ProSoft TECHNOLOGY

Where Automation Connects.

MVI69E-LDM-MQTT

MQ Telemetry Transport

CompactLogix® Platform

February 10, 2021

QUICK START GUIDE

Your Feedback Please

We always want you to feel that you made the right decision to use our products. If you have suggestions, comments, compliments or complaints about our products, documentation, or support, please write or call us.

How to Contact Us

ProSoft Technology, Inc.
+1 (661) 716-5100
+1 (661) 716-5101 (Fax)
www.prosoft-technology.com
support@prosoft-technology.com

MVI69E-LDM-MQTT Quick Start Guide

February 10, 2021

Content Disclaimer

This documentation is not intended as a substitute for and is not to be used for determining suitability or reliability of these products for specific user applications. It is the duty of any such user or integrator to perform the appropriate and complete risk analysis, evaluation and testing of the products with respect to the relevant specific application or use thereof. Neither ProSoft Technology nor any of its affiliates or subsidiaries shall be responsible or liable for misuse of the information contained herein. Information in this document including illustrations, specifications and dimensions may contain technical inaccuracies or typographical errors. ProSoft Technology makes no warranty or representation as to its accuracy and assumes no liability for and reserves the right to correct such inaccuracies or errors at any time without notice. If you have any suggestions for improvements or amendments or have found errors in this publication, please notify us.

For professional users in the European Union

If you wish to discard electrical and electronic equipment (EEE), please contact your dealer or supplier for further information.

Prop 65 Warning – Cancer and Reproductive Harm – www.P65Warnings.ca.gov

Important Installation Instructions

Power, Input, and Output (I/O) wiring must be in accordance with Class I, Division 2 wiring methods, Article 501-4 (b) of the National Electrical Code, NFPA 70 for installation in the U.S., or as specified in Section 18-1J2 of the Canadian Electrical Code for installations in Canada, and in accordance with the authority having jurisdiction. The following warnings must be heeded:

WARNING - EXPLOSION HAZARD - SUBSTITUTION OF COMPONENTS MAY IMPAIR SUITABILITY FOR CLASS I, DIV. 2;

WARNING - EXPLOSION HAZARD - WHEN IN HAZARDOUS LOCATIONS, TURN OFF POWER BEFORE REPLACING OR WIRING MODULES

WARNING - EXPLOSION HAZARD - DO NOT DISCONNECT EQUIPMENT UNLESS POWER HAS BEEN SWITCHED OFF OR THE AREA IS KNOWN TO BE NON-HAZARDOUS.

Class 2 Power

Agency Approvals and Certifications

Please visit our website: www.prosoft-technology.com

Contents

1 Start Here

MQTT is a lightweight messaging protocol, ideal for passing IIoT (Industrial Internet of Things) data from remote locations. For more information, see the MQTT v3.1.1 Specification.

This Quick Start Guide describes how to:

- Obtain sample applications.
- Run MQTT application programs.
- Setup your LDM development environment.
- Customize and build your own MQTT application programs.

1.1 MQTT-LDM Generic and SparkplugB

1.1.1 Using the MQTT LDM Generic Implementation

The **mqtt-ldm** is a software library available for free from the ProSoft Technology website.

This document provides step-by-step information on how to enable communication between Rockwell Automation®'s CompactLogix® PLC and a simple MQTT broker, using MQTT Explorer.

The goal is to read data from the PLC and publish to topics on an MQTT broker. The module also allows subscription to topics on an MQTT broker to receive new values published by other MQTT clients and then write them to the PLC. This is accomplished by running the sample application on the MVI69E-LDM while connected to the MQTT broker.

The MVI69E-LDM acts as a Message Queuing Telemetry Transport (MQTT) client. This document references two public MQTT brokers that are available on the Internet for testing.

Sample data exchanges can be accomplished in unencrypted mode and in encrypted mode.

The sample application can be used as-is, or you can follow the step-by-step instructions on how to build it from source code. The sample application features may be extended to suit your needs.

1.1.2 Using the SparkplugB Implementation

This document provides step-by-step information on how to enable communication between a CompactLogix PLC and Inductive Automation®'s Ignition, using MQTT SparkplugB.

The goal is to read data from the CompactLogix PLC, then publish it by a topic to the MQTT broker. Another MQTT client will subscribe to that topic in that MQTT broker, so that the client can present the PLC information to the user. Additionally, the MQTT client can publish data of that topic to the MQTT broker. The LDM-MQTT client can subscribe to it and write the data to the PLC.

In the MQTT protocol, one MQTT client does not require to be programmatically linked to another MQTT client.

The MVI69E-LDM acts as a Message Queuing Telemetry Transport (MQTT) client. This document references the Ignition Gateway (by Inductive Automation), a Windows Service. It will require an up-to-date installation of three files from Cirrus link Solutions. It provides Sparkplug B-based MQTT software modules that perform the service of MQTT Distributor (*Broker*), the MQTT Engine (*Subscribing Client*), and the MQTT Transmission (*Publishing Client*).

Videos are available to help you to become familiar with the Ignition implementation of the MQTT and its use by the LDM-MQTT module:

Video 1: What is MQTT?

https://inductiveautomation.com/resources/video/what-is-mqtt

Video 2: How MQTT Works

https://inductiveautomation.com/resources/video/how-mqtt-works

Video 3: MQTT Sparkplug Specification

https://inductiveautomation.com/resources/video/mqtt-sparkplug-specification

Video 4: MQTT & Ignition

https://inductiveautomation.com/resources/video/mqtt-ignition

Video 5: MQTT Distributor Module

https://inductiveautomation.com/resources/video/mqtt-distributor-module

Video 6: MQTT Transmission Module

https://inductiveautomation.com/resources/video/mqtt-transmission-module

Video 7: Using the MQTT Transmission Module to Publish Data

https://inductiveautomation.com/resources/video/using-the-mqtt-transmission-module-to-publish-data

Video 8: MQTT Engine Module

https://inductiveautomation.com/resources/video/mqtt-engine-module

Video 9: Allow Outbound Tag Writes

https://inductiveautomation.com/resources/video/allow-outbound-tag-writes

Video 10: Primary Host ID Setting

https://inductiveautomation.com/resources/video/primary-host-id-setting

Video 11: How to Set Up Transport Layer Security

https://inductiveautomation.com/resources/video/how-to-set-up-transport-layer-security

Video 12: Set Up Store-and-Forward System

https://inductiveautomation.com/resources/video/set-up-storeandforward-system

Note: Each video contains a transcript available on its webpage.

2 Obtaining Sample Applications

2.1 MVI69E-LDM-MQTT Zip File

The MVI69E-LDM-MQTT zip file is available at www.prosoft-technology.com. This file contains both the **Generic** and **SparkplugB** implementations.

1. Navigate to the **MVI69E-LDM** product webpage.
2. Create a folder on your PC named *C:\Workspace* and download the ***MVI69E-LDM-MQTT-xxx.zip*** (where **xxx** is version number) to this folder.
3. Unzip the file in this folder.
4. Make note of the location of the firmware file.

The interface library contains the following components:

c:\Workspace\	Subfolder	Description
aws-iot-device-sdk-embedded-C		Open source library AWS IoT Device SDK C v4.0.0
cJSON		cJSON, open source C library to parse JSON formatted configuration file
mqtt-ldm		MQTT-LDM Library root folder
	build	Location where target binaries are created during build
	docker	Toolchain to build source code and Docker® configuration files to start container with build environment
	mqtt-ldm-lib	Source code of the library mqtt-ldm-lib, wrapper on top of the AWS IoT Device SDK
	scripts	Build scripts
mqtt-ldm-sample-app-mvi69e		Source code of the sample application, with default configuration file; firmware build scripts and some runtime scripts; and optional Visual Studio 2017 solution and project files; Visual Studio 2017 solution and project files for sample application (optional)
	src	Source code of the sample application
	test-*	Folders with sample configurations file and certificates to connect to different MQTT brokers
	Firmware\ mvi69e-ldm.firmware_<version>_<date>.firmware	MVI69E-LDM MQTT sample application
mvi69e-ldm		Source code of dependency libraries required to communicate with the PLC
tahu		Eclipse Tahu, an open source library with implementation of the Sparkplug format of encoding/decoding
LDM_MQTT_.ACD		CompactLogix Ladder Logic file

2.2 Obtain MQTT Explorer

If you are implementing MQTT-LDM Generic, navigate to http://mqtt-explorer.com and download MQTT Explorer to a Windows 10 PC. You do not need to perform this step if implementing SparkplugB.

Note: The MQTT Explorer Windows installer version had issues with storing connection settings; therefore, the portable version is recommended.

2.3 Obtain Ignition

If you are implementing MQTT-LDM SparkplugB, navigate to https://inductiveautomation.com/downloads/.

You will need to complete a form to gain access to a free trial of Ignition.

3 Connecting to the MVI69E-LDM

3.1 Physical Connections

1. With the MVI69E-LDM in the CompactLogix rack, connect the top Ethernet port to your local network, and connect to the Windows 10 PC.
2. Use the middle Ethernet port to connect to the network where the MQTT broker is running.
3. Use the bottom port to connect the Windows 10 PC via USB to a 1756-EN2T module. This is for debugging purposes using TeraTerm (Open Source Telnet terminal).

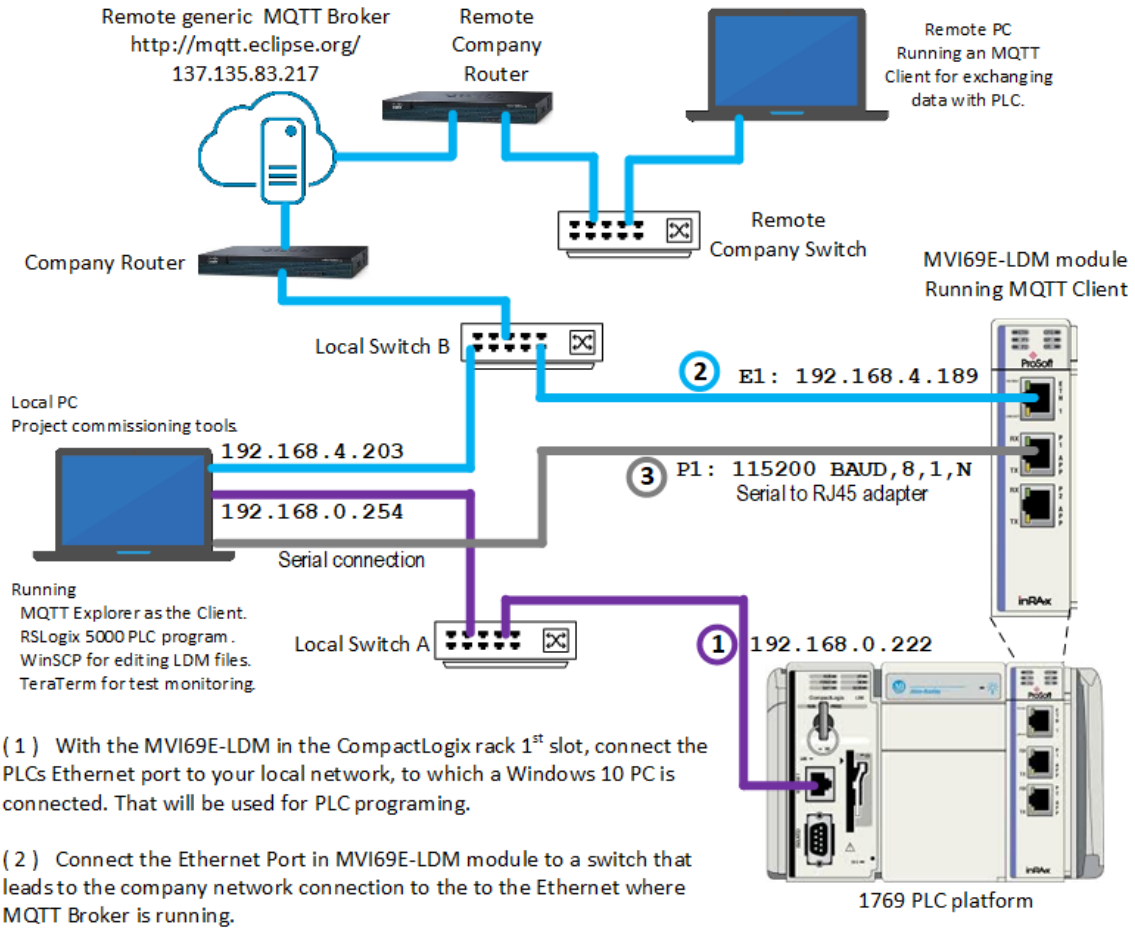

Figure 1: *MVI69E-LDM-MQTT Generic Type Communication Topology*

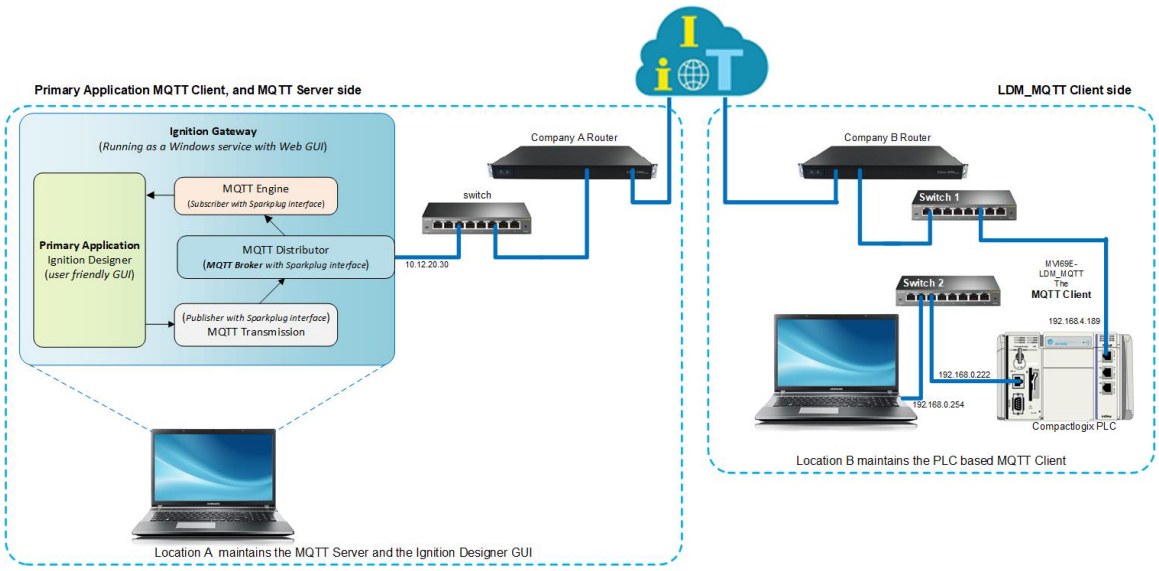

Figure 2: *MVI69E-LDM-MQTT SparkplugB Communication Topology*

3.2 Configuring the CompactLogix PLC

1. Open the LDM_MQTT.ACD program and change the appropriate CompactLogix chassis type to match your hardware and firmware.
2. Download LDM_MQTT.ACD file to the CompactLogix processor by choosing **COMMUNICATIONS > WHO ACTIVE > DOWNLOAD**.
3. Install the sample application.

Note: If there is an application currently running on the MVI69E-LDM, be sure to back it up before proceeding.

3.2.1 Firmware Update

1. Obtain the .firmware file from the MVI69E-LDM-MQTT zip file, or that was built in the "Creating a Build" section in this document.
2. Download the .firmware file to the module via the module's webpage. Refer to the MVI69E-LDM Developer Manual for details.

