Industrial Assembly Solutions - ADH

Atlas Copco Industrial Assembly Solutions
Atlas Copco IAS GmbH

Visitor Address: Gewerbestr. 52
75015 Bretten – Germany
Visit Atlas Copco at: http://www.atlascopco.com

Developer Installation Guide

MQTT – SYS6000 V4

Author: **Carlos Santiago Lamas Camacho**
Tel.: **+49 163 1375568**
e-mail: **carlos.lamascamacho@atlascopco.com**

Version: **V 1.0**

Date: **Tuesday, May 6, 2025**

1 Revision History

Date	Version	Modifications	by
3.12.2025	V1.0		

Table of contents

2 Introduction

Welcome to the MQTT Developer Installation Guide for the **SYS6000 V4 System**. This guide is designed to provide detailed instructions for installing and configuring MQTT (Message Queuing Telemetry Transport) specifically for the **SYS6000 V4 system.**

In this guide, you will find step-by-step instructions to ensure a smooth installation and configuration process. The guide covers everything from the initial requirements and data needed from the customer to the deployment, configuration, and testing of MQTT on the **SYS6000 V4 system**.

3 Requirements

3.1 Firmware

Firmware

The firmware is a specific class of software that provides low-level control for a device's hardware. It is typically stored in the device's read-only memory and is essential for the device's functionality. Firmware updates can improve performance, add new features, or fix bugs.

Accessing Firmware via HTTP

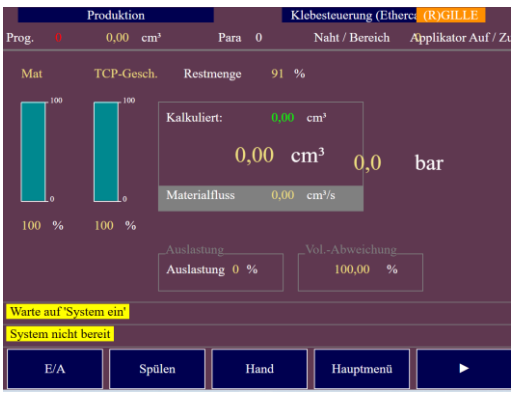
HTTP (Hypertext Transfer Protocol) is a protocol used for transmitting hypertext requests and information on the internet. To access the firmware page, you need to enter the IP address of the SYS6000 V4 controller in your web browser.

Here is a list of available controllers for the SYS6000 V4 system:

- **Controller 1: IP Address - 10.49.38.107**
- **Controller 2: IP Address - 10.49.38.171**
- **Controller 3: IP Address – 10.49.38.159**

Steps to Access Firmware Sys6000

1. Open your web browser: Use any web browser of your choice (e.g., Chrome, Firefox, Edge).
2. Enter the IP address: Type the IP address of the SYS6000 V4 controller into the address bar of your browser and press Enter.



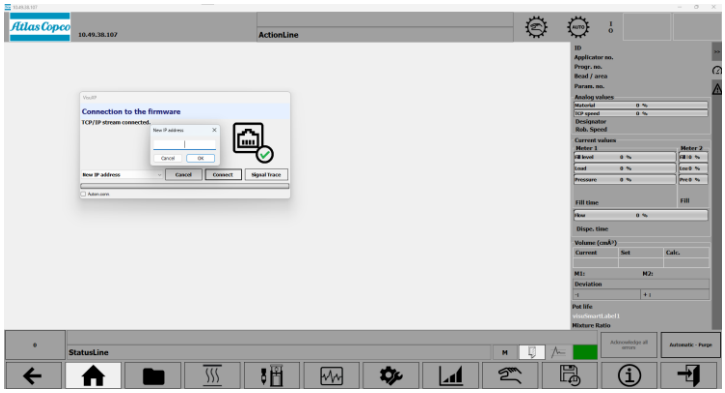
3. Go to the Main Menu (Hauptmenü) and click on "Info". There you will find details about the **hardware, MAC address**, **Linux version**, and more.

Important Note:

Pay attention to the Linux version. The version of Linux determines the MQTT installation.

3.2 VisuXP

Visualization software compatible with SYS6000 V4 or later versions. You can install it from the same directory where you find the hardware information.



To access, you need to enter the respective **IP address**. While to make modifications, you will need a user and password that will be assigned to you.

3.3 MQTT

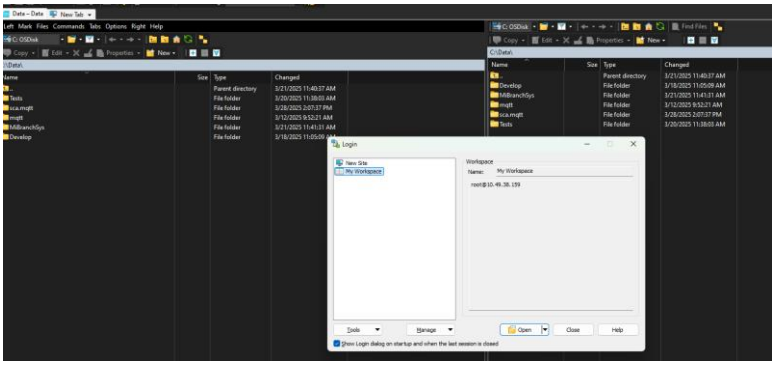
Lightweight **messaging protocol** ideal for low-power devices and networks with limited bandwidth. It is widely used in industrial applications for its efficiency and reliability in transmitting data between devices.

