



EDM MQTT PROTOCOL

October 22nd, 2025
Document ver. 1.12

© Crystal Instruments Corporation

Contents

EDM MQTT	7
MQTT Introduction	7
Understanding EDM MQTT Network.....	8
EDM MQTT Broker	9
Broker Parameter Settings	10
Connected Clients	12
Broker Log	12
EDM MQTT Client.....	13
Client Connection Parameter Settings	13
Sparkplug Setting.....	14
Publish Setting	15
TLS Setting	16
Advanced Setting	17
Messages	18
Subject Prefix.....	19
How to Connect to EDM MQTT Network	21
Controlling Multiple EDM Suite Apps.....	21
How to Connect to AWS IoT Core Thing	23
Creating an AWS IoT Core Thing Basic	23
Creating an AWS IoT Core Thing Policy.....	27
Connecting to AWS IoT Core Thing with EDM MQTT Client.....	32
Publishing Payload.....	36
Receiving Messages.....	37
App Universal Topics	38
App/Message.....	38
App/Status.....	38
App/Error	38
App/System.....	38
App/System/Status	39
App/System/TimeStatus	39
App/Test.....	40
App/Test/Status.....	40
App/Test/Command.....	40

App/Test/CommandEx	44
App/Test/RecordStatus	45
App/Test/LimitStatus	45
App/Test/Channels	46
App/Test/ChannelStatus	47
App/Test/Parameters	48
App/Test/Signals	48
App/Test/List	49
App/Test/SignalData	49
App/Test/SingleSignalData	50
App/Test/CompressedSignalData	50
App/Test/CompressedSingleSignalData	50
App/Test/ReportFile	50
App/Test/ReportTemplates	51
App/Test/ReportNotes	51
App/Test/RecordFile	51
App/Test/SignalProperty	52
App/Test/RunFolder	52
App/Test/AdvancedStatus	52
Global Parameter Topics	53
GlobalParameter/Request	53
GlobalParameter/Response	53
DSA Topics	56
DSA/Test/Command	56
DSA/Test/CommandEx	58
DSA/Test/DSASatus	58
DSA/Test/TriggerSettings	59
DSA/Test/TriggerFrame	59
VCS Topics	59
VCS/Test/Command	59
VCS/Test/CommandEx	66
VCS/Test/Stage	66
VCS/Test/ControlUpdated	67
VCS/Test/Shaker	67

VCS/Test/Mass	68
VCS/Test/RandomStatus	68
VCS/Test/SineStatus.....	68
VCS/Test/ShockStatus.....	69
VCS/Test/TWRStatus.....	70
VCS/Test/MIMORandomStatus	70
VCS/Test/MIMOSineStatus	71
VCS/Test/MIMOShockStatus.....	72
VCS/Test/MIMOTWRStatus.....	73
THV Topic	74
THV/Test/THStatus	74
THV/Test/VibStatus	74
EDM MQTT CLIENT DEMO PROGRAM	75
EDM MQTT Client Demo Package	76
Demo Package Content.....	77
How to Install the EDM MQTT Client Demo Program	77
EDM MQTT Client C# Demo Showcase	77
How to Build the MQTT Client C# Demo	77
C# Program GUI.....	79
Connecting and Running a Test.....	92
EDM MQTT Client Python Scripts	96
MQTT Client API Script	97
Running a Test and Plotting Signal Data.....	97
Automating THD Measurement	102
EDM MQTT Client LabVIEW Demo	106
LabVIEW Front Panel	108
LabVIEW Block Diagram	110
EDM MQTT Client Java Demo.....	114
Java Scripts	114
Java GUI app.....	114
EDM MQTT Client C++ Demo.....	116
C++ CLI app	116
Connecting	117
Commands	117

EDM MQTT C# Sparkplug Demo	118
C# Demo App	118
C# Demo Code.....	119
Project Structure.....	119
Dependencies	120
Build Steps.....	120
HOW THE USER'S SOFTWARE READS CI DATA FILES (ATFX API)	121
END USER LICENSE AGREEMENT FOR CRYSTAL INSTRUMENTS SOFTWARE	ERROR! BOOKMARK NOT DEFINED.

Information in this document is subject to change without notice. No part of this document may be reproduced or transmitted in any form, for any purpose, without the written permission of Crystal Instruments Corporation (“Crystal Instruments”).

By installing, copying or using the Software, the user agrees to be bound by the terms of the Crystal Instruments End User License Agreement which is a legally binding agreement between the user (“the Licensee”) and Crystal Instruments for the Crystal Instruments software, which includes software components, tools, and written documentation (“Software”).

Crystal Instruments makes no warranties on the Software, whether express or implied, nor implied warranties of merchantability or fitness for a particular purpose. Crystal Instruments does not warrant your data, that the software will meet your requirements, or that the operation will be reliable or error free. The Licensee of the Software assumes the entire risk of use of the Software and the results obtained from the use of the software. Crystal Instruments shall not be liable for any incidental or consequential damages, including loss of data, lost profits, the cost of cover, or other special or indirect damages.

Copyright © 2005-2025 Crystal Instruments Corporation. All rights reserved.

All trademarks and registered trademarks used herein are the property of their respective holders.

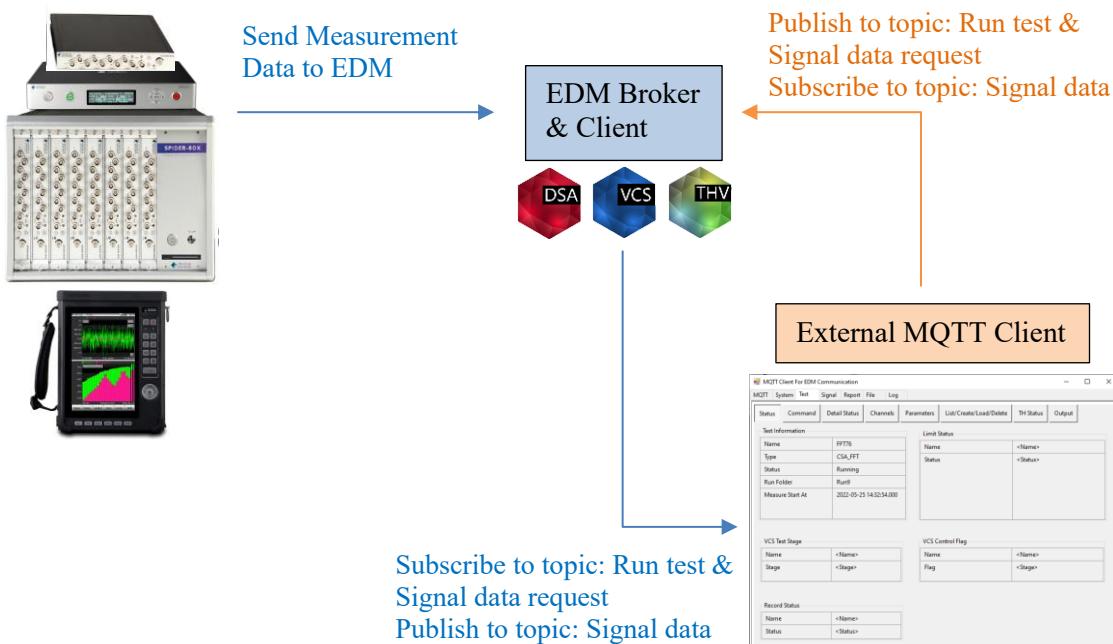
EDM MQTT

MQTT Introduction

MQTT is an OASIS standard messaging protocol for the Internet of Things (IoT). It is designed as an extremely lightweight publish/subscribe messaging transport that is ideal for connecting remote devices with a small code footprint and minimal network bandwidth.

By implementing MQTT in EDM, users can monitor status of environmental tests (vibration, temperature, humidity) run in EDM VCS and measurements taken in EDM DSA and even remotely run a test.

Here is the EDM MQTT Publish/Subscribe Architecture.



MQTT client can be

- The MQTT Client in EDM (publisher) publishes test status and measurements to an MQTT broker.
- A software (subscriber) written by a user to subscribe to topic from an MQTT broker to get status and measurements from EDM MQTT client or send a command to perform various EDM test controls.

MQTT broker can be

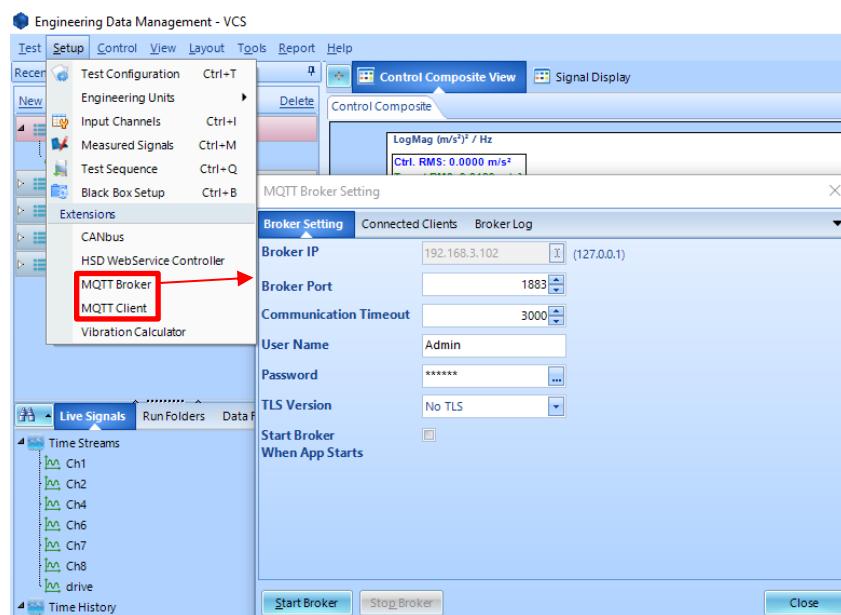
- The MQTT Broker in EDM communicates to MQTT clients (both publisher and subscriber).
- A MQTT Broker hosted on the Internet by a third-party provider or on a LAN.

EDM has a built-in MQTT Client and MQTT Broker, available in EDM VCS and EDM DSA. The built-in MQTT Client also supports the Sparkplug™ specification, which can be enabled/disabled.

Please refer to <https://mqtt.org/> for MQTT technical content.

Please refer to <https://sparkplug.eclipse.org/> for Sparkplug™ specification.

In the following topics for the MQTT Broker and MQTT Client windows, these can be accessed via **Setup** and under **Extensions**.

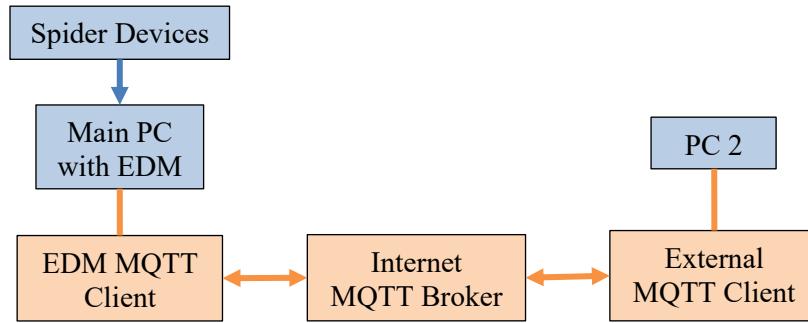


Understanding EDM MQTT Network

When all MQTT clients are in the same **local area network (LAN)**, users have the option to utilize the built-in MQTT Broker feature of EDM. This involves configuring the built-in MQTT Client of EDM to connect to the Broker for data publishing, while other MQTT Clients from various applications connect to the same Broker to subscribe and retrieve data. This setup facilitates communication and system integration efficiently, requiring only one broker within the LAN.