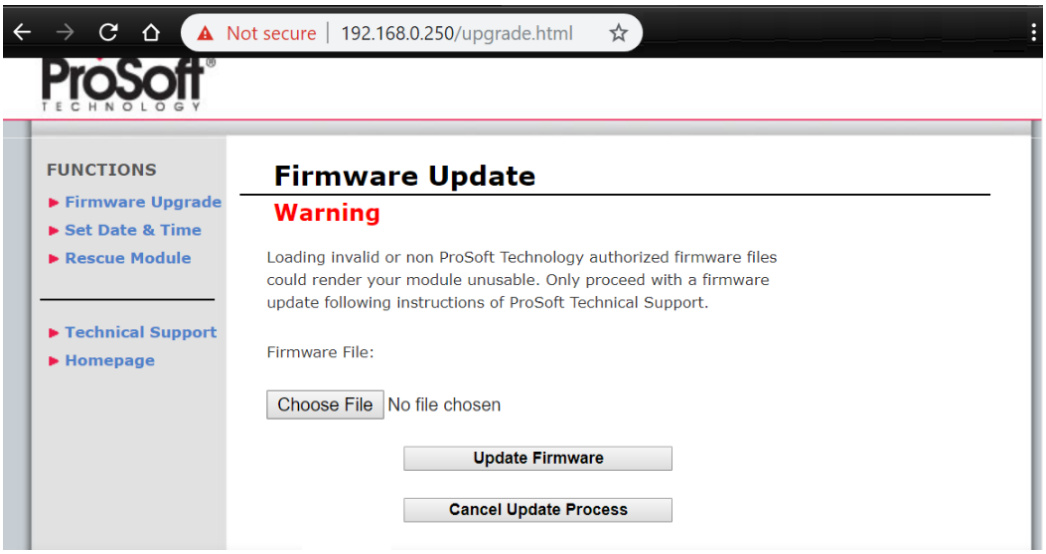

Figure 3: *Firmware Update Page*

3. At the end of the Firmware Update process, the module reboots and connections to the module are lost during reboot.
4. The sample application starts running automatically after reboot. It still needs to be configured.

3.3 Configuring the MVI69E-LDM's EtherNet/IP Address

Configure the MVI69E-LDM's Ethernet port IP addresses by modifying the `/etc/network/interfaces` file on the module. Refer to Figure 1 for a sample communication topology.

Refer to the *MVI69E-LDM Developer Manual* for detailed information about the interfaces file. Use an FTP client such as **WinSCP**™ to edit the **eth0** and **eth1** sections of the interface file.

Interfaces file	
Parameters	**Values**
# We always want the loopback interface.	
#	
auto lo	
iface lo inet loopback	
#An example Ethernet card setup:	(broadcast and gateway are optional)
#	
auto	eth0
iface eth0	inet static
address	102.168.0.250
network	192.168.0.0
netmask	255.255.255.0
broadcast	192.168.0.255
gateway	192.169.0.1

Table 1: *Network Interfaces. Modify octets per network requirements.*

4 MQTT Generic Type Sample Application

This chapter pertains to installing, configuring, and running MQTT generic type implementations.

4.1 Configuring the Sample Applications

Configure the MVI69E-LDM MQTT sample application by modifying the `root/psft/sample/mqtt/config.json` file on the module.

MQTT reserves port 1883 for unencrypted communication and port 8883 for encrypted communication.

4.1.1 MQTT Generic config.json File

If you are using MQTT Generic, edit the **config.json** file parameter values in the MVI69E-LDM-MQTT module according to the '*Values for Unencrypted messaging*' column indicated in the following table:

Parameters	Values for unencrypted messaging	Vales for encrypted messaging
{		
"MqttServer": {		
"Type":	"Generic",	"Generic",
"Host":	"137.135.83.217",	"137.135.83.217",
"Port":	1883,	8883,
"Timeout"	5000,	5000,
"DoNotUseTls":	1,	0,
"DisableCertificateValidation":	1,	1,
"RootCaFileName":	"root ca.cer",	"root ca.cer",
"ClientCertPublicFileName":	"client_cert_public_key.cer",	"client_cert_public_key.cer",
"ClientCertPrivateFileName":	"client_cert_private_key.pem",	"client_cert_private_key.pem",
"UserName":	"",	"",
"Password":	"",	"",
"ClientId":	"",	"",
"WillTopic":	"Will",	"Will",
"WillMessage":	"69EPSFT-LDM Disconnected",	"69EPSFT-LDM Disconnected",
"PublishRetryInterval":	10000,	10000,
"MaxPublishRetries":	10,	10,
"MaxPublishInterval":	5000	5000
"PublishTopicPrefix":	"69EPSFT",	"69E PSFT",
"SubscribeTopicPrefix":	"",	"",
"PublishOOS":	1,	1,
"PublishRetain":	1,	1,
},		
The remaining section of the "config.json" files are applicable to the PLC communications. No editing required.		

Table 1: *MQTT Configuration File*

Two public MQTT brokers/servers and clients can be found here:

- **test.mosquitto.org**: current IP = 5.196.95.208. (Subject to change)
- **mqtt.eclipse.org**: current IP = 137.135.83.217. (Subject to change)

For this sample configuration, '**mqtt.eclipse.org**' is accessed remotely and used as the broker. The locally installed MQTT Explorer is used as the client.

4.1.2 SparkplugB config.json File

If you are implementing SparkplugB, please skip the following section and restart with "Config.json Configuration Requirements" on page 24.

4.2 Unencrypted Data Exchange

The home webpage for the MQTT broker is at https://mqtt.eclipse.org. It has no diagnostic support.

Figure 4: *mqtt.eclipse.org MQTT broker home page*

Rather than verifying the MVI69E-LDM module's communication on the eclipse site, the communication with the remote broker can be verified by reviewing the log messages on the module. To do that, open a browser and enter the following URL:

http://192.168.4.189/log/messages.txt

(Replace **192.168.4.189** with the MVI69E-LDM module's IP address.)

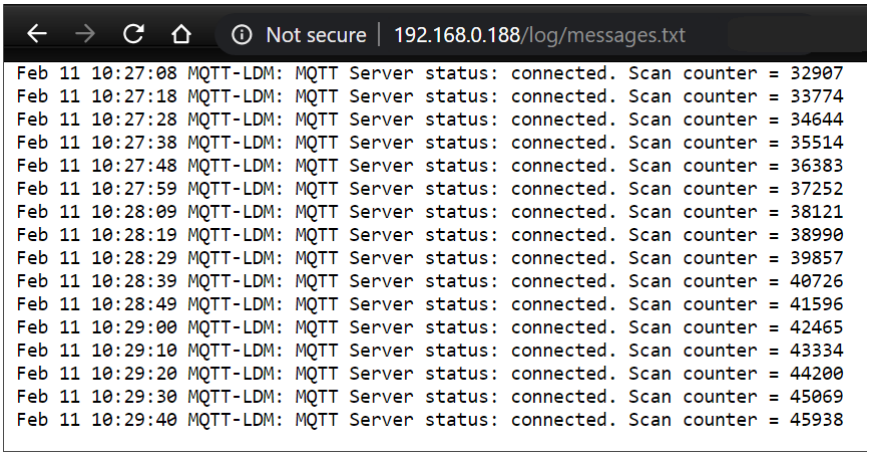

Figure 5: *MVI69E-LDM-MQTT successful communications with the remote MQTT broker*

This log file is located in the MVI69-LDM module at this path:

"root/www/html/log/messages.txt"

4.3 MQTT Explorer Client Configuration

The MQTT Explorer client must be configured so that the data from the PLC is displayed in the MQTT Explorer.

1 Launch MQTT Explorer. The *MQTT Connection* dialog displays with two default connections.

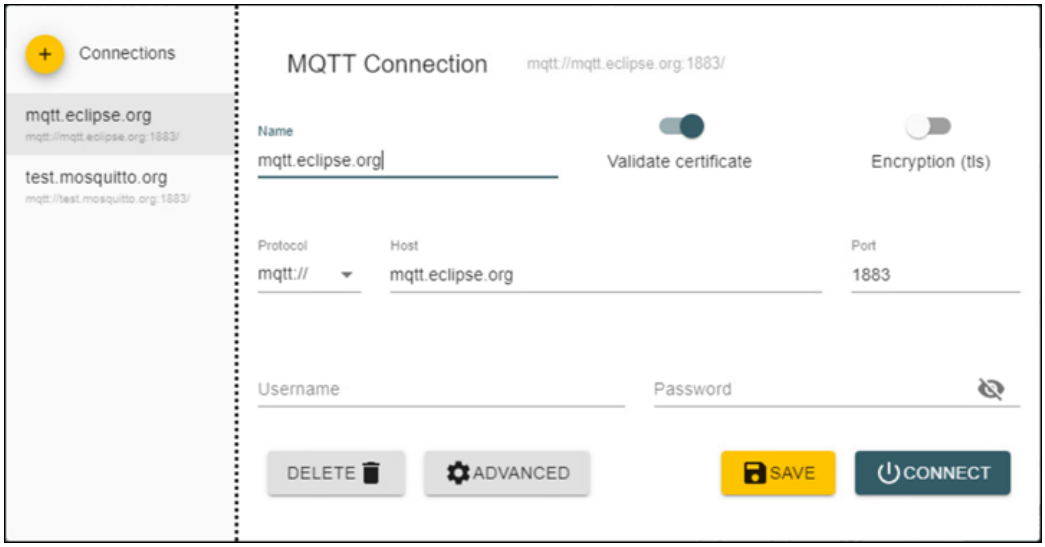

Figure 6: *Initial MQTT Connection dialog*

2 Select mqtt.eclipse.org from the list of connections on the left.
3 Edit the settings in the panel on the right as follows:

- **Name:** mqtt.eclipse.org
- **Validate certificate:** Off
- **Encryption:** Off
- **Protocol:** mqtt://
- **Host:** 137.135.83.217
- **Port:** 1883

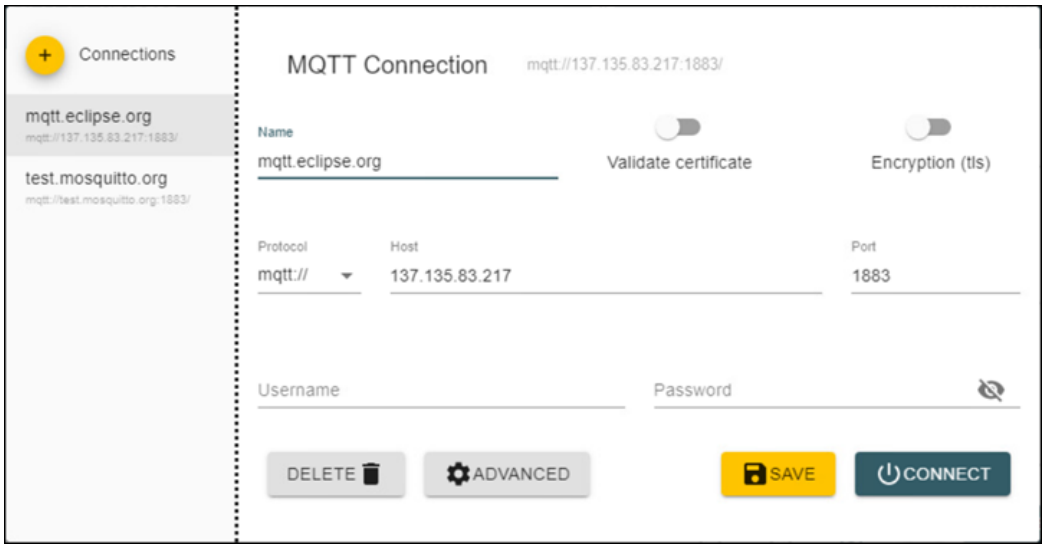

Figure 7: *Settings for the mqtt.eclipse.org connection*

4 Click on the **ADVANCED** button.
5 In the *Topic* field, enter the following values one by one, clicking the **ADD** button after each:

- 69EPSFT/SteamSensor/Temperature
- 69EPSFT/SteamSensor/Pressure
- 69EPSFT/SteamSensor/TotalFlow
- 69EPSFT/SteamSensor/TemperatureLimit
- 69EPSFT/SteamSensor/InletValve
- 69EPSFT/SteamSensor/Publish_Subscribe
- 69EPSFT/SteamSensor/String
- 69EPSFT/SteamSensor/ReadPLCclock

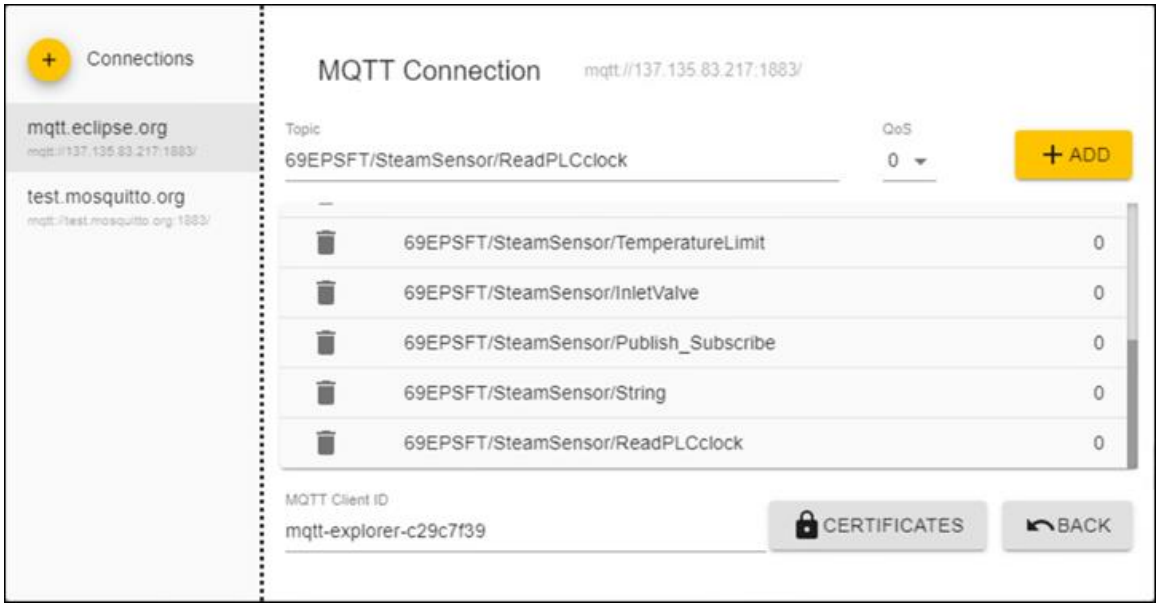

Figure 8: *Topics for the mqtt.eclipse.org connection*

6 Click on the **BACK** button.

7 Click **SAVE** and then the **CONNECT** button.

MQTT Explorer displays the topics entered in step 5.