You can find the respective versions in the PDP directory under **Data Component Services > Releases > MQTT > SYS6000V4 > 5.6 > Binaries.**

- **Self-contained:** For Linux versions lower than 3.2.42 (includes .NET libraries).
- **Framework:** For Linux versions 3.2.42 or higher.

3.4 WinSCP

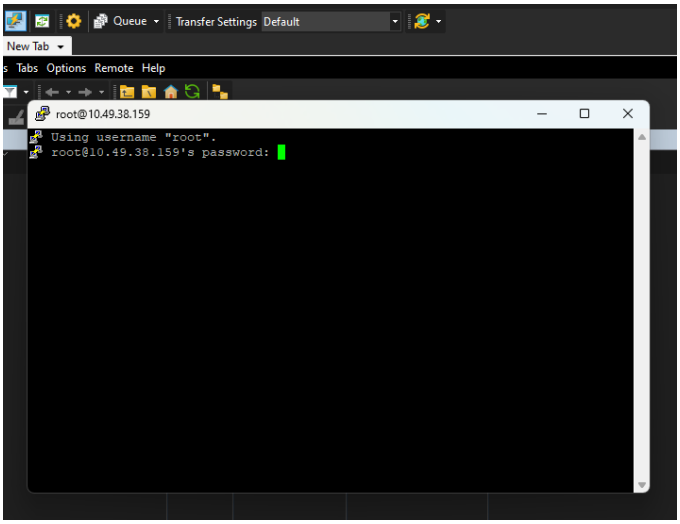
WinSCP is a **file manager** for Windows that supports FTP, SFTP, WebDAV, and SCP protocols. It allows you to move documents from your computer to the SYS6000 system without any issues.



- **Download Link**: https://winscp.net/eng/download.php

3.5 PuTTY

PuTTY is a open-source **terminal emulator** that supports various network protocols, including SSH, Telnet, and SCP. It is used for remote access and file transfers. It provides a terminal solution for executing commands, such as starting the firmware, on the SYS6000 system.



- **Download Link**: https://www.chiark.greenend.org.uk/~sgtatham/putty/latest.html

3.6 MQTT Explorer

MQTT Explorer is a **graphical tool for monitoring and debugging MQTT** communication. It allows users to visualize messages published on a broker, subscribe to various topics, and analyze real-time data, making it easier to integrate and maintain MQTT-based systems.

- **Download Link**: https://mqtt-explorer.com/

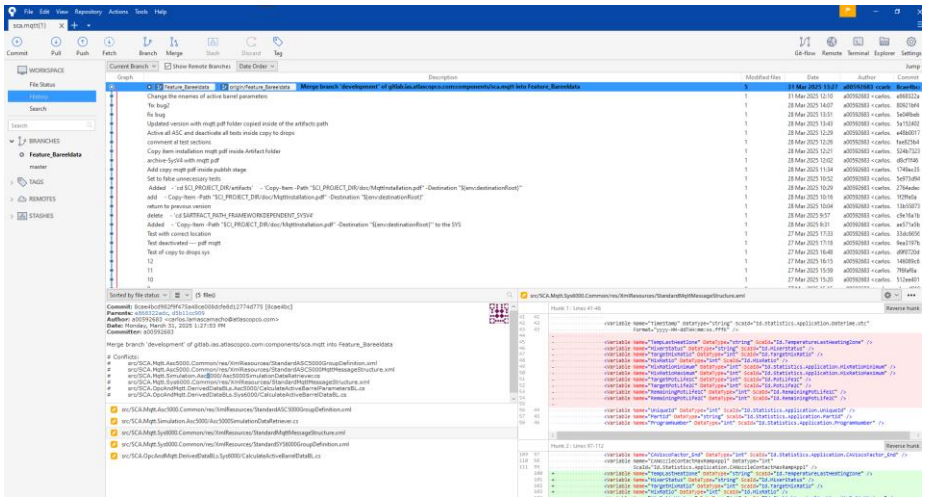
3.7 Visual Studio

Integrated development environment (IDE) designed for developing, debugging, and testing applications in various programming languages, including C#, Python, and C++. It provides advanced features such as intelligent code completion, integrated version control, and powerful debugging tools, making it an essential tool for software development.

- **Download Link**: https://visualstudio.microsoft.com/

3.8 SourceTree

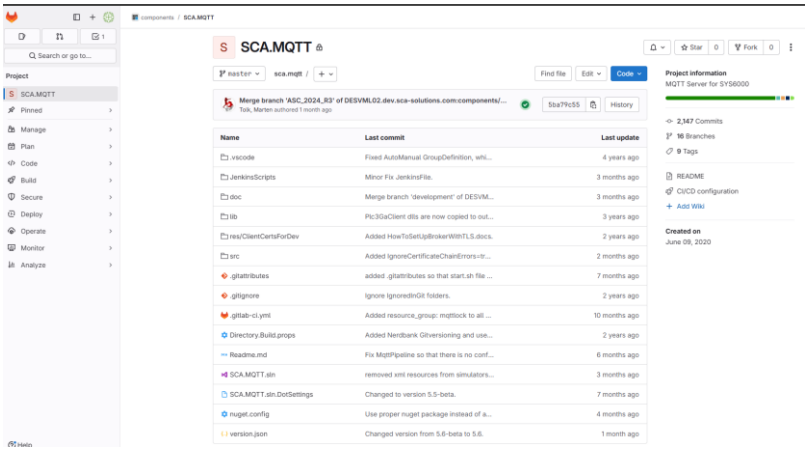
Graphical Git client that simplifies version control management by providing an intuitive interface for handling repositories, branches, commits, and merges.



- **Download Link**: https://www.sourcetreeapp.com/

3.9 GitLab

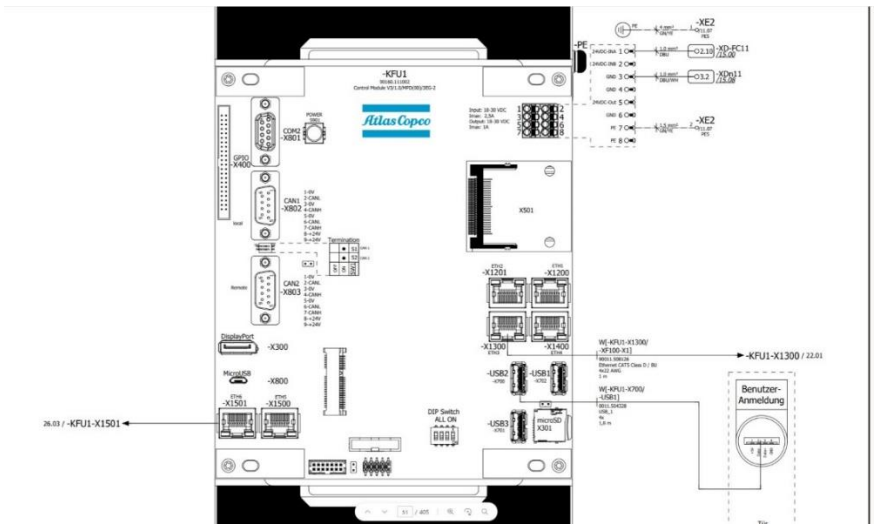
In our **GitLab repository SCA.MQTT**, we follow a structured branching strategy to ensure stability and efficient development. The **master branch** contains the stable, production-ready code, while the **development branch** is used for integrating and testing new features before merging into the main branch. For each new feature or task, a separate feature branch is created, allowing isolated development and testing before being merged back into the development branch. This workflow ensures a clear and organized process, maintaining code quality and facilitating collaboration.



- Repository: https://10.49.38.12/components/sca.mqtt/-/tree/master?ref_type=heads

3.10 Hardware (Console Module V3)

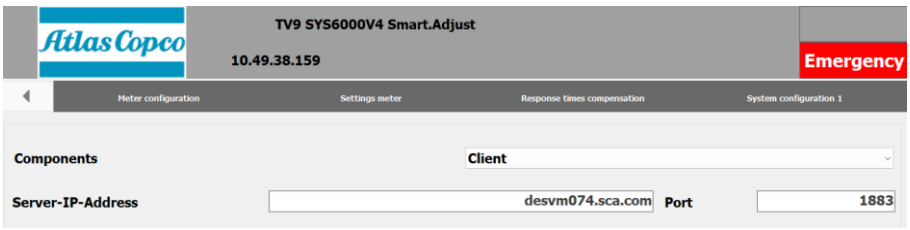
The controller unit manages **various ports and connections**, ensuring optimal operation of the system. Some of the key components include ports labeled X1, X2, X3, etc., with specific pin configurations. Functions of certain ports and switches are annotated for easy identification.



Ensure that the computer is connected to the **ETH3** port of the control module to be connected to the network and Linux.

4 Data needed from the costumer

1. **MQTT Broker IP Address**
2. In case of **secure Mqtt connection**, we need the following data
    - Mqtt broker **host name** (CommonName in Server Certificate)
    - **Client certificate** and private key in case of
    - **CA certificate** (used to sign client and server certificate)

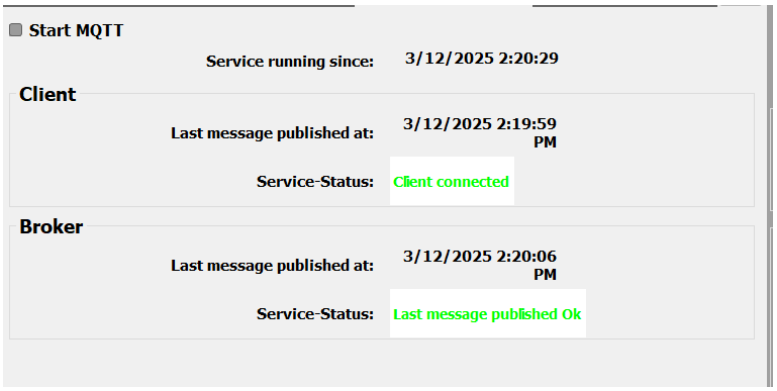


5 Firmware Backup

1. Access WinSCP and enter the IP address of the firmware as the host using the port **22**.
2. Open PuTTY and use the username **root** and the password **sca** to log in.

- You can use the following commands in PuTTY:
    - **systemctl start sys6000**
    - **systemctl stop sys6000**
    - **systemctl restart sys6000**

3. **Backup Important folders** such as MQTT by stopping the SYS6000 service and using WinSCP to transfer those folders from the SYS6000 system to your computer.
4. Also ensure that **MQTT is not running**, You can use the VisuXP to activate or deactivate the service.



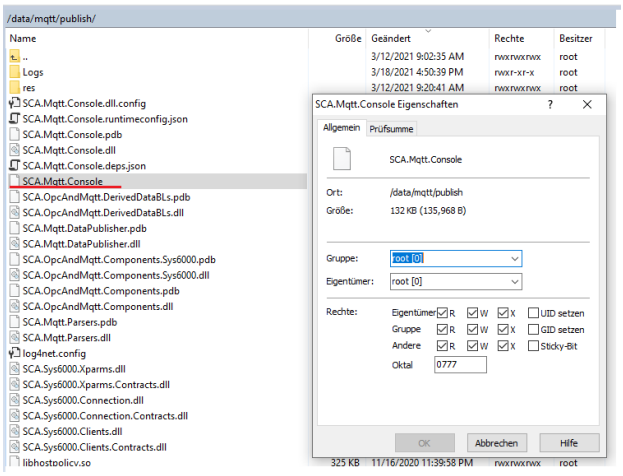
6 Deploy MQTT

6.1 Via Script

1. Copy the files extracted from the step 3.3 to **/data/mqtt.**
2. The publish folder should exist at **/data/mqtt/publish.**
3. Give execute rights to **install.sh**
4. Execute **install.sh.**

6.2 Manually

1. Copy **mqtt.service** in mqtt folder to **/etc/systemd/system/**
2. Navigate to the mqtt folder and replace the it by pasting the new documents with the respective version (**Depending on the Linux version**)
3. Give permissions **0664** to **mqtt.service** file as it follows:
4. Give permissions **0666** to full **publish folder** (in case logs are activated).
5. Give full permission to **SCA.Mqtt.Console**



6.3 Secure MQTT

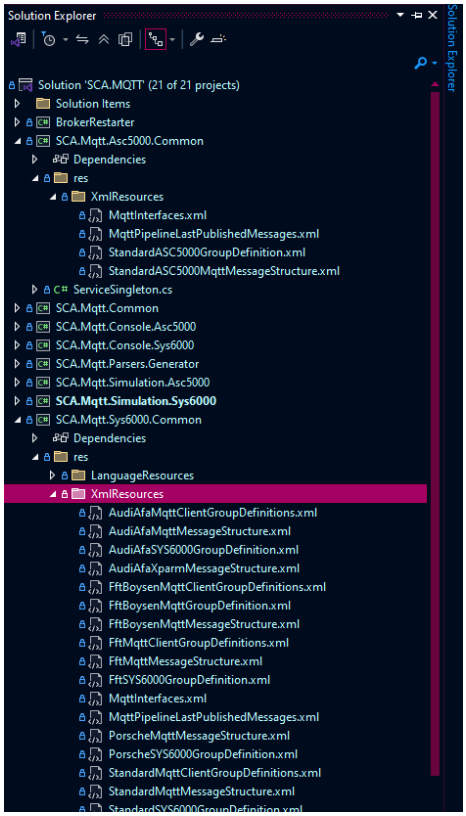
If secure mqtt is needed, then do the following steps:

1. Get the **Client.crt** and get the **ClientPrivate.key** from the customer.
2. Create **client.pfx** using **Openssl** command given below

   openssl pkcs12 -export -in Client.crt -inkey ClientPrivate.key -out Client.pfx
3. **Client.pxf** will be generated in the folder where you executed this command.
4. Copy the created **Client.pfx** to **/data/mqtt/publish/res/clientCert**

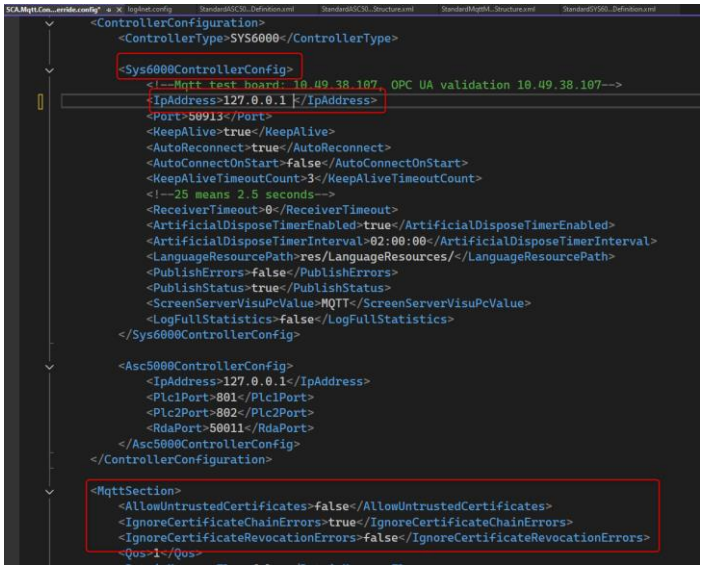
Important Note:

You can find the XML definition and message files for each system and client in the following directory: **/data/mqtt/publish/res/XmlResources/** This directory contains files for different clients and systems, such as Audi, Fiat, Ford, etc.



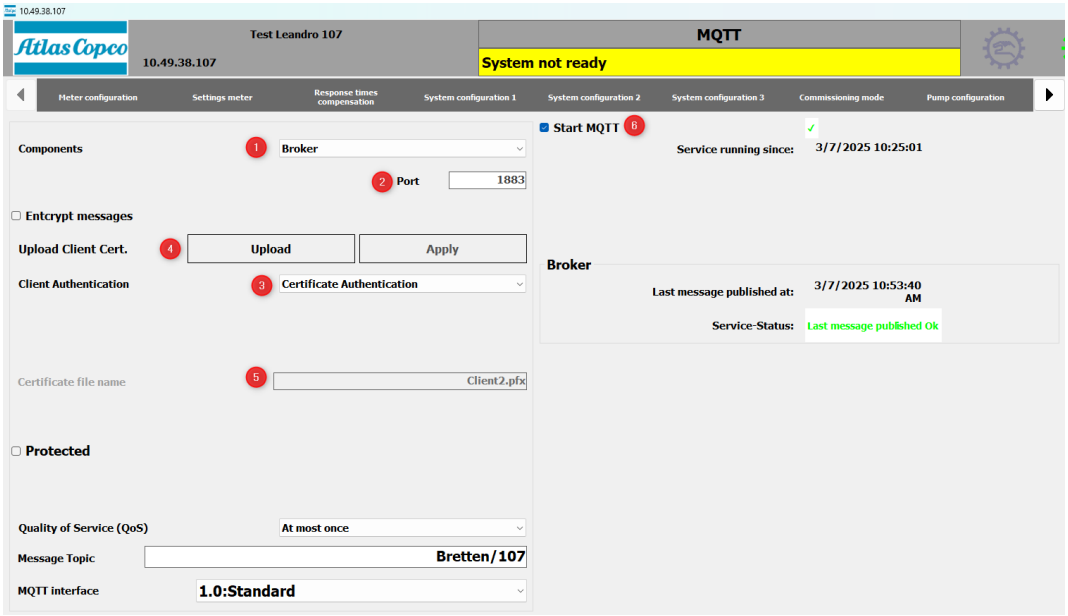
7 Configure and Start MQTT

1. Go to **/data/mqtt/publish** and copy the contents of **SCA.Mqtt.Console.dll.config** to a new file **SCA.Mqtt.Console.dll.override.config**. The customer specific configuration should be added/changed in **SCA.Mqtt.Console.dll.override.config.**
2. Make sure the following settings are correct in **SCA.Mqtt.Console.dll.config** and in **SCA.Mqtt.Console.dll.override.config** if this file exists.
    - **IpAddress** in Sys6000ControllerConfig is **127.0.0.1** (1)
    - **AllowUntrustedCertificates** in MqttSection is **false** (2). Setting this to ‘true’ is not recommended (but could solve problems with untrusted certificate).
    - **IgnoreCertificateChainErrors** in MqttSection is **invisible or false**. Setting this to ‘true’ is not recommended (but could solve problems with untrusted certificate)



8 VisuXP MQTT Settings

1. In VisuXP go to Settings, navigate to the **MQTT tab** and configure the following settings:



2. Enter the **IP Address** of the MQTT Broker.

- For an **insecure MQTT connection:**
  - Use the MQTT Server IP of the customer or the PC where the Mosquitto broker is installed.
- For a **secure MQTT connection**:
  - If AllowUntrustedCertificates in MqttSection is set to false, use the common name of the Server certificate.
  - If AllowUntrustedCertificates is set to true (not recommended), you can use the MQTT server's IP address.

3. Enter the Port number. Default is **1883**, unless changed in the Mosquitto configuration file.

4. For **Client Authentication** select a Client Authentication method:

  - **Anonymous** (anyone can connect).

- **Certificate Authentication** (required when Secure MQTT is enabled):
  - Provide the **Certificate File Name** (e.g., client.pfx).
- **Username and Password / broker requires username,**

5. For **MQTT Interface** select the appropriate interface based on the customer setup.

6. Check **Start MQTT** to initiate the connection.

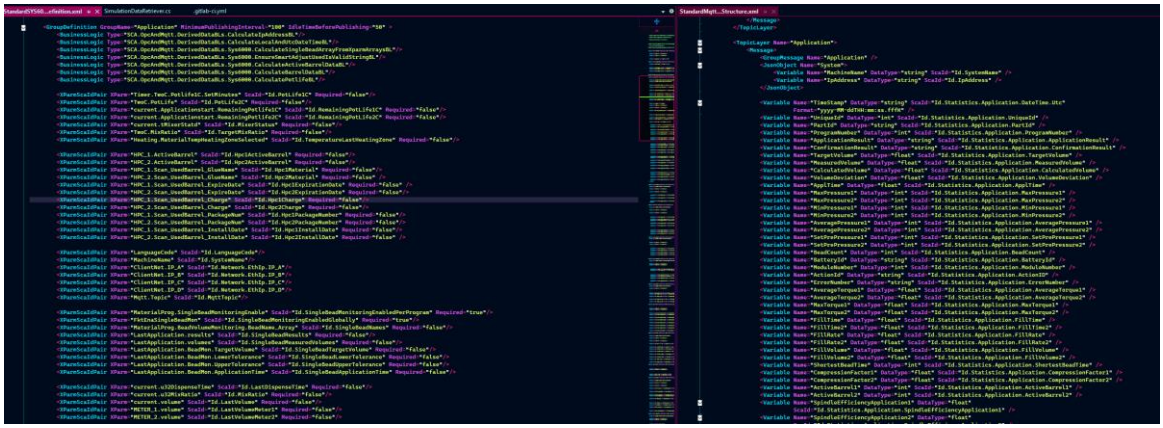
- If successful, a **green checkmark** will appear.
- If you see a red cross, check the logs for errors in **/data/mqtt/publish/Log.**

9 Add parameters

When adding new parameters to the MQTT structure in SYS6000 V4, it is important to correctly define them in the **GroupDefinition** file and structure their representation in the **Message Structure** file. Each parameter requires the following elements:

- **ScaId (Unique ID)**: A unique identifier for the parameter.
- **XParm (Real-time Data Source)**: The source from which the data is obtained in real time.
- **JSON Name**: The name under which the parameter will appear in the JSON message sent to the client.

The **GroupDefinition** file defines the variables, associates them with real-time data sources, and includes business logic rules for additional processing.



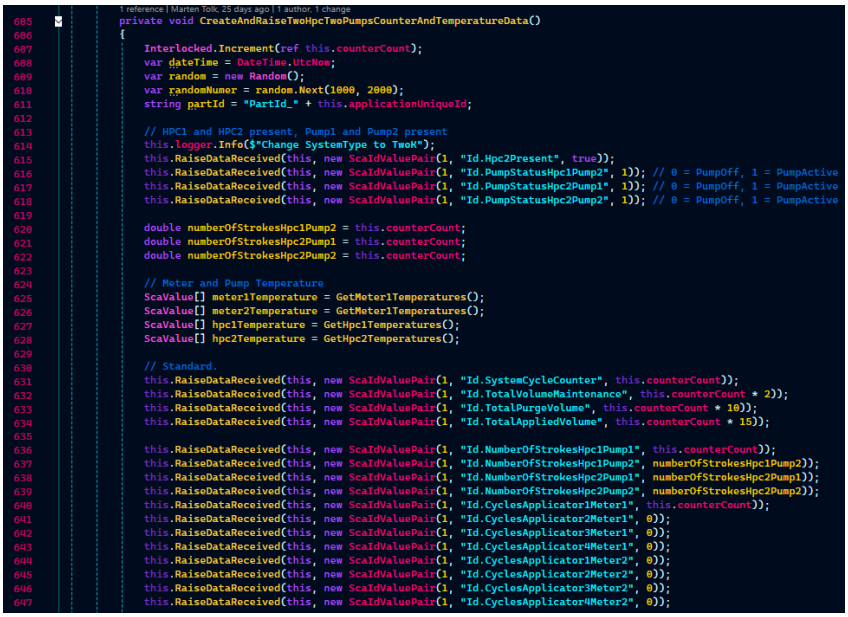
Once defined in **GroupDefinition**, the parameters must be included in **MessageStructure**, ensuring they appear in the MQTT messages. Each **Variable** element in the **MessageStructure** references the **ScaId** from **GroupDefinition**.

- The **DataType** should match the expected data format.
- The JSON structure ensures the client receives the correct values.
- Ensure the **ScaId** used in both **GroupDefinition** and **Message Structure** are consistent.
- If a parameter requires preprocessing, add a **BusinessLogic** rule in **GroupDefinition**.
- Validate new parameters using **MQTT Explorer** to confirm correct data transmission.

By following these steps, new parameters can be seamlessly integrated into the SYS6000 V4 MQTT system while maintaining data integrity and reliability.

10 Simulation

The project includes a dedicated simulation module called **SimulationDataRetriever**. This module is specifically designed to generate realistic but artificial parameter values, allowing for an effective way to test different functionalities without needing a live production environment.



```
1 reference | Marten Tolk, 25 days ago | 1 author, 1 change
private void CreateAndRaiseTwoHpcTwoPumpsCounterAndTemperatureData()
{
    Interlocked.Increment(ref this.counterCount);
    var dateTime = DateTime.UtcNow;
    var random = new Random();
    var randomNumer = random.Next(1000, 2000);
    string partId = "PartId_" + this.applicationUniqueId;

    // HPC1 and HPC2 present, Pump1 and Pump2 present
    this.logger.Info($"Change SystemType to TwoK");
    this.RaiseDataReceived(this, new ScaIdValuePair(1, "Id.Hpc2Present", true));
    this.RaiseDataReceived(this, new ScaIdValuePair(1, "Id.PumpStatusHpc1Pump2", 1)); // 0 = PumpOff, 1 = PumpActive
    this.RaiseDataReceived(this, new ScaIdValuePair(1, "Id.PumpStatusHpc2Pump1", 1)); // 0 = PumpOff, 1 = PumpActive
    this.RaiseDataReceived(this, new ScaIdValuePair(1, "Id.PumpStatusHpc2Pump2", 1)); // 0 = PumpOff, 1 = PumpActive

    double numberOfStrokesHpc1Pump2 = this.counterCount;
    double numberOfStrokesHpc2Pump1 = this.counterCount;
    double numberOfStrokesHpc2Pump2 = this.counterCount;

    // Meter and Pump Temperature
    ScaValue[] meter1Temperature = GetMeter1Temperatures();
    ScaValue[] meter2Temperature = GetMeter1Temperatures();
    ScaValue[] hpc1Temperature = GetHpc1Temperatures();
    ScaValue[] hpc2Temperature = GetHpc2Temperatures();

    // Standard.
    this.RaiseDataReceived(this, new ScaIdValuePair(1, "Id.SystemCycleCounter", this.counterCount));
    this.RaiseDataReceived(this, new ScaIdValuePair(1, "Id.TotalVolumeMaintenance", this.counterCount * 2));
    this.RaiseDataReceived(this, new ScaIdValuePair(1, "Id.TotalPurgeVolume", this.counterCount * 10));
    this.RaiseDataReceived(this, new ScaIdValuePair(1, "Id.TotalAppliedVolume", this.counterCount * 15));

    this.RaiseDataReceived(this, new ScaIdValuePair(1, "Id.NumberOfStrokesHpc1Pump1", this.counterCount));
    this.RaiseDataReceived(this, new ScaIdValuePair(1, "Id.NumberOfStrokesHpc1Pump2", numberOfStrokesHpc1Pump2));
    this.RaiseDataReceived(this, new ScaIdValuePair(1, "Id.NumberOfStrokesHpc2Pump1", numberOfStrokesHpc2Pump1));
    this.RaiseDataReceived(this, new ScaIdValuePair(1, "Id.NumberOfStrokesHpc2Pump2", numberOfStrokesHpc2Pump2));
    this.RaiseDataReceived(this, new ScaIdValuePair(1, "Id.CyclesApplicator1Meter1", this.counterCount));
    this.RaiseDataReceived(this, new ScaIdValuePair(1, "Id.CyclesApplicator2Meter1", 0));
    this.RaiseDataReceived(this, new ScaIdValuePair(1, "Id.CyclesApplicator3Meter1", 0));
    this.RaiseDataReceived(this, new ScaIdValuePair(1, "Id.CyclesApplicator4Meter1", 0));
    this.RaiseDataReceived(this, new ScaIdValuePair(1, "Id.CyclesApplicator1Meter2", 0));
    this.RaiseDataReceived(this, new ScaIdValuePair(1, "Id.CyclesApplicator2Meter2", 0));
    this.RaiseDataReceived(this, new ScaIdValuePair(1, "Id.CyclesApplicator3Meter2", 0));
    this.RaiseDataReceived(this, new ScaIdValuePair(1, "Id.CyclesApplicator4Meter2", 0));
```

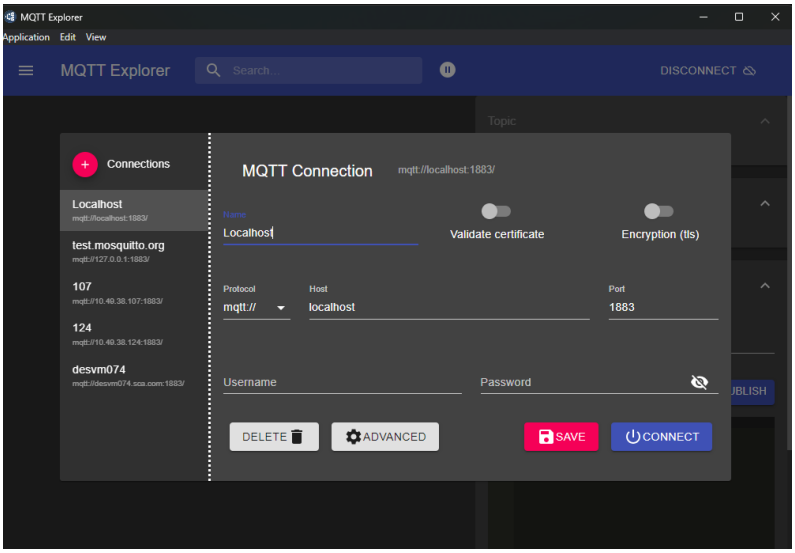
The **SimulationDataRetriever** works by providing fake but plausible data that mimics real-world scenarios. It generates values for different parameters such as active barrels, material types, expiration dates, usernames, charge numbers, package numbers, and installation dates. These values are systematically assigned and updated, simulating real changes that would occur in an actual production setup.

By running the simulation, users can send these generated MQTT messages and visualize them in **MQTT Explorer.** This provides a clear and structured way to verify that the messages are properly formatted, transmitted, and received by the system. Furthermore, this approach allows developers and engineers to analyze how the system responds to different data inputs, ensuring that all business logic and validation rules function as expected.

11 Validation

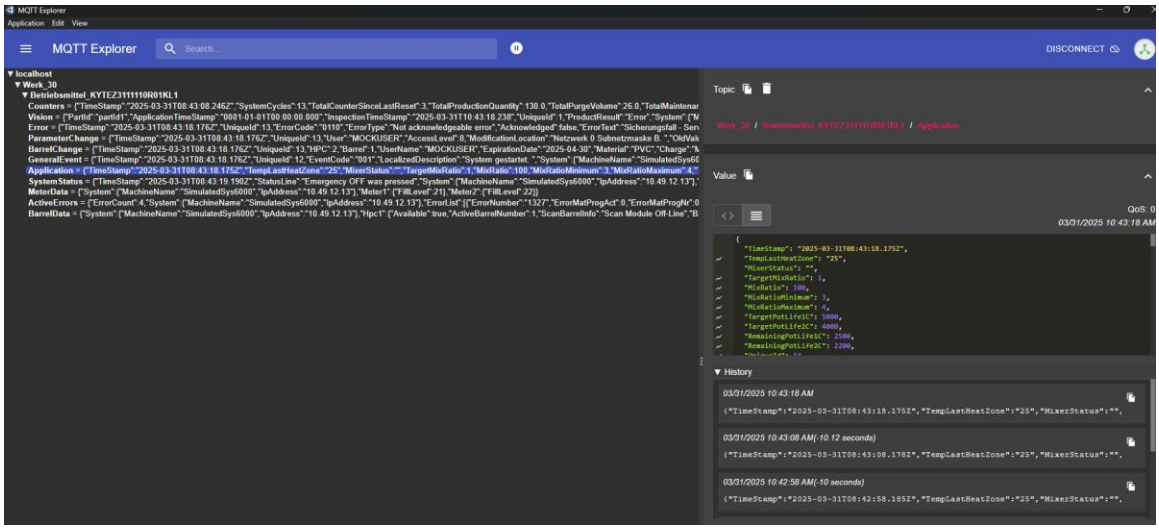
To verify the connection to the MQTT broker, follow these steps:

1. **Open MQTT Explorer** and click on the **"+"** button under "Connections" to create a new connection.



2. In the **MQTT Connection** window, enter the following details:
    - **Name:** A descriptive name for the connection.
    - **Protocol:** Keep the default setting as mqtt://.
    - **Host:** Enter the broker's IP address or hostname.
    - **Port:** Set to 1883 (default for non-TLS connections) or 8883 (for TLS connections, if enabled).
3. If authentication is required, enter the **Username** and **Password** assigned to you.
4. If the broker uses encryption, enable the **Encryption (TLS)** toggle and, if needed, select **Validate Certificate**.
5. Click **Save** to store the connection settings.
6. Click **Connect** to establish the connection.

If the connection is successful, you will see a hierarchical list of topics on the left panel. You can expand these topics to inspect the messages being transmitted.



To verify data transmission:

- If messages are being published on the topic, they will appear in real-time.

If the connection fails:

- Check that the broker address, port, and credentials are correct.
- Ensure the MQTT broker is running and accessible from your network.
- If using TLS, verify that the correct certificates are installed.