However, when some MQTT clients are located outside the LAN, such as in remote locations or over the internet, a different approach is necessary. In this scenario, both local and remote clients need to connect to an MQTT broker hosted on the internet. Consequently, the built-in Broker feature should be disabled, and the built-in MQTT Client of EDM should be configured to connect to the external MQTT Broker for data publishing.

Below is a graphical representation illustrating an example network layout for better understanding.

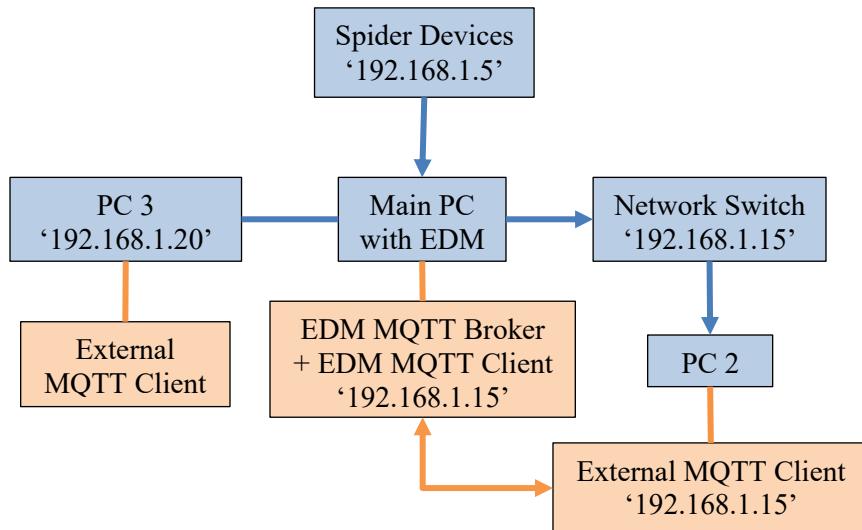


In a LAN network scenario where, multiple computers connect to the main PC with EDM via ethernet, the placement of the MQTT broker in relation to the IP addresses of the spider devices is important.

The MQTT broker IP address does not need to be on the same IP address as the spider devices, such as '192.168.1.5'.

If the MQTT broker's IP address matches that of the main PC's network adapter, for example, '192.168.1.15', which is connected to a network switch, then external MQTT clients can successfully connect to the MQTT broker if they are also on the same network switch.

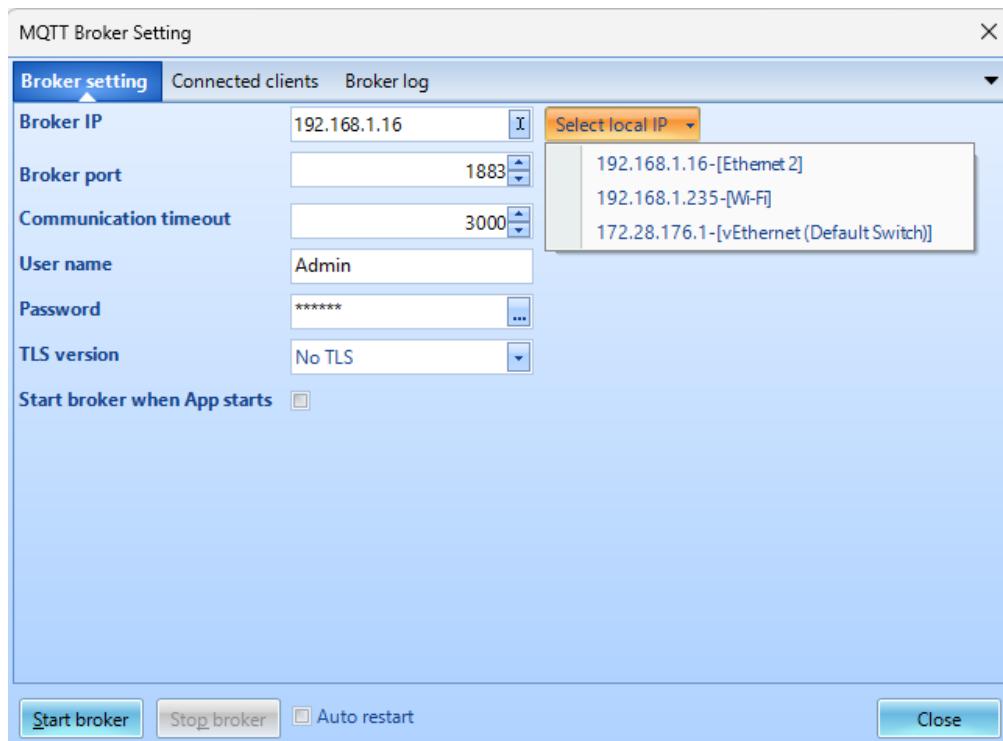
If an MQTT client is on another network adapter of the main PC, such as '192.168.1.20', which is not directly connected to the network switch but is still connected to the main PC, it will not be able to establish a connection with the MQTT broker.



EDM MQTT Broker

EDM built in MQTT broker for LAN network for local MQTT clients.

Broker Parameter Settings



- **Broker IP** – The MQTT Broker IP Address that clients can connect to. It must be different from known used IP addresses such as the spider devices.
- **Select local IP** – A dropdown menu that displays the computer's network adapter IPv4 addresses.
- **Broker port** – The MQTT Broker Port that clients established a connection through.
- **Communication timeout** – A timer in milliseconds that will stop the broker if it becomes unresponsive due to a command or action.
- **User name** – The MQTT Broker ID that clients use to identify.
- **Password** – The MQTT Broker Password that clients use to login.
- **TLS version** – Transport Layer Security Version, which is to provide secure message communication between the broker and clients.
- **Start broker when app starts** – A checkbox that determines whether the broker should start when EDM starts.

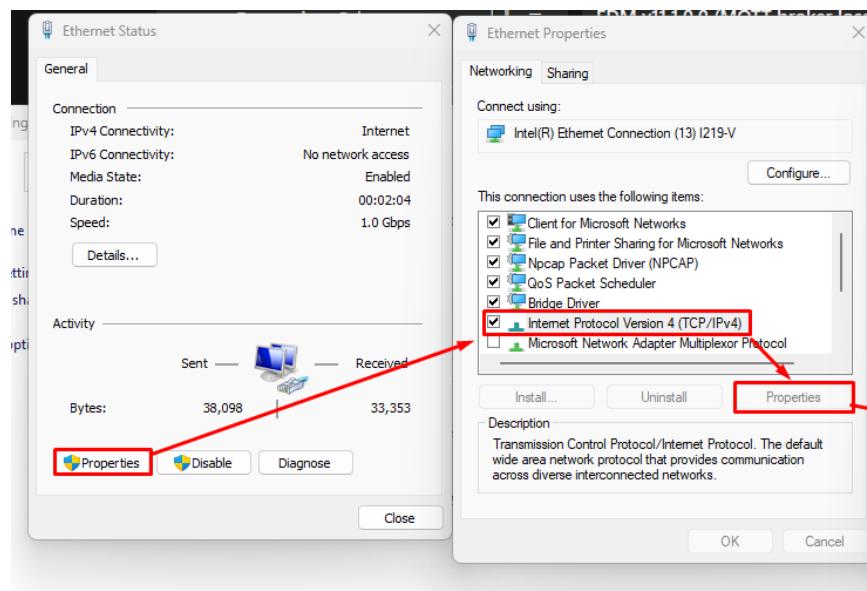
- **Auto restart:** Restart the MQTT Broker if it somehow stops while EDM is running. This does not restart the MQTT broker if launching EDM.

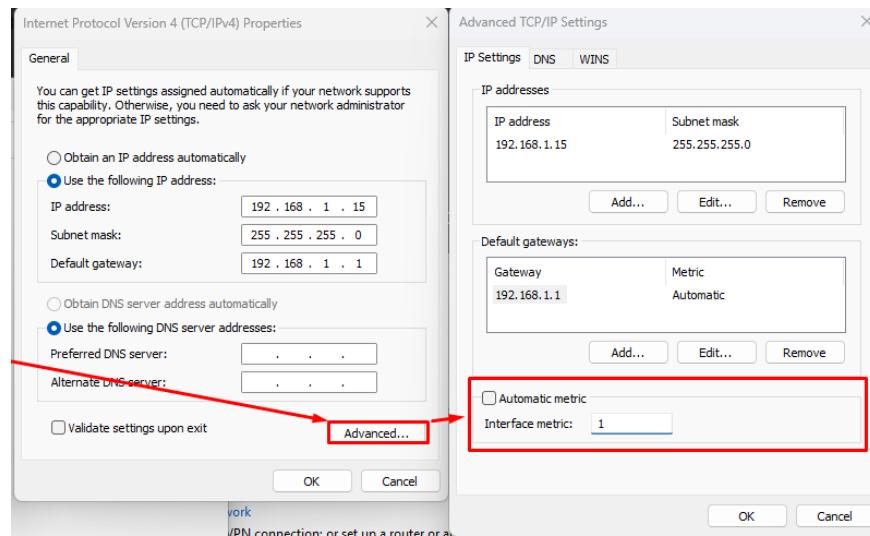
Ensuring EDM Broker IP is Set to a Specific IP

NOTE: The below workaround is no longer necessary as the EDM MQTT Broker will save whatever IP address set in the Broker IP field even after restarting EDM. The workaround will be kept just in case it is needed as well as for previous versions of EDM below 11.1.

When EDM restarts and EDM Broker window pops up, the IP address may be different than what it was set to previously. This is because EDM sets the broker IP to the first network adapter as shown via Control Panel > Network and Internet > Network and Sharing Center.

To ensure that EDM selects a correct network adapter IP address, such as the ethernet adapter connected to a spider system, users would need to change the network adapter IPv4 automatic metric.





Connected Clients

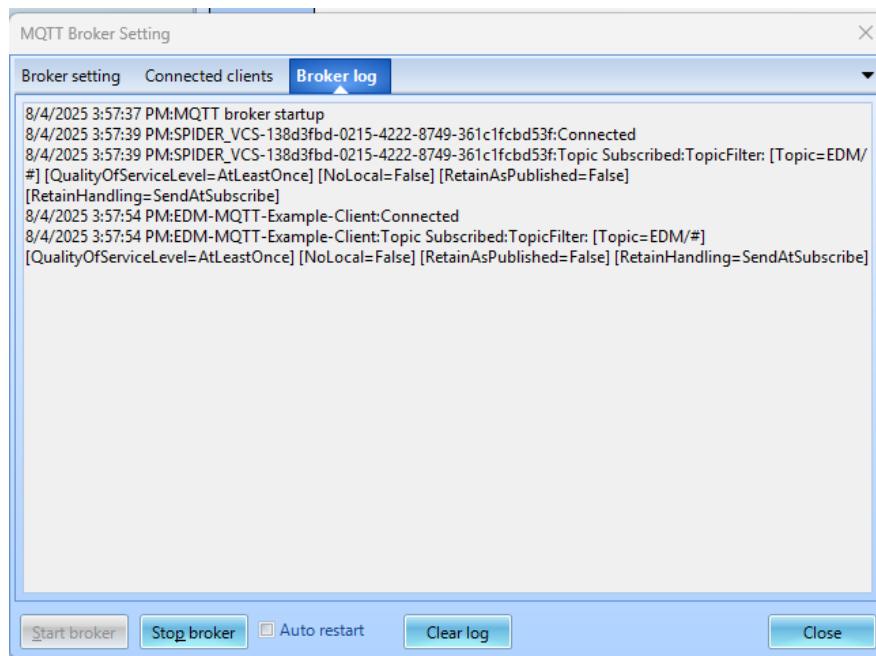
Show all MQTT clients connected to this Broker

MQTT Broker Setting			
Broker Setting	Connected Clients	Broker Log	
Client ID	User Name	Protocol Version	End Point
EDM-MQTT-Example-Client	Admin	V311	192.168.10.10:1028
SPIDER_VCS-635e70e4-91e7-404d-b... Admin		V311	192.168.10.11:56263

Broker Log

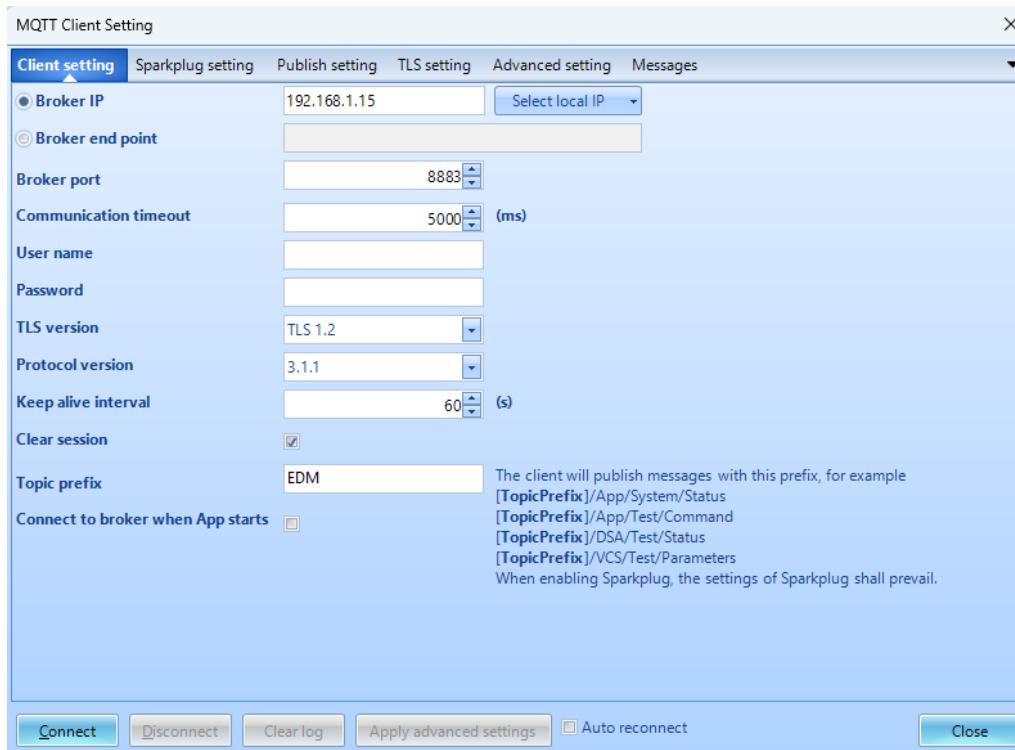
Shows received messages, established connections and so on between the MQTT broker and connected MQTT clients.

Pay attention to the log when the Broker settings window is open, as the log will not be saved after closing.



EDM MQTT Client

Client Connection Parameter Settings

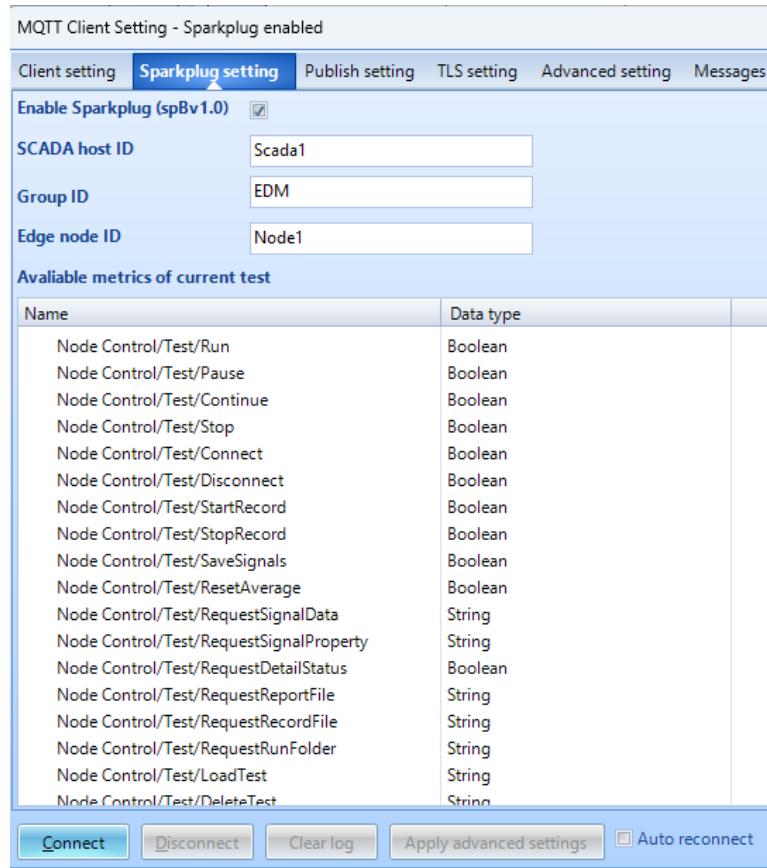


- **Broker IP** – The MQTT Broker IP Address that clients can connect to. It must be different from known used IP addresses such as the spider devices.
- **Select local IP** – A dropdown menu that displays the computer's network adapter IPv4 addresses.
- **Broker end point** – An endpoint that EDM MQTT client can connect to online. Additional information must be provided via other parameters in the Client setting tab and the TLS setting tab. For more details, go to [How to Connect to AWS IoT Core Thing](#).
- **Broker port** – The MQTT Broker Port that clients established a connection through.
- **Communication timeout** – A timer in milliseconds that will stop the broker if it becomes unresponsive due to a command or action.
- **User name** – The MQTT Broker ID that clients use to identify.
- **Password** – The MQTT Broker Password that clients use to login.
- **TLS version** – Transport Layer Security Version, which is to provide secure message communication between the broker and clients.
- **Protocol version** – The MQTT Client protocol version.
- **Keep alive interval** – Defines the maximum time interval between messages received from a client if the server detects that the network connection to a client has been dropped without a long TCP/IP timeout.
- **Clear session** – Determines if the connection should subscribe to topics that retain QoS 1 and 2 messages, otherwise treats the connection as clean.
- **Topic prefix** – The MQTT Client Prefix that establishes the identity of the client.
- **Connect to broker when App starts** – A checkbox that determines whether the client should connect to the broker if it has started when EDM starts.
- **Auto reconnect:** Reconnect the EDM MQTT Client to the MQTT Broker if it somehow loss connection while EDM is running. This does not reconnect the EDM MQTT client to the broker if launching EDM.

Sparkplug Setting

If users need to connect to a Broker that requires Sparkplug™ spB data parsing, such as Ignition (<https://inductiveautomation.com/ignition/>), the Sparkplug™ specification of the EDM built-in MQTT client must be enabled.

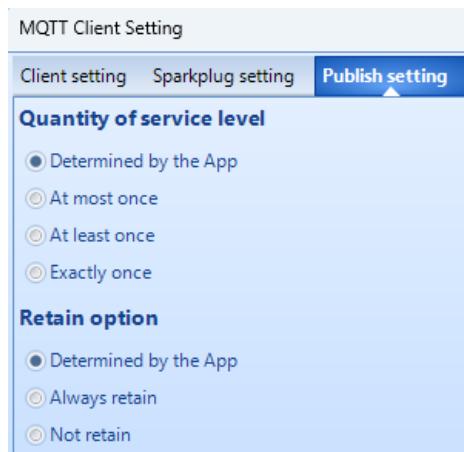
The list of metrics are commands available that can be used for sparkplug applications.



- **SCADA host ID:** Used to connect to SCADA applications.
- **Group ID:** A group of nodes that external MQTT clients point to.
- **Edge node ID:** The specific MQTT client node ID to send commands to and request data from. Other MQTT clients must have different node ID or there may be a conflict of communication.

Publish Setting

EDM will automatically select the **Quality of service** (QoS) level of message publishing and whether to retain or not. Users can change this in the **Publish Setting** tab.



Quality of Service Level has 3 levels and the option to let EDM determine the QoS level of the messages:

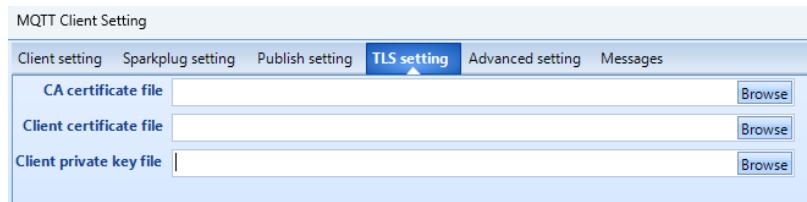
- **At most once** – Level 0 – Simplest, low overhead message, where the client publishes the message and there is no acknowledgement by the broker.
- **At least once** – Level 1 – Guaranteed message, where the client will keep publishing the message until the broker sends back an acknowledgement.
- **Exactly once** – Level 2 – Highest level message, where the client publishes a message to the broker to establish a handshake to ensure the message sent from the client is sent once.

Retain has 3 options:

- **Determined by the app** – Lets EDM determine the retained message option.
- **Always Retain** – Lets the broker remember retained message for topics, in which if clients subscribe to these topics, they will immediately receive the retained message.
- **Not Retain** – Lets the broker not remember the message.

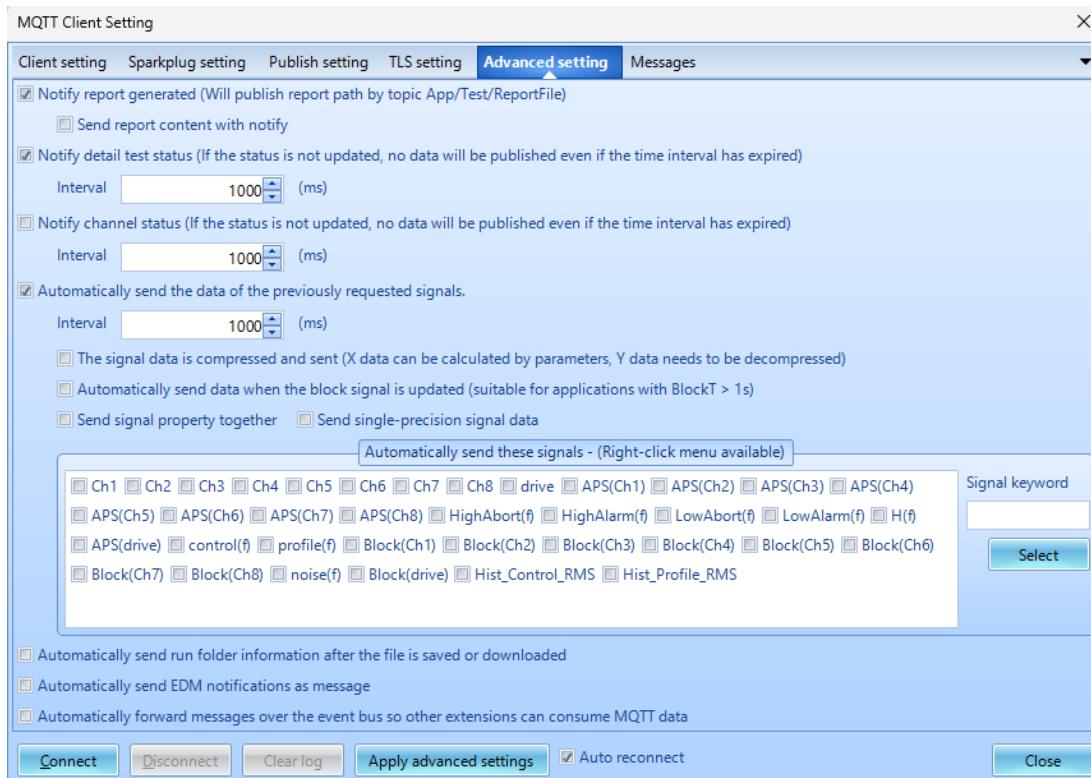
TLS Setting

The Transport Layer Security setting tab for including additional certificate files and key file for EDM MQTT client to connect to an endpoint. These are necessary to establish a connection and must be downloaded and stored somewhere where EDM can access them. These files are generated when an endpoint is created and are usually required to download them.



Advanced Setting

Here are some advance settings that the EDM MQTT client can publish on set time intervals.



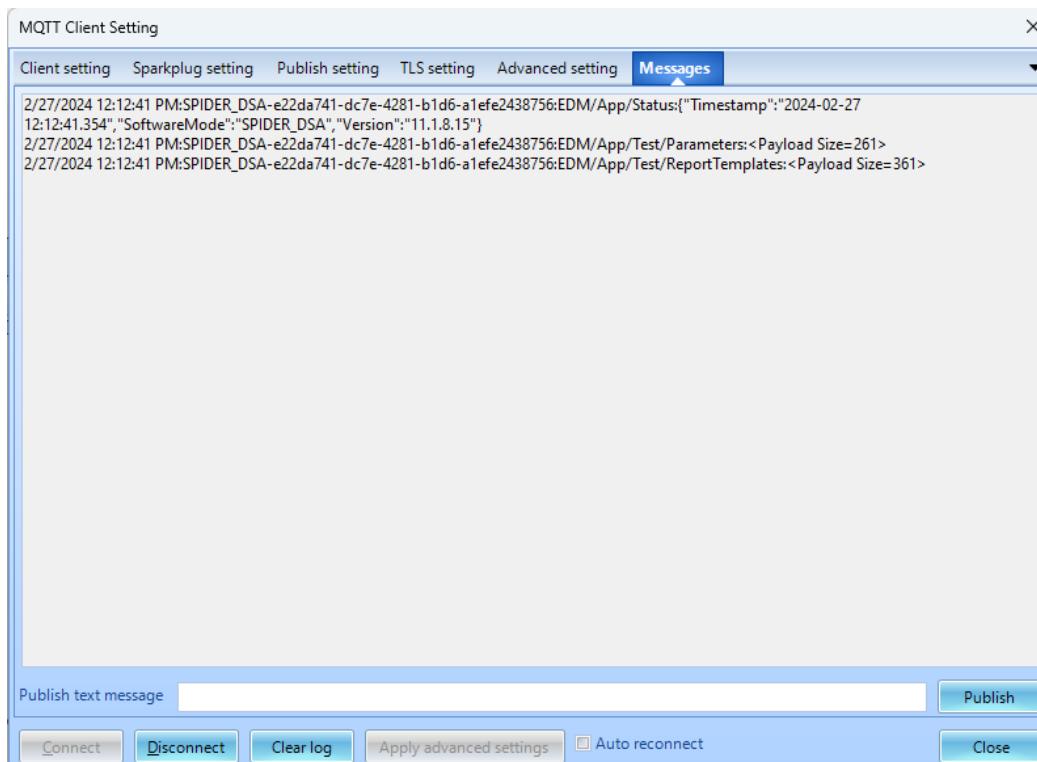
- **Notify report generated** – EDM MQTT Client will notify other clients that a report has been generated, in which will send a message with the report's file path.
 - **Send report content with notify** – EDM's Broker will also send a message that contains the report contents.
- **Notify detail test status** – EDM MQTT Client will notify other clients of the test detail status within a set interval and if there is any data to update.
- **Notify channel status** – EDM MQTT Client will notify other clients of the test's channel status within a set interval and if there is any data to update.
- **Automatically send data of previously requested signals** – EDM MQTT Client will send data of previously requested signals every set interval as well as the signal's

property if selected.

- **The signal data is compressed and sent** – Signal data can be compressed and sent, which will greatly reduce the size of the payload in the message. It needs to be sent and received using the TOPIC_APP_TEST_COMPRESSED_SIGNALDATA topic.
When compressed sending is enabled, the X-axis data will no longer be sent in array form, but will be calculated based on the parameters XStart/XDelta/XLength/XSequenceType
- **Automatically send data when the block signal is updated** – Signal data can be automatically sent according to BlockT time, that is, signal data is automatically sent when the Block signal has data update.
- **Send signal property together** – Signal data can be sent with signal properties.
- **Send single-precision signal data** – Signal data with single precision.
- **Automatically send these signals** – A list of selectable signals that are measured and displayed here that can be sent to other MQTT clients.
- **Automatically send run folder information** – EDM MQTT Client will send run folder information if the files have been saved or downloaded.
- **Automatically send EDM notifications as message** – EDM MQTT Client will send any EDM notifications to MQTT Clients of events that occur in EDM.
- **Automatically forward messages over the event bus so other extensions can consume MQTT data** – EDM MQTT Client will send messages to the event bus for other non MQTT extensions to use.

Messages

Shows published and received messages between the MQTT clients. It can also send simple text messages between clients.



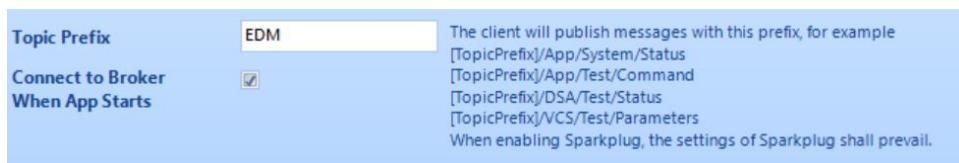
Subject Prefix

To distinguish the topics published by different clients, enable topic prefix.

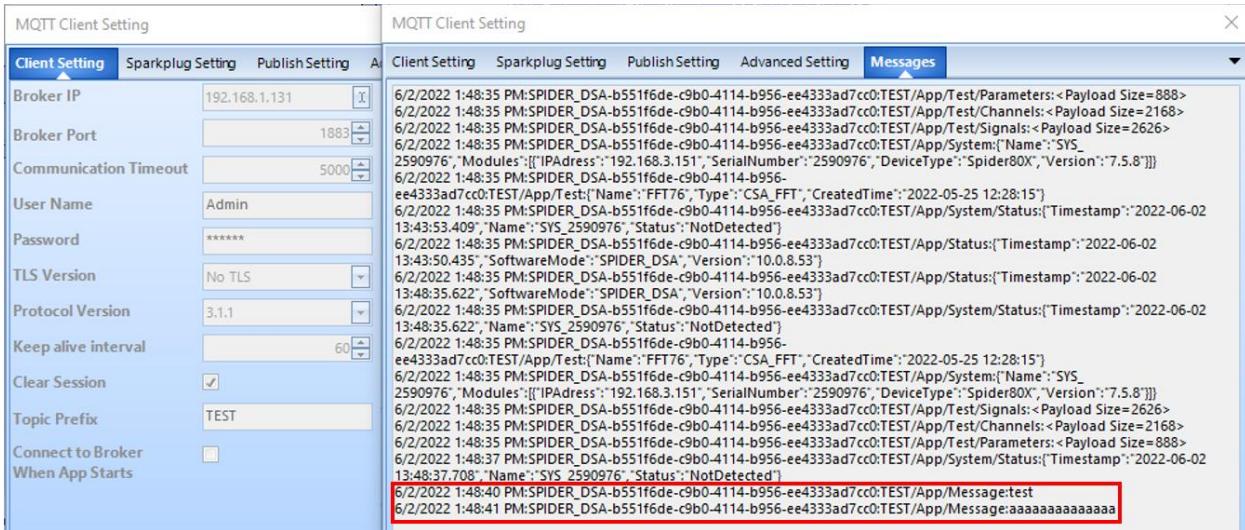
The **default prefix** is **EDM**, and the actual topic is EDM/App/Message, EDM/App/Test, etc.
 The prefix can be customized.

(The prefix is omitted in the following topic descriptions)

If there are multiple EDM clients that need to publish topics, be sure to set different topic prefixes to distinguish them.



An example of how the Topic Prefix works with text messaging between clients. The below EDM MQTT client topic prefix is TEST and the external MQTT client connecting to EDM topic prefix is also TEST. Both clients send messages and can receive them.

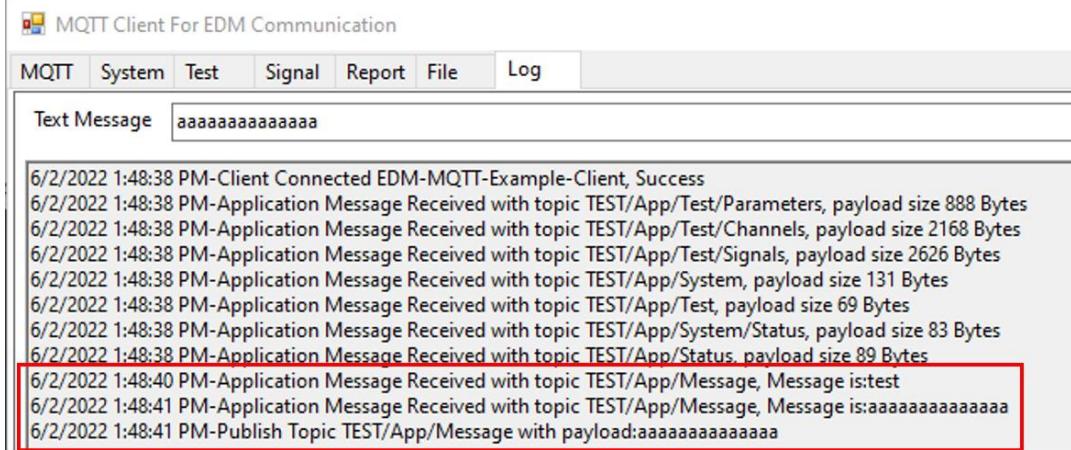


The screenshot shows two overlapping windows titled "MQTT Client Setting". The top window has tabs for "Client Setting", "Sparkplug Setting", "Publish Setting", and "Advanced Setting", with "Messages" selected. The bottom window also has these tabs, with "Messages" also selected. Both windows show a log of messages received by the client. The logs are identical, showing a series of application messages from the EDM system. The last two messages in the log are highlighted with a red box:

```

6/2/2022 1:48:40 PM:SPIDER_DSA-b551f6de-c9b0-4114-b956-ee433ad7cc0:TEST/App/Message:test
6/2/2022 1:48:41 PM:SPIDER_DSA-b551f6de-c9b0-4114-b956-ee433ad7cc0:TEST/App/Message:aaaaaaaaaaaaaaa

```



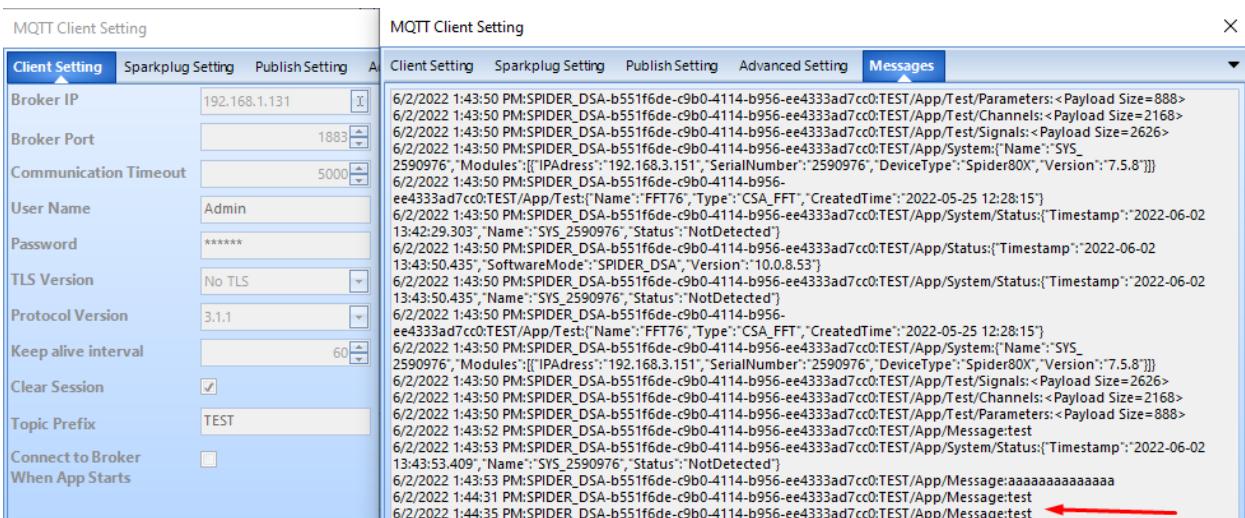
The screenshot shows the "MQTT Client For EDM Communication" interface. It has tabs for "MQTT", "System", "Test", "Signal", "Report", "File", and "Log". The "Log" tab is selected, displaying a "Text Message" log. The log shows the same sequence of messages as the previous screenshot, with the last two messages highlighted with a red box:

```

6/2/2022 1:48:38 PM-Client Connected EDM-MQTT-Example-Client, Success
6/2/2022 1:48:38 PM-Application Message Received with topic TEST/App/Test/Parameters, payload size 888 Bytes
6/2/2022 1:48:38 PM-Application Message Received with topic TEST/App/Test/Channels, payload size 2168 Bytes
6/2/2022 1:48:38 PM-Application Message Received with topic TEST/App/Test/Signals, payload size 2626 Bytes
6/2/2022 1:48:38 PM-Application Message Received with topic TEST/App/System, payload size 131 Bytes
6/2/2022 1:48:38 PM-Application Message Received with topic TEST/App/Test, payload size 69 Bytes
6/2/2022 1:48:38 PM-Application Message Received with topic TEST/App/System/Status, payload size 83 Bytes
6/2/2022 1:48:38 PM-Application Message Received with topic TEST/App>Status, payload size 89 Bytes
6/2/2022 1:48:40 PM-Application Message Received with topic TEST/App/Message, Message is:test
6/2/2022 1:48:41 PM-Application Message Received with topic TEST/App/Message, Message is:aaaaaaaaaaaaaaa
6/2/2022 1:48:41 PM-Publish Topic TEST/App/Message with payload:aaaaaaaaaaaaaaa

```

The EDM topic prefix is still TEST, but now the external MQTT client topic prefix is EDM. Both clients send messages, however, neither can receive the messages. This is due to the different topic prefix.

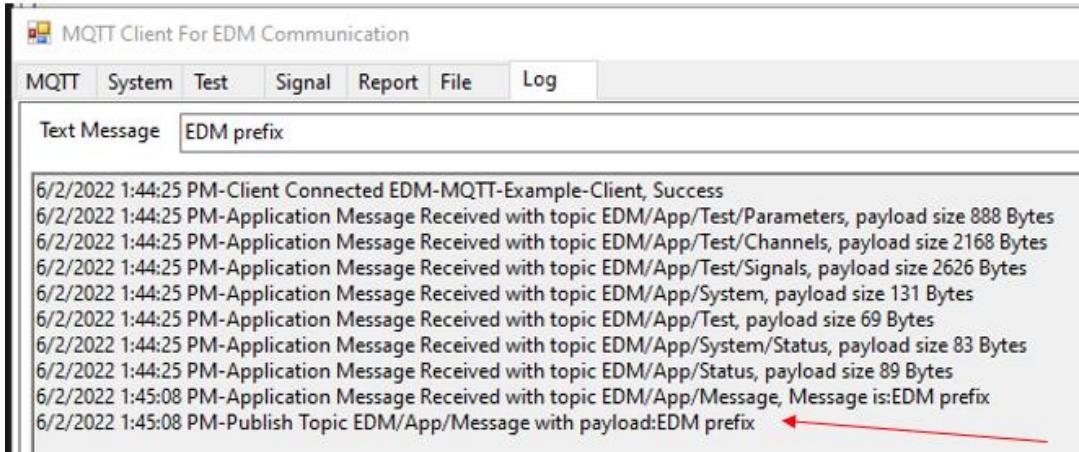


The screenshot shows the "MQTT Client Setting" window again, with the "Messages" tab selected. The log shows the same sequence of messages. A red arrow points to the last message in the log, which is the second-to-last message in the sequence:

```

6/2/2022 1:43:53 PM:SPIDER_DSA-b551f6de-c9b0-4114-b956-ee433ad7cc0:TEST/App/Message:test
6/2/2022 1:43:53 PM:SPIDER_DSA-b551f6de-c9b0-4114-b956-ee433ad7cc0:TEST/App/Message:aaaaaaaaaaaaaaa
6/2/2022 1:44:31 PM:SPIDER_DSA-b551f6de-c9b0-4114-b956-ee433ad7cc0:TEST/App/Message:test
6/2/2022 1:44:35 PM:SPIDER_DSA-b551f6de-c9b0-4114-b956-ee433ad7cc0:TEST/App/Message:test

```



How to Connect to EDM MQTT Network

To connect to EDM MQTT network, users must make sure that the following Client and Broker Settings in EDM matches with their client program:

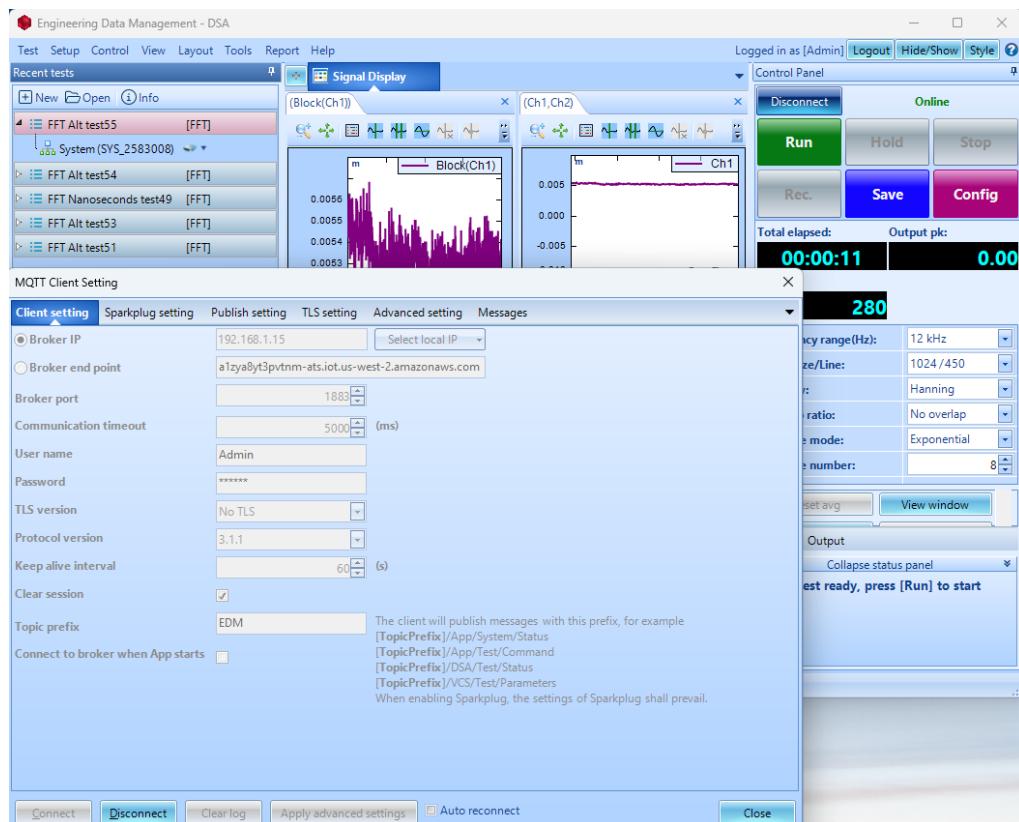
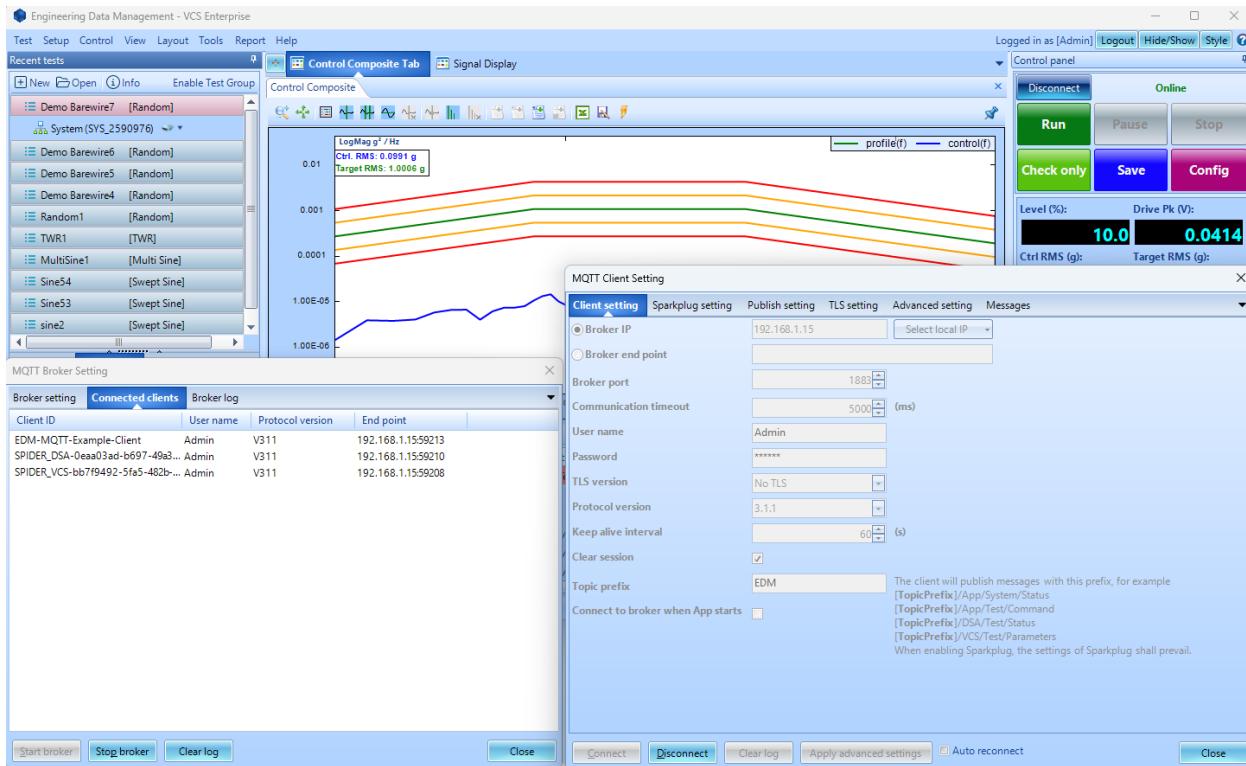
- Broker IP Address
- Broker port
- User name
- Password
- TLS version

Controlling Multiple EDM Suite Apps

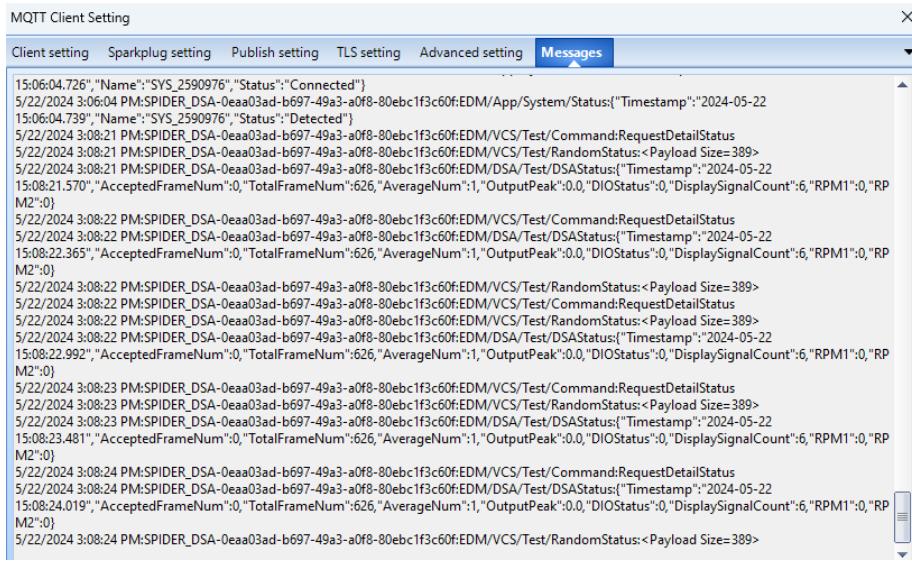
Multiple EDM suite apps (DSA, VCS, etc.) can be controlled simultaneously or from one computer by having their own EDM MQTT Client connected to one MQTT Broker.

An example shown in the images below, where VCS has a local MQTT broker started and VCS, DSA and a demo MQTT client are connected to the MQTT broker. The demo MQTT client can send commands to the EDM clients as well as receive data from those EDM clients.

EDM will be able to filter commands meant for specific EDM suite apps, such as proceeding a Random pretest for VCS or turning on a trigger for DSA.



For more advanced or customized external MQTT clients, they can separate which EDM client sent a data packet, via [Prefix]/[EDM suite app type]/etc.



How to Connect to AWS IoT Core Thing

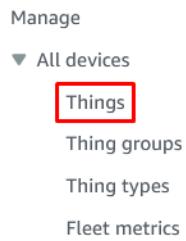
The following section will show how to create a basic AWS IoT Core Thing and how to connect to this Thing with EDM MQTT client.

Creating an AWS IoT Core Thing Basic

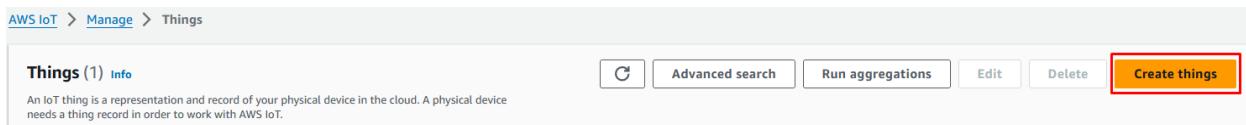
This section will cover how to create a simple AWS IoT Core Thing. Any advanced settings will not be covered in this manual.

To create an AWS IoT Core Thing, search for “**iot core**” and select **IoT Core** under Services.

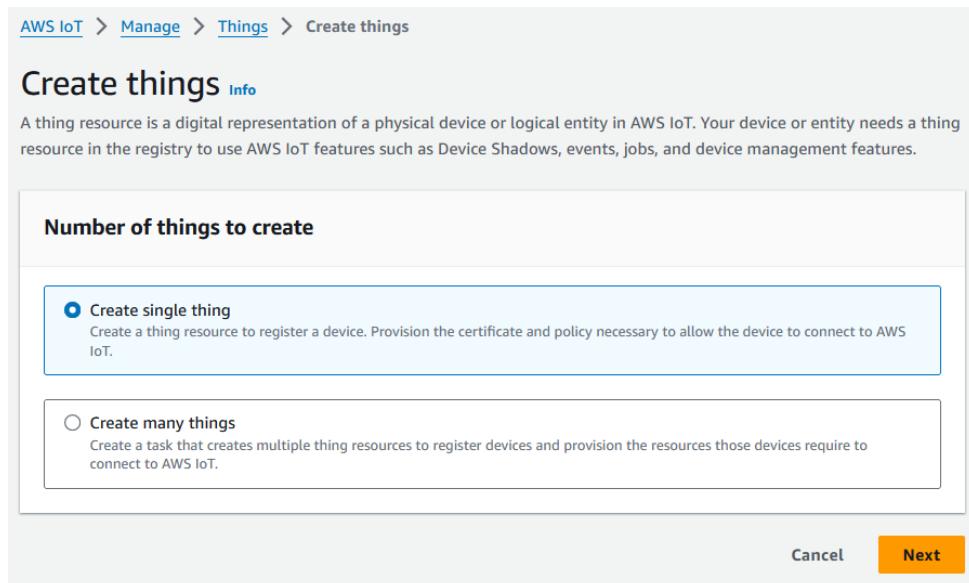
Upon loading the service page, on the left menu bar, go to **All devices > Things**.



Then click **Create things**.



For this example, we are creating a single thing. Click next.



Then specify the Thing's name and click next.

Specify thing properties Info

A thing resource is a digital representation of a physical device or logical entity in AWS IoT. Your device or entity needs a thing resource in the registry to use AWS IoT features such as Device Shadows, events, jobs, and device management features.

Thing properties Info

Thing name

Enter a unique name containing only: letters, numbers, hyphens, colons, or underscores. A thing name can't contain any spaces.

Additional configurations

You can use these configurations to add detail that can help you to organize, manage, and search your things.

- ▶ **Thing type** - *optional*
- ▶ **Searchable thing attributes** - *optional*
- ▶ **Thing groups** - *optional*
- ▶ **Billing group** - *optional*
- ▶ **Packages and versions** - *optional*

Device Shadow Info

Device Shadows allow connected devices to sync states with AWS. You can also get, update, or delete the state information of this thing's shadow using either HTTPS or MQTT topics.

No shadow

Named shadow
Create multiple shadows with different names to manage access to properties, and logically group your devices properties.

Unnamed shadow (classic)
A thing can have only one unnamed shadow.

[Cancel](#)
[Next](#)

Leave the certificate option as default: Auto-generate a new certificate, then click next.

Configure device certificate - *optional* Info

A device requires a certificate to connect to AWS IoT. You can choose how to register a certificate for your device now, or you can create and register a certificate for your device later. Your device won't be able to connect to AWS IoT until it has an active certificate with an appropriate policy.

Device certificate

- Auto-generate a new certificate (recommended)
Generate a certificate, public key, and private key using AWS IoT's certificate authority.
- Use my certificate
Use a certificate signed by your own certificate authority.
- Upload CSR
Register your CA and use your own certificates on one or many devices.
- Skip creating a certificate at this time
You can create a certificate for this thing and attach a policy to the certificate at a later time.

Cancel Previous Next

If a policy was created beforehand for this IoT core Thing, it can be attached at this point. Otherwise, creating a policy will be covered after this section.

Attach policies to certificate - *optional* Info

AWS IoT policies grant or deny access to AWS IoT resources. Attaching policies to the device certificate applies this access to the device.

Policies (1)

[Create policy](#) 

Select up to 10 policies to attach to this certificate.

Filter policies

< 1 >
⚙️

	Name
<input type="checkbox"/>	TestPolicyMQTT

Cancel Previous Create thing

Upon creating the Thing, a window should pop up with downloading the certificates and keys. These are required to download and save locally on the computer.

Download certificates and keys

Download certificate and key files to install on your device so that it can connect to AWS.

Device certificate

You can activate the certificate now, or later. The certificate must be active for a device to connect to AWS IoT.

Device certificate
[REDACTED].pem.crt

[Deactivate certificate](#) [!\[\]\(ea8b2ffcc3f5bfa59cc6a40e7c545e82_img.jpg\) Download](#)

Key files

The key files are unique to this certificate and can't be downloaded after you leave this page. Download them now and save them in a secure place.

⚠ This is the only time you can download the key files for this certificate.

Public key file
[REDACTED]-public.pem.key

[!\[\]\(74a49162860860a93b8e684c08e4f6e0_img.jpg\) Download](#)

Private key file
[REDACTED]-private.pem.key

[!\[\]\(9b1a2b2c1d2c6e5be40888cb341beaf1_img.jpg\) Download](#)

Root CA certificates

Download the root CA certificate file that corresponds to the type of data endpoint and cipher suite you're using. You can also download the root CA certificates later.

Amazon trust services endpoint
RSA 2048 bit key: Amazon Root CA 1

[!\[\]\(987a086189953867080e0d8bee144d33_img.jpg\) Download](#)

Amazon trust services endpoint
ECC 256 bit key: Amazon Root CA 3

[!\[\]\(315db76506aedfdbddf57c693471624b_img.jpg\) Download](#)

If you don't see the root CA certificate that you need here, AWS IoT supports additional root CA certificates. These root CA certificates and others are available in our developer guides. [Learn more](#) 

[**Done**](#)

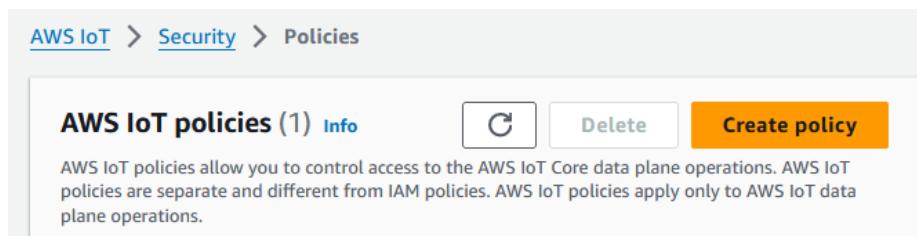
Creating an AWS IoT Core Thing Policy

This section will cover creating an IoT Core Thing policy to attach to a Thing certificate.

To create a policy, go to **Security > Policies**.

- ▼ Security
- Intro
 - Certificates
 - Policies
 - Certificate authorities
 - Certificate signing [New](#)
 - Role aliases
 - Authorizers
 - Audit
 - Detect

Then click **Create policy**.



Then specify a policy name and an effect that allows any device to access any IoT resource.

Create policy Info

AWS IoT Core policies allow you to manage access to the AWS IoT Core data plane operations.

Policy properties

AWS IoT Core supports named policies so that many identities can reference the same policy document.

Policy name

A policy name is an alphanumeric string that can also contain period (.), comma (,), hyphen(-), underscore (_), plus sign (+), equal sign (=), and at sign (@) characters, but no spaces.

▶ Tags - optional

Policy statements
Policy examples

Policy document Info

Builder
JSON

An AWS IoT policy contains one or more policy statements. Each policy statement contains actions, resources, and an effect that grants or denies the actions by the resources.

<p>Policy effect</p> <input style="width: 100%; height: 25px; border: 1px solid #ccc; padding: 2px;" type="text" value="Allow"/>	<p>Policy action</p> <input style="width: 100%; height: 25px; border: 1px solid #ccc; padding: 2px;" type="text" value="iot:*"/>
<p>Policy resource</p> <input style="width: 100%; height: 25px; border: 1px solid #ccc; padding: 2px;" type="text" value="*"/>	
<input style="border: 1px solid #ccc; padding: 2px 10px;" type="button" value="Remove"/>	
<input style="border: 1px solid #ccc; padding: 2px 10px;" type="button" value="Add new statement"/>	

Cancel
Create

The below **JSON** is the same as the above **Builder** policy effect.

Policy document [Info](#)
[Builder](#)
[JSON](#)

An AWS IoT policy contains one or more policy statements. Each policy statement contains actions, resources, and an effect that grants or denies the actions by the resources.

Policy document

```

1 [ { 
2   "Version": "2012-10-17",
3   "Statement": [
4     {
5       "Effect": "Allow",
6       "Action": "iot: *",
7       "Resource": "*"
8     }
9   ]
10 }
```

After creating the policy, find the newly created Thing and click its name.

Things (2) [Info](#)

An IoT thing is a representation and record of a physical or virtual device that needs a thing record in order to work with AWS IoT. A thing can be a sensor, a motor, a lightbulb, or anything else that can be controlled over the network.

<input type="text"/> Filter things by: name, type, created time	
<input type="checkbox"/>	Name
<input type="checkbox"/>	ExampleThingMQTT

Go to the **Certificates** tab and click the certificate name.

ExampleThingMQTT [Info](#)

Thing details

Name
ExampleThingMQTT

ARN
 [REDACTED]/ExampleThingMQTT

Attributes Certificates Thing groups Device Shad

Certificates (1) [Info](#)

The device certificates attached to this thing resource.

Find certificates

<input type="checkbox"/>	Certificate ID
<input type="checkbox"/>	6809dcf12a412e55790 [REDACTED]

Then attach a policy to the certificate. And that will cover the basics of creating an AWS IoT Thing.

Policies Things Noncompliance

Policies (0) [Info](#)

AWS IoT policies allow you to control access to the AWS IoT Core data plane operations.

[C](#) Detach policies Attach policies

Policies

Choose policies to attach to this certificate. The certificate can have up to 10 policies attached to it.

Choose AWS IoT policy

example

ExampleMQTTPolicy

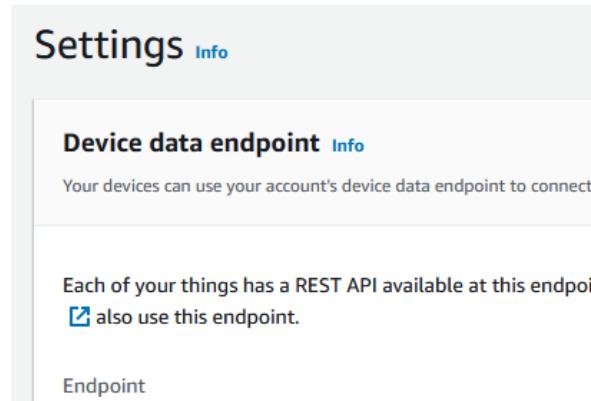
[Cancel](#) [Attach policies](#)

Connecting to AWS IoT Core Thing with EDM MQTT Client

There are several key values that will be needed to connect to a AWS IoT Thing from EDM MQTT client. Some of which can be found in the AWS IoT Settings page.

- Device software
- Billing groups
- Settings 
- Feature spotlight
- Documentation 

In the Settings page, the endpoint and TLS version can be found here and copied.



The screenshot shows the 'Settings' page with the 'Device data endpoint' section highlighted. The 'Device data endpoint' field contains the value 'Endpoint'. Below it, there is a note: 'Each of your things has a REST API available at this endpoint. [Also use this endpoint](#)'.

The TLS version specified in the security policy corresponds to TLS version 1.2. TLS15 corresponds to TLS version 1.5.

Select security policy | [Info](#)
[To customize your TLS settings](#)
[IoTSecurityPolicy_TLS12](#)

Then going to the EDM MQTT client, input the data into their respective fields. Another key value to know is the **port** for AWS is set to **8883**.

MQTT Client Setting

	Client setting	Sparkplug setting	Publish setting	TLS setting	Advanced setting	Messages
Broker IP	192.168.1.15				Select local IP	
Broker end point						.aws.amazonaws.com
Broker port	8883					
Communication timeout	5000	(ms)				
User name						
Password						
TLS version	TLS 1.2					
Protocol version	3.1.1					
Keep alive interval	60	(s)				
Clear session	<input checked="" type="checkbox"/>					
Topic prefix	EDM		The client will publish to [TopicPrefix]/App/Sys [TopicPrefix]/App/Test [TopicPrefix]/DSA/Test			
Connect to broker when App starts	<input type="checkbox"/>					

Then go to the **TLS setting** tab and browse for the file paths of the certificates downloaded when creating the AWS IoT Thing. After that, click **connect** in the EDM MQTT client and it should connect to the AWS IoT Thing.

MQTT Client Setting

	Client setting	Sparkplug setting	Publish setting	TLS setting	Advanced setting	Messages
CA certificate file	[REDACTED]-certificate.pem.crt			<input type="button" value="Browse"/>		
Client certificate file	[REDACTED]-certificate.pem.crt			<input type="button" value="Browse"/>		
Client private key file	[REDACTED]-private.pem.key			<input type="button" value="Browse"/>		

To see if the connection is working and data can be sent from EDM to AWS via MQTT, go to **Test > MQTT test client**. This client can be used to subscribe to various EDM topics that will receive data from EDM when it publishes it.

AWS IoT

- Monitor
- Connect
 - Connect one device
 - ▶ Connect many devices
- Test
 - ▶ Device Advisor
 - MQTT test client**
 - Device Location [New](#)
- Manage
 - ▶ All devices
 - ▶ Greengrass devices
 - ▶ LPWAN devices
 - Software packages [New](#)

[AWS IoT](#) > [MQTT test client](#)

MQTT test client Info

You can use the MQTT test client to monitor the MQTT messages being published by your device. You can also publish messages to inform devices and apps of changes and events. You can subscribe to topics to receive messages.

▶ **Connection details**

You can update the connection details by choosing Disconnect and making up a new connection.

Subscribe to a topic
Publish to a topic

Topic filter [Info](#)
 The topic filter describes the topic(s) to which you want to subscribe. The topic filter must begin with a slash (/).

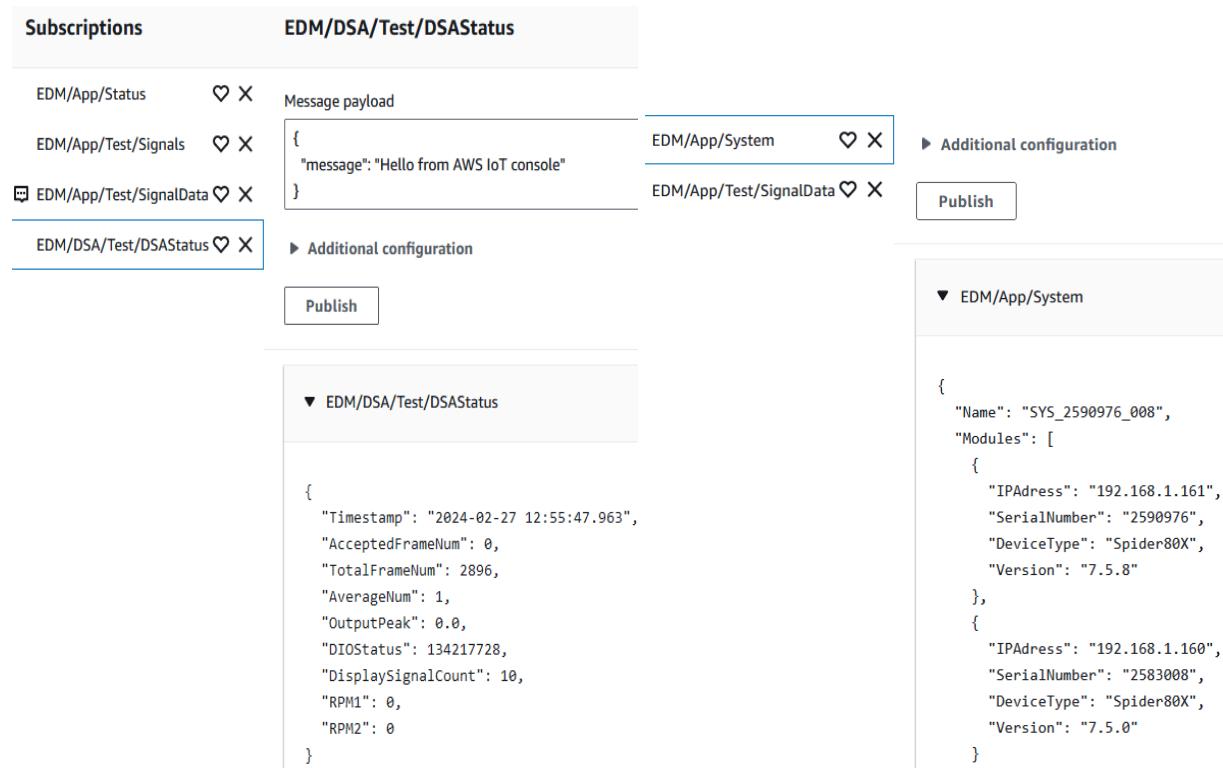
▶ **Additional configuration**

Subscribe

MQTT Client Setting

Client setting	Sparkplug setting	Publish setting	TLS setting	Advanced setting	Messages
<pre>2/27/2024 12:55:01 PM:SPIDER_DSA-e22da741-dc7e-4281-b1d6-a1efe2438756:EDM/App/Test/SignalData:< Payload Size=55090> 2/27/2024 12:55:01 PM:SPIDER_DSA-e22da741-dc7e-4281-b1d6-a1efe2438756:EDM/DSA/Test/DSAStatus:{"Timestamp":"2024-02-27 12:55:01.864", "AcceptedFrameNum":0, "TotalFrameNum":1743, "AverageNum":1, "OutputPeak":0.0, "DIOStatus":134217728, "DisplaySignalCount":10, "RPM1":0, "RPM2":0} 2/27/2024 12:55:02 PM:SPIDER_DSA-e22da741-dc7e-4281-b1d6-a1efe2438756:EDM/App/Test/SignalData:< Payload Size=55353> 2/27/2024 12:55:03 PM:SPIDER_DSA-e22da741-dc7e-4281-b1d6-a1efe2438756:EDM/DSA/Test/DSAStatus:{"Timestamp":"2024-02-27 12:55:02.903", "AcceptedFrameNum":0, "TotalFrameNum":1768, "AverageNum":1, "OutputPeak":0.0, "DIOStatus":134217728, "DisplaySignalCount":10, "RPM1":0, "RPM2":0} 2/27/2024 12:55:03 PM:SPIDER_DSA-e22da741-dc7e-4281-b1d6-a1efe2438756:EDM/App/Test/SignalData:< Payload Size=55438> 2/27/2024 12:55:04 PM:SPIDER_DSA-e22da741-dc7e-4281-b1d6-a1efe2438756:EDM/DSA/Test/DSAStatus:{"Timestamp":"2024-02-27 12:55:03.954", "AcceptedFrameNum":0, "TotalFrameNum":1795, "AverageNum":1, "OutputPeak":0.0, "DIOStatus":134217728, "DisplaySignalCount":10, "RPM1":0, "RPM2":0} 2/27/2024 12:55:04 PM:SPIDER_DSA-e22da741-dc7e-4281-b1d6-a1efe2438756:EDM/App/Test/SignalData:< Payload Size=55414> 2/27/2024 12:55:05 PM:SPIDER_DSA-e22da741-dc7e-4281-b1d6-a1efe2438756:EDM/DSA/Test/DSAStatus:{"Timestamp":"2024-02-27 12:55:04.966", "AcceptedFrameNum":0, "TotalFrameNum":1821, "AverageNum":1, "OutputPeak":0.0, "DIOStatus":134217728, "DisplaySignalCount":10, "RPM1":0, "RPM2":0} 2/27/2024 12:55:05 PM:SPIDER_DSA-e22da741-dc7e-4281-b1d6-a1efe2438756:EDM/App/Test/SignalData:< Payload Size=55389> 2/27/2024 12:55:06 PM:SPIDER_DSA-e22da741-dc7e-4281-b1d6-a1efe2438756:EDM/DSA/Test/DSAStatus:{"Timestamp":"2024-02-27 12:55:05.986", "AcceptedFrameNum":0, "TotalFrameNum":1846, "AverageNum":1, "OutputPeak":0.0, "DIOStatus":134217728, "DisplaySignalCount":10, "RPM1":0, "RPM2":0} 2/27/2024 12:55:06 PM:SPIDER_DSA-e22da741-dc7e-4281-b1d6-a1efe2438756:EDM/App/Test/SignalData:< Payload Size=55470> 2/27/2024 12:55:07 PM:SPIDER_DSA-e22da741-dc7e-4281-b1d6-a1efe2438756:EDM/DSA/Test/DSAStatus:{"Timestamp":"2024-02-27 12:55:07.014", "AcceptedFrameNum":0, "TotalFrameNum":1864, "AverageNum":1, "OutputPeak":0.0, "DIOStatus":134217728, "DisplaySignalCount":10, "RPM1":0, "RPM2":0} 2/27/2024 12:55:07 PM:SPIDER_DSA-e22da741-dc7e-4281-b1d6-a1efe2438756:EDM/App/Test/SignalData:< Payload Size=55275> 2/27/2024 12:55:08 PM:SPIDER_DSA-e22da741-dc7e-4281-b1d6-a1efe2438756:EDM/DSA/Test/DSAStatus:{"Timestamp":"2024-02-27 12:55:08.043", "AcceptedFrameNum":0, "TotalFrameNum":1897, "AverageNum":1, "OutputPeak":0.0, "DIOStatus":134217728, "DisplaySignalCount":10, "RPM1":0, "RPM2":0}</pre>					





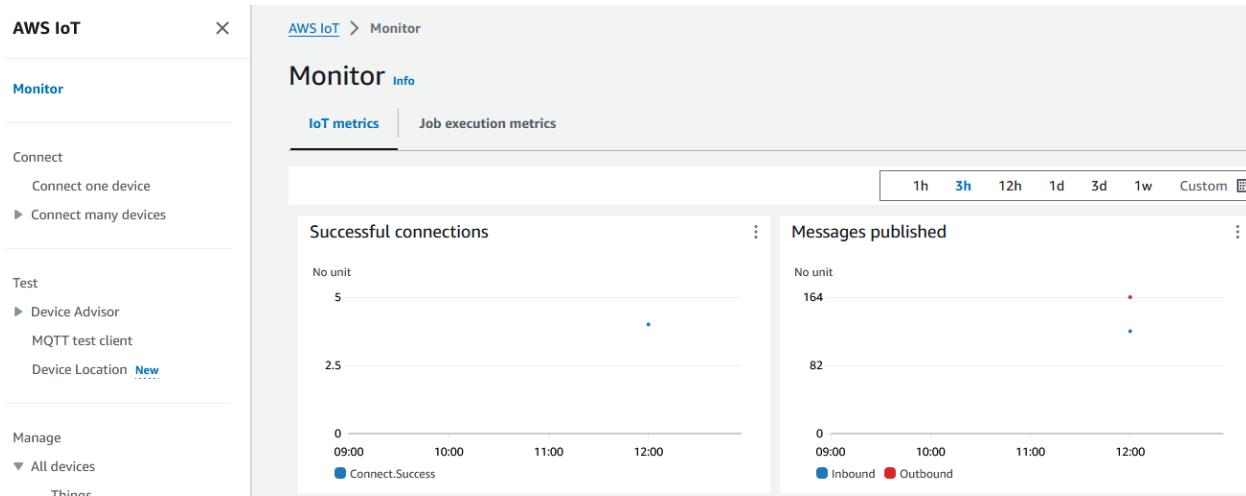
The screenshot shows the AWS IoT Device Management interface under the 'Subscriptions' section. A subscription named 'EDM/DSA/Test/DSASatus' is selected. The message payload is defined as:

```
{
  "message": "Hello from AWS IoT console"
}
```

Below the payload, there are 'Additional configuration' and 'Publish' buttons. To the right, a detailed view of the published message structure is shown:

```
{
  "Name": "SYS_2590976_008",
  "Modules": [
    {
      "IPAddr": "192.168.1.161",
      "SerialNumber": "2590976",
      "DeviceType": "Spider80X",
      "Version": "7.5.8"
    },
    {
      "IPAddr": "192.168.1.160",
      "SerialNumber": "2583008",
      "DeviceType": "Spider80X",
      "Version": "7.5.0"
    }
  ]
}
```

Another location that can be helpful is the Monitor page that will monitor the overall status of the AWS IoT Thing.



Publishing Payload

For MQTT Clients with custom coding to publish a command to EDM Broker.

Parameter Name	Type	Description
IsLevel0	bool	Determines the message
IsLevel1	bool	QoS level

IsLevel2	bool	
Level	MqttQualityOfServiceLevel	AtMostOnce AtLeastOnce ExactlyOnce
Name	string	
Payload	byte[]	Command and or value in UTF8 encoding bytes
Retain	bool	
Topic	string	Prefix and topic

Data example:

options	{EDM.MQTT.PublishOptionsModel}
↳ IsLevel0	true
↳ IsLevel1	false
↳ IsLevel2	false
↳ Level	AtMostOnce
↳ Name	null
↳ ↳ Payload	{byte[7]}
↳ Retain	false
↳ Topic	"EDM/VCS/Test/Command"

The model is later used in the MQTT Client publish method for the MQTT message builder class.

```
var applicationMessage = new MqttApplicationMessageBuilder()
    .WithTopic(options.Topic)
    .WithQualityOfServiceLevel(options.Level)
    .WithRetainFlag(options.Retain)
    .WithPayload(options.Payload)
    .Build();
```

Receiving Messages

EDM can send back messages with the **topic** and **JSON** in byte[] that can be deserialize as noted in the various topics below.

Commands generally do not return a message with JSON.

↳ e.ApplicationMessage	{MQTTnet.MqttApplicationMessage}
↳ ContentType	null
↳ CorrelationData	null
↳ Dup	false
↳ MessageExpiryInterval	null
↳ ↳ Payload	{byte[7]}
↳ PayloadFormatIndicator	null
↳ QualityOfServiceLevel	AtLeastOnce
↳ ResponseTopic	null
↳ Retain	false
↳ ↳ SubscriptionIdentifiers	null
↳ Topic	"EDM/VCS/Test/Command"
↳ TopicAlias	null
↳ ↳ UserProperties	null

```
public class MqttApplicationMessage
{
    public MqttApplicationMessage();

    public string Topic { get; set; }
    public byte[] Payload { