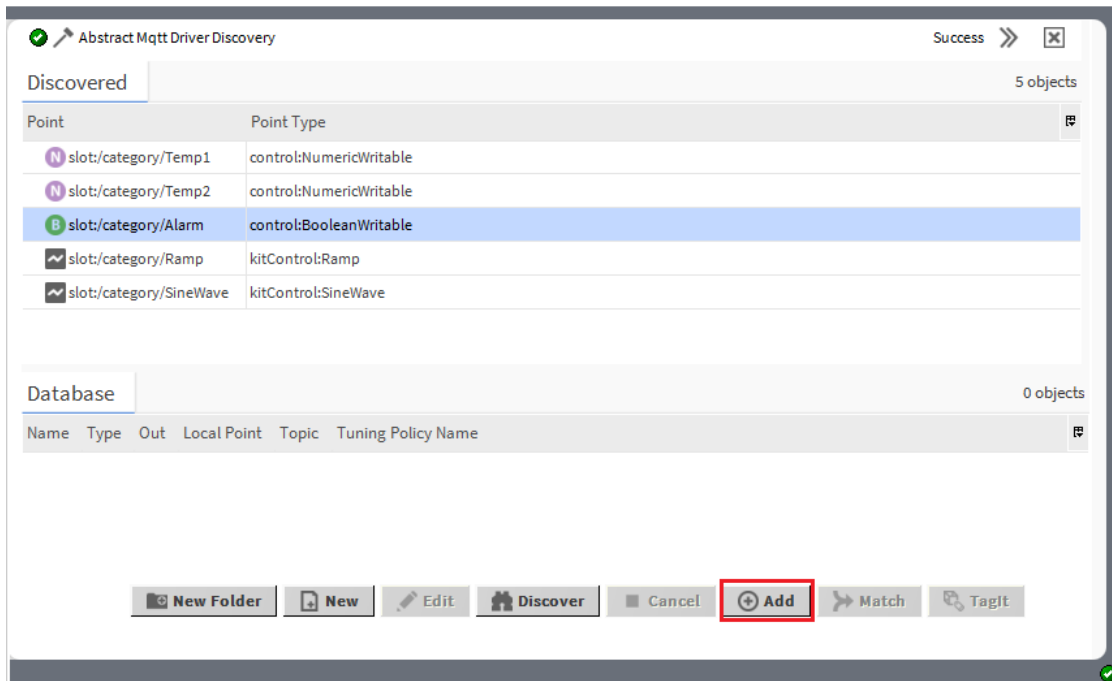


Step 4 To define a BQL query, click the search icon (🔍).

The **Choose Root** window opens.

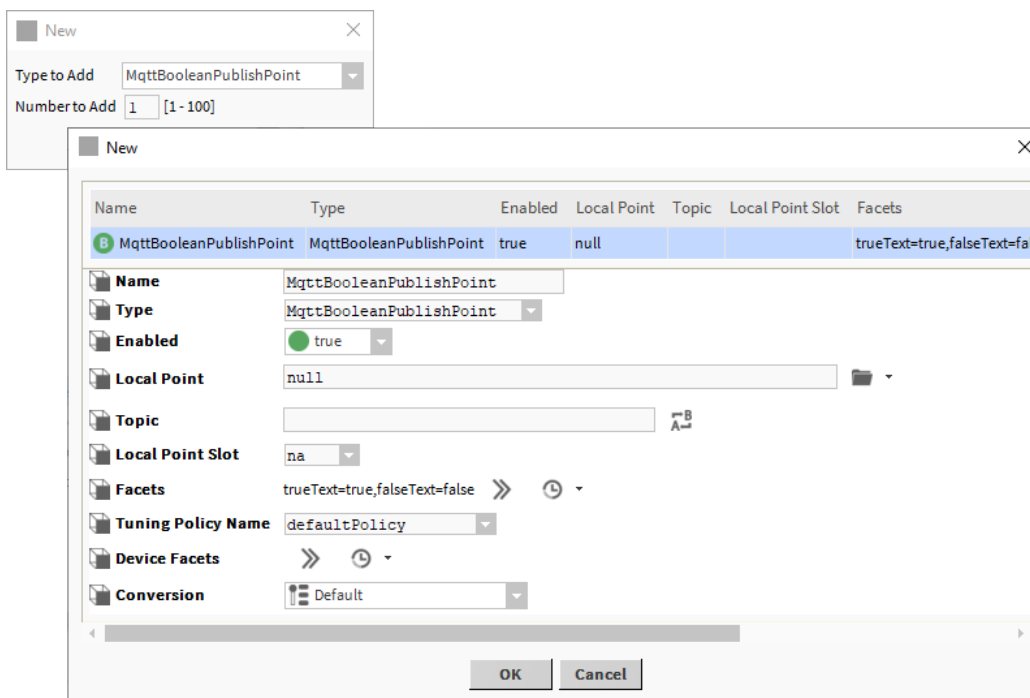
Step 5 Navigate to the location of points that are available in the station to discover (**In:** property), select the type of point from the **Of type:** drop-down list and click **OK**.

The discovery job runs, populating the **Discovered** pane with points.



Step 6 To add a discovered point to the station database, select the point(s) in the **Discovered** and click **Add**.

The **Add** window opens.



Step 7 Give the point a **Name**, configure the other properties as needed and click **OK**.

The added point begins updating with the real-time values.

Adding points manually

You can add individual points manually using the palette or **New** button in the **Mqtt Client Driver Point Manager**.

Prerequisites: The station is open; network and devices have been added.

Step 1 Expand **Config**→**Drivers**→**AbstractMqttDriverNetwork** and expand the device.

Step 2 Expand the **AbstractMqttPoints** folder in the palette.

Step 3 Do one of the following:

- Drag a point from the palette to the **Points** node in the Nav tree, name the point and click **OK**.
- Double-click the **Points** folder, click **New**, select the **Type** to **Add** and **Number** to **Add** from the drop-down lists and click **OK**.

If you used the **New** button, the software opens the **Add** point window.

Step 4 If the **Add** point window is open, configure the point(s) and click **OK**.

Publishing data

To publish data to the broker, you link a data point output value to an MQTT publish point of the proper data type. This procedure is the same when publishing Boolean, string and enum data.

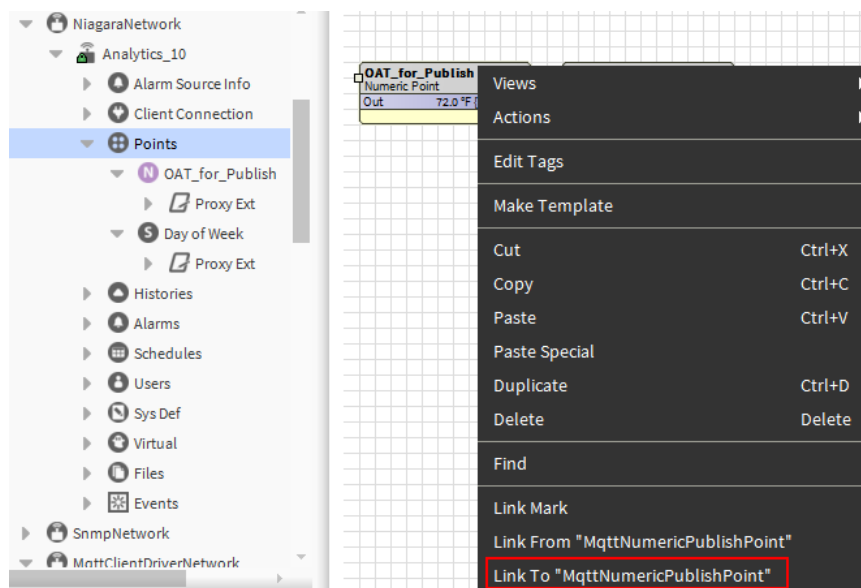
Prerequisites: You have added the points you need to the station.

Step 1 Expand **Config**→**Drivers** and expand the device.

Step 2 Right-click the **Points** node and select **Views**→**Wire Sheet**.

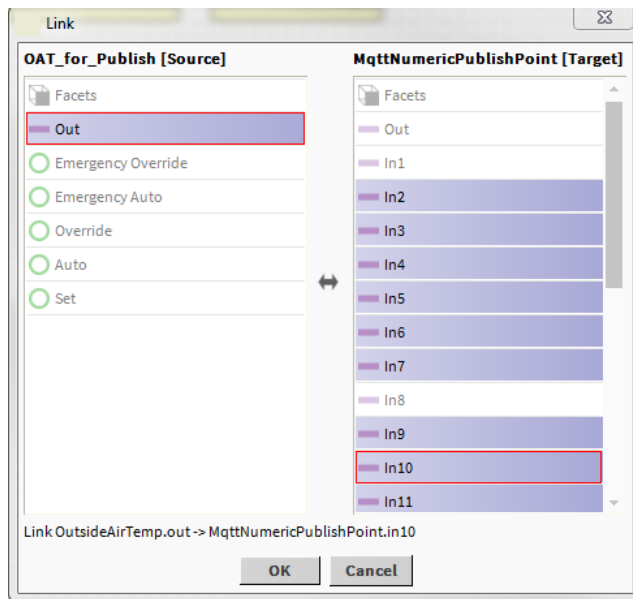
The **Wire Sheet** view opens.

Step 3 Right-click the point glyph and select a **Link Mark** option from the list.

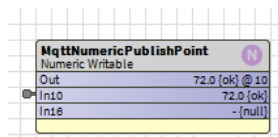


Step 4 For a numeric point, right-click it and select the **Link To "MqttNumericPublishPoint"** from the menu.

The **Link** window opens.



Step 5 Connect the output of the data point to the input of the **MqttNumericPublishPoint**. The data should now be transferring to the broker.



Chapter 2 Abstract MQTT Driver with GCP Authenticator

Topics covered in this chapter

- ◆ Setting up Google Cloud Platform (GCP)
- ◆ Setting up station to connect to GCP
- ◆ Creating device in GCP
- ◆ Sending messages from Niagara to GCP
- ◆ Sending messages GCP Console to Niagara

The MQTT authenticators plug in to the MQTT device according to establish a secure connection between a broker and a client device. You choose which authenticator to use for each device.

The **GcpAuthenticator** authenticates a secure connection between a client device and a broker on a Google Cloud Platform. It uses JwtKeys for connection.

Setting up Google Cloud Platform (GCP)

Before connecting to Niagara some configurations are done in GCP Console. This section provides information about the GCP Configuration.

- Step 1 Navigate to GCP Console. Click the drop-down list displayed in below image:
 Select A Project window opens.
- Step 2 Select **New Project**.
 A **New Project** window opens.
- Step 3 Enter the Project name of your choice and browse the location. Click **Create**.
 A new project is created.
- Step 4 Navigate to **Topics** using the navigation menu on the top left or using the search bar at top of the screen. (Search: Topics).
 a. Select Topics from the list.
 b. Create Topics (tempData, device, data).
- Step 5 Navigate to **IoT Core** using the navigation menu on the top left or using the search bar at top of the screen. (Search: IoT Core).
- Step 6 Create a new Registry in IoT Core.
 a. Enter Registry ID.
 b. Select region as europe-west1 or us-central1.
 NOTE: At the time of testing choosing Asia-east1 did not connect and produced an output error as: "Invalid Protocol Version".
 c. Select a topic from the list created in step 4.
- Step 7 Select **Add** under additional topics and select **tempData**. Enter **degree** as a subfolder.
- Step 8 Under the **device** state topic, select **device** from the list. **Projects/PROJECT_NAME/topics/device**. Click **Create**.

Setting up station to connect to GCP

The following steps describe how to prepare your Niagara station to connect it to GCP Platform.

Prerequisites:

- New Registry is created GCP.
- You are working in Workbench and your connected station is open.
- The **abstractMqttDriver** palette is open.

- Step 1 To create a new **AbstractMqttDriverNetwork**, expand **Config**→**Drivers**, and double-click **Drivers** to open the **Driver Manager** view.
- Step 2 In the **Driver Manager** view, click **New**, select **AbstractMqttDriverNetwork** from the drop-down list, and click **OK**.
- Step 3 In the open **abstractMqttDriver** palette, expand the **Devices** folder.
- Step 4 Drag the **GcpMqttDevice** component to your **AbstractMqttDriverNetwork** in the Nav tree and click **OK**.
- Step 5 Expand **GcpMqttDevice**, and double-click **authenticator**.
Gcp Authenticator Property sheet opens.
- Step 6 Configure the following parameters:
 - a. **Broker Endpoint**: Enter `mqtt.googleapis.com` as broker endpoint.
 - b. **Client ID**: It is auto-populated when you connect to GCP.
 - c. **Broker Port**: Enter `8883` as the port number.
- Step 7 Expand **Gcp Iot Parameters** and configure the following parameters:
 - a. **Project ID**: Enter ID of project created in GCP.
 - b. **Cloud Region**: Enter the Cloud region of registry created in GCP console.
 - c. **Registry ID**: Enter the ID of GCP Registry created.
 - d. **Device ID**: Enter the ID of the device created in GCP console.
- Step 8 To save the configuration, click **Save**.

Creating device in GCP

Once the **GcpMqttDevice** is configured in the Workbench, you need to add the device in GCP console.

Prerequisites:

GcpMqttDevice is added from the palette to the driver.

- Step 1 Navigate to IoT Core. Select devices from list.
- Step 2 Create new device/devices. Enter following parameters:
 - a. **Device ID**: Enter Device ID
 - b. Under Authentication. Enter or Upload public key of device created in Workbench.

NOTE: One public key per authenticator is added in Workbench. I.e., when an **Gcp Authenticator** is added from the palette, a public key is automatically created inside the mentioned directory if the station is started from **Application Director** `C:\ProgramData\Niagara4.x\Brand\stations\stationName\shared\JwtKeys\AbstractMqttDriverNetwork`. If the station is started from Niagara Console public key is automatically created inside the mentioned directory `C:\Users\user_name\Niagara4.x\Brand\stations\stationName\shared\JwtKeys\AbstractMqttDriverNetwork`.

- Step 3 Select the public key corresponding to the device name in Workbench. I.e., If device name in Workbench is device1 then public key will be device1.pem. Select the correct public key and upload or copy and paste the contents into the text box in IoT Core.
- Step 4 Click **Create**.
- Step 5 Navigate to Workbench and update device Id under **Gcp IoT Parameters** with device Id of newly created device in GCP console, then connect to device from Workbench.
- Step 6 To connect to GCP ,right click on **GcpMqttDevice Actions**→**Connect**.

Sending messages from Niagara to GCP

This procedure describes the steps to send messages from Niagara to GCP.

Prerequisites:

Connection is established between GCP and Niagara.

- Step 1 Navigate to **Config**→**Drivers**→**AbstractMqttDriverNetwork**→**GcpMqttDevice** and double-click **Points**.
MqttClientDriverPointManager window opens.
- Step 2 To add **MqttStringPublishPoint** click **New..**.
A **New** window opens.
- Step 3 Select **MqttStringPublishPoint** from the drop-down list and click **OK**.
Another **New** window opens.
- Step 4 Enter the following parameters:
 - a. **Name**: Enter the name to be given to the point.
 - b. **Local Point**: Enter the file path where the publish point is created.
 - c. **Topic**: Enter the topic name as /devices/device_name/state, Enter the device name set in GCP console.
- Step 5 Click **OK**. Navigate to AX property sheet of the point created.
- Step 6 Navigate to **Points**→**Proxy Ext**→**Set Value**. Select any **In** slot and set value to desired text message. for example `Hello World`.
- Step 7 Go to GCP Console, **IoT Core**→**Devices** and select the correct device in which the message was sent to. Select Configuration and State.
- Step 8 In the Cloud Update section you should see a row with your message in base64 format.
- Step 9 Click the row with the message, select text to see the message in text format.

Sending messages GCP Console to Niagara

This procedure describes the steps to send messages from GCP to Niagara.

Prerequisites:

Connection is established between GCP and Niagara.

- Step 1 Navigate to **Config**→**Drivers**→**AbstractMqttDriverNetwork**→**GcpMqttDevice** and double-click **Points**.
MqttClientDriverPointManager window opens.
- Step 2 To add **MqttStringSubscribePoint** click **New..**.
A **New** window opens.