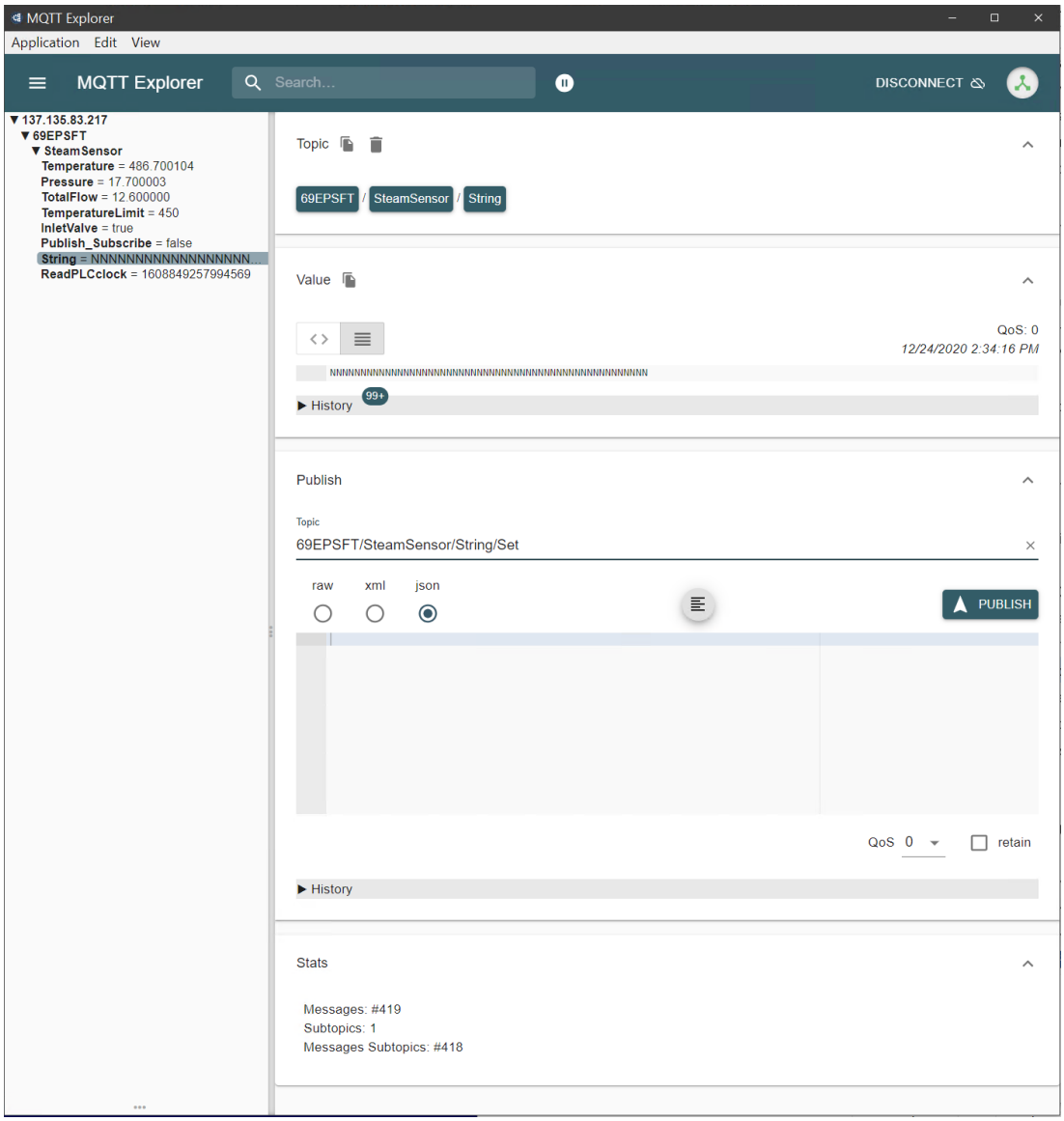

Figure 9: *MQTT Explorer Main Display*

8 Expand the *Publish* section in the center panel, and add "***/Set***" to end of the topic name (so it will be equal to the value of the "*SubscribeTopic": "SteamSensor/Publish_Subscribe/Set*" field, in the config.json file). Select payload format type *raw*.

9 Type the ward "**true**" or the number "**1**" in the field indicated in screen capture below. Click on the **PUBLISH** button.

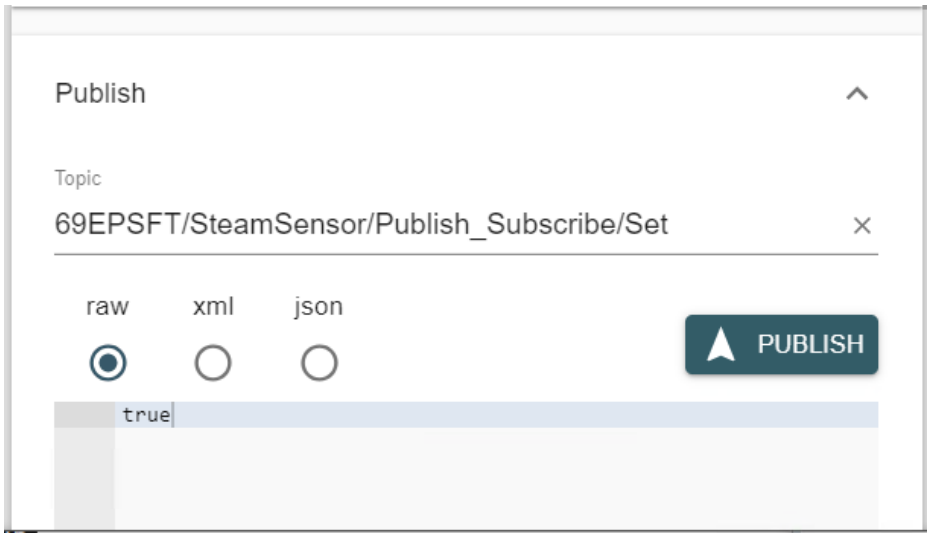

Figure 10: *Preparing the string for publishing to the broker*

In the PLC, tag **Local:1:I.Data[9].0** bit was set to TRUE by the remote client that has enabled the PLC to receive tags data from the remote MQTT client.

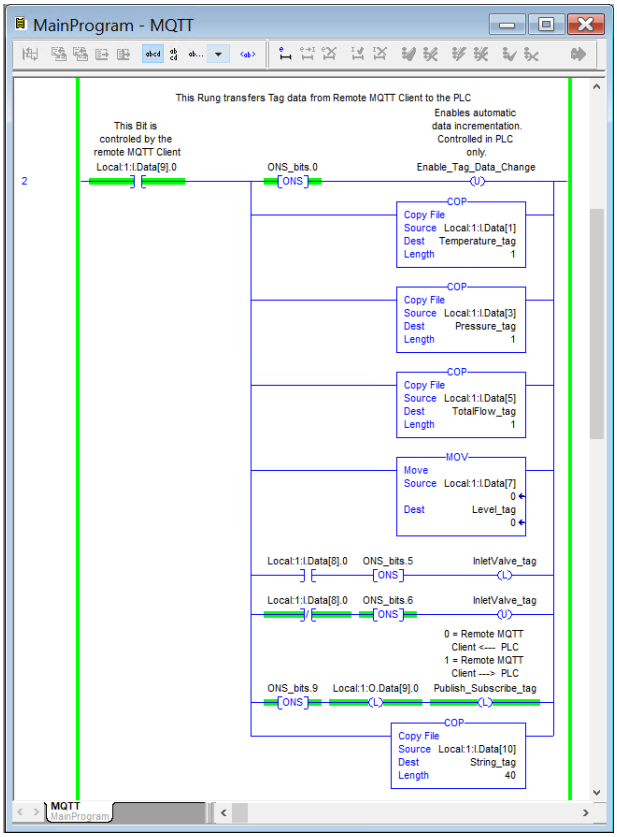

Figure 11: *Preparing the PLC Program for receiving the published string from the broker*

10 In the MQTT Explorer, enter multiple **1**'s in the *PSFT/SteamSensor/String/Set* field indicated in the following figure:

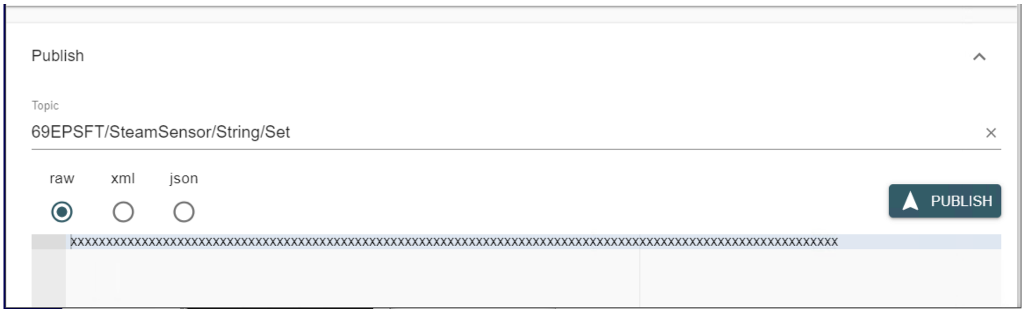

Figure 12: *Typing in the desired string values*

11 Click on the **PUBLISH** button.

12 Go back to the PLC program and observe that the multiple **1**'s are displayed in the PLC in the *String_tag* tag.

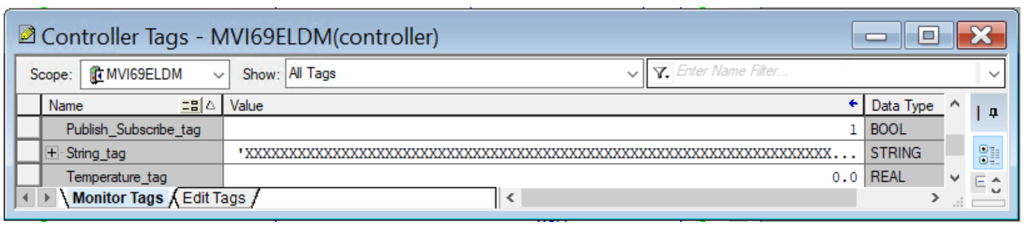

Figure 13: *PLC receives the String Value*

13 The successful publishing process confirms that the data is being transferred in both directions.

4.4 Encrypted Data Exchange

1. In **config.json** file edit this parameter as follows, **"Port": 8883,** and **"DoNotUseTls": 0,** to enable encrypted data exchange.
2. Reboot the MVI69E-LDM module.
3. Edit the MQTT Explorer by activating the *Encryption* and set *Port* to **8883**.

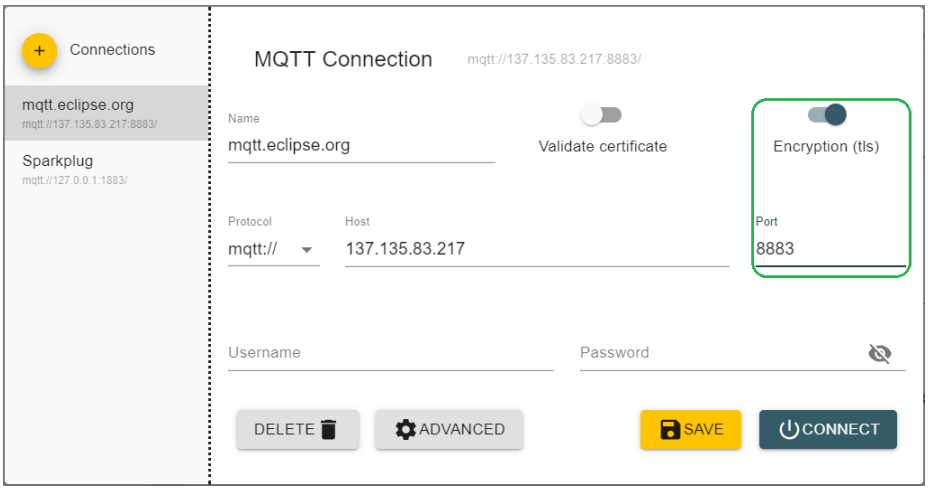

Figure 14: *MQTT Connection dialog in MQTT Explorer*

4. Click **Save**, then **Connect**.
5. Observe that all parameters are changing their values frequently. This confirms that the encrypted data is being exchanged between the two remote MQTT clients.

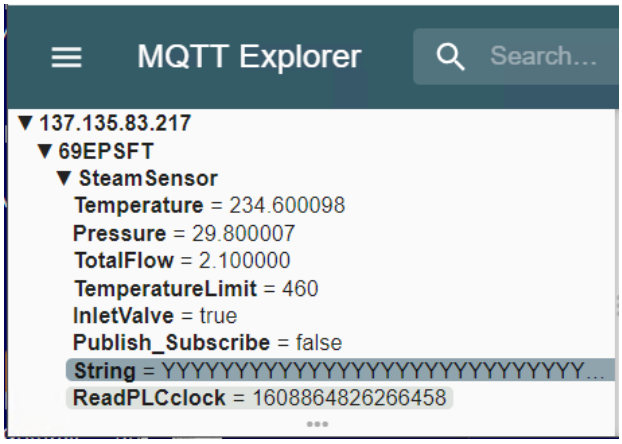

Figure 15: *All subscribed tags are displaying changing encrypted data.*

6 The Wireshark capture indicates that the data is encrypted.

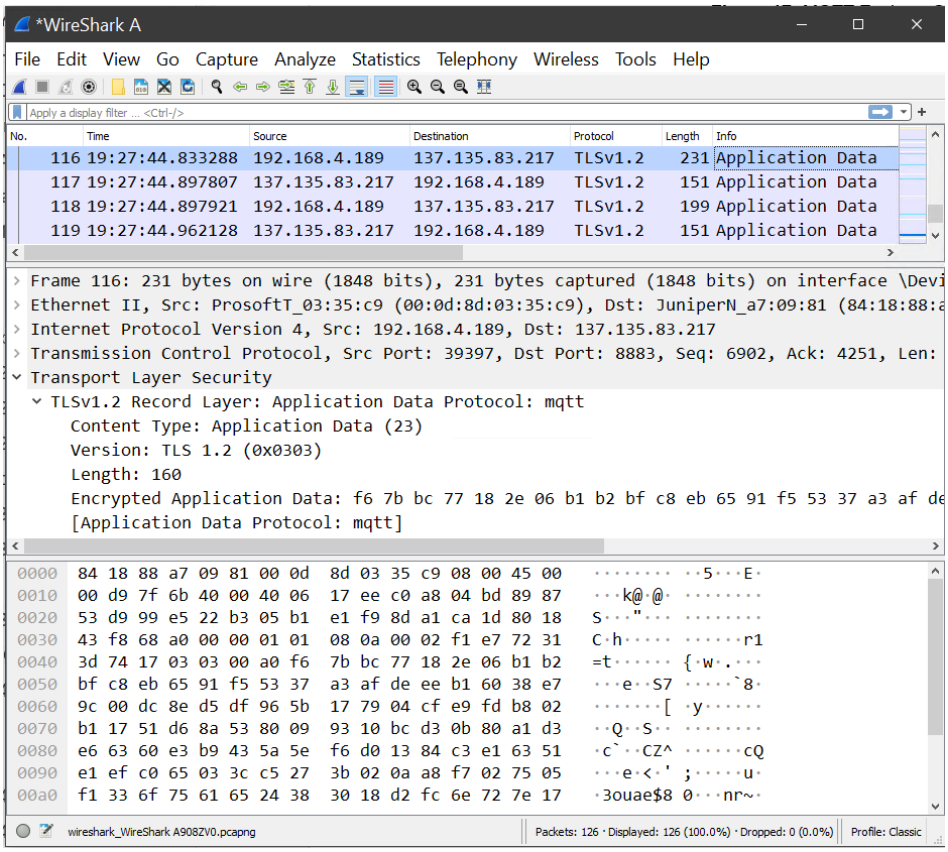

Figure 16: *Encrypted data in Wireshark*

This concludes the Generic MQTT setup instructions with unencrypted and encrypted communications.

5 MQTT SparkplugB Example

Use this chapter to install, configure, and run the MVI69E-LDM module with MQTT Sparkplug-B communications. Set the configuration file parameters, configure the CompactLogix PLC, and begin data exchange.

5.1 Config.json Configuration Requirements

This section give sample configuration settings when using Sparkplug-B.

config.json "MqttServer" section file parameters		
Parameters	**Values for unencrypted messaging**	**Values for encrypted messaging**
{		
"MqttServer": {		
"Type":	"Sparkplug",	"Sparkplug",
"Host":	"192.168.4.200",	"192.168.4.200",
"Port":	1883,	8883,
"Timeout"	5000,	5000,
"DoNotUseTls":	1,	0,
"DisableCertificateValidation":	1,	0,
"RootCaFileName":	"root ca.cer",	"root ca.cer",
"ClientCertPublicFileName":	"client_cert_public_key.cer",	"client_cert_public_key.cer",
"ClientCertPrivateFileName":	"client_cert_private_key.pem",	"client_cert_private_key.pem",
"UserName":	"ldm",	"ldm",
"Password":	"ldm",	"ldm",
"GroupId":	"ProSoft",	"ProSoft",
"UUID":	"UniqueUUID",	"UniqueUUID",
"ClientId":	"MVI69E-1",	"MVI69E-1",
"WillTopic":	"spBv1.0/ProSoft/NDEATH/MVI69E-1",	"spBv1.0/ProSoft/NDEATH/MVI69E-1",
"WillMessage":	"Node MVI69E-1 is disconnected",	"Node MVI69E-1 is disconnected",
"PublishRetryInterval":	1000,	1000,
"MaxPublishRetries":	10,	10,
"MaxPublishInterval":	5000	5000
"PublishTopicPrefix":	"",	"",
"SubscribeTopicPrefix":	"",	"",
"PublishQOS":	1,	1,
"PublishRetain":	1,	1,
},		
The remaining section of the "config.json" files are applicable to the PLC communications. No editing is required.		

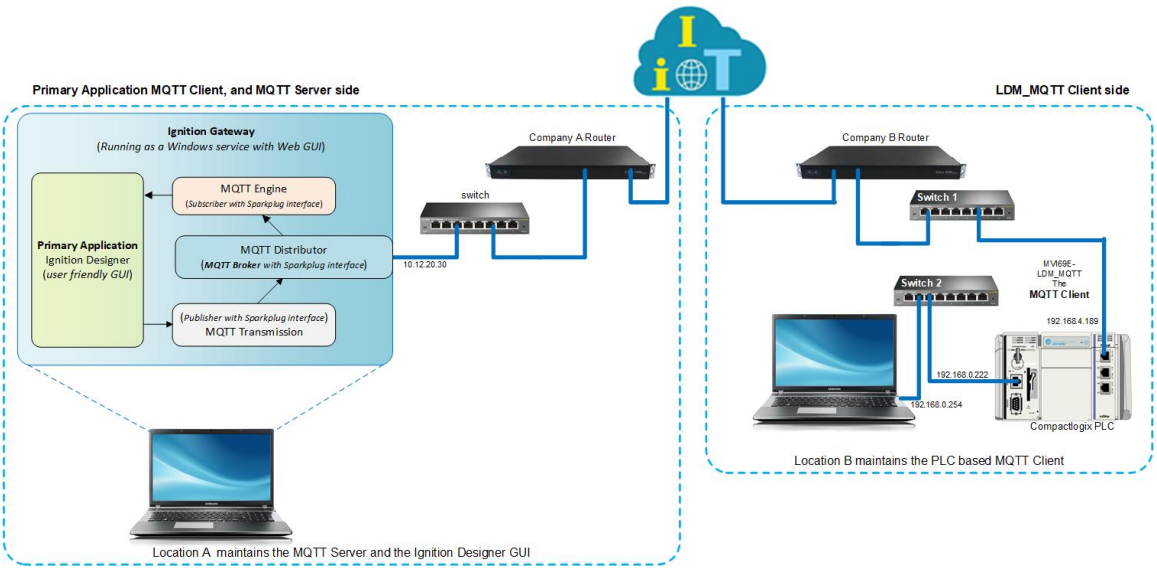

Figure 17: *Sparkplug-B Configuration*

5.2 Configuring the CompactLogix PLC

1. Open the MVI69E_LDM_MQTT_SparkplugB_Sample_Ladder.ACD program and change the appropriate chassis type to match your hardware and firmware.
2. Download MVI69E_LDM_MQTT_SparkplugB_Sample_Ladder.ACD file to the CompactLogix processor by choosing **COMMUNICATIONS > WHO ACTIVE > DOWNLOAD**.

5.3 Ignition

Ignition requires that MQTT-related modules be installed for the LDM-MQTT module's Sparkplug communications requirements. This is a trial version of Ignition that lasts two hours. It can be restarted any number of times.

1. Go to the following website to learn more about Ignition: https://docs.inductiveautomation.com/display/DOC80/Introducing+Ignition
2. Navigate to the following website to download and install (selecting the default options) Ignition v8.1.1 (Stable), current as of this publication: https://inductiveautomation.com/downloads/ignition/8.1.1. Follow the Ignition installation prompts. Make notes of the credentials required for the installation process. Ignition is not the MQTT Sparkplug-B broker. It is an industrial user interface that among other protocols also accommodates third-party files that provide the MQTT Sparkplug-B broker/server and client functionality.
3. Fetch the following Cirrus Link Solutions, MQTT Modules for Ignition. https://inductiveautomation.com/downloads/third-party-modules/8.1.1

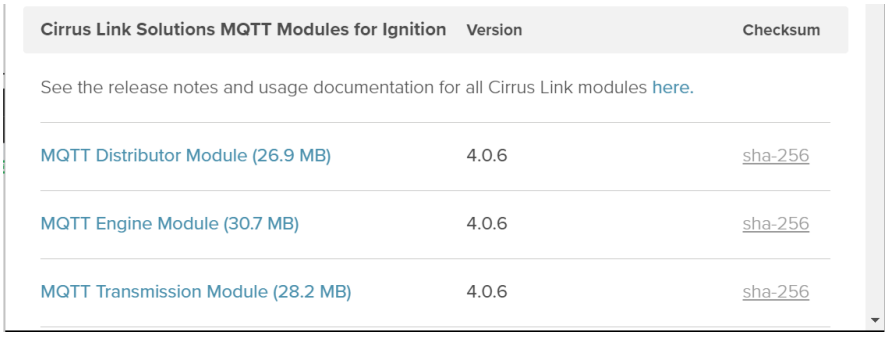

Figure 18: *MQTT Modules for Ignition*

Note: Make note of the hard drive location where these module files are saved.

5.3.1 Installing .modl Files

Once Ignition is running, select the **CONFIG** gear icon (left side of the window) and sign in as required. When the Configuration menu options display:

1 Select the **SYSTEM/MODULES** option.
2 Scroll to the bottom of the page and select the **INSTALL OR UPGRADE A MODULE** link.

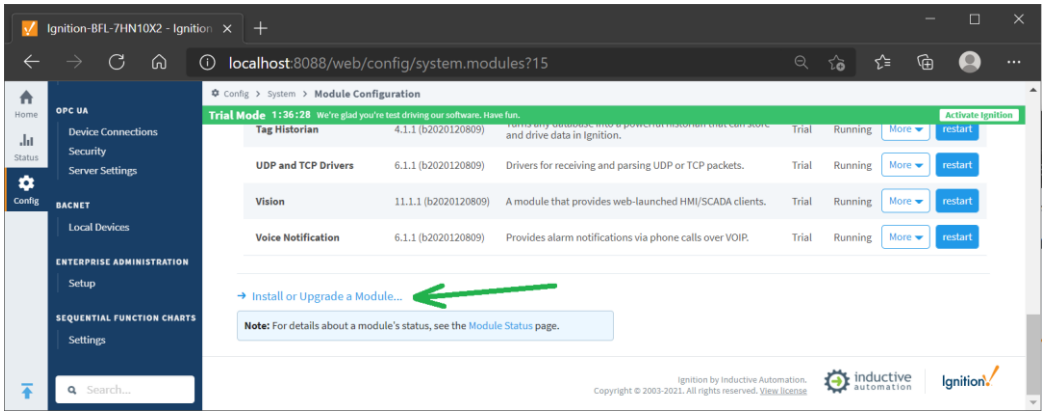

Figure 19: *Install or Upgrade a Module link*

3 Select **CHOOSE FILE**.

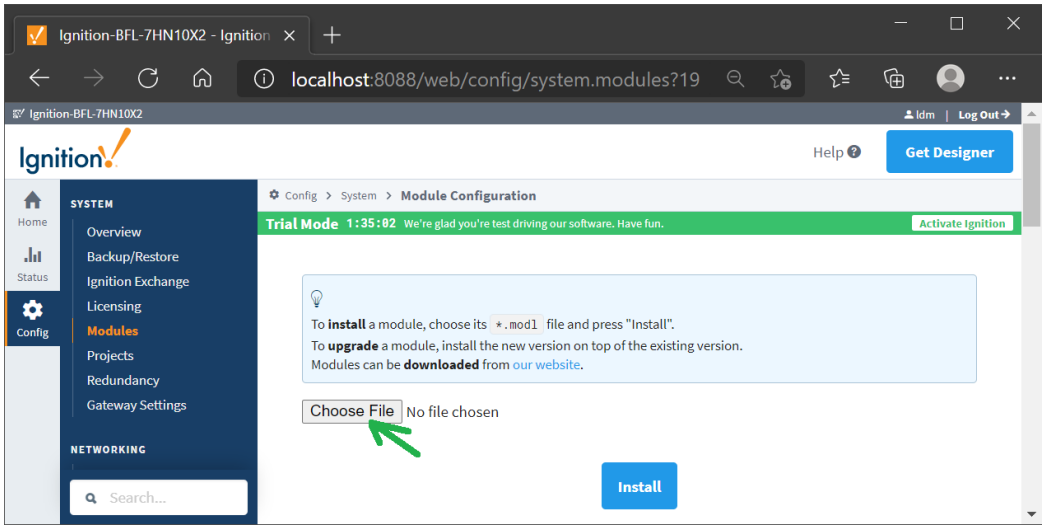

Figure 20: *Selecting the module files*

4 Navigate to the MQTT Module (*.modl files) storage location on the hard drive and follow the web instructions for installing them into the Ignition Gateway.

5.4 Configuring the Unencrypted Sparkplug Data Exchange

This section explains how to configure Ignition to communicate with the PLC.

5.4.1 Logging in to Ignition

1 Using the credentials chosen during the Ignition installation process, log into the Ignition web GUI http://localhost:8088/web/.

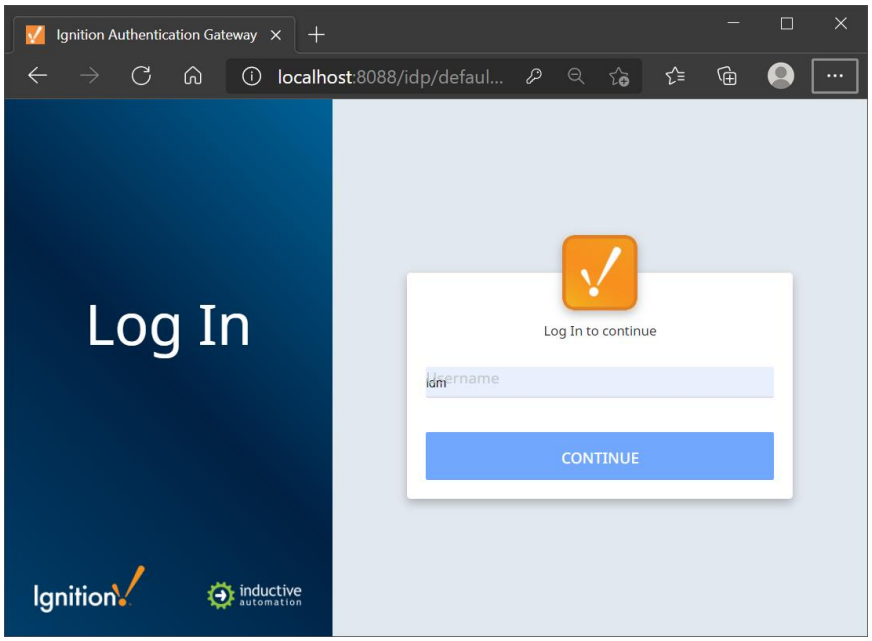

Figure 21: *Ignition Log In screen. This user interface may look different with your setup.*

The home screen opens.

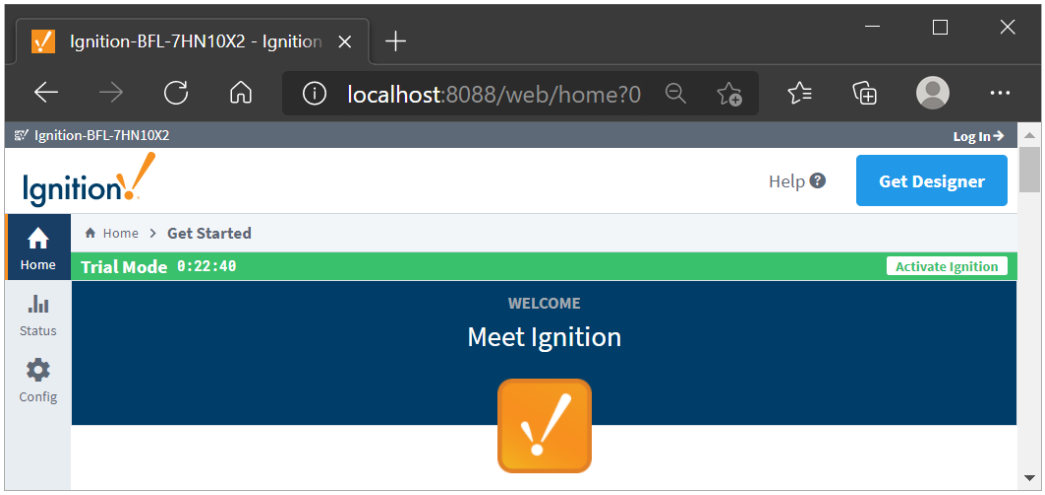

Figure 22: *Ignition Home Screen. Expand the screen vertically to see the options.*

5.4.2 Configuring the MQTT Broker Distributor within Ignition

This section covers the basic configuration for the MQTT Distributor file that behaves as the MQTT broker within the Ignition application.

1 Click on the **CONFIG** > **MQTT DISTRIBUTOR *SETTINGS*** selection. Ignition may ask for your credentials.

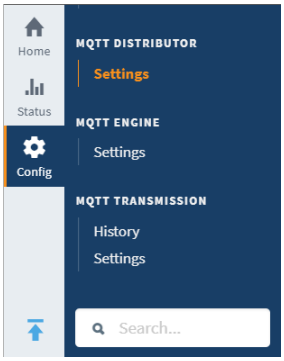

Figure 23: *Config View (MQTT Distributor Settings)*

The following parameters are displayed:

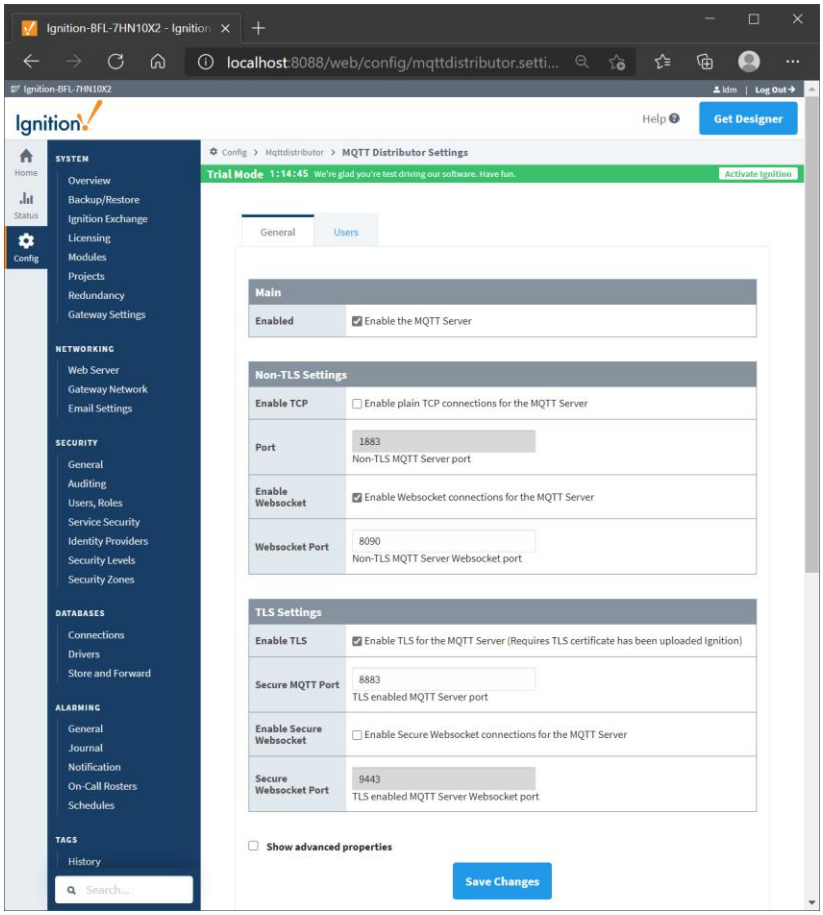

Figure 24: *Distributor Settings. This is the MQTT broker.*

2 Click on the *Users* tab and then click on the **CREATE NEW MQTT USERS** option.

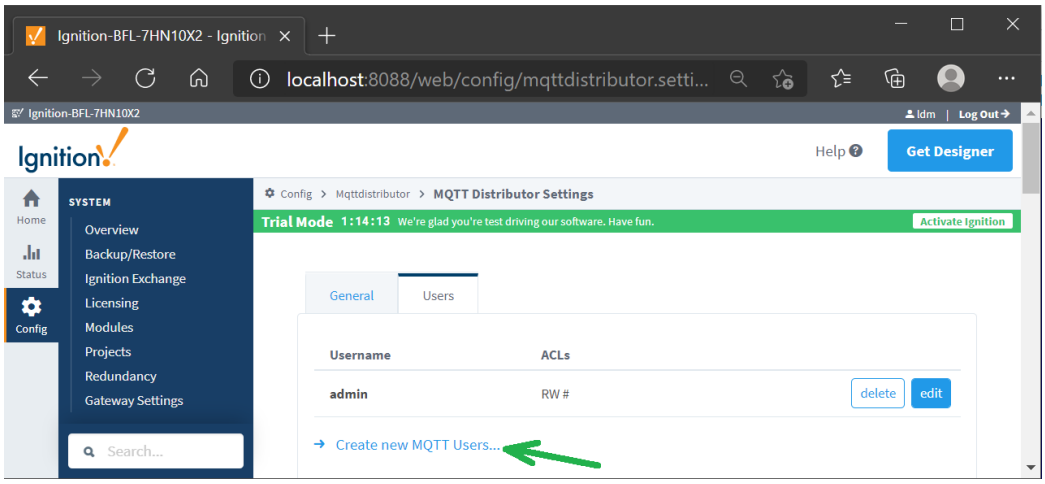

Figure 25: *Create new MQTT Users option*

3 Enter the parameter values as indicated in the following screen capture:

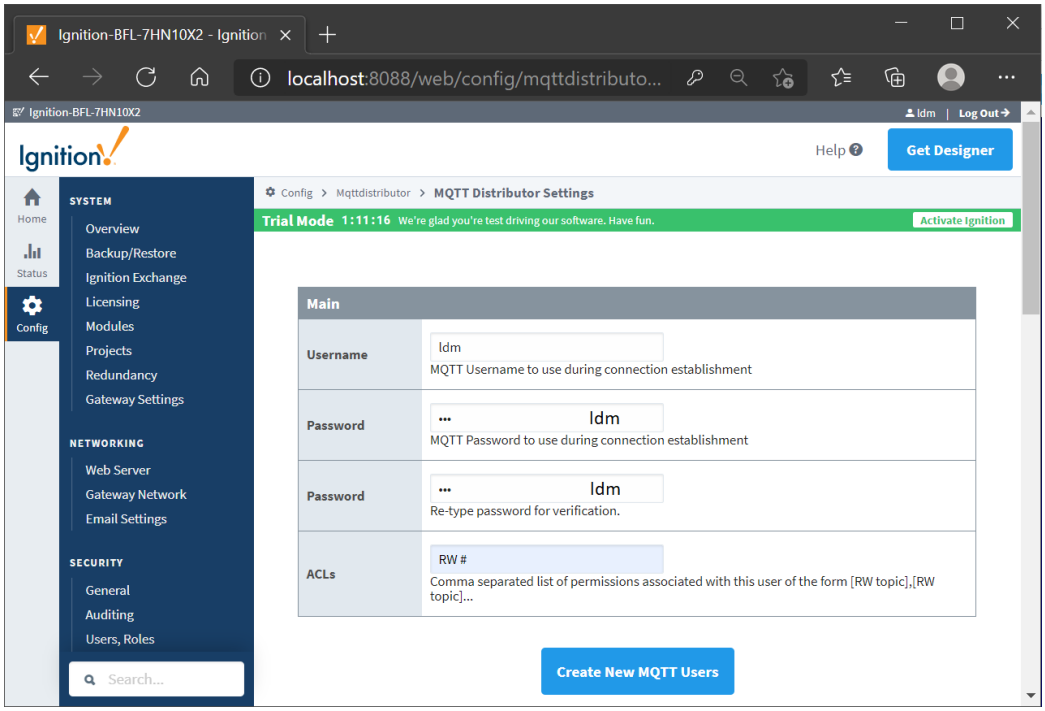

Figure 26: *Assigning the new user to the MQTT broker*

4 Click on the **CREATE NEW MQTT USERS** button to save the credential values.
5 Click on the **GENERAL** button to go back to the main *Distributor* settings and click on the **SAVE CHANGES** button.

5.4.3 Configuring the MQTT Subscribing Client and MQTT Engine Within Ignition

This section covers the basic configuration for the MQTT Engine file that behaves as the MQTT subscribing client within the Ignition application.

1 Click on the Config > MQTT Engine *Settings* button.
2 Enter the indicated values for the specific parameters as displayed in the *General* tab, as shown below:

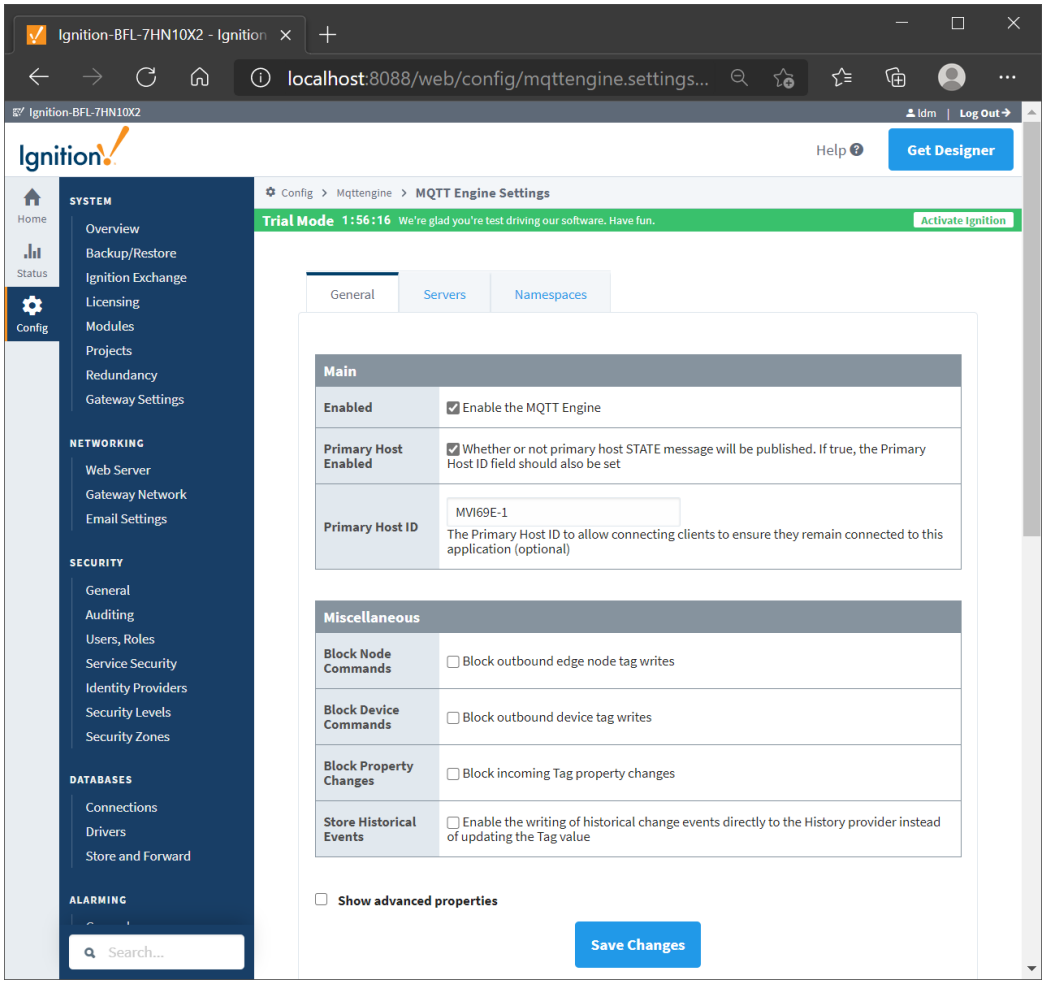

Figure 27: *Configuration Parameters for MQTT Engine. This is the MQTT Subscribing Client.*

3 Click the **SAVE CHANGES** button.

5.4.4 Configuring the MQTT Publishing Client and MQTT Transmission Within Ignition

This section covers the basic configuration for the MQTT Transmission file that behaves as the MQTT Publishing Client within the Ignition application.

1 Click on the Config > MQTT Transmission *Settings* button.

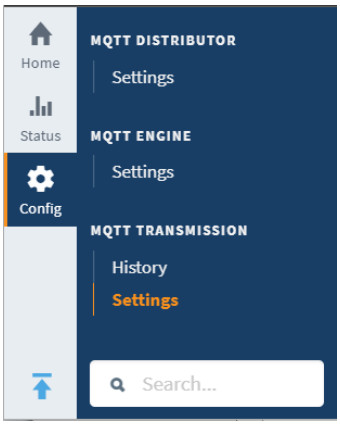

Figure 28: *Engine Settings*

2 Enter the values as indicated below: Enabled check mark required only.

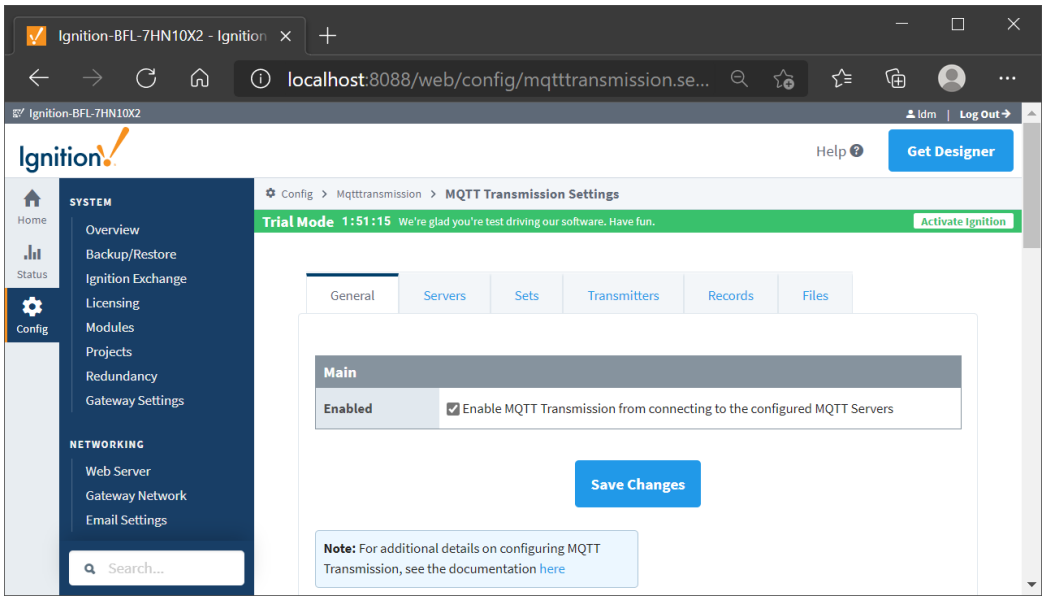

Figure 29: *MQTT Transmission Parameters*

3 Click on the **SAVE CHANGES** button.

5.4.5 *Verify Ignition to PLC Communication*

1 In the Ignition application, navigate to the following location:

Status > Systems > Tags > MQTT Engine > Edge Nodes > ProSoft MQTT LDM Gateways> MVI69E-1 > SteamSensor

2 Verify that LDM-MQTT is successfully reading data (subscribing) from the PLC to the Ignition application.

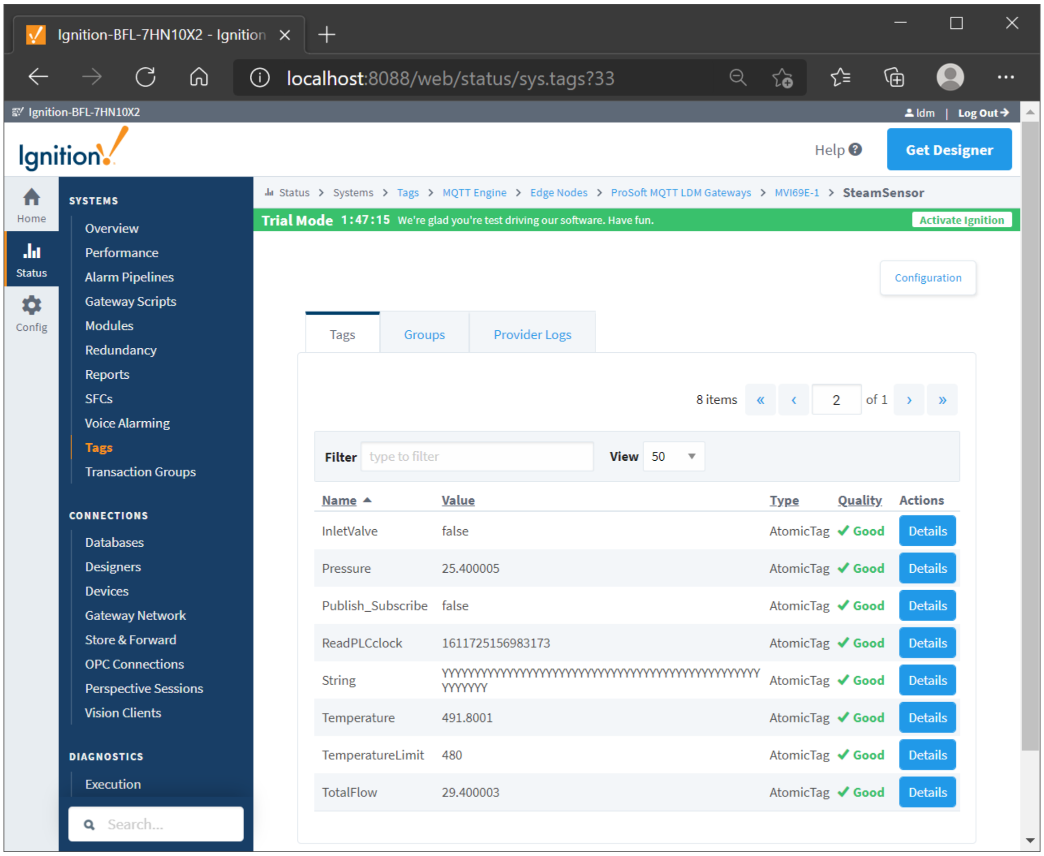

Figure 30: *Data successfully being read from the PLC to Ignition*

3 Verify that all tag data *Quality* is reported as **Good**. This data is coming from the PLC.

5.5 Installing the Ignition Designer Software

1 Navigate to the Ignition website and click on the **GET DESIGNER** button.

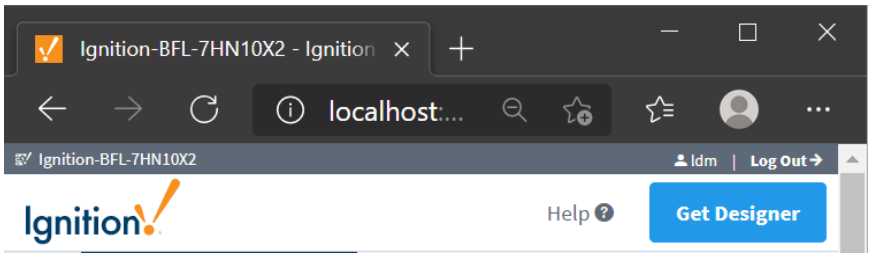

Figure 31: *Get Designer Button*

2 Once downloaded, run the **Designer Launcher Setup** application.
3 Follow the instructions to complete the installation.

5.5.1 Using Ignition Designer to Send Data to the PLC

This section covers the configuration of the Ignition Designer tool and sending data to the PLC. It will show the data exchange with the PLC using the MVI69E-LDM-MQTT in unsecured mode.

1 Run the Ignition Designer Launcher.
2 Edit the parameters if desired to suit your installation. Then click on the **SAVE CHANGES** button.

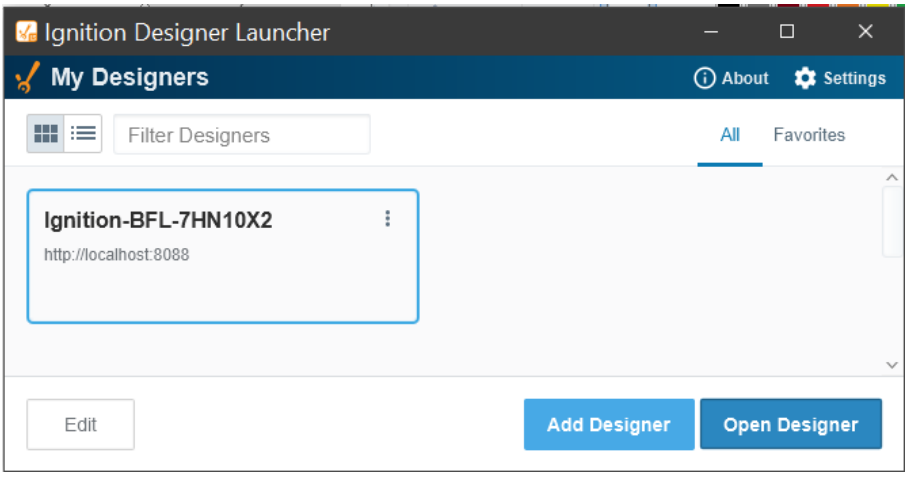

Figure 32: *Ignition Designer Launcher*

3 Click on the **LAUNCH** button to activate the Ignition Designer.

Figure 33: *Designer Login*

4 Use the same credentials as when the ignition was initially installed.
5 The *Open/Create Project* dialog pops up. Click on the **NEW PROJECT** button.

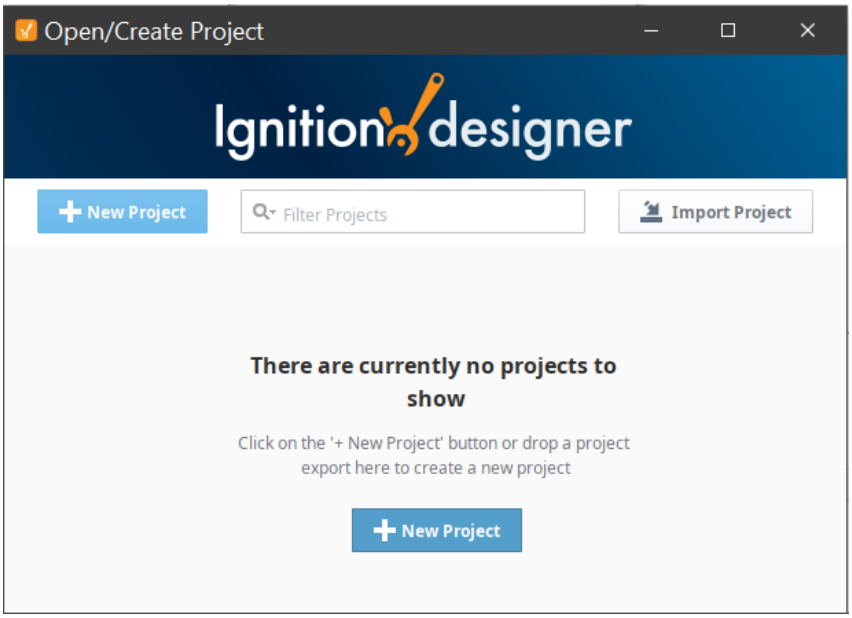

Figure 34: *New Project*

6 Set up a new project as in the following example.

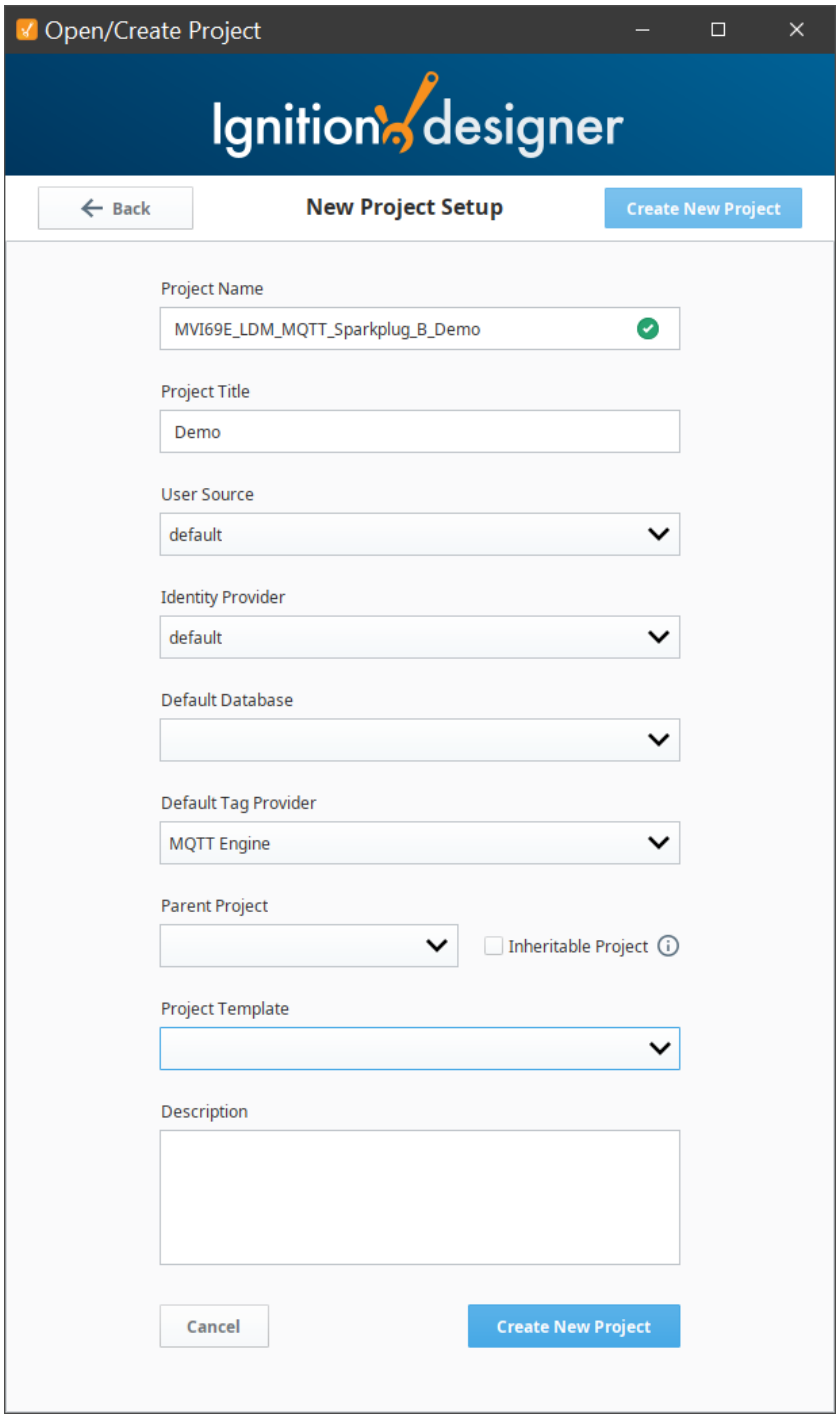

Figure 35: *Setting up the new Project*

7 Click the **CREATE NEW PROJECT** button.

Figure 36: *Creating a new project*

8 The new project is created. Drill down to Edge Nodes > ProSoft MQTT LDM Gateways > MVI69E-1 > SteamSensor. Observe that data for most of the tags is changing every few seconds.

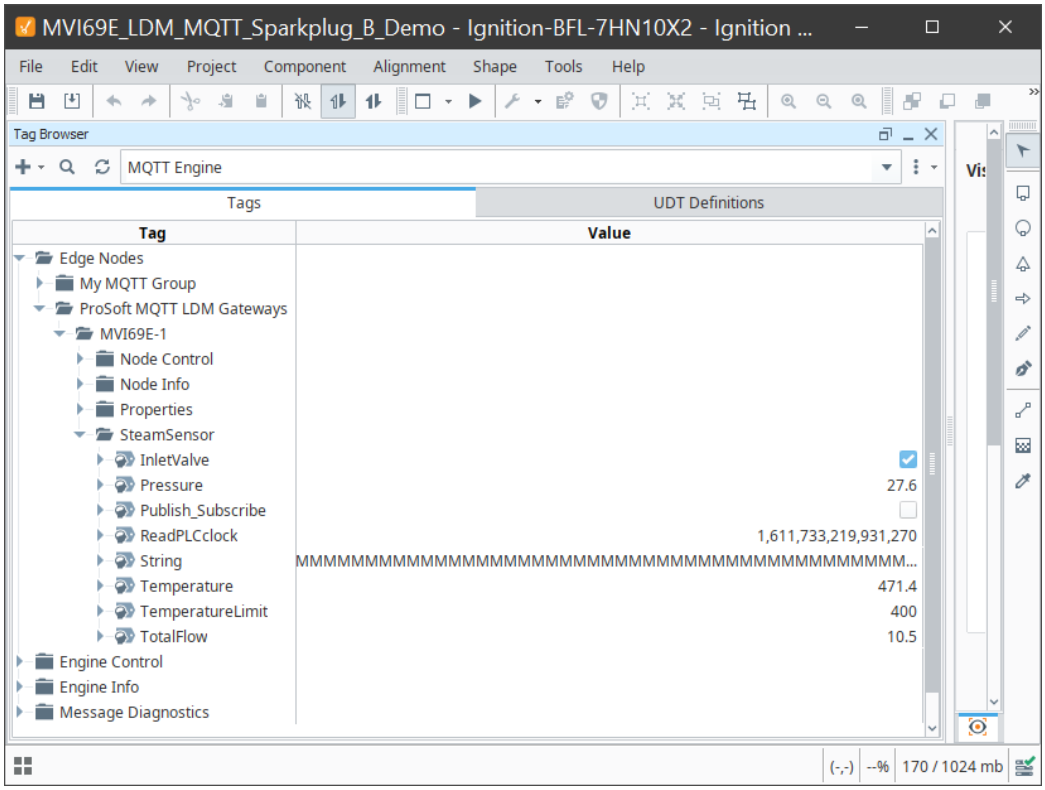

Figure 37: *The new project is receiving data from the PLC.*

9 Select the **Publish_Subscribe** tag as indicated in the following image.

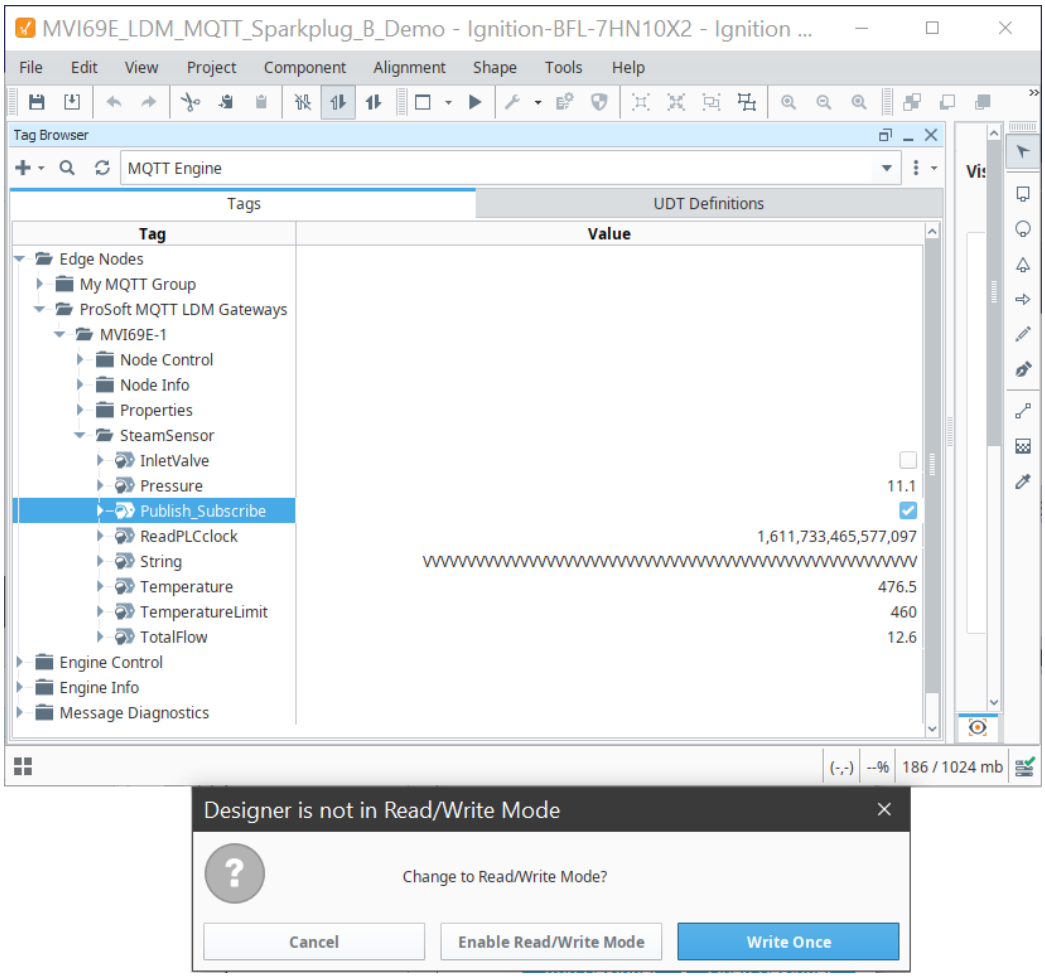

Figure 38: *Selecting the Publish_Subscribe tag*

10 Click on the [**Write Once**] button. This will cause the MVI69E-LDM-MQTT to prepare the PLC to receive data from the Ignition Designer.

11 Select the **String** tag and write multiple 9s(as an example) as indicated in the following image.

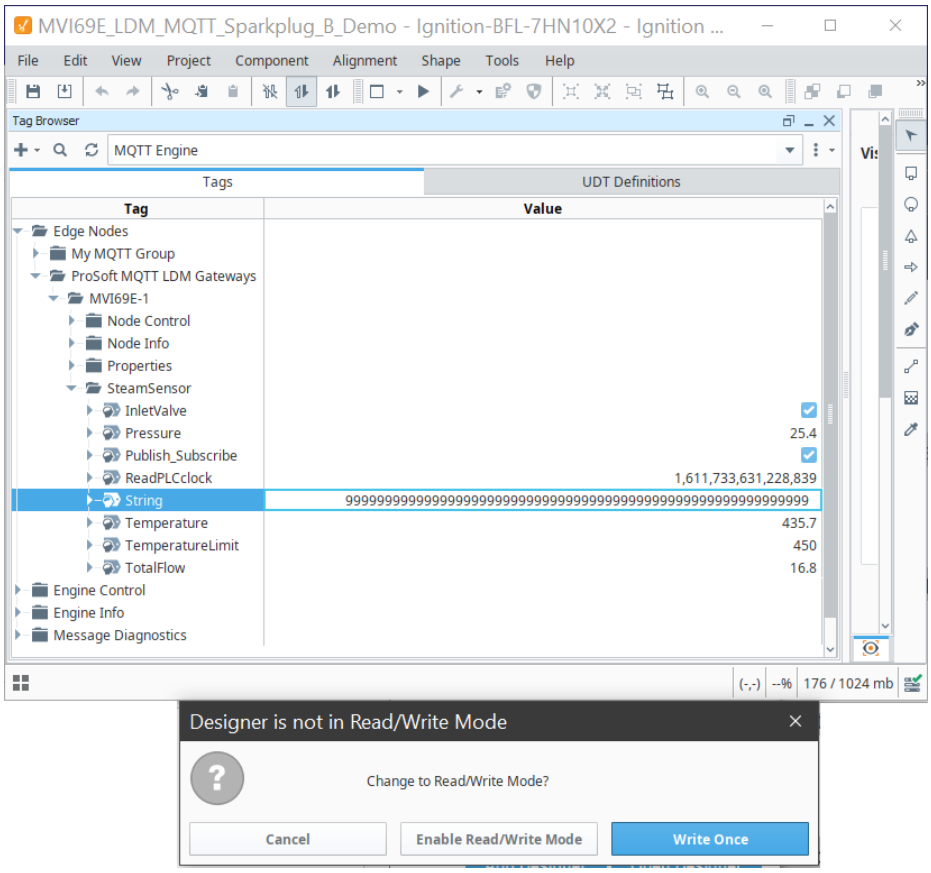

Figure 39: *Selecting the String tag*

12 Click on the [**Write Once**] blue button. This will cause the MVI69E-LDM-MQTT to send the string of 9s to the PLC.

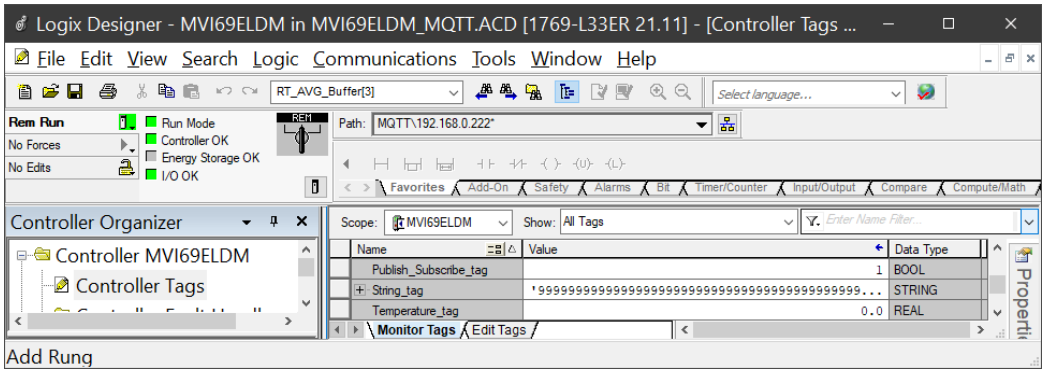

Figure 40: *Sending the string to the PLC*

This concludes the MVI69E-LDM-MQTT client publishing and subscribing functionality within the Ignition environment.

6 Prerequisites for Customizing the Sample Application

6.1 MVI69E-LDM-MQTT Zip File

Note: The MVI69E-LDM-MQTT zip file contains the generic version as well as SparkplugB.

1. Go to https://www.prosoft-technology.com and navigate to the MVI69E-LDM product page. Download the **mqtt-ldm-MVI69-xxx.zip** (where xxx is the version number).
2. Create the folder *C:\Workspace* on your PC and unzip into this folder.

6.2 Turn on Hyper-V

On the Windows 10 PC, ensure that Hyper-V is turned on.

Note: VMware can be used instead of Hyper-V, although Hyper-V is the recommended method.

6.3 Docker®

The MVI69E-LDM development tools run in Linux. If you have experience with a previous ProSoft Technology LDM module, you may have setup a Linux Debian 6 Virtual Machine. For MQTT, this guide steps you through using a Docker® container on a Windows 10 PC.

Docker Desktop for Windows is required to run the toolchain from a container running 32 bit Debian Stretch OS.

1. Locate Docker Desktop for Windows here: https://www.docker.com/products/docker-desktop. Note that it should run in Linux Containers mode (Default).
2. Ensure that PowerShell is enabled in order to run Docker commands. Information on how to enable or install PowerShell can be found here: https://docs.microsoft.com/en-us/powershell/scripting/install/installing-windows-powershell?view=powershell-6
3. Note that container **psft** will be left running after script completion. If you want to stop the container and remove it, you can modify script file build.ps1 (uncomment command Docker container stop psft at the end). SSH server will be also running in the container, so it is possible to connect to it using command from Windows console:
   - ssh user@localhost -p 6622
   - When asked for password, enter "**password**".
   - ssh user@localhost -p 6622
   - The port number is 6622

7 Development Setup

7.1 Create User

Some Docker files will be stored in the Windows 10 *\User* folder. You can either use your existing Windows login ID or create a new one.

In addition, the root folder of source code files (C:\Workspace) needs to be shared in order to access it from the build container. In order to access this shared folder from Docker container **psft**, Windows user credentials are required (user name, password, and the shared folder name should be passed as command line arguments when the script *build.ps1* is called).

7.2 Sharing the C:/Workspace Folder

To provide access to source code files from Docker container, share the C:\Workspace folder as shown in the following figure:

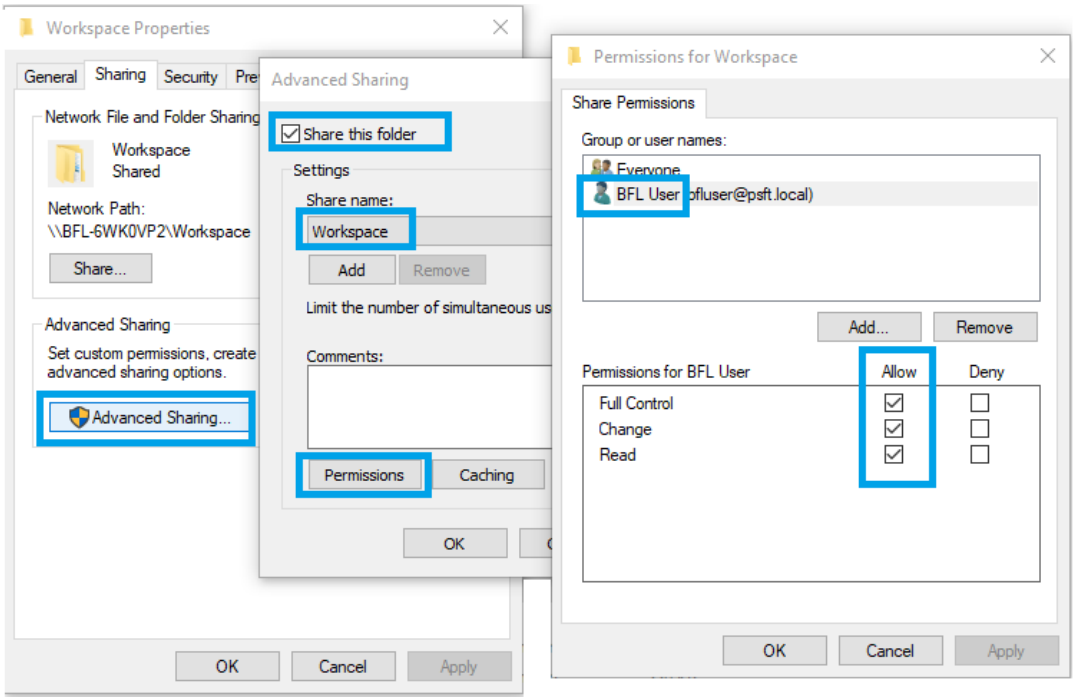

Figure 41: *Sharing the C:\Workspace Folder*

8 Creating a Build

1. Obtain the IP address of your Windows 10 PC.
2. Open PowerShell console.
3. Navigate to `C:\Workspace\mqtt-ldm\scripts`.
4. Run script `build.ps1` as follows:

   ```
   ./build.ps1 -SHARED_FOLDER //192.168.1.73/Workspace -
   SHARED_FOLDER_USER bfluser –SHARED_FOLDER_PASSWORD passwd
   ```

   Make the following substitutions in the command:

   - Replace the IP address shown above (**192.168.1.73**) with the PC's IP address.
   - Replace the user ID shown above (**bfluser**) with your user ID (see "Config.json Configuration Requirements" on page 24).
   - Replace the password shown above (**passwd**) with the password for your user ID.

Important: A Debian 9 image should be polled from Docker Hub and required components installed to it, including toolchain files. The first run of the script can take 15 minutes or more depending on Internet speed. Consecutive runs will take seconds.

The firmware image is created:

```
C:\Workspace\mqtt-ldm-sample-app-mvi69e\firmware\mvi69e-
ldm.firmware_<version number>_<date>.firmware
```

9 Configuration File Details

The configuration file is named **config.json** (in JSON format). Edit it manually on the PC and move it to the module, or access it directly on the module over FTP (example: WinSCP). The file is in the folder */psft/sample/mqtt*.

Note that settings differ depending on which MQTT broker the MVI69E-LDM is connecting. Some MQTT brokers accept any client ID, while some require it to follow specific rules. Also, different brokers have different requirements on TLS client certificates.

The folder `C:\Workspace\mqtt-ldm-sample-app-mvi69e\test-generic` houses an example configuration file and certificates for generic MQTT brokers.

The file config.json and certificate files included in the firmware by default pertain to a generic MQTT broker, such as Eclipse Mosquitto™.

Note: After changing the configuration file, you must restart the **mqtt-ldm-sample-app-mvi69e** process.

9.1 Configuration File Structure

This section describes the elements of the configuration file.

9.1.1 MQTT Server Settings

These settings are used to connect to the MQTT broker:

Parameter	Description
UserName and Password	Set according to configuration settings for Ignition MQTT Distributor user.
Group ID	The *group_id* element of Sparkplug topic namespace.
UUID	(Optional) Universally Unique Identifier component of Sparkplug payload.
Client ID	The *edge_node*_id element of topic namespace.
Type	Type of the MQTT broker to connect to: **Generic**: Any MQTT broker, such as open source broker Eclipse Mosquitto™. **SparkPlug**: MQTT broker supporting Sparkplug-B protocol.
Host	IP address of the MQTT broker.
WillTopic	WillTopic field of the MQTT connect request payload. It should follow the following format: spBv1.0/**<GroupId>**/NDEATH/**<ClientId>**. **Example**: spBv1.0/Prosoft MQTT LDM Gateways/NDEATH/MVI69E-1.
WillMessage	Will Message field of MQTT Connect request payload.
PublishRetryInterval	Message retransmit if no response is received within this time for Publish messages sent to the MQTT broker.
MaxPublishRetries	Maximum number of retries to successfully send a Publish message.
PublishTopicPrefix	Prefix added before tag-specific topic name, used to publish values for tags. **Example:** If the tag-specific topic name is *SteamSensor/Pressure*, and option *PublishTopicPrefix* is set to PSFT, then the final topic name on which tag value is published would be *PSFT/SteamSensor/Pressure*.
SubscribeTopicPrefix	Similar to *PublishTopicPrefix* but used on topic names for Subscribe requests.
MaxPublishInterval	Maximum time interval between publishing of values for each tag. The tag values are read from the PLC with a scan rate specific for each tag. If the value is not changed, then it is not published. If time elapsed since the latest publishing of a value for a tag is greater than *MaxPublishInterval*, then it is published even if value is not changed.
PublishQOS	Value of the QoS field in MQTT Publish messages. Can be **0** or **1**.
PublishRetain	Value of the RETAIN field of the MQTT Publish message. Can be **0** or **1**.

9.1.2 PLC Path

This defines the connection string to connect to the PLC.

9.1.3 Sync Time with PLC

A flag indicating if the system time should be synchronized with the PLC.

Value	Description
0	Default value. Synchronize once, if current system year is less than 2019, which is usually case after system restart. This helps prevent a system reset when PLC time is not set.
1	Synchronize one time after the first successful connection to the PLC.
2	Synchronize after every successful connection to the PLC.

9.1.4 Status Print Interval

This setting is optional.

The MQTT broker periodically checks the connection status. Messages are written to the log system on every status change. Additional status messages are written at the interval defined in this parameter, whether or not the status has changed.

The default value is 10 (seconds).

9.1.5 Tags

The following table includes tags that are defined in the PLC with the settings to map them to MQTT messages:

Parameter	Description
Tag	Name of the tag in PLC. Defined for MVI69E-LDM only.
DataType	Data Type of the tag in the PLC. Possible values are: BOOL, SINT, INT, DINT, LINT, USINT, UINT, UDINT, ULINT, REAL, LREAL, BYTE, WORD, DWORD, LDWORD, STRING82
ScanRate	Defines how often the tag value is read from the PLC, in milliseconds.
Access	Defines if the tag is read-only (value is RD), or readable and writable (value is RDWR).
Topic	Name of the MQTT topic to which the MQTT-LDM subscribes. The MQTT-LDM will receive messages on this topic and publish the appropriate data values to the PLC.
Subscribe Topic	Optional. Ignored if the *Access* field is set to RD (read only). If this option is omitted, or has an empty value, the system uses the same topic used for publishing. In this case, the MQTT-LDM receives its own Publish messages as well. Although the received value is written only if it is different than the last read value, there is no guarantee that an older value is not written due to race conditions. It is recommended to use a different Subscribe Topic name, rather than a Publish Topic name. In the sample configuration files, the Subscribe Topic name is composed by adding ***/*Set** at the end of the topic name.

9.2 Configuring Generic MQTT Brokers

An example configuration file is located in the **mqtt-ldm-sample-app-mvi69e\test-generic** folder.

9.2.1 Generic MQTT Broker

This is located in the **mqtt-ldm-sample-app-mvi69e\test-generic** folder.

The file config.json contains a default configuration file, which is received during a firmware update. This file contains settings for the generic MQTT broker. There are no specific requirements pertaining to *Client ID*, *Topic Namespace*, or other settings.

The node *MqttServer* of the configuration file has the following default values:

Parameter	Description
Type	Set to **Generic**.
HostName	Set to **192.168.0.254** (should be modified to IP address of the PC where MQTT broker is running).
Port	Set to **1883**.
DoNotUseTls	Set to **1** (unencrypted communication mode).

You can connect to different MQTT brokers by modifying only the *HostName* field. Different installations of MQTT brokers are considered in the next section.

9.2.2 Online MQTT Brokers

There are couple of MQTT brokers available online:

- mqtt.eclipse.org
- test.mosquitto.org

For both, the port number for unencrypted TCP communication is **1883**.

For encrypted communication: **8883**.

Communication Type	Port Number
Unencrypted	1883
Encrypted	8883

9.2.3 Install MQTT Locally

If complete control over the MQTT broker is needed, or if there is no access to the Internet from the network where the MVI69E-LDM is installed, it is possible to use a local version of the MQTT broker.

For example, open-source broker Mosquitto can be downloaded from https://mosquitto.org/download/. Once installed, it starts as a Windows service Mosquitto Broker and is ready to accept connections at port 1883.

In the configuration file, if the *MqttServer/HostName* field is set to the IP address of the MQTT broker, values for the PLC should be published into it. This will verify that you can use third-party MQTT clients.

As a result, the tag value in the PLC is updated and the newly published values are reported in MQTT Explorer and plotted on the history graph.

9.3 Running the Sample Application

With the configuration complete, restart the application.

To restart the application, either reboot the module, or connect to the module over Telnet terminal and run the following command:

```
/etc/init.d/S88-mqtt stop
```

and then

```
/etc/init.d/S88-mqtt start
```

10 MQTT-LDM Library

This chapter pertains to developers building custom applications using the library. It describes high-level design of the library and main API functions required to use it from customer applications.

10.1 Component Diagram

Interaction between components is illustrated in the following component diagram:

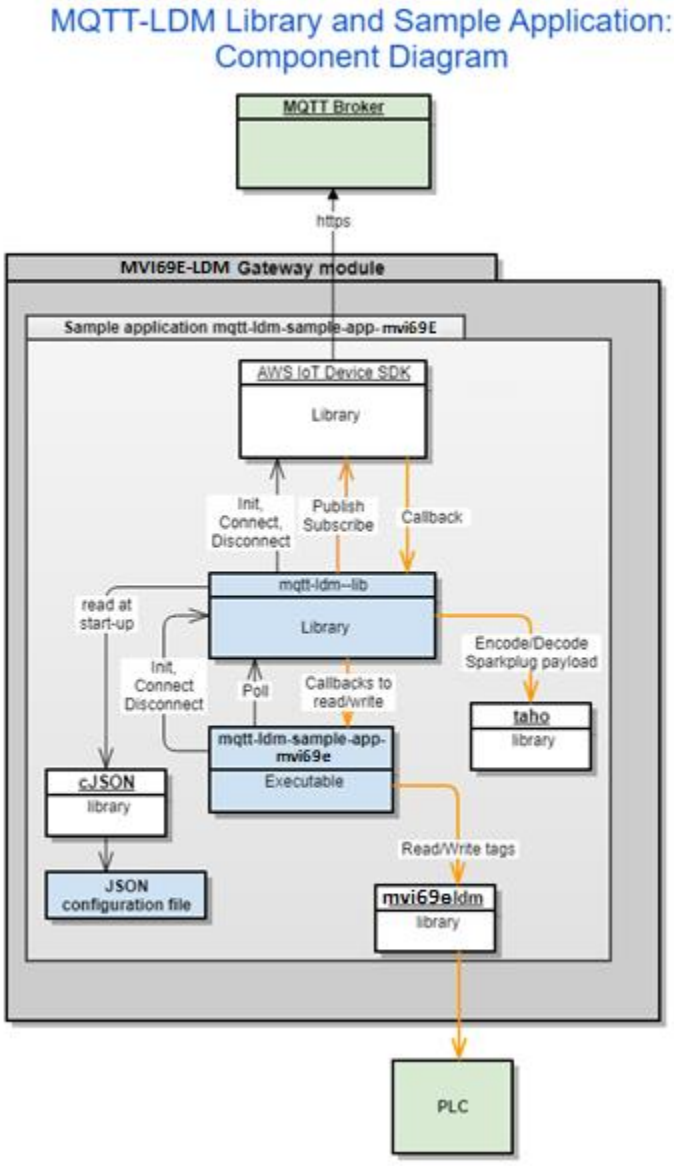

Figure 42: *Component Diagram*

10.2 Main API Functions and Data Flow

10.2.1 Functions Implemented by the Library

The functions defined in the header file `C:\Workspace\mqtt-ldm\mqtt-ldm-lib\inc\mqtt-ldm-lib.h` are shown in the following examples. They are expected to be called from the user application:

int mqtt_ldm_initialize(const char* path_to_config_file);

- This function should be called once at application start-up to initialize internal structures and start its threads.
- Its input argument is a path to the JSON configuration file.
- If successful, the function returns 0.

int mqtt_ldm_connect(void);

- This function connects to the MQTT broker.
- If successful, the function returns 0.

int mqtt_ldm_disconnect(void);

- This function disconnects from the MQTT broker.

int mqtt_ldm_is_connected(void);

- Returns a non-zero value if a connection to the MQTT broker is established.

int mqtt_ldm_is_connecting(void);

- Returns a non-zero value if a connection to the MQTT broker is in progress.

int mqtt_ldm_clean(void);

- Frees resources used by the MQTT-LDM library and by the AWS IoT SDK, it should be called before application exit.

uint32_t get_tick_count(void);

- Helper function. Returns number of ticks since computer start, in milliseconds.

int mqtt_ldm_poll(void);

- This function must be called by the user application in a continuous loop. It reads the current value for each tag (by calling function **mqtt_read_value**) and if the value is changed, it publishes it on the MQTT broker.

void ldm_log(enum ldm_log_level log_level, const char *format, ...);

- Used by the user application to log messages into log file. Passed arguments are log-level, C style format string, and optional data to log.

10.2.2 Callback Function Declarations

Functions - Callbacks, called by MQTT-LDM Library during runtime, should be implemented by the user application. The sample application provides a default implementation of the callback functions.

int mqtt_ldm_read_value(plc_tag* tag, plc_value* value);

- Called to read the value of a tag from the PLC.
- The first input argument, tag, is the tag name to be read from the PLC.
- The value read from the PLC is returned in the *output argument* value.
- If successful, the read value is returned in the *output argument* value and the function returns 0.

int mqtt_ldm_write_value(plc_tag* tag, plc_value* value);

- Called to write a new value to a tag.
- The first input argument, tag, is the tag name to be written to the PLC.
- The input argument value is the value to be written to the PLC tag.
- New values are received from MQTT broker by subscribing to specified subscription topics in the configuration file.

int mqtt_ldm_is_connected_to_plc(void);

- If the module is connected to the PLC, it returns 1. Otherwise, it returns 0.

int mqtt_ldm_get_status(char is_verbose, char buffer, uint16_t max_size);**

- This function is used to get the current status of the communication with the PLC.
- If the input argument *is_verbose* is not 0, then more detailed information is returned.
- When argument *is_verbose* has a non-zero value, sample application **mqtt-ldm-sample-app-mvi69e** returns content of log file in the buffer.
- The result is copied into the memory buffer. It may be pre-allocated by the caller. In this case, its size is passed in the *max_size argument*. If the buffer points to NULL, then *max_size* can still be used to limit the size of returned text.
- If successful, return 0.

Examples of the use of these functions can be found in the source code of the sample application.

10.3 Logging

The logging feature uses the standard Linux daemon syslog. The script S10-syslog, which configures and starts the feature, is located in the folder of the sample application (**mqtt-ldm-sample-app-mvi69e**). It is included in the firmware and installed in the module's folder `/etc/init.d` and starts syslog daemon at system boot.

Messages are logged into the file `/www/html/log/messages.txt`, under the web server's public files folder. Therefore, this file can be accessed via the module's web server: `http://192.168.0.250/log/messages.txt`

Logging is configured to limit the file size to 32 Kb, with a maximum number of files set to **1**. When the log file size exceeds 32Kb, the active log file is archived into file *messages.txt.0*, and logging continues in a new file *messages.txt*. The archived copy can be accessed via URL: `http://192.168.0.250/log/messages.txt.0`

10.4 Data Flow for Reading Tag Values

Refer to "Component Diagram" on page 47. The orange lines in the diagram illustrate data flow.

In order to read data from the PLC and publish it to the MQTT broker, the user application must periodically call the function *mqtt-ldm-poll*. In this function, MQTT-LDM iterates over all configured tags: if the time specified by the *Scan Rate* parameter has elapsed since last read call, it performs a reading of the current value from the PLC by calling *mqtt_ldm_read_value*. If this value is newer than the last read value, it is reported to the MQTT broker in a Publish message.

If the last Publishing of the tag value time is greater than *MaxPublishInterval*, then the value is published, even if the value has not changed.

10.5 Data Flow for Writing Tag Values

During the connection, if there are writable tags, mqtt-ldm-lib subscribes to topics used to receive Publish messages. It then registers a callback function on AWS IoT SDK, which is called when the corresponding Publish message is received. If the received value is different than the existing value, it is then written to the PLC.

11 Firmware

11.1 Firmware Contents

The sample application installed to the MVI69E-LDM contains files that are included in the **MVI69E-LDM-MQTT zip** file. The contents of the zip file were copied to the Windows 10 PC's **C:\Workspace** folder in Chapter 2.

Although they are installed on the module with the module webpage's *Firmware Download* process, the files can be FTP'd independently as well.

When the module is rebooted after the upgrade, the sample application starts via script: `/etc/init.d/S88-mqtt`.

If the module has other custom files and/or the firmware update is not desirable, then the following files from **C:\Workspace** can be installed over FTP connection to the module:

#	File location on Windows 10 PC	FTP to folder on MVI69E-LDM	Description
1	C:\Workspace\mqtt-ldm-sample-app-mvi69e\firmware **mvi69e-ldm.firmware_<version>_<date>.firmware**	/psft/sample/mqtt	MQTT-LDM sample application. After building a custom application, copy the .firmware file to the module.
2	C:\Workspace\mqtt-ldm-sample-app-mvi69e\ **config.json**	/psft/sample/mqtt	Configuration file. Its structure is described in detail in section *Configuration File Details* starting on page 43.
3	C:\Workspace\mqtt-ldm-sample-app-mvi69e\ **root_ca.cer**	/psft/sample/mqtt	Root CA certificate of the server's SSL certificate's chain.
4	C:\Workspace\mqtt-ldm-sample-app-mvi69e\ **S10-syslog**	/etc/init.d	Optional,for logging. The service syslog should be started in order to enable logging. The service can be started automatically at system reboot by copying of the script file S10-syslog to the folder */etc/init.d*. Note that the script configures log file location as **/www/html/log/messages.txt**, i.e. under embedded web server's content location. This allows viewing of the log file's content at: **http://192.168.0.250/log/messages.txt** (IP address of the module).
5	C:\Workspace\mqtt-ldm-sample-app-mvi69e\ **S88-mqtt**	/etc/init.d	This script starts the sample application on the MVI69E-LDM after module reboot.
6	C:\Workspace\mqtt-ldm-sample-app-mvi69e\test-generic	/psft/sample/mqtt	Configuration file and certificates, specific for the Generic type of MQTT brokers.

11.2 Run the Application

After the installation and configuration are complete, the application can be started either automatically after device reboot (if the script **/etc/init.d/S88-mqtt** was installed), or it can be started manually via telnet terminal by navigating to the folder */spft/sample/mqtt* and running the command **./mqtt-ldm-sample-app-mvi69e**.

12 Visual Studio 2017 Project

You can use Visual Studio 2017 to build the sample application. Before doing so, make sure you have first completed the steps in "Prerequisites for Customizing the Sample Application" and "Development Setup."

12.1 Visual Studio Build

The Visual Studio 2017 solution file is located at:

C:\Workspace\mqtt-ldm-sample-app-mvi69e\mqtt-ldm-mvi69e.sln

This file has two projects:

- `mqtt-ldm--lib`
- `mqtt-ldm-sample-app-mvi69e`.

1 In the *Solution Explorer*, click on **mqtt-ldm-sample-app-mvi69e**, right-click and choose **PROPERTIES**.

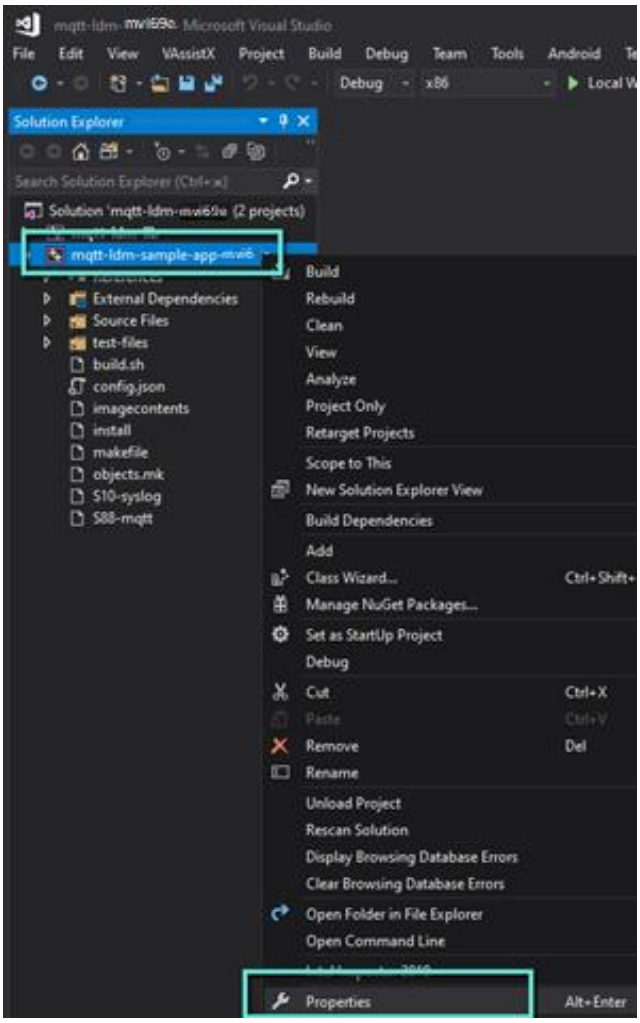

Figure 43: *Solution Explorer*

2 Choose *Build Events* from the left panel. Notice the *Command Line* on the right. Copy this *Command Line* to Notepad, and modify it as follows:
   - Set the correct **IP address**. This is your Windows 10 PC's IP address.
   - Set the **userid** and **password**.

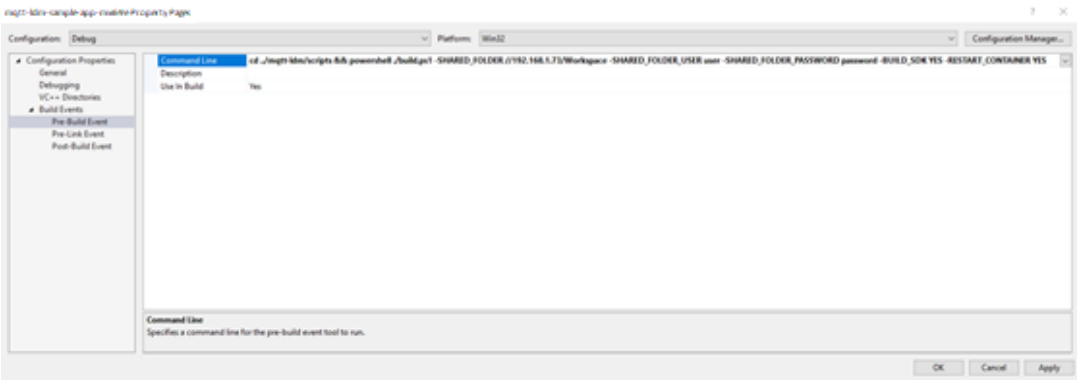

Figure 44: *Command Line in Visual Studio- Pre Build Events*

3 Place the updated command line back into Visual Studio.

Note: During the building of the project, the PowerShell script (build.ps1) is executed. Your PC security policies might prevent it from running. To run this script, open a command prompt (as Administrator) and enter the following commands:

Powershell

```
Set-ExecutionPolicy –ExecutionPolicy RemoteSIgn –Scope
LocalMachine

Set-ExecutionPolicy –ExecutionPolicy RemoteSign –Scope
CurrentUser
```

4 In *Solution Explorer*, select **mqtt-ldm-sample-app-mvi69e**, right-click and choose **BUILD**.

5 The warning "You are building a Docker image from Windows against a non-Windows Docker host…" is OK.

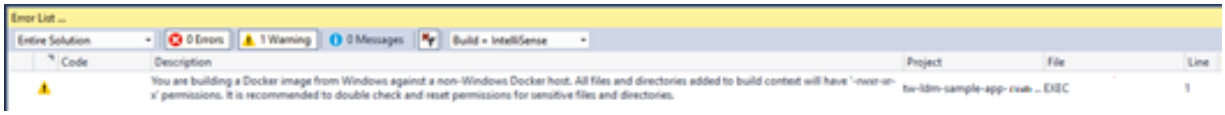

Figure 45: *Error List*

6 The first run can take about 15 minutes. The firmware file created is located here: C:\Workspace\mqtt-ldm-sample-app-mvi69e\firmware\mvi69e-ldm.firmware_<version number>_<date>.firmware

13 Support, Service & Warranty

13.1 Contacting Technical Support

ProSoft Technology, Inc. is committed to providing the most efficient and effective support possible. Before calling, please gather the following information to assist in expediting this process:

- Product Version Number
- System architecture
- Network details

If the issue is hardware related, we will also need information regarding:

- Module configuration and associated ladder files, if any
- Module operation and any unusual behavior
- Configuration/Debug status information
- LED patterns
- Details about the interfaced serial, Ethernet or Fieldbus devices

Note: *For technical support calls within the United States, ProSoft's 24/7 after-hours phone support is available for urgent plant-down issues.*

North America (Corporate Location)	**Europe / Middle East / Africa Regional Office**
Phone: +1.661.716.5100 info@prosoft-technology.com Languages spoken: English, Spanish REGIONAL TECH SUPPORT support@prosoft-technology.com	Phone: +33.(0)5.34.36.87.20 france@prosoft-technology.com Languages spoken: French, English REGIONAL TECH SUPPORT support.emea@prosoft-technology.com
Latin America Regional Office	**Asia Pacific Regional Office**
Phone: +52.222.264.1814 latinam@prosoft-technology.com Languages spoken: Spanish, English REGIONAL TECH SUPPORT support.la@prosoft-technology.com	Phone: +60.3.2247.1898 asiapc@prosoft-technology.com Languages spoken: Bahasa, Chinese, English, Japanese, Korean REGIONAL TECH SUPPORT support.ap@prosoft-technology.com

For additional ProSoft Technology contacts in your area, please visit:
https://www.prosoft-technology.com/About-Us/Contact-Us.

13.2 Warranty Information

For complete details regarding ProSoft Technology's TERMS & CONDITIONS OF SALE, WARRANTY, SUPPORT, SERVICE AND RETURN MATERIAL AUTHORIZATION INSTRUCTIONS, please see the documents at:
www.prosoft-technology/legal