- Step 3 Select **MqttStringSubscribePoint** from the drop-down list and click **OK**.
Another **New** window opens.
- Step 4 Enter the following parameters:
- a. **Name:** Enter the name to be given to the point.
 - b. **Topic:** Enter the topic name as /devices/device_name/commands/#, Enter the device name set in GCP console,
- Step 5 Click **OK**. Navigate to AX Property Sheet of the point created.
- Step 6 Right click on device, **Actions**→**Subscribe All**.
- Step 7 Go to GCP Console, **IoT Core**→**Devices** and select the correct device in which the point was set.
- Step 8 Select **Send Command**. Select Text. Type the command to send.
- Step 9 Input subfolder (Topic) in subfolder window. Click Send Command.
- Step 10 Return to Niagara console and view the **AX property Sheet** of the device. Expand Points and the output of the subscribe point should be the text or message sent from the console.

Chapter 3 Abstract MQTT Driver with Microsoft Azure

Topics covered in this chapter

- ◆ Azure SAS Tokens
- ◆ Connecting Abstract MQTT Driver to Microsoft Azure IoT Hub

As of Niagara 4.12, the Abstract MQTT driver includes an authenticator for providing connections to the Microsoft Azure IoT Hub platform.

In general, there are two available methods to directly authenticate with Azure IoT Hub as an MQTT client:

- X.509 certificates
- Shared Access Signature (SAS) tokens

The Abstract MQTT driver currently only supports SAS tokens.

Azure SAS Tokens

The Abstract MQTT Driver supports the use of SAS tokens for authentication to the Azure IoT Hub.

The authentication process is as follows:

- Add an **AzureMqttSasAuthenticator** component to the Abstract MQTT Driver device.
- The device's connection string is copied from Azure IoT Hub to a Niagara station.
- The authenticator securely stores the connection string and generates a new SAS token, and connects to the IoT Hub.
- The authenticator automatically generates a new token when the current token is close to expiring.

The benefits of using SAS tokens over certificates include:

- Simpler configuration during station commissioning.
- Secure encrypted storage of tokens and access key at rest in the Niagara key store.
- You can configure more frequently token expiry.
- The authenticator generates a new replacement token prior to expiry thereby avoiding repeated certificate maintenance in the future.
- Administrators can revoke the original connection string in the Azure IoT Hub at any time.

Connecting Abstract MQTT Driver to Microsoft Azure IoT Hub

The following section describes how you connect your Niagara Abstract MQTT Driver to Microsoft Azure IoT Hub using SAS tokens.

Setting up Azure IoT Hub on Azure portal

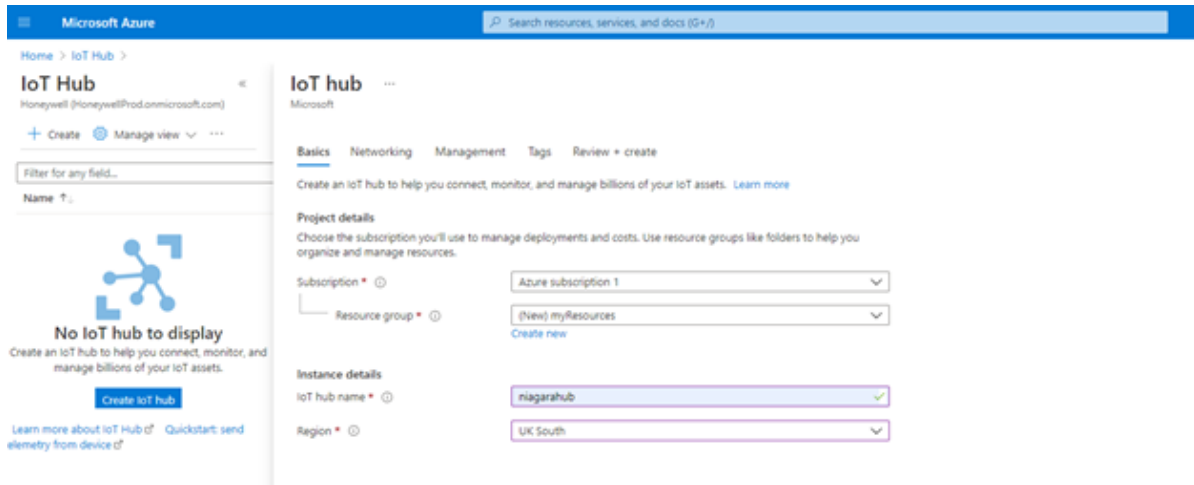
Below are the steps involved to set up an Azure IoT Hub on the Microsoft Azure portal.

Prerequisites:

- You have created and logged in to your Microsoft Azure account.
- You have selected the account subscription plan.

- Port 8883 of the host `azure-devices.net` is accessible in the customer environment.

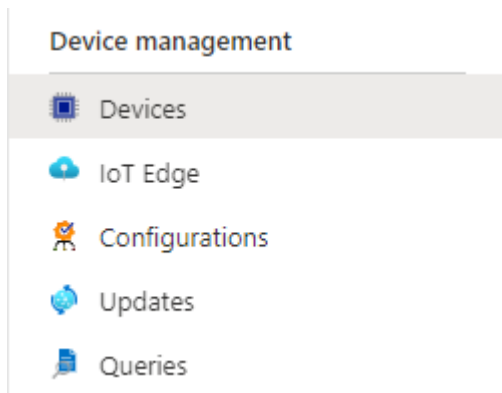
Step 1 On the Microsoft Azure website, search for IoT Hub and navigate to the IoT Hub homepage.



Step 2 Create an IoT Hub, choose your subscription plan and region, and select the appropriate network access for your scenario.

Wait for the deployment to complete.

Step 3 Once complete, click on your newly created hub and navigate to **Device Management**→**Devices**.



Step 4 Add a **New Device** and keep the default selections.

Device ID * ⓘ

NiagaraStation

Authentication type ⓘ

Symmetric key X.509 Self-Signed X.509 CA Signed

Auto-generate keys ⓘ

☒

Connect this device to an IoT hub ⓘ

Enable Disable

Parent device ⓘ

No parent device

[Set a parent device](#)

Step 5 To auto-generate keys and enable the device, select **Symmetric key** as the **Authentication type**.

Step 6 Once created, click on your device name to open the **Device Configuration** view.

Home > IoT Hub > niagarahub >

NiagaraStation ⓘ ...

niagarahub

Save Message to Device Direct Method Add Module Identity Device twin Manage keys Refresh

Device ID ⓘ NiagaraStation ⓘ

Primary Key ⓘ ⓘ

Secondary Key ⓘ ⓘ

Primary Connection String ⓘ ⓘ

Secondary Connection String ⓘ ⓘ

Enable connection to IoT Hub ⓘ ☒ Enable ☐ Disable

Parent device ⓘ No parent device ⓘ

Step 7 Click the **Copy** icon to the right side of either your device's **Primary Connection String** or **Secondary Connection String**.

Setting up station to connect to Azure IoT Hub

The following steps describe how to prepare your Niagara station to connect it to your created Azure IoT Hub using SAS tokens.

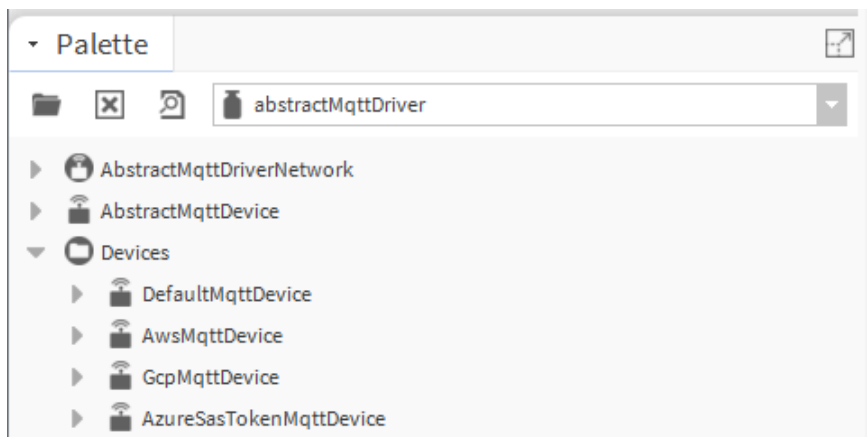
Prerequisites:

- You are working in Workbench and your connected station is open.
- The `abstractMqttDriver` palette is open.

Step 1 To create a new **AbstractMqttDriverNetwork**, expand **Config→Drivers**, and double-click **Drivers** to open the **Driver Manager** view.

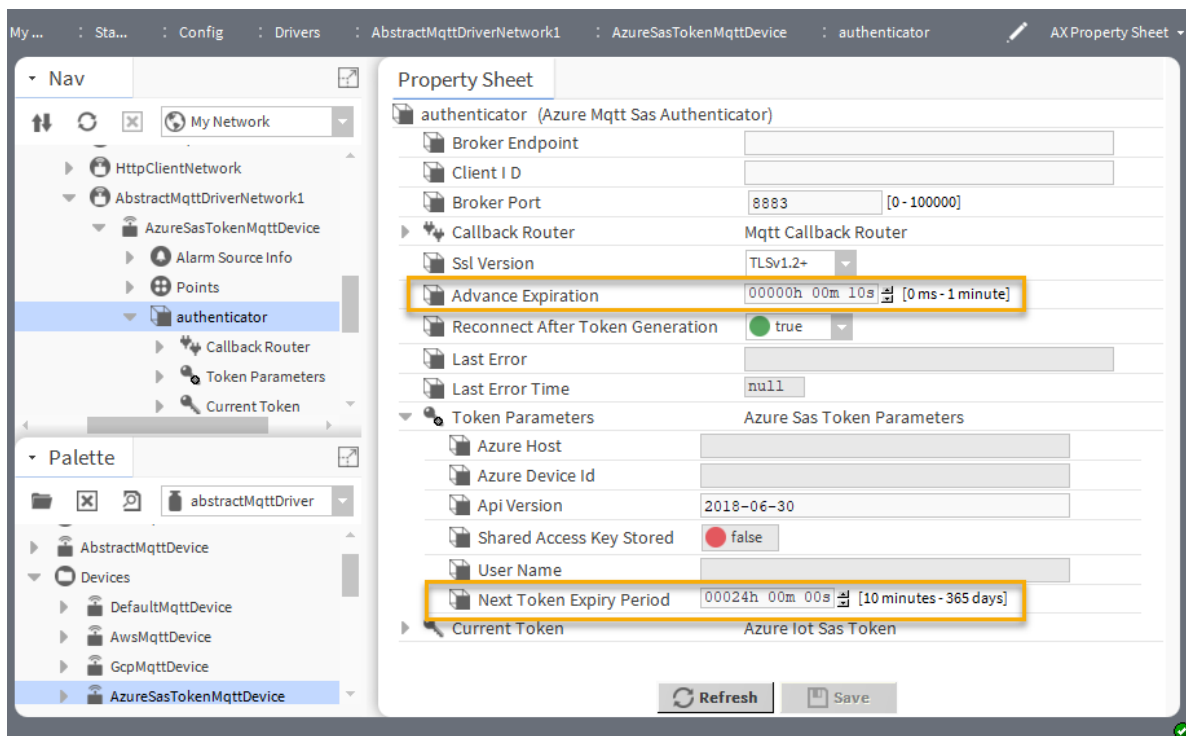
Step 2 In the **Driver Manager** view, click **New**, select **AbstractMqttDriverNetwork** from the drop-down menu, and click **OK**.

Step 3 In the open **abstractMqttDriver** palette, expand the **Devices** folder.



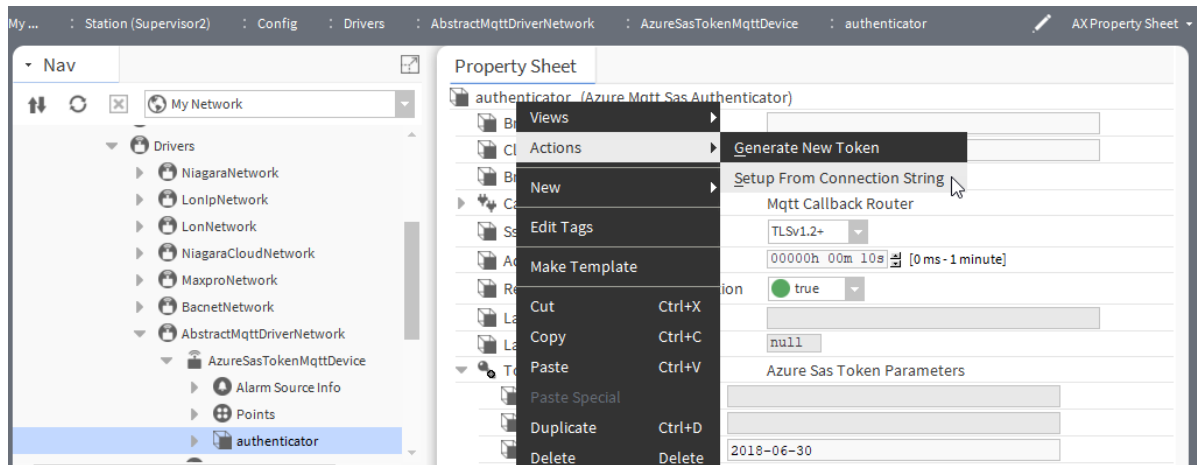
Step 4 Drag the **AzureSasTokenMqttDevice** component to your **AbstractMqttDriverNetwork** in the Nav tree, name it and click **OK**.

Step 5 Expand **AzureSasTokenMqttDevice**, double-click **authenticator** and consider the following two optional steps:

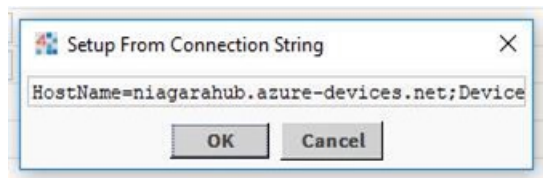


- Optional: For **Advance Expiration**, set an appropriate value to define the time period prior to the token's expiry when a new token is generated replacing the existing token. The default value is 10 seconds.
- Optional: Expand **Token Parameters** and choose an appropriate value for **Next Token Expiry Period**. By default, the token expires every 24 hours.

Step 6 Right-click **authenticator** and select **Actions**→**Setup From Connection String**.



Step 7 In the **Setup From Connection String** window, paste in the connection string as copied from the Azure IoT Hub device configuration (see section “Setting up Azure IoT Hub on Azure portal”).



The authenticator populates the following fields and connects the parent device to Azure IoT Hub:

- Broker Endpoint
- Client ID
- Token Parameters
- Current Token

Status Message	Connected.	
Points	Mqtt Client Driver Point Device Ext	
Connection Lost Waiting Time	00000h 00m 30s	[1second - +inf]
Send Enum As	TAG	
authenticator	Azure Mqtt Sas Authenticator	
Broker Endpoint	niagarahub.azure-devices.net	
Client I D	NiagaraStation	
Broker Port	8883	[0-100000]
Callback Router	Mqtt Callback Router	
Ssl Version	TLSv1.2+	
Advance Expiration	00000h 00m 10s	[0 ms - 1 minute]
Reconnect After Token Generation	true	
Last Error		
Last Error Time	null	
Token Parameters	Azure Sas Token Parameters	
Azure Host	niagarahub.azure-devices.net	
Azure Device Id	NiagaraStation	
Api Version	2018-06-30	
Shared Access Key Stored	true	
User Name	niagarahub.azure-devices.net/NiagaraStat.	
Next Token Expiry Period	00024h 00m 00s	[10 minutes - 365 days]
Current Token	Azure Iot Sas Token	
Token Stored	true	
Last Generated Time	17-Mar-2022 04:40 PM GMT	
Next Expiry Time	18-Mar-2022 04:40 PM GMT	

NOTE: The authenticator automatically generates a new token before the current token expires and reconnects the device using the new token. To prompt this procedure manually, you can at any time invoke the **Generate New Token** action on the authenticator.

Chapter 4 Components

Topics covered in this chapter

- ◆ abstractMqttDriver-AbstractMqttDriverNetwork
- ◆ abstractMqttDriver-AbstractMqttDevice
- ◆ Aws Jitp Mqtt Authenticator (abstractMqttDriver-AwsJitpMqttAuthenticator)
- ◆ abstractMqttDriver-AwsMqttAuthenticator
- ◆ abstractMqttDriver-AzureMqttSasAuthenticator
- ◆ abstractMqttDriver-AzureSasTokenParameters
- ◆ abstractMqttDriver-AzureIotSasToken
- ◆ abstractMqttDriver-GenericMqttAuthenticator
- ◆ abstractMqttDriver-GcpAuthenticator
- ◆ abstractMqttDriver-GcpIotParameters
- ◆ abstractMqttDriver-TokenParameters
- ◆ abstractMqttDriver-MqttCallbackRouter
- ◆ abstractMqttDriver-PointCallbackHandler
- ◆ abstractMqttDriver-MqttClientDriverPointDeviceExt
- ◆ abstractMqttDriver-MqttClientDriverDeviceFolder
- ◆ abstractMqttDriver-MqttClientDriverPointFolder
- ◆ abstractMqttDriver-MqttBooleanObjectPublishExt
- ◆ abstractMqttDriver-MqttBooleanObjectSubscribeExt
- ◆ abstractMqttDriver-MqttNumericObjectPublishExt
- ◆ abstractMqttDriver-MqttNumericObjectSubscribeExt
- ◆ abstractMqttDriver-MqttStringObjectPublishExt
- ◆ abstractMqttDriver-MqttStringObjectSubscribeExt
- ◆ abstractMqttDriver-MqttEnumObjectPublishExt
- ◆ abstractMqttDriver-MqttEnumObjectSubscribeExt

Components include services, folders and other model building blocks associated with a module. Each of the following Mqtt components are briefly described in this section.

This topic contains a short description of a component or the component plugin view.

abstractMqttDriver-AbstractMqttDriverNetwork

This component is the top-level container component for an **AbstractMqttDriverNetwork** in a station.

This component is located in the **abstractMqttDriver** palette. Like other drivers, you should install this component under the **Drivers** node of a station. The default view of this component is the **Mqtt Client Driver Device Manager** view where you can add new devices, remove or edit devices in a tabular layout.

Figure 2 AbstractMqttDriverNetwork Property Sheet

Property Sheet

AbstractMqttDriverNetwork (Abstract Mqtt Driver Network)

Status {ok}

Enabled true

Fault Cause

Health Ok [03-Oct-17 5:11 PM IST]

Alarm Source Info Alarm Source Info

Monitor Ping Monitor

Tuning Policies Tuning Policy Map

Poll Scheduler N Poll Scheduler

Refresh Save

To access these properties, expand **Config→Drivers**, right-click **AbstractMqttDriverNetwork** and click, **Views→AX Property Sheet**.

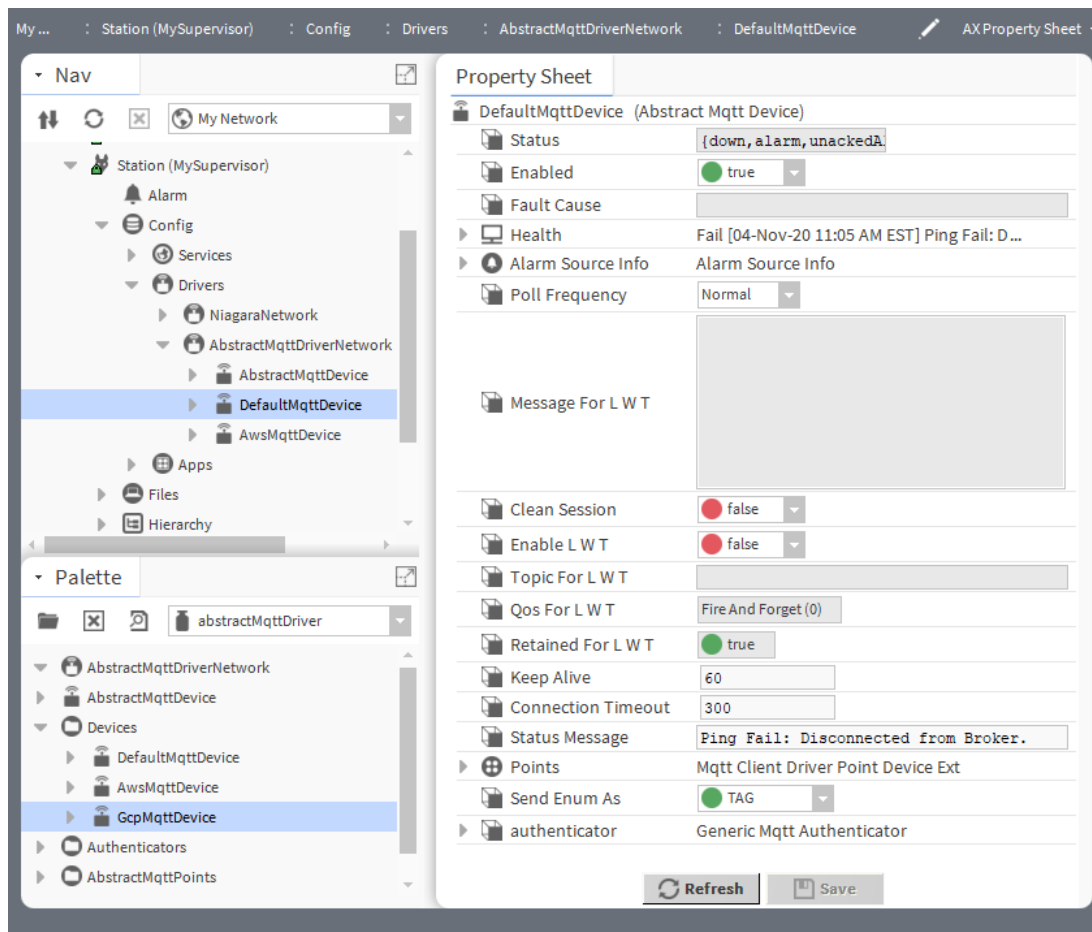
Property	Value	Description
Status	read-only	Reports the condition of the entity or process at last polling. {ok} indicates that the component is licensed and polling successfully. {down} indicates that the last check was unsuccessful, perhaps because of an incorrect property, or possibly loss of network connection. {disabled} indicates that the Enable property is set to false. {fault} indicates another problem. Refer to Fault Cause for more information.
Enabled	true or false	Activates (true) and deactivates (false) use of the object (network, device, point, component, table, schedule, descriptor, etc.).
Fault Cause	read-only	Indicates the reason why a system object (network, device, component, extension, etc.) is not working (in fault). This property is empty unless a fault exists.
Health	read-only	Reports the status of the network, device or component. This advisory information, including a time stamp, can help you recognize and troubleshoot problems but it provides no direct management controls. The <i>Niagara Drivers Guide</i> documents the these properties.
Alarm source info	additional properties	Contains a set of properties for configuring and routing alarms when this component is the alarm source. For property descriptions, refer to the <i>Niagara Alarms Guide</i>
Monitor	additional properties	Configures a network's ping mechanism, which verifies network health. This includes verifying the health of all connected objects (typically, devices) by pinging each device at a repeated interval. The <i>Niagara Drivers Guide</i> documents these properties.

Property	Value	Description
Tuning Policies	additional properties	Selects a network tuning policy by name. This policy defines stale time and minimum and maximum update times. During polling, the system uses the tuning policy to evaluate both write requests and the acceptability (freshness) of read requests. These properties are documented in the <i>Niagara Drivers Guide</i> .
Poll Scheduler	additional properties	Configures the frequency with which the driver polls points and devices. "Poll Service properties" in the <i>Niagara Drivers Guide</i> documents these properties.

abstractMqttDriver-AbstractMqttDevice

This driver supports four MQTT device components that communicate with the broker: **AbstractMqttDevice**, **DefaultMqttDevice**, **AwsMqttDevice** and **GcpMqttDevice**. The only difference among them is that the **AbstractMqttDevice** does not provide communication security, which means that it does not include an **authenticator** component. You configure the device by setting its properties to match the settings for the broker.

Figure 3 DefaultMqttDevice properties



To access these properties, expand **Config→Drivers** and double-click a device object in the Nav tree.

In addition to the standard properties (Status, Enabled, Fault Cause, Health, Alarm Source Info, and Poll Frequency), these properties are unique to these components.

Property	Value	Description
Message for L W T	read-only	Reports the Last Will and Testament (last known good value) when an unexpected error occurs.
Clean Session	true or false (default)	Controls the persistence of the session between broker and client. true enables a clean session that is not persistent. All information is lost when the client disconnects from the broker for any reason. false disables the clean session creating a persistent session. The driver preserves this session until the client requests a clean session again. If a session is already available, the driver uses it and delivers any queued messages to the client.
Enable L W T	true or false (default)	Turns on (true) and off (false) the Last-Will-and-Testament feature.
Topic for L W T	read-only	Reports the topic that contains the Last Will and Testament value.

Property	Value	Description
Qos for L W T	drop-down list	<p>Defines the Qos (Quality of Service) level, which configures how persistently the broker and client attempt to ensure that a message is received.</p> <p><code>Fire And Forget (0)</code> delivers the message once with no confirmation.</p> <p><code>Atleast Once (1)</code> delivers the message at least once with confirmation required.</p> <p><code>Exactly Once (2)</code> delivers the message exactly once by using a four-step handshake.</p> <p>Messages may be sent at any Qos level, and clients may attempt to subscribe to topics at any Qos level.</p>
Retained for L W T	true (default) or false	<p>Configures the broker to save the message for the specified topic as its Last Will and Testament (last known good value).</p> <p>It is useful for newly-connected subscribers to receive the last retained message for the topic immediately after subscribing and extremely helpful to provide status updates from components and devices on individual topics.</p>
Keep Alive	seconds	<p>Defines the longest period during which the broker and client can remain connected without sending a message. The broker must disconnect a client that does not send a PINGREQ (ping request) or another message within this period.</p> <p>To ensure that the connection remains open and both client and broker remain connected, the client sends this time interval to the broker during the establishment of the connection. This provides Abstract MQTT with a work-around for the issue of half-open connections.</p>
Connection Timeout	seconds	Specifies the maximum amount of time the client waits after sending a request to the broker.
Status Message	text	Reports the current status of the component.
Points	folder	Contains the discovered points.
Send Enum As	drop-down list (defaults to TAG)	<p>Configures how to transmit an enum value.</p> <p>TAG</p> <p>ORDINAL</p>
authenticator (available using DefaultMqttDevice component only)	additional properties	Defines the authenticator component to use for this device. An authenticator secures the connection between the client and broker.

Actions

These actions are available on both components. To perform an action, right-click the device in the Nav tree and click **Actions** followed by selecting the action.

Action	Description
Ping	This checks the device status.
Connect	Establishes a connection with a broker.
Disconnect	Disconnects session with a broker.
Subscribe all	Subscribes clients for all topics.
Unsubscribe all	Unsubscribes clients from all topics

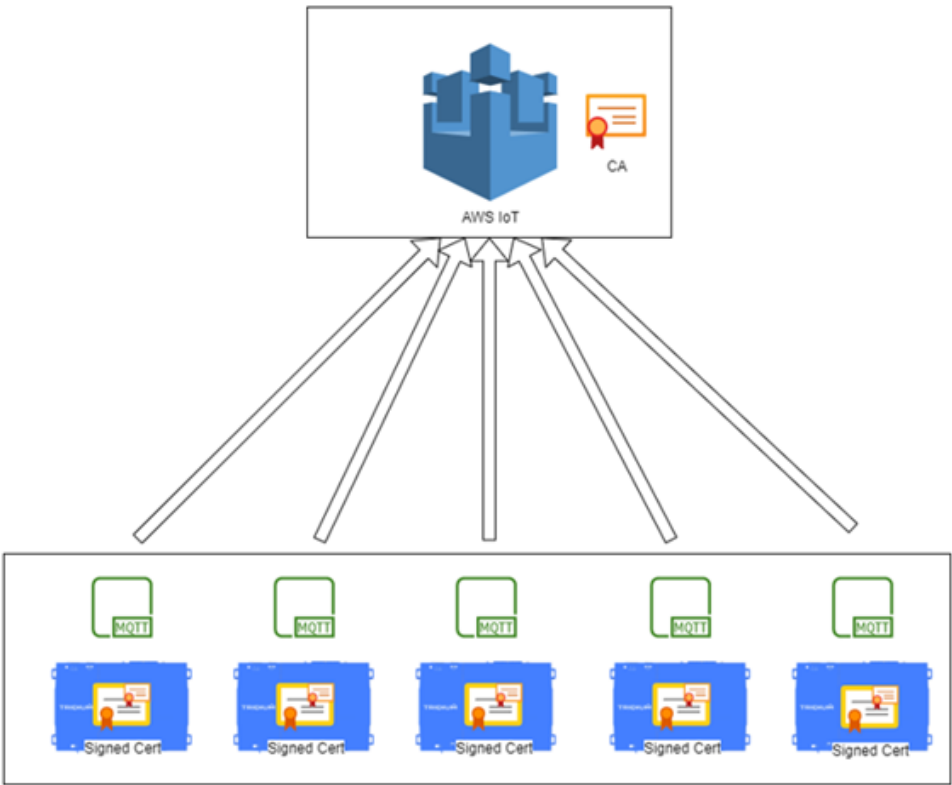
NOTE:

The Ip address, Client ID, and port number of the broker must be entered for Connect and Disconnect actions to work.

Aws Jitp Mqtt Authenticator (abstractMqttDriver-AwsJitpMqttAuthenticator)

The **Aws Jitp Mqtt Authenticator** component connects to Amazon Web Services (AWS) utilizing the **Just In Time Provisioning** (JITP) functionality as configured in the `awsUtils` module. See *"Configuring Just In Time Provisioning"* in the *"Niagara AWS Utils Guide"* for more details.

Just In Time Provisioning allows a fleet of devices to automatically connect to AWS with auto-generated certificates as means of authentication. The major difference to the existing AWS MQTT authenticator is that the JITP authenticator does not require an AWS user to manually configure the device in AWS IoT, or to generate and sign their device certificate. This is performed in conjunction with the Signing Service, which automatically supplies signing certificates to each authenticator. In addition, certificates are also renewed without any user intervention required. For more information, see *"Signing Service"* in the *"Niagara Signing Service Guide"*.



authenticator

Aws Jitp Mqtt Authenticator

Broker Endpoint

Client I D

Broker Port

8883

[0 - 100000]

Callback Router

Mqtt Callback Router

Certificate Alias And Password

aws-AwsJitpMqttDevice

Cert Requester

Fox Signing Requester

Property	Value	Description
Broker Endpoint	string	Defines the broker endpoint with your AWS IoT service endpoint.
Client ID	read-only	Automatically populated when the signed certificate is retrieved from the Signing Service. The value will match the Common Name of the certificate.
Broker Port	numeric value [0–100000]	Automatically set to the AWS default port 8883.
Callback Router	additional properties	Specifies Callback Type and Point Callback Handler .

March 3, 2023

31

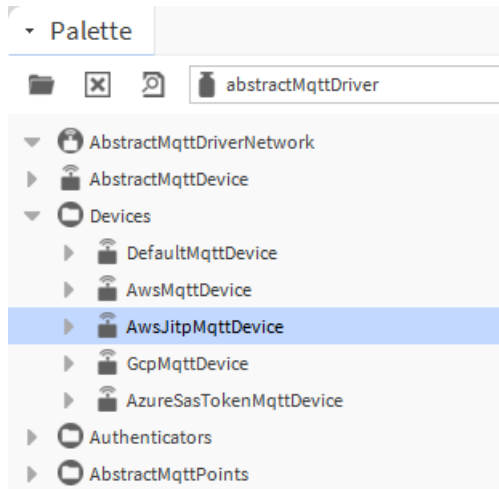
Property	Value	Description
Certificate Alias and Password	additional properties	Specifies alias and password for the certificate used to authenticate with AWS. Alias is automatically generated in the format 'aws_deviceName'
Cert Requester	additional properties	Contains components that submit a CSR to the Supervisor Signing Service and obtain the signed certificate to install in the User Key Store .

Automatic install

To use this authenticator, you can automatically install an MQTT device on each Niagara station in your network using a Niagara provisioning task from a Supervisor station. As the device is added to the station, it will automatically onboard with the Signing Service, obtain a signed device certificate and connect to AWS. For more information, see “Running Install AWS MQTT Device task” in the “Niagara AWS Utils Guide.”

Manual install

You can also manually install a single device by dragging the **AwsJitpMqttDevice** component from the **abstractMqttDriver** palette.

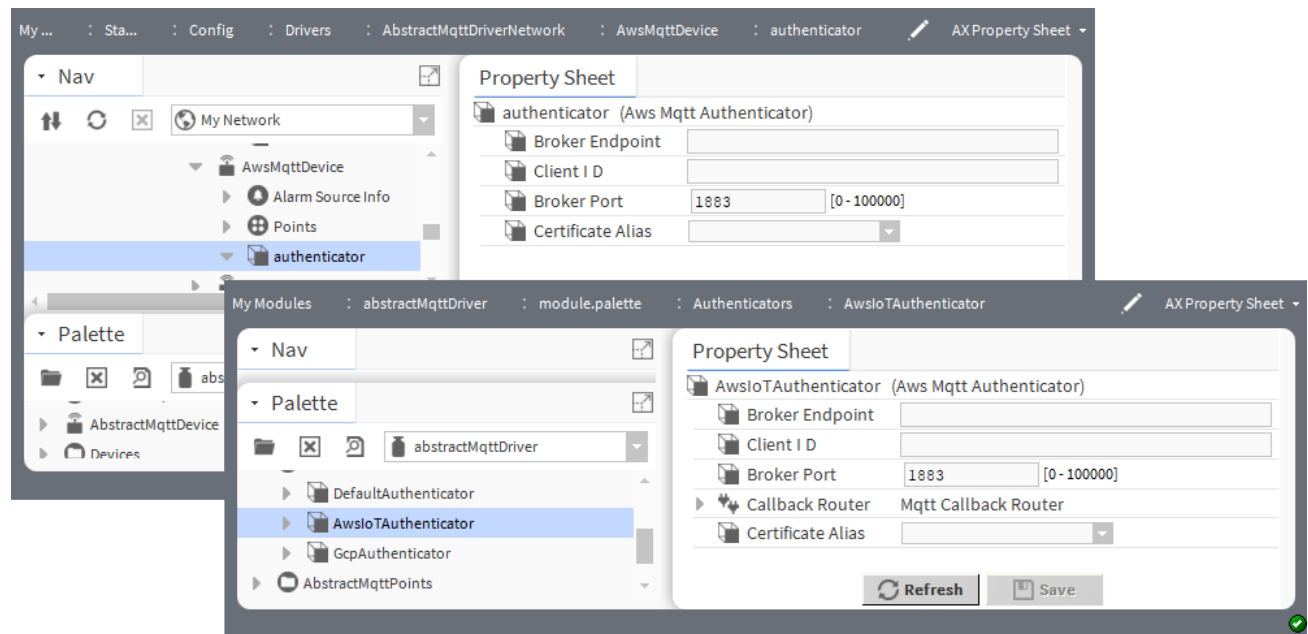


- Populate the broker endpoint with your AWS IoT service endpoint and change the port if different from the AWS default.
- **Certificate Alias** will be populated automatically. We recommend that you enter a password to protect your device certificate in the Niagara **User Key Store**.
- On **Cert Requester**, invoke the **Onboard** action and expand this component to monitor progress. An admin user will need to approve the onboarding request in the Supervisor. For more details, see “*Signing Service*” in the “Niagara Signing Service Guide”.

abstractMqttDriver-AwsMqttAuthenticator

This component sets up a secure connection between an AWS client and broker using certificates and keys.

Figure 4 AwsMqttAuthenticator properties



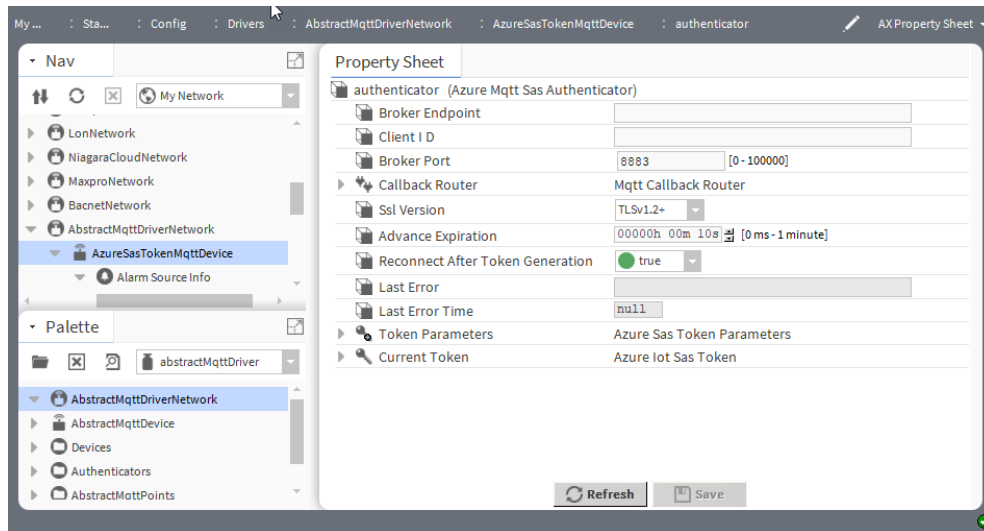
To access these properties, expand **Config→Drivers→AbstractMqttDriverNetwork**, double-click an **AwsMqttDevice** and expand or double-click **authenticator**.

Property	Value	Description
Broker Endpoint	IP address, URL or host name	Specifies the IP address, URL or Hostname of the broker.
Client ID	text string	Identifies the client to the server. If not specified, the system generates a unique Client ID. No other client should currently use or in the future use this unique string. If two devices share the same Client ID, Status reports {down} or {fault}, unexpected behaviors happen and communication issues between client and broker result.
Broker Port	number	Specifies the port number of the broker.
Callback Router (available under AwsIoTAuthenticator from the palette)	additional properties	Directs a client request to an appropriate function. A separate topic documents these router functions.
Certificate Alias	drop-down list	Selects the name of the client certificate, which secures the connection between the client and broker. For AWS IoT devices, the registration process creates this certificate, which you download from the AWS IoT portal and import into the User Key Store .

abstractMqttDriver-AzureMqttSasAuthenticator

This component sets up a secure connection between the Niagara a Abstract MQTT device and the Azure IoT Hub.

Figure 5 AzureMqttSasAuthenticator properties



Azure Mqtt Sas Authenticator is contained in the **abstractMqttDriver** palette.

To access these properties, expand **Config**→**Drivers**→**AbstractMqttDriverNetwork**, double-click **AzureSasTokenMqttDevice** and double-click **authenticator**.

Property	Value	Description
Broker Endpoint	IP address, URL or hostname	Specifies the IP address, URL or Hostname of the broker.
Client ID	text string	Identifies the client to the server. If not specified, the system generates a unique Client ID. No other client should currently use or in the future use this unique string. If two devices share the same Client ID, Status reports {down} or {fault}, unexpected behaviors happen and communication issues between client and broker result.
Broker Port	number	Specifies the port number of the broker.
Callback Router	additional properties	Directs a client request to an appropriate function. A separate topic documents these router functions.
Ssl Version	enum	Reports the TLS (Transfer Layer Security) version currently being used.
Advance Expiration	hours, minutes, seconds (defaults to 10 seconds)	Defines the time period prior to the token's expiry that a new token will be generated replacing the existing token.
Reconnect After Token Generation	true (default) or false	When set to true, the authenticator automatically reconnects the parent MQTT device after a new token is generated.
Last Error	read-only	The most recent error caused by token management functions.
Last Error Time	read-only	The time of the most recent error.

Actions

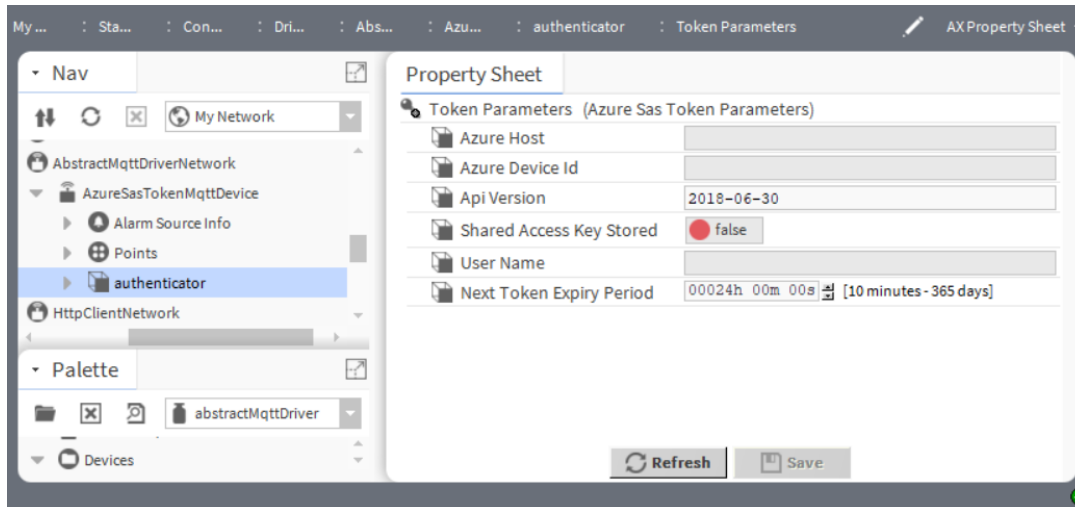
To perform an action, right-click **authenticator**, select **Actions** and select the action.

- **Generate New Token:** Manually generates a new SAS token to use for connections. It replaces any existing token.
- **Setup From Connection String:** Configures the authenticator and generates a new SAS token from that configuration.

abstractMqttDriver-AzureSasTokenParameters

This component displays the configuration values that are used when generating a new Azure SAS token.

Figure 6 abstractMqttDriver-AzureSasTokenParameters properties



Azure Sas Token Parameters is contained in the **abstractMqttDriver** palette.

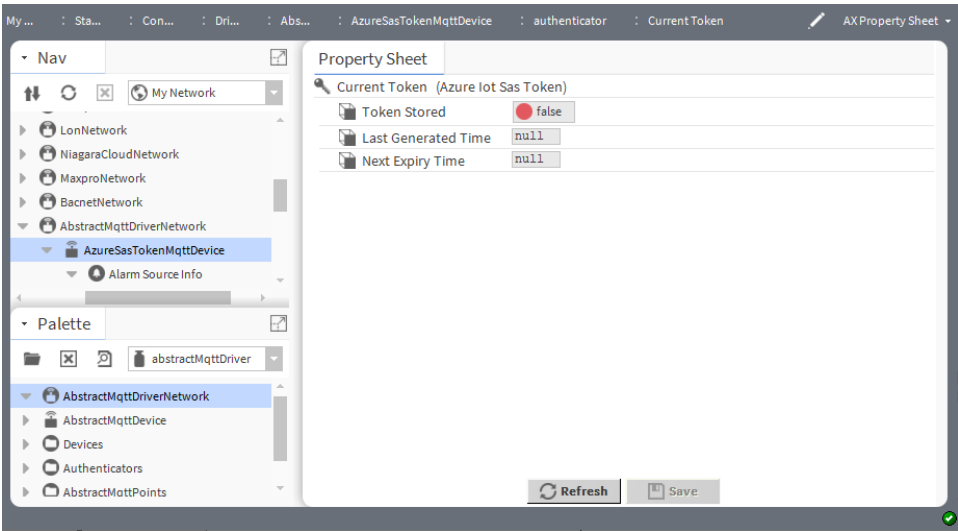
To access these properties, expand **Config→Drivers→AbstractMqttDriverNetwork→AzureSasTokenMqttDevice**, double-click **AzureSasTokenMqttDevice**, double-click **authenticator** and double-click **Token Parameters**.

Property	Value	Description
Azure Host	read-only	Specifies the hostname of the Azure IoT Hub.
Azure Device Id	read-only	Specifies the name of the Azure device in IoT Hub.
Api Version	string	Constitutes the version of the Azure IoT Hub API to use for token generation.
Shared Access Key Stored	read-only	Set to True if the automatic configuration has stored your device's access key in the secure key store. This is required for future token generation.
User Name	read-only	Consists of a combination of host, device Id, and API version.
Next Token Expiry Period	hours, minutes, seconds (defaults to 24 hours)	Defines the length in time the next generated SAS token will be valid.

abstractMqttDriver-AzureIotSasToken

This component represents an instance of a current Azure IoT Hub SAS token.

Figure 7 abstractMqttDriver-AzureIotSasToken properties



Azure Iot Sas Token is contained in the `abstractMqttDriver` palette.

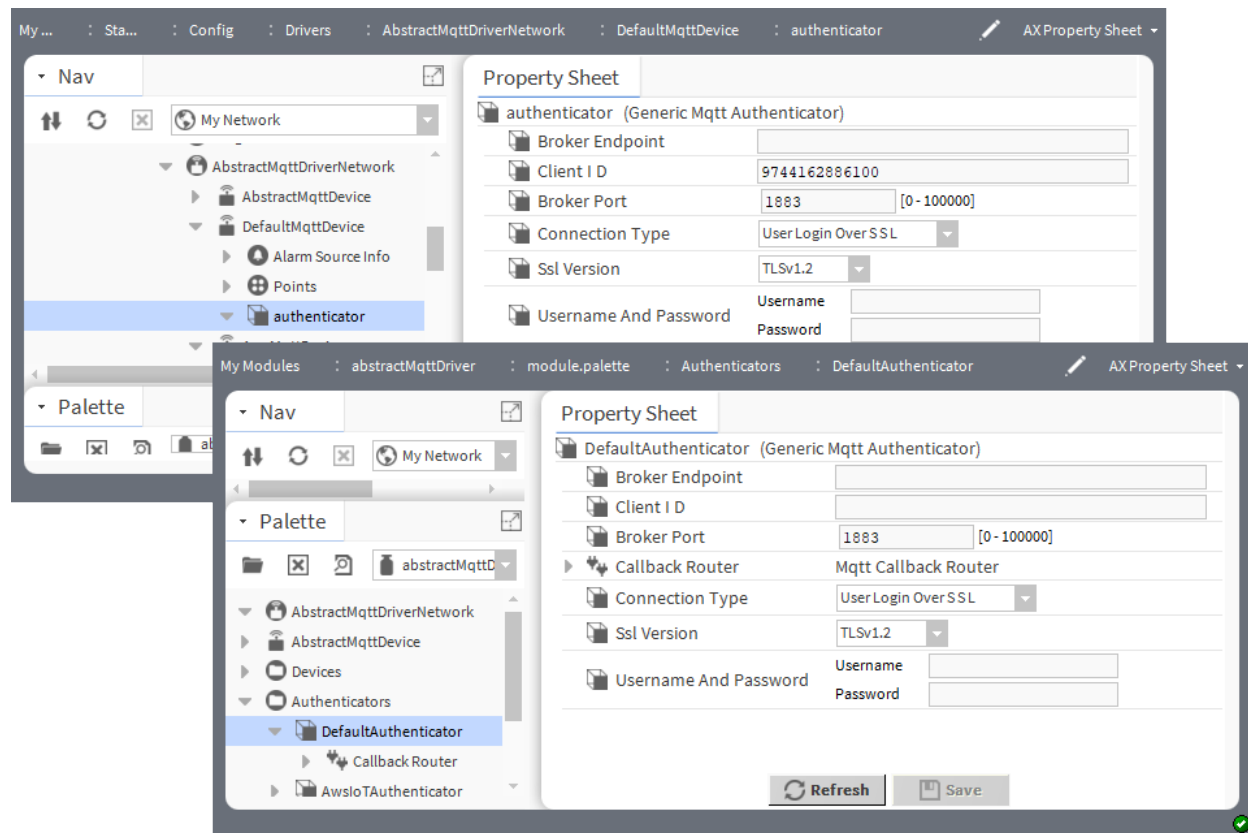
To access these properties, expand **Config→Drivers→AbstractMqttDriverNetwork**, double-click **AzureSasTokenMqttDevice**, double-click **authenticator** and double-click **Current Token**.

Property	Value	Description
Token Stored	read-only	Set to <code>true</code> if the SAS token is stored in the secure key store.
Last Generated Time	read-only	Indicates the date and time the current token was generated.
Next Expiry Time	read-only	Indicates the time when the current token expires.

abstractMqttDriver-GenericMqttAuthenticator

This is the default authenticator. It configures a connection between client and broker that is not secure. You might use this type of connection when debugging connection problems. This should not be your preferred authenticator.

Figure 8 Generic Authenticator properties



To access these properties, expand **Config→Drivers→AbstractMqttDriverNetwork**, expand a device and double-click **authenticator**.

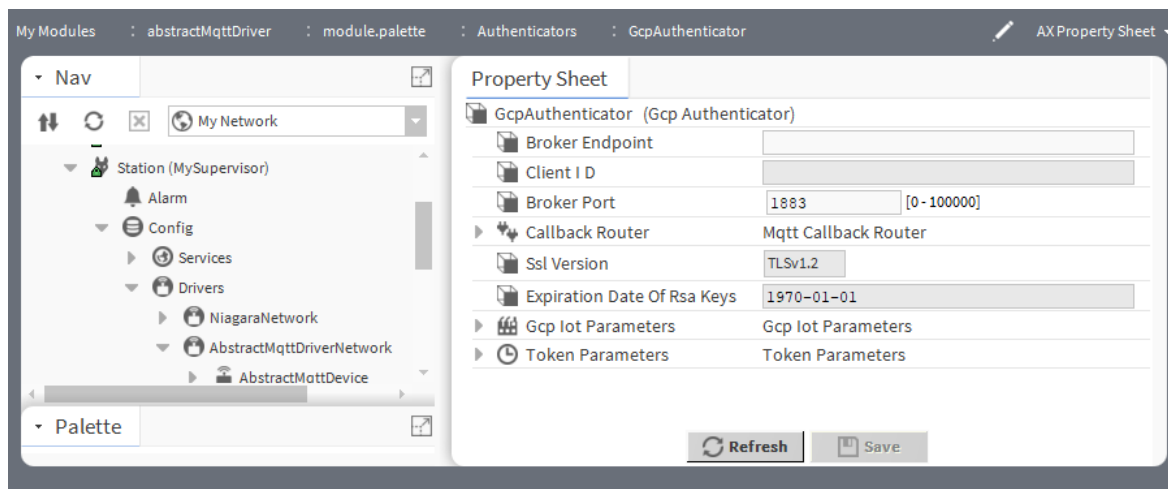
Property	Value	Description
Broker Endpoint	IP address, URL or host name	Specifies the IP address, URL or Hostname of the broker.
Client ID	text string	Identifies the client to the server. If not specified, the system generates a unique Client ID. No other client should currently use or in the future use this unique string. If two devices share the same Client ID, Status reports {down} or {fault}, unexpected behaviors happen and communication issues between client and broker result.
Broker Port	number	Specifies the port number of the broker.
Callback Router (available under AwsIoTAuthenticator from the palette)	additional properties	Directs a client request to an appropriate function. A separate topic documents these router functions.

Property	Value	Description
Connection Type	drop-down (defaults to User Login Over SSL)	<p>Defines the type of authentication for this connection.</p> <p>Anonymous establishes a connection with the broker that is not secure. Choosing this type of connection is not recommended and generates a warning message. If you agree, the system logs your choice in its audit file. To use this type of connection, add this information using the security dashboard.</p> <p>Anonymous Over SSL establishes a one-way secure connection using TLS (Transport Layer Security) where the broker is configured with certificates that include keys. Choosing this type of connection, which is secure in only one direction, is not recommended and generates a warning message. If you agree, the system logs your choice in its audit file.</p> <p>Using Login Over SSL requires you to enter the username and password for the broker. This is the recommended Connection Type.</p>
User name and Password	text	<p>Defines the credentials required by the broker. The broker may or may not require these credentials.</p> <p>If Connection Type is User Login Over SSL, you must enter the Username and Password.</p>

abstractMqttDriver-GcpAuthenticator

This component provides secure communication between a device and the Google Cloud Platform (GCP).

Figure 9 GcpAuthenticator properties



To access this view, expand **Config**→**Drivers**→**AbstractMqttDriverNetwork**, double-click an **GcpMqttDevice** and expand or double-click **authenticator**.

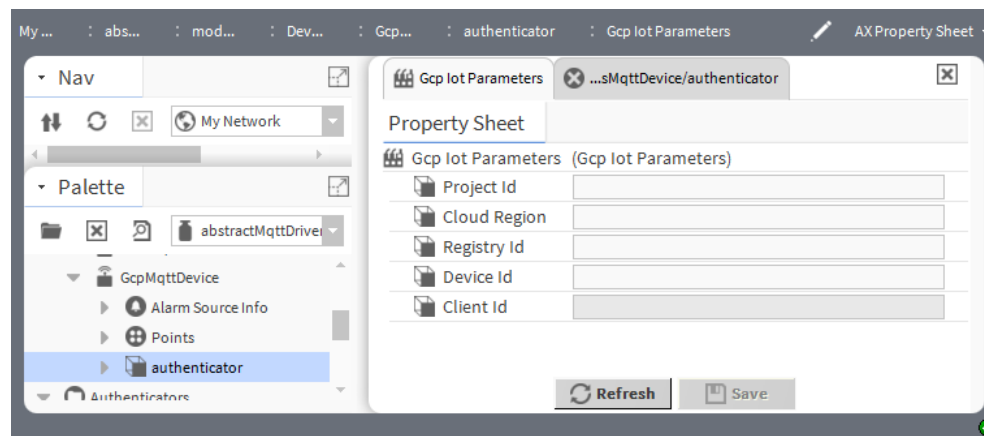
Property	Value	Description
Broker Endpoint	IP address, URL or host name	Specifies the IP address, URL or Hostname of the broker.
Client ID	text string	Identifies the client to the server. If not specified, the system generates a unique Client ID.

Property	Value	Description
		No other client should currently use or in the future use this unique string. If two devices share the same Client ID, Status reports {down} or {fault}, unexpected behaviors happen and communication issues between client and broker result.
Broker Port	number	Specifies the port number of the broker.
Callback Router	additional properties	Directs a client request to an appropriate function. A separate topic documents these router functions.
Ssl Version	read-only	Reports the TLS (Transfer Layer Security) version currently being used.
Expiration Date of Rsa Keys	read-only	Reports when the RSA (Rivest-Shamir-Adleman) pair of cryptosystem keys expires.
Gcp Iot Parameters	additional properties	Configures properties required by the Google Cloud Platform. A separate topic documents these properties.
Token Parameters	additional properties	Configures properties related to the RSA token. A separate topic documents these properties.

abstractMqttDriver-GcplotParameters

This component configures the Google Cloud Platform Internet of Things properties.

Figure 10 Gcp Iot Parameters properties



To view these properties, expand **Config**→**Drivers**→**GcpMqttDevice**, double-click **authenticator** and double-click **Gcp Iot Parameters**.

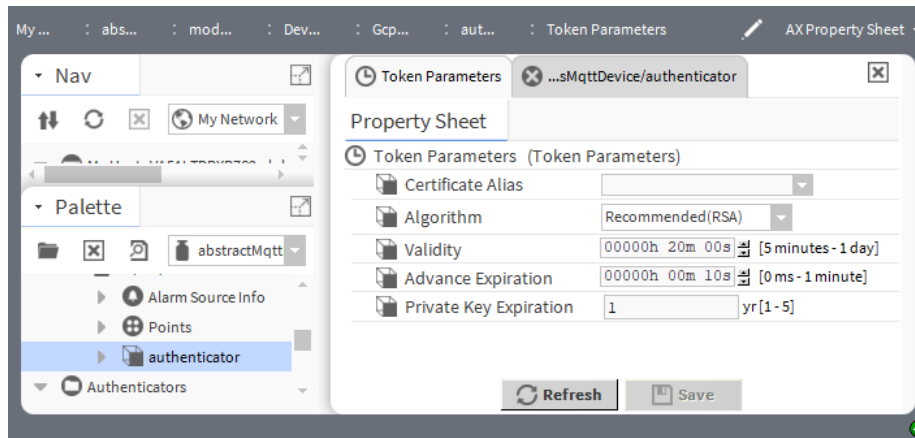
Property	Value	Description
Project Id	text	Defines the string ID of the cloud project that owns the registry and device.
Cloud Region	text	Defines the geographical region where the Google Cloud Platform device registry is located, for example <code>us-central1</code> .
Registry Id	text	Defines the identifier of the device registry, for example <code>registry1</code> .

Property	Value	Description
Device Id	text	Defines the identifier for the device, for example, <code>thing1</code> . This identifier must be unique within the registry.
Client Id	read-only	Reports the ID of the client.

abstractMqttDriver-TokenParameters

This component configures security-related properties for Google Cloud Platform devices.

Figure 11 Token Parameters properties



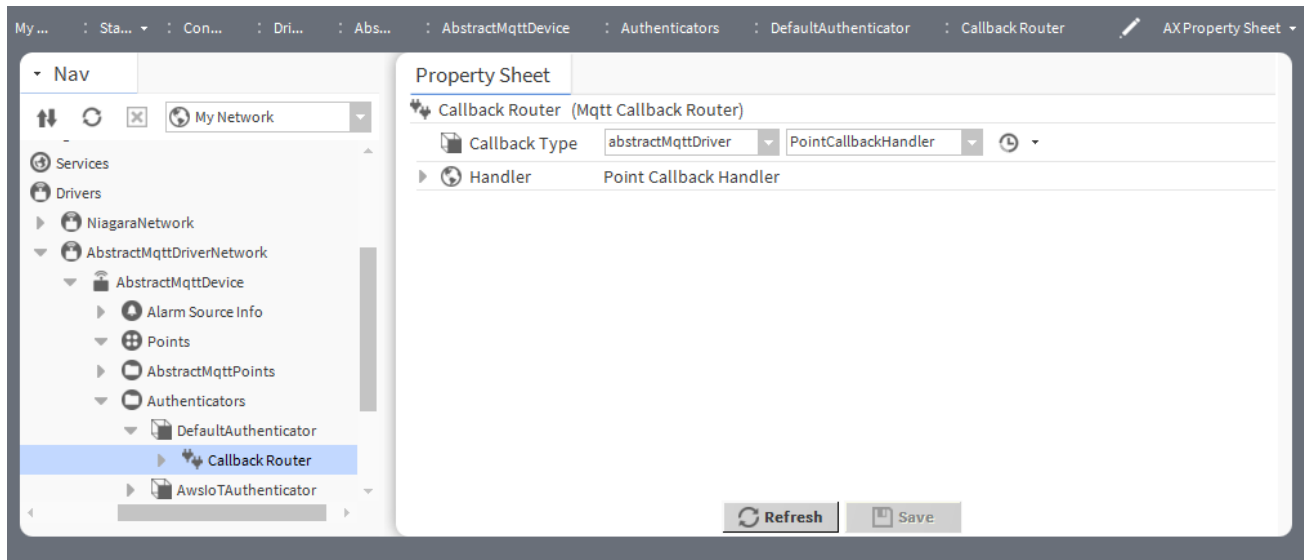
To view these properties, expand **Config**→**Drivers**→**GcpMqttDevice**, double-click **authenticator** and double-click **Token Parameters**.

Property	Value	Description
Certificate Alias	drop-down list	Selects the name of the client certificate, which secures the connection between the client and broker.
Algorithm	drop-down list	Selects the algorithm used to generate private and public keys.
Validity	hours minutes seconds (defaults to 20 minutes)	Configures a time value that specifies the limited period for which the token is valid.
Advance Expiration	hours minutes seconds (defaults to 10 seconds)	Specifies how far in advance to send notification that the token validity time is expiring.
Private Key Expiration	years	Specifies how long the private key is valid.

abstractMqttDriver-MqttCallbackRouter

This component configures callback routing. Several routing methods specify a callback function (also called a handler function).

Figure 12 MqttCallbackRouter properties



To access these properties, in the station expand **Config→Drivers→AbstractMqttDriverNetwork→Devices** any device, **Authenticators**, expand any authenticator and double-click **Callback Router**.

Type	Value	Description
Callback Type	drop-down list	Select the type of driver and callback point from the drop-down list.
Handler	container	

abstractMqttDriver-PointCallbackHandler

This component handles the information received regarding points.

To access these properties, in the station expand **Config→Drivers→AbstractMqttDriverNetwork→Devices**, any device **Authenticator**, any authenticator **Callback Route** and double-click **Handler**.

abstractMqttDriver-MqttClientDriverPointDeviceExt

This component is a container for Mqtt client points.

The **Mqtt Client Driver Point Manager** is the default view for this component. It has no properties of its own.

abstractMqttDriver-MqttClientDriverDeviceFolder

This component implements a folder under the network.

Typically, you add such folders using the **New Folder** button in the **Mqtt Client Driver Device Manager** view of the network. Each **MqttClientDriverDeviceFolder** has its own **Mqtt Client Driver Device Manager** view.

abstractMqttDriver-MqttClientDriverPointFolder

This component implements a folder under the **Points** extension.

Typically, you add such folders using the **New Folder** button in the **Mqtt Client Driver Point Manager** view of the **Points** extension. Each `MqttClientDriverPointFolde`r has its own **Mqtt Client Driver Point Manager** view.

abstractMqttDriver-MqttBooleanObjectPublishExt

This component configures the driver to publish Boolean data.

This component is in the **abstractMqttDriver** palette under the **AbstractMqttPoints** folder.

Figure 13 Mqtt Boolean Object Publish Point Ext properties

Property Sheet	
Proxy Ext (Mqtt Boolean Object Publish Ext)	
Status	{stale}
Fault Cause	
Enabled	<input checked="" type="checkbox"/> true
Device Facets	>> ⌚
Conversion	Default
Tuning Policy Name	defaultPolicy
Read Value	- {ok}
Write Value	- {ok}
Topic	
QoS	Fire And Forget (0)
Retained	<input checked="" type="checkbox"/> true
Publish Message On Change	<input checked="" type="checkbox"/> true

To access these properties, expand **Config→Drivers→AbstractMqttDriverNetwork**, expand a device, expand the **Points** folder, expand this point and double-click the **Proxy Ext**.

In addition to the standard properties (Status, Fault Cause and Enabled) these properties are unique to this component.

Property	Value	Description
Device facets	additional properties	<p>Determine how values are formatted for display depending on the context and the type of data. Examples include engineering units and decimal precision for numeric types, and descriptive value (state) text for boolean and enum types.</p> <p>With the exception of proxy points (with possible defined device facets), point facets do not affect how the framework processes the point's value.</p> <p>Besides control points, various other components have facets too. For example, many <code>kitControl</code> and <code>schedule</code> components have facets. Details about point facets apply to these components too, unless especially noted.</p> <p>You access facets by clicking an Edit button or a chevron >>. Both open an Edit Facets window.</p>
Conversion	Drop-down list (defaults to <code>Default</code>)	<p>Defines how the system converts proxy extension units to parent point units.</p> <p><code>Default</code> automatically converts similar units (such as Fahrenheit to Celsius) within the proxy point.</p> <p>NOTE: In most cases, the standard <code>Default</code> conversion is best.</p>

Property	Value	Description
		<p><code>Linear</code> applies to voltage input, resistive input and voltage output writable points. Works with linear-acting devices. You use the <code>Scale</code> and <code>Offset</code> properties to convert the output value to a unit other than that defined by device facets.</p> <p><code>Linear With Unit</code> is an extension to the existing linear conversion property. This specifies whether the unit conversion should occur on "Device Value" or "Proxy Value". The new linear with unit convertor, will have a property to indicate whether the unit conversion should take place before or after the scale/offset conversion.</p> <p><code>Reverse Polarity</code> applies only to Boolean input and relay output writable points. Reverses the logic of the hardware binary input or output.</p> <p><code>500 Ohm Shunt</code> applies to voltage input points only. It reads a 4-to-20mA sensor, where the <code>Ui</code> input requires a 500 ohm resistor wired across (shunting) the input terminals.</p> <p><code>Tabular Thermistor</code> applies to only a Thermistor input point and involves a custom resistance-to-temperature value response curve for Type 3 Thermistor temperature sensors.</p> <p><code>Thermistor Type 3</code> applies to an Thermistor Input point, where this selection provides a "built-in" input resistance-to-temperature value response curve for Type 3 Thermistor temperature sensors.</p> <p><code>Generic Tabular</code> applies to non-linear support for devices other than for thermistor temperature sensors with units in temperature. Generic Tabular uses a lookup table method similar to the "Thermistor Tabular" conversion, but without predefined output units.</p>
Tuning Policy Name	drop-down list	<p>Selects a network tuning policy by name. This policy defines stale time and minimum and maximum update times.</p> <p>During polling, the system uses the tuning policy to evaluate both write requests and the acceptability (freshness) of read requests.</p>
Read Value	read-only	Displays the last value read from the device, expressed in device facets.
Write value	read-only	Displays the last value written using device facets.
Topic	text	Defines a string that the broker uses to filter messages for each client. A topic consists of one or more topic levels. Each level is separated by a forward slash (topic level separator).
Qo S	drop-down list	<p>Defines the Qos (Quality of Service) level, which configures how persistently the broker and client attempt to ensure that a message is received.</p> <p><code>Fire And Forget (0)</code> delivers the message once with no confirmation.</p> <p><code>Atleast Once (1)</code> delivers the message at least once with confirmation required.</p>

Property	Value	Description
		Exactly Once (2) delivers the message exactly once by using a four-step handshake. Messages may be sent at any Qos level, and clients may attempt to subscribe to topics at any Qos level.
Retained	true (default) or false	Configures the broker to save the message for the specified topic as its Last Will and Testament (last known good value). It is useful for newly-connected subscribers to receive the last retained message for the topic immediately after subscribing and extremely helpful to provide status updates from components and devices on individual topics.
Publish Message on Change	true (default) or false	Determines if the client publishes a message when a value changes.

abstractMqttDriver-MqttBooleanObjectSubscribeExt

This component configures the subscription to Boolean data.

This component is in the **abstractMqttDriver** palette under the **AbstractMqttPoints** folder.

Figure 14 Mqtt Boolean Object Subscribe Point Ext Property Sheet

Property Sheet

Proxy Ext (Mqtt Boolean Object Subscribe Ext)

Status	{stale}
Fault Cause	
Enabled	<input checked="" type="checkbox"/> true
Device Facets	>> ⌚
Conversion	Default
Tuning Policy Name	defaultPolicy
Read Value	- {ok}
Write Value	- {ok}
Topic	
QoS	Fire And Forget (0)

To access these properties, expand **Config→Drivers→AbstractMqttDriverNetwork**, expand a device, expand the **Points** folder, expand this point and double-click the **Proxy Ext**.

In addition to the standard properties (Status, Fault Cause and Enabled) these properties are unique to this component.

Property	Value	Description
Device facets	additional properties	<p>Determine how values are formatted for display depending on the context and the type of data. Examples include engineering units and decimal precision for numeric types, and descriptive value (state) text for boolean and enum types.</p> <p>With the exception of proxy points (with possible defined device facets), point facets do not affect how the framework processes the point's value.</p> <p>Besides control points, various other components have facets too. For example, many <code>kitControl</code> and <code>schedule</code> components have facets. Details about point facets apply to these components too, unless especially noted.</p> <p>You access facets by clicking an Edit button or a chevron >>. Both open an Edit Facets window.</p>
Conversion	Drop-down list (defaults to <code>Default</code>)	<p>Defines how the system converts proxy extension units to parent point units.</p> <p><code>Default</code> automatically converts similar units (such as Fahrenheit to Celsius) within the proxy point.</p> <p>NOTE: In most cases, the standard <code>Default</code> conversion is best.</p> <p><code>Linear</code> applies to voltage input, resistive input and voltage output writable points. Works with linear-acting devices. You use the <code>Scale</code> and <code>Offset</code> properties to convert the output value to a unit other than that defined by device facets.</p> <p><code>Linear With Unit</code> is an extension to the existing linear conversion property. This specifies whether the unit conversion should occur on "Device Value" or "Proxy Value". The new linear with unit convertor, will have a property to indicate whether the unit conversion should take place before or after the scale/offset conversion.</p> <p><code>Reverse Polarity</code> applies only to Boolean input and relay output writable points. Reverses the logic of the hardware binary input or output.</p> <p><code>500 Ohm Shunt</code> applies to voltage input points only. It reads a 4-to-20mA sensor, where the <code>Ui</code> input requires a 500 ohm resistor wired across (shunting) the input terminals.</p> <p><code>Tabular Thermistor</code> applies to only a Thermistor input point and involves a custom resistance-to-temperature value response curve for Type 3 Thermistor temperature sensors.</p> <p><code>Thermistor Type 3</code> applies to an Thermistor Input point, where this selection provides a "built-in" input resistance-to-temperature value response curve for Type 3 Thermistor temperature sensors.</p> <p><code>Generic Tabular</code> applies to non-linear support for devices other than for thermistor temperature sensors with units in temperature. <code>Generic Tabular</code> uses a lookup table method similar to the "Thermistor Tabular" conversion, but without predefined output units.</p>

Property	Value	Description
Tuning Policy Name	drop-down	Selects a network tuning policy by name. This policy defines stale time and minimum and maximum update times. During polling, the system uses the tuning policy to evaluate both write requests and the acceptability (freshness) of read requests.
Read Value	read-only	Displays the last value read from the device, expressed in device facets.
Write value	read-only	Displays the last value written using device facets.
Topic	text	Defines a string that the broker uses to filter messages for each client. A topic consists of one or more topic levels. Each level is separated by a forward slash (topic level separator).
Qos	drop-down list	Defines the Qos (Quality of Service) level, which configures how persistently the broker and client attempt to ensure that a message is received. <code>Fire And Forget (0)</code> delivers the message once with no confirmation. <code>Atleast Once (1)</code> delivers the message at least once with confirmation required. <code>Exactly Once (2)</code> delivers the message exactly once by using a four-step handshake. Messages may be sent at any Qos level, and clients may attempt to subscribe to topics at any Qos level.

abstractMqttDriver-MqttNumericObjectPublishExt

This component configures the publishing of numeric data.

This component is in the **abstractMqttDriver** palette under the **AbstractMqttPoints** folder.

Figure 15 Mqtt Numeric Object Publish Ext properties

Property Sheet

Proxy Ext (Mqtt Numeric Object Publish Ext)

- Status: {stale}
- Fault Cause: [Empty]
- Enabled: ☒ true
- Device Facets: >> [Clock icon]
- Conversion: [Default]
- Tuning Policy Name: defaultPolicy
- Read Value: - {ok}
- Write Value: - {ok}
- Topic: [Empty]
- Qo S: Fire And Forget (0)
- Retained: ☒ true
- Publish Message On Change: ☒ true

To access these properties, expand **Config→Drivers→AbstractMqttDriverNetwork**, expand a device, expand the **Points** folder, expand this point and double-click the **Proxy Ext**.

In addition to the standard properties (Status, Fault Cause and Enabled) these properties are unique to this component.

Property	Value	Description
Device facets	additional properties	<p>Determine how values are formatted for display depending on the context and the type of data. Examples include engineering units and decimal precision for numeric types, and descriptive value (state) text for boolean and enum types.</p> <p>With the exception of proxy points (with possible defined device facets), point facets do not affect how the framework processes the point's value.</p> <p>Besides control points, various other components have facets too. For example, many <code>kitControl</code> and <code>schedule</code> components have facets. Details about point facets apply to these components too, unless especially noted.</p> <p>You access facets by clicking an Edit button or a chevron <code>>></code>. Both open an Edit Facets window.</p>
Conversion	Drop-down list (defaults to <code>Default</code>)	<p>Defines how the system converts proxy extension units to parent point units.</p> <p><code>Default</code> automatically converts similar units (such as Fahrenheit to Celsius) within the proxy point.</p> <p>NOTE: In most cases, the standard <code>Default</code> conversion is best.</p> <p><code>Linear</code> applies to voltage input, resistive input and voltage output writable points. Works with linear-acting devices. You use the <code>Scale</code> and <code>Offset</code> properties to convert the output value to a unit other than that defined by device facets.</p> <p><code>Linear With Unit</code> is an extension to the existing linear conversion property. This specifies whether the unit conversion should occur on "Device Value" or "Proxy Value". The new linear with unit convertor, will have a property to indicate whether the unit conversion should take place before or after the scale/offset conversion.</p> <p><code>Reverse Polarity</code> applies only to Boolean input and relay output writable points. Reverses the logic of the hardware binary input or output.</p> <p><code>500 Ohm Shunt</code> applies to voltage input points only. It reads a 4-to-20mA sensor, where the <code>Ui</code> input requires a 500 ohm resistor wired across (shunting) the input terminals.</p> <p><code>Tabular Thermistor</code> applies to only a Thermistor input point and involves a custom resistance-to-temperature value response curve for Type 3 Thermistor temperature sensors.</p> <p><code>Thermistor Type 3</code> applies to an Thermistor Input point, where this selection provides a "built-in" input resistance-to-temperature value response curve for Type 3 Thermistor temperature sensors.</p> <p><code>Generic Tabular</code> applies to non-linear support for devices other than for thermistor temperature sensors with units in temperature. <code>Generic Tabular</code> uses a lookup table method</p>

Property	Value	Description
		similar to the “Thermistor Tabular” conversion, but without predefined output units.
Tuning Policy Name	drop-down list	Selects a network tuning policy by name. This policy defines stale time and minimum and maximum update times. During polling, the system uses the tuning policy to evaluate both write requests and the acceptability (freshness) of read requests.
Read Value	read-only	Displays the last value read from the device, expressed in device facets.
Write value	read-only	Displays the last value written using device facets.
Topic	text	Defines a string that the broker uses to filter messages for each client. A topic consists of one or more topic levels. Each level is separated by a forward slash (topic level separator).
Qo S	drop-down list	Defines the Qos (Quality of Service) level, which configures how persistently the broker and client attempt to ensure that a message is received. <code>Fire And Forget (0)</code> delivers the message once with no confirmation. <code>Atleast Once (1)</code> delivers the message at least once with confirmation required. <code>Exactly Once (2)</code> delivers the message exactly once by using a four-step handshake. Messages may be sent at any Qos level, and clients may attempt to subscribe to topics at any Qos level.
Retained	true (default) or false	Configures the broker to save the message for the specified topic as its Last Will and Testament (last known good value). It is useful for newly-connected subscribers to receive the last retained message for the topic immediately after subscribing and extremely helpful to provide status updates from components and devices on individual topics.
Publish Message on Change	true (default) or false	Determines if the client publishes a message when a value changes.

abstractMqttDriver-MqttNumericObjectSubscribeExt

This component configures the subscription to numeric data.

This component is in the **abstractMqttDriver** palette under the **AbstractMqttPoints** folder.

Figure 16 Mqtt Numeric Object Subscribe Ext properties

Property Sheet

Proxy Ext (Mqtt Numeric Object Subscribe Ext)

Status	{stale}
Fault Cause	
Enabled	<input checked="" type="checkbox"/> true
Device Facets	>> ⌚
Conversion	Default
Tuning Policy Name	Default Policy
Read Value	0.00 {ok}
Write Value	- {null} @ def
Topic	topic\komal
Qo S	Fire And Forget (0)

Refresh Save

To access these properties, expand **Config→Drivers→AbstractMqttDriverNetwork**, expand a device, expand the **Points** folder, expand this point and double-click the **Proxy Ext**.

In addition to the standard properties (Status, Fault Cause and Enabled) these properties are unique to this component.

Property	Value	Description
Device facets	additional properties	<p>Determine how values are formatted for display depending on the context and the type of data. Examples include engineering units and decimal precision for numeric types, and descriptive value (state) text for boolean and enum types.</p> <p>With the exception of proxy points (with possible defined device facets), point facets do not affect how the framework processes the point's value.</p> <p>Besides control points, various other components have facets too. For example, many <code>kitControl</code> and schedule components have facets. Details about point facets apply to these components too, unless especially noted.</p> <p>You access facets by clicking an Edit button or a chevron >>. Both open an Edit Facets window.</p>
Conversion	Drop-down list (defaults to Default)	<p>Defines how the system converts proxy extension units to parent point units.</p> <p>Default automatically converts similar units (such as Fahrenheit to Celsius) within the proxy point.</p> <p>NOTE: In most cases, the standard Default conversion is best.</p> <p>Linear applies to voltage input, resistive input and voltage output writable points. Works with linear-acting devices. You use the Scale and Offset properties to convert the output value to a unit other than that defined by device facets.</p> <p>Linear With Unit is an extension to the existing linear conversion property. This specifies whether the unit conversion should occur on "Device Value" or "Proxy Value". The new linear with unit convertor, will have a property to indicate whether the unit conversion should take place before or after the scale/offset conversion.</p>

Property	Value	Description
		<p><code>Reverse Polarity</code> applies only to Boolean input and relay output writable points. Reverses the logic of the hardware binary input or output.</p> <p><code>500 Ohm Shunt</code> applies to voltage input points only. It reads a 4-to-20mA sensor, where the Ui input requires a 500 ohm resistor wired across (shunting) the input terminals.</p> <p><code>Tabular Thermistor</code> applies to only a Thermistor input point and involves a custom resistance-to-temperature value response curve for Type 3 Thermistor temperature sensors.</p> <p><code>Thermistor Type 3</code> applies to an Thermistor Input point, where this selection provides a “built-in” input resistance-to-temperature value response curve for Type 3 Thermistor temperature sensors.</p> <p><code>Generic Tabular</code> applies to non-linear support for devices other than for thermistor temperature sensors with units in temperature. Generic Tabular uses a lookup table method similar to the “Thermistor Tabular” conversion, but without predefined output units.</p>
Tuning Policy Name	drop-down	<p>Selects a network tuning policy by name. This policy defines stale time and minimum and maximum update times.</p> <p>During polling, the system uses the tuning policy to evaluate both write requests and the acceptability (freshness) of read requests.</p>
Read Value	read-only	Displays the last value read from the device, expressed in device facets.
Write value	read-only	Displays the last value written using device facets.
Topic	text	Defines a string that the broker uses to filter messages for each client. A topic consists of one or more topic levels. Each level is separated by a forward slash (topic level separator).
Qos	drop-down	<p>Defines the Qos (Quality of Service) level, which configures how persistently the broker and client attempt to ensure that a message is received.</p> <p><code>Fire And Forget (0)</code> delivers the message once with no confirmation.</p> <p><code>Atleast Once (1)</code> delivers the message at least once with confirmation required.</p> <p><code>Exactly Once (2)</code> delivers the message exactly once by using a four-step handshake.</p> <p>Messages may be sent at any Qos level, and clients may attempt to subscribe to topics at any Qos level.</p>

abstractMqttDriver-MqttStringObjectPublishExt

This component configure the properties that publish string data.

This component is in the **abstractMqttDriver** palette under the **AbstractMqttPoints** folder.

Figure 17 Mqtt String Object Publish Ext properties

Property Sheet

Proxy Ext (Mqtt String Object Publish Ext)

Status	{ok}
Fault Cause	
Enabled	<input checked="" type="checkbox"/> true
Device Facets	>> ⌚
Conversion	Default
Tuning Policy Name	Default Policy
Read Value	- {ok}
Write Value	- {null} @ def
Topic	topic\komal
Qo S	Fire And Forget (0)
Retained	<input checked="" type="checkbox"/> true
Publish Message On Change	<input checked="" type="checkbox"/> true

Refresh Save

To access these properties, expand **Config→Drivers→AbstractMqttDriverNetwork**, expand a device, expand the **Points** folder, expand this point and double-click the **Proxy Ext**.

In addition to the standard properties (Status, Fault Cause and Enabled) these properties are unique to this component.

Property	Value	Description
Device facets	additional properties	<p>Determine how values are formatted for display depending on the context and the type of data. Examples include engineering units and decimal precision for numeric types, and descriptive value (state) text for boolean and enum types.</p> <p>With the exception of proxy points (with possible defined device facets), point facets do not affect how the framework processes the point's value.</p> <p>Besides control points, various other components have facets too. For example, many <code>kitControl</code> and schedule components have facets. Details about point facets apply to these components too, unless especially noted.</p> <p>You access facets by clicking an Edit button or a chevron <code>>></code>. Both open an Edit Facets window.</p>
Conversion	Drop-down list (defaults to <code>Default</code>)	<p>Defines how the system converts proxy extension units to parent point units.</p> <p><code>Default</code> automatically converts similar units (such as Fahrenheit to Celsius) within the proxy point.</p> <p>NOTE: In most cases, the standard <code>Default</code> conversion is best.</p> <p><code>Linear</code> applies to voltage input, resistive input and voltage output writable points. Works with linear-acting devices. You use the <code>Scale</code> and <code>Offset</code> properties to convert the output value to a unit other than that defined by device facets.</p> <p><code>Linear With Unit</code> is an extension to the existing linear conversion property. This specifies whether the unit conversion</p>

Property	Value	Description
		<p>should occur on “Device Value” or “Proxy Value”. The new linear with unit convertor, will have a property to indicate whether the unit conversion should take place before or after the scale/offset conversion.</p> <p><code>Reverse Polarity</code> applies only to Boolean input and relay output writable points. Reverses the logic of the hardware binary input or output.</p> <p><code>500 Ohm Shunt</code> applies to voltage input points only. It reads a 4-to-20mA sensor, where the Ui input requires a 500 ohm resistor wired across (shunting) the input terminals.</p> <p><code>Tabular Thermistor</code> applies to only a Thermistor input point and involves a custom resistance-to-temperature value response curve for Type 3 Thermistor temperature sensors.</p> <p><code>Thermistor Type 3</code> applies to an Thermistor Input point, where this selection provides a “built-in” input resistance-to-temperature value response curve for Type 3 Thermistor temperature sensors.</p> <p><code>Generic Tabular</code> applies to non-linear support for devices other than for thermistor temperature sensors with units in temperature. Generic Tabular uses a lookup table method similar to the “Thermistor Tabular” conversion, but without predefined output units.</p>
Tuning Policy Name	drop-down list	<p>Selects a network tuning policy by name. This policy defines stale time and minimum and maximum update times.</p> <p>During polling, the system uses the tuning policy to evaluate both write requests and the acceptability (freshness) of read requests.</p>
Read Value	read-only	Displays the last value read from the device, expressed in device facets.
Write value	read-only	Displays the last value written using device facets.
Topic	text	Defines a string that the broker uses to filter messages for each client. A topic consists of one or more topic levels. Each level is separated by a forward slash (topic level separator).
Qo S	drop-down list	<p>Defines the Qos (Quality of Service) level, which configures how persistently the broker and client attempt to ensure that a message is received.</p> <p><code>Fire And Forget (0)</code> delivers the message once with no confirmation.</p> <p><code>Atleast Once (1)</code> delivers the message at least once with confirmation required.</p> <p><code>Exactly Once (2)</code> delivers the message exactly once by using a four-step handshake.</p> <p>Messages may be sent at any Qos level, and clients may attempt to subscribe to topics at any Qos level.</p>

Property	Value	Description
Retained	true (default) or false	Configures the broker to save the message for the specified topic as its Last Will and Testament (last known good value). It is useful for newly-connected subscribers to receive the last retained message for the topic immediately after subscribing and extremely helpful to provide status updates from components and devices on individual topics.
Publish Message on Change	true (default) or false	Determines if the client publishes a message when a value changes.

abstractMqttDriver-MqttStringObjectSubscribeExt

This component configures the properties for subscribing string data.

This component is in the **abstractMqttDriver** palette under the **AbstractMqttPoints** folder.

Figure 18 Mqtt String Object Subscribe Ext properties

Property Sheet

Proxy Ext (Mqtt String Object Subscribe Ext)

- Status: {stale}
- Fault Cause:
- Enabled: ☒ true
- Device Facets: >> ⌚
- Conversion: Default
- Tuning Policy Name: defaultPolicy
- Read Value: - {ok}
- Write Value: - {ok}
- Topic:
- QoS: Fire And Forget (0)

To access these properties, expand **Config→Drivers→AbstractMqttDriverNetwork**, expand a device, expand the **Points** folder, expand this point and double-click the **Proxy Ext**.

In addition to the standard properties (Status, Fault Cause and Enabled) these properties are unique to this component.

Property	Value	Description
Device facets	additional properties	<p>Determine how values are formatted for display depending on the context and the type of data. Examples include engineering units and decimal precision for numeric types, and descriptive value (state) text for boolean and enum types.</p> <p>With the exception of proxy points (with possible defined device facets), point facets do not affect how the framework processes the point's value.</p> <p>Besides control points, various other components have facets too. For example, many <code>kitControl</code> and schedule components have facets. Details about point facets apply to these components too, unless especially noted.</p> <p>You access facets by clicking an Edit button or a chevron >>. Both open an Edit Facets window.</p>
Conversion	Drop-down list (defaults to <code>Default</code>)	<p>Defines how the system converts proxy extension units to parent point units.</p> <p><code>Default</code> automatically converts similar units (such as Fahrenheit to Celsius) within the proxy point.</p> <p>NOTE: In most cases, the standard <code>Default</code> conversion is best.</p> <p><code>Linear</code> applies to voltage input, resistive input and voltage output writable points. Works with linear-acting devices. You use the <code>Scale</code> and <code>Offset</code> properties to convert the output value to a unit other than that defined by device facets.</p> <p><code>Linear With Unit</code> is an extension to the existing linear conversion property. This specifies whether the unit conversion should occur on "Device Value" or "Proxy Value". The new linear with unit convertor, will have a property to indicate whether the unit conversion should take place before or after the scale/offset conversion.</p> <p><code>Reverse Polarity</code> applies only to Boolean input and relay output writable points. Reverses the logic of the hardware binary input or output.</p> <p><code>500 Ohm Shunt</code> applies to voltage input points only. It reads a 4-to-20mA sensor, where the <code>Ui</code> input requires a 500 ohm resistor wired across (shunting) the input terminals.</p> <p><code>Tabular Thermistor</code> applies to only a Thermistor input point and involves a custom resistance-to-temperature value response curve for Type 3 Thermistor temperature sensors.</p> <p><code>Thermistor Type 3</code> applies to an Thermistor Input point, where this selection provides a "built-in" input resistance-to-temperature value response curve for Type 3 Thermistor temperature sensors.</p> <p><code>Generic Tabular</code> applies to non-linear support for devices other than for thermistor temperature sensors with units in temperature. <code>Generic Tabular</code> uses a lookup table method similar to the "Thermistor Tabular" conversion, but without predefined output units.</p>

Property	Value	Description
Tuning Policy Name	drop-down list	Selects a network tuning policy by name. This policy defines stale time and minimum and maximum update times. During polling, the system uses the tuning policy to evaluate both write requests and the acceptability (freshness) of read requests.
Read Value	read only	Displays the last value read from the device, expressed in device facets.
Write value	read-only	Displays the last value written using device facets.
Topic	text	Defines a string that the broker uses to filter messages for each client. A topic consists of one or more topic levels. Each level is separated by a forward slash (topic level separator).
Qos	drop-down list	Defines the Qos (Quality of Service) level, which configures how persistently the broker and client attempt to ensure that a message is received. <code>Fire And Forget (0)</code> delivers the message once with no confirmation. <code>Atleast Once (1)</code> delivers the message at least once with confirmation required. <code>Exactly Once (2)</code> delivers the message exactly once by using a four-step handshake. Messages may be sent at any Qos level, and clients may attempt to subscribe to topics at any Qos level.

abstractMqttDriver-MqttEnumObjectPublishExt

This component configures the properties for publishing enum data.

This component is in the **abstractMqttDriver** palette under the **AbstractMqttPoints** folder.

Figure 19 Mqtt Enum Object Publish Point Ext properties

Property Sheet

Proxy Ext (Mqtt Enum Object Publish Ext)

Status	{stale}
Fault Cause	
Enabled	<input checked="" type="checkbox"/> true
Device Facets	» ⌚
Conversion	Default
Tuning Policy Name	defaultPolicy
Read Value	- {ok}
Write Value	- {ok}
Topic	
Qo S	Fire And Forget (0)
Retained	<input checked="" type="checkbox"/> true
Publish Message On Change	<input checked="" type="checkbox"/> true

To access these properties, expand **Config→Drivers→AbstractMqttDriverNetwork**, expand a device, expand the **Points** folder, expand this point and double-click the **Proxy Ext**.

In addition to the standard properties (Status, Fault Cause and Enabled) these properties are unique to this component.

Property	Value	Description
Device facets	additional properties	<p>Determine how values are formatted for display depending on the context and the type of data. Examples include engineering units and decimal precision for numeric types, and descriptive value (state) text for boolean and enum types.</p> <p>With the exception of proxy points (with possible defined device facets), point facets do not affect how the framework processes the point's value.</p> <p>Besides control points, various other components have facets too. For example, many <code>kitControl</code> and schedule components have facets. Details about point facets apply to these components too, unless especially noted.</p> <p>You access facets by clicking an Edit button or a chevron >>. Both open an Edit Facets window.</p>
Conversion	Drop-down list (defaults to <code>Default</code>)	<p>Defines how the system converts proxy extension units to parent point units.</p> <p><code>Default</code> automatically converts similar units (such as Fahrenheit to Celsius) within the proxy point.</p> <p>NOTE: In most cases, the standard <code>Default</code> conversion is best.</p> <p><code>Linear</code> applies to voltage input, resistive input and voltage output writable points. Works with linear-acting devices. You use the <code>Scale</code> and <code>Offset</code> properties to convert the output value to a unit other than that defined by device facets.</p> <p><code>Linear With Unit</code> is an extension to the existing linear conversion property. This specifies whether the unit conversion should occur on "Device Value" or "Proxy Value". The new linear with unit convertor, will have a property to indicate whether the unit conversion should take place before or after the scale/offset conversion.</p> <p><code>Reverse Polarity</code> applies only to Boolean input and relay output writable points. Reverses the logic of the hardware binary input or output.</p> <p><code>500 Ohm Shunt</code> applies to voltage input points only. It reads a 4-to-20mA sensor, where the <code>Ui</code> input requires a 500 ohm resistor wired across (shunting) the input terminals.</p> <p><code>Tabular Thermistor</code> applies to only a Thermistor input point and involves a custom resistance-to-temperature value response curve for Type 3 Thermistor temperature sensors.</p> <p><code>Thermistor Type 3</code> applies to an Thermistor Input point, where this selection provides a "built-in" input resistance-to-temperature value response curve for Type 3 Thermistor temperature sensors.</p> <p><code>Generic Tabular</code> applies to non-linear support for devices other than for thermistor temperature sensors with units in temperature. <code>Generic Tabular</code> uses a lookup table method</p>

Property	Value	Description
		similar to the “Thermistor Tabular” conversion, but without predefined output units.
Tuning Policy Name	drop-down list	Selects a network tuning policy by name. This policy defines stale time and minimum and maximum update times. During polling, the system uses the tuning policy to evaluate both write requests and the acceptability (freshness) of read requests.
Read Value	read-only	Displays the last value read from the device, expressed in device facets.
Write value	read-only	Displays the last value written using device facets.
Topic	text	Defines a string that the broker uses to filter messages for each client. A topic consists of one or more topic levels. Each level is separated by a forward slash (topic level separator).
QoS	drop-down list	Defines the QoS (Quality of Service) level, which configures how persistently the broker and client attempt to ensure that a message is received. <code>Fire And Forget (0)</code> delivers the message once with no confirmation. <code>Atleast Once (1)</code> delivers the message at least once with confirmation required. <code>Exactly Once (2)</code> delivers the message exactly once by using a four-step handshake. Messages may be sent at any QoS level, and clients may attempt to subscribe to topics at any QoS level.
Retained	true (default) or false	Configures the broker to save the message for the specified topic as its Last Will and Testament (last known good value). It is useful for newly-connected subscribers to receive the last retained message for the topic immediately after subscribing and extremely helpful to provide status updates from components and devices on individual topics.
Publish Message on Change	true (default) or false	Determines if the client publishes a message when a value changes.

abstractMqttDriver-MqttEnumObjectSubscribeExt

This component configures the properties for subscribing enum data.

This component is in the **abstractMqttDriver** palette under the **AbstractMqttPoints** folder.

Figure 20 Mqtt Enum Object Subscribe Point Ext properties

Property Sheet

Proxy Ext (Mqtt Enum Object Subscribe Ext)

Status	{stale}
Fault Cause	
Enabled	<input checked="" type="checkbox"/> true
Device Facets	>> ⌚
Conversion	Default
Tuning Policy Name	defaultPolicy
Read Value	- {ok}
Write Value	- {ok}
Topic	
Qo S	Fire And Forget (0)

To access these properties, expand **Config→Drivers→AbstractMqttDriverNetwork**, expand a device, expand the **Points** folder, expand this point and double-click the **Proxy Ext**.

In addition to the standard properties (Status, Fault Cause and Enabled) these properties are unique to this component.

Property	Value	Description
Device facets	additional properties	<p>Determine how values are formatted for display depending on the context and the type of data. Examples include engineering units and decimal precision for numeric types, and descriptive value (state) text for boolean and enum types.</p> <p>With the exception of proxy points (with possible defined device facets), point facets do not affect how the framework processes the point's value.</p> <p>Besides control points, various other components have facets too. For example, many <code>kitControl</code> and schedule components have facets. Details about point facets apply to these components too, unless especially noted.</p> <p>You access facets by clicking an Edit button or a chevron >>. Both open an Edit Facets window.</p>
Conversion	Drop-down list (defaults to <code>Default</code>)	<p>Defines how the system converts proxy extension units to parent point units.</p> <p><code>Default</code> automatically converts similar units (such as Fahrenheit to Celsius) within the proxy point.</p> <p>NOTE: In most cases, the standard <code>Default</code> conversion is best.</p> <p><code>Linear</code> applies to voltage input, resistive input and voltage output writable points. Works with linear-acting devices. You use the <code>Scale</code> and <code>Offset</code> properties to convert the output value to a unit other than that defined by device facets.</p> <p><code>Linear With Unit</code> is an extension to the existing linear conversion property. This specifies whether the unit conversion should occur on "Device Value" or "Proxy Value". The new linear with unit convertor, will have a property to indicate whether the unit conversion should take place before or after the scale/offset conversion.</p>

Property	Value	Description
		<p><code>Reverse Polarity</code> applies only to Boolean input and relay output writable points. Reverses the logic of the hardware binary input or output.</p> <p><code>500 Ohm Shunt</code> applies to voltage input points only. It reads a 4-to-20mA sensor, where the <code>Ui</code> input requires a 500 ohm resistor wired across (shunting) the input terminals.</p> <p><code>Tabular Thermistor</code> applies to only a Thermistor input point and involves a custom resistance-to-temperature value response curve for Type 3 Thermistor temperature sensors.</p> <p><code>Thermistor Type 3</code> applies to an Thermistor Input point, where this selection provides a “built-in” input resistance-to-temperature value response curve for Type 3 Thermistor temperature sensors.</p> <p><code>Generic Tabular</code> applies to non-linear support for devices other than for thermistor temperature sensors with units in temperature. Generic Tabular uses a lookup table method similar to the “Thermistor Tabular” conversion, but without predefined output units.</p>
Tuning Policy Name	drop-down list	<p>Selects a network tuning policy by name. This policy defines stale time and minimum and maximum update times.</p> <p>During polling, the system uses the tuning policy to evaluate both write requests and the acceptability (freshness) of read requests.</p>
Read Value	read-only	Displays the last value read from the device, expressed in device facets.
Write value	read-only	Displays the last value written using device facets.
Topic	text	Defines a string that the broker uses to filter messages for each client. A topic consists of one or more topic levels. Each level is separated by a forward slash (topic level separator).
Qos	drop-down	<p>Defines the Qos (Quality of Service) level, which configures how persistently the broker and client attempt to ensure that a message is received.</p> <p><code>Fire And Forget (0)</code> delivers the message once with no confirmation.</p> <p><code>Atleast Once (1)</code> delivers the message at least once with confirmation required.</p> <p><code>Exactly Once (2)</code> delivers the message exactly once by using a four-step handshake.</p> <p>Messages may be sent at any Qos level, and clients may attempt to subscribe to topics at any Qos level.</p>

Chapter 5 Plugins

Topics covered in this chapter

- ◆ Abstract Mqtt Driver Device Ux Manager
- ◆ Mqtt Client Driver Point Manager

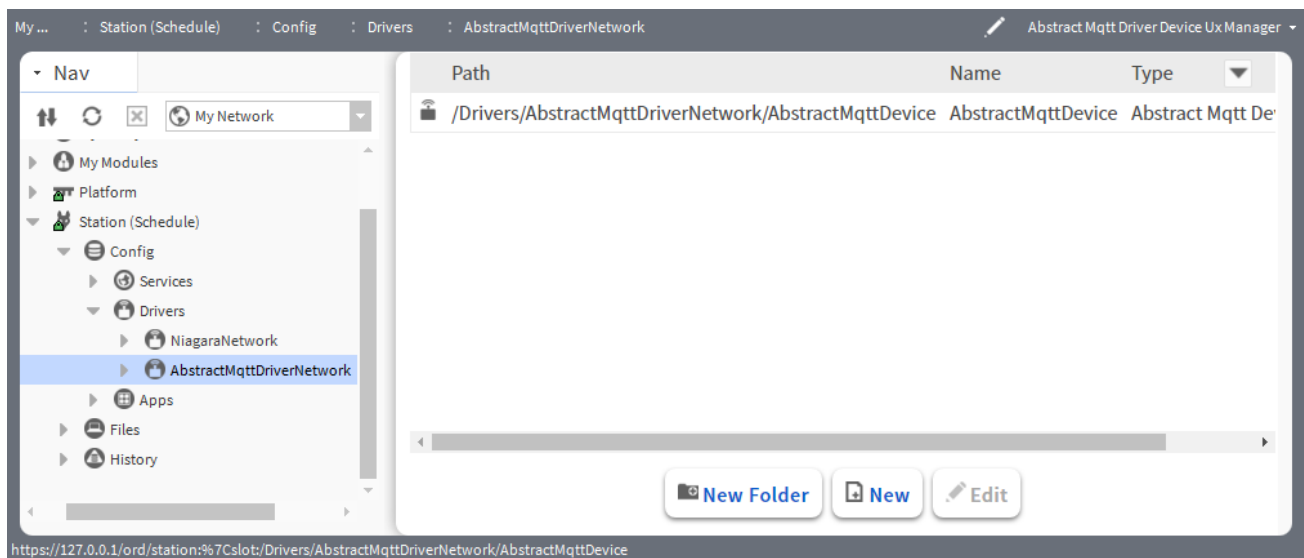
Plugins provide views of components and can be accessed in many ways. For example, double-click a component in the **Nav** tree to see its default view. In addition, you can right-click on a component and select from its **Views** menu.

For summary documentation on any view, select **Help→On View** (F1) from the menu or press F1 while the view is open.

Abstract Mqtt Driver Device Ux Manager

The default view of the AbstractMqttDriverNetwork is the **Abstract Mqtt Driver Device Ux Manager**.

Figure 21 Abstract Mqtt Driver Device Ux Manager view



To open this view, **Config→Drivers**, double-click the **AbstractMqttDriverNetwork**.

Columns

Following are the columns and buttons for this view:

Column name	Description
Path	Displays the path of the device.
Name	Displays the name of the device.
Type	Displays the type of device.

Buttons

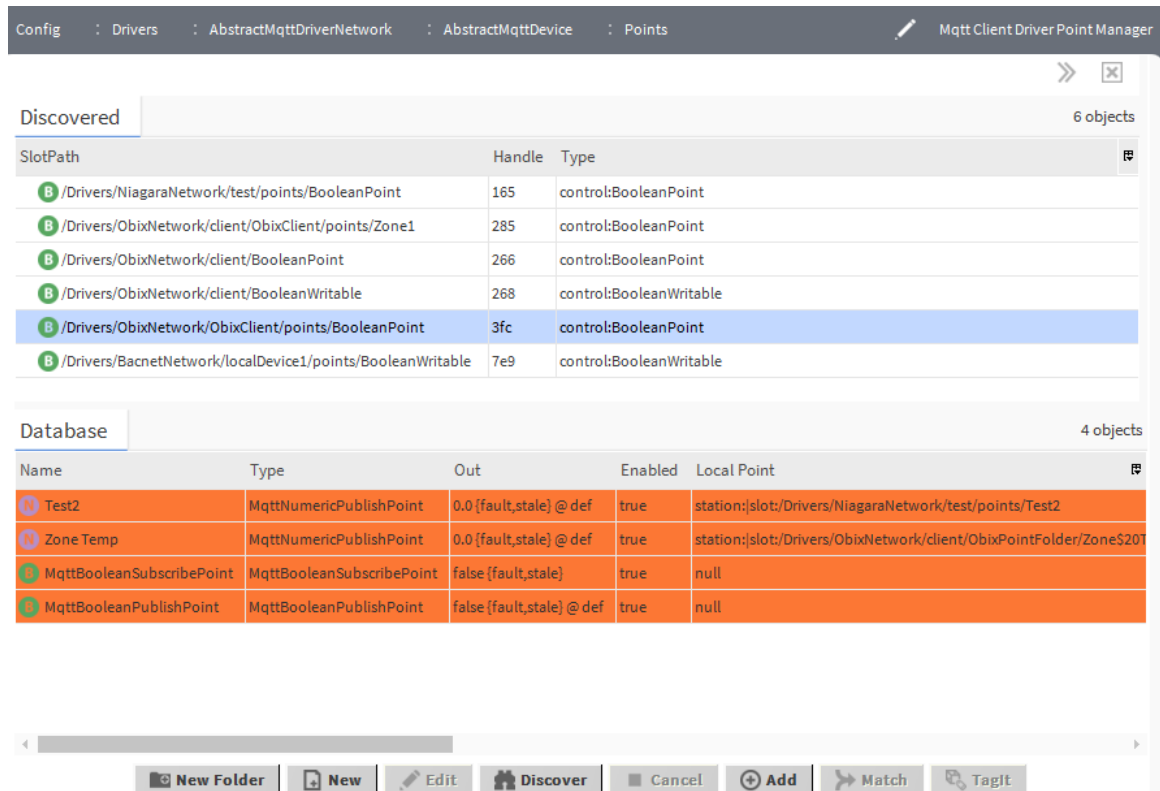
- **New Folder** creates a new folder for devices. Each such folder provides its own set of manager views.
- **New** creates a new device record in the database.

- **Edit** opens the device's database record for updating.

Mqtt Client Driver Point Manager

The **Mqtt Client Driver Point Manager** is default view of the **Points** container.

The **Mqtt Client Driver Point Manager** view splits into two panes when you click the **Discover** button.



To open this view, right-click **Points** container and select **Views→Mqtt Client Driver Point Manager**.

Discovered pane

As in other manager views, when you click **Discover** this manager goes to Learn mode, splits into two panes, and executes a discover job. Discover also returns multiple objects, each of which occupies one row in this view.

Following are the columns for Discovered pane

Column name	Description
SlotPath	Displays the slot sheet path.
Handle	Displays unique ID for the point.
Type	Displays the point type.

Database Pane

Following are the columns and buttons for database pane.

Column name	Description
Name	Displays the name of the point.
Type	Displays the point type.
Out	Displays the output.
Enabled	Indicates if the point is enabled.
Local point	Displays the path where the points are located.
Local Point Slot	Displays the slot of local point.
Facets	Displays the details of facets.
Tuning Policy	Displays the configured tuning policy.
Conversion	Displays the conversion type.
Fault cause	Display the reason for a fault.
Read value	Displays the last status of the object.

Buttons

- **New Folder** creates a new folder for devices. Each such folder provides its own set of manager views.
- **New** creates a new device record in the database.
- **Edit** opens the device's database record for updating.
- **Discover** runs a discover job to locate installed devices, which appear in the **Discovered** pane. This view has a standard appearance that is similar to all **Device Manager** views.
- **Cancel** ends the current discovery job.
- **Add** inserts into the database a record for the discovered and selected object.
- **Match** associates a discovered device with a record that is already in the database.
- **TagIt** associates metadata, such as location or unique configuration with the object.

Chapter 6 Windows

Topics covered in this chapter

- ◆ New Device Windows
- ◆ Add point windows

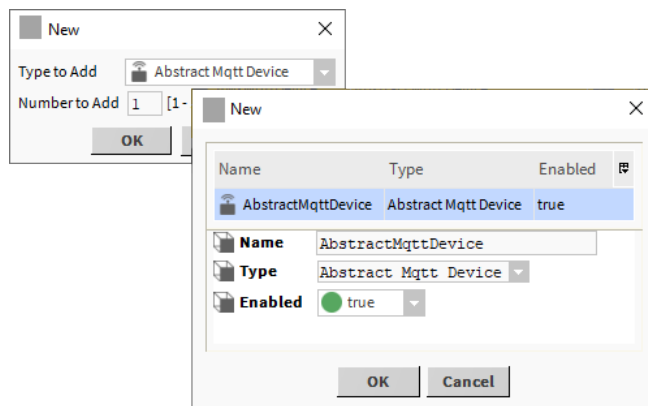
Windows create and edit database records or collect information when accessing a component. You access them by dragging a component from a palette into a station or by clicking a button.

Windows do not support **On View (F1)** and **Guide on Target** help. To learn about the information each contains, search the help system for key words.

New Device Windows

This window configures device properties.

Figure 22 New device window



To access these windows, expand **Config→Drivers**, double-click **AbstractMqttDriverNetwork** and click **New**.

Property	Value	Description
Type to Add	drop-down list	Selects the name of the device to add.
Number to Add	number (defaults to 1)	Configures how many devices to add.
Name	text	Provides descriptive text that reflects the identity of the entity or logical grouping. In this case, it defines the device name.
Type	drop-down list	Selects the type of device.
Enabled	true (default) or false	Activates (true) and deactivates (false) use of the object (network, device, point, component, table, schedule, descriptor, etc.).

Add point windows

After a discovery job runs, this window opens to configure the points as they are added to the database.

Figure 23 New point window properties

The screenshot shows two overlapping 'New' windows. The top window is partially obscured by a larger one below it. Both windows have a 'Type to Add' dropdown set to 'MqttBooleanPublishPoint' and a 'Number to Add' field set to '1' with a range '[1 - 100]'.

The larger 'New' window in the foreground displays a table with the following columns: Name, Type, Enabled, Local Point, Topic, Local Point Slot, and Facets. The first row is highlighted in blue and contains the following values: 'MqttBooleanPublishPoint', 'MqttBooleanPublishPoint', 'true', 'null', an empty field, and 'trueText=true,falseText=fal'.

Below the table is a configuration panel with the following fields and values:

- Name:** MqttBooleanPublishPoint
- Type:** MqttBooleanPublishPoint
- Enabled:** true (radio button selected)
- Local Point:** null
- Topic:** (empty text field)
- Local Point Slot:** na
- Facets:** trueText=true,falseText=false
- Tuning Policy Name:** defaultPolicy
- Device Facets:** (empty field with a plus icon)
- Conversion:** Default

At the bottom of the window are 'OK' and 'Cancel' buttons.

To access these properties, expand **Config**→**Drivers**→**AbstractMqttDriverNetwork**, expand a device, double-click **Points** and click **New**.

Property	Value	Description
Type to Add	drop-down list	Selects from among eight point types.
Number to Add	number	Defines how many points of the selected type to add.
Name	text	Provides descriptive text that reflects the identity of the entity or logical grouping.
Type	drop-down list	Selects the type of point.
Enabled	true (default) or false	Activates (<i>true</i>) and deactivates (<i>false</i>) use of the object (network, device, point, component, table, schedule, descriptor, etc.).
Local Point	identifier (defaults to the ord)	Identifies the point in the station. When added from the discovered points, Local Point refers to the ord of the point. If null, the point is not linked to any point by default. You must link the point manually, if needed.
Topic	text	Defines a string that the broker uses to filter messages for each client. A topic consists of one or more topic levels. Each level is separated by a forward slash (topic level separator).
Local Point Slot	for subscribe points: slot number; for publish points, defaults to out.	Links directly to the functional block. Options are available for subscribe points only.

Property	Value	Description
Facets	additional properties	<p>Determine how values are formatted for display depending on the context and the type of data. Examples include engineering units and decimal precision for numeric types, and descriptive value (state) text for boolean and enum types.</p> <p>With the exception of proxy points (with possible defined device facets), point facets do not affect how the framework processes the point's value.</p> <p>Besides control points, various other components have facets too. For example, many <code>kitControl</code> and <code>schedule</code> components have facets. Details about point facets apply to these components too, unless especially noted.</p> <p>You access facets by clicking an Edit button or a chevron >>. Both open an Edit Facets window.</p>
Tuning Policy Name	drop-down list	<p>Selects a network tuning policy by name. This policy defines stale time and minimum and maximum update times.</p> <p>During polling, the system uses the tuning policy to evaluate both write requests and the acceptability (freshness) of read requests.</p>

Property	Value	Description
Device Facets	additional properties	<p>Determine how values are formatted for display depending on the context and the type of data. Examples include engineering units and decimal precision for numeric types, and descriptive value (state) text for boolean and enum types.</p> <p>With the exception of proxy points (with possible defined device facets), point facets do not affect how the framework processes the point's value.</p> <p>Besides control points, various other components have facets too. For example, many <code>kitControl</code> and schedule components have facets. Details about point facets apply to these components too, unless especially noted.</p> <p>You access facets by clicking an Edit button or a chevron >>. Both open an Edit Facets window.</p>
Conversion	drop-down list (defaults to <code>Default</code>)	<p>Defines how the system converts proxy extension units to parent point units.</p> <p><code>Default</code> automatically converts similar units (such as Fahrenheit to Celsius) within the proxy point.</p> <p>NOTE: In most cases, the standard <code>Default</code> conversion is best.</p> <p><code>Linear</code> applies to voltage input, resistive input and voltage output writable points. Works with linear-acting devices. You use the <code>Scale</code> and <code>Offset</code> properties to convert the output value to a unit other than that defined by device facets.</p> <p><code>Linear With Unit</code> is an extension to the existing linear conversion property. This specifies whether the unit conversion should occur on "Device Value" or "Proxy Value". The new linear with unit convertor, will have a property to indicate whether the unit conversion should take place before or after the scale/offset conversion.</p> <p><code>Reverse Polarity</code> applies only to Boolean input and relay output writable points. Reverses the logic of the hardware binary input or output.</p> <p><code>500 Ohm Shunt</code> applies to voltage input points only. It reads a 4-to-20mA sensor, where the <code>Ui</code> input requires a 500 ohm resistor wired across (shunting) the input terminals.</p> <p><code>Tabular Thermistor</code> applies to only a Thermistor input point and involves a custom resistance-to-temperature value response curve for Type 3 Thermistor temperature sensors.</p> <p><code>Thermistor Type 3</code> applies to an Thermistor Input point, where this selection provides a "built-in" input resistance-to-temperature value response curve for Type 3 Thermistor temperature sensors.</p> <p><code>Generic Tabular</code> applies to non-linear support for devices other than for thermistor temperature sensors with units in temperature. <code>Generic Tabular</code> uses a lookup table method similar to the "Thermistor Tabular" conversion, but without predefined output units.</p>

Index

A

Abstract MQTT Driver	19
Abstract Mqtt Driver Device Ux Manager	61
abstractMqttDriver-AbstractMqttDevice	27
abstractMqttDriver- AbstractMqttDriverNetwork	25
abstractMqttDriver-AwsJitpMqttAuthenticator	30
abstractMqttDriver-AwsMqttAuthenticator	32
abstractMqttDriver-AzureIotSasToken	35
abstractMqttDriver- AzureMqttSasAuthenticator	33
abstractMqttDriver-AzureSasTokenParameters	35
abstractMqttDriver-GcpAuthenticator	38
abstractMqttDriver-GcplotParameters	39
abstractMqttDriver-GenericMqttAuthenticator	36
abstractMqttDriver-MqttCallbackRouter	40
abstractMqttDriver- MqttClientDriverDeviceFolder	41
abstractMqttDriver- MqttClientDriverPointDeviceExt	41
abstractMqttDriver- MqttClientDriverPointFolder	41
abstractMqttDriver-TokenParameters	40
authenticator adding	9
default	36
authenticators	8, 32, 38
AwsMqttDevice	27
Azure SAS tokens authentication	19

B

Boolean publish point	42
subscribe point	44
broker	7

C

certificate authentication	9
client	7
components	25, 30, 35
Connecting Abstract MQTT Driver to Microsoft Azure IoT Hub	19

D

DefaultMqttDevice	27
device	7
adding	9
components	27, 33
document change log	5

E

enum publish point	55
subscribe point	57

G

GCP	15–17
GCP Authenticator	15
GCP Console	17
GcpMqttDevice	16, 27
guide	5

I

introduction	7
--------------------	---

L

license requirement	8
---------------------------	---

M

Microsoft Azure	19
Abstract MQTT driver	19
Microsoft Azure IoT Hub	19
MQTT	7
Mqtt Client Driver Point Manager	62

N

network	8
component	25
New device window	65
New point	65
numeric publish point	46
subscribe point	48

P

plugins	61
PointCallbackHandler	41
points adding	12
discovering	10
publish	7, 12

R

requirements	8
--------------------	---

S

secure communication	9
Setting up Azure IoT Hub on Azure portal.....	19
Setting up station to connect to Azure IoT Hub....	21
software version	8
string	
publish point	50
subscribe point	53
subscribe	7

W

windows.....	65
--------------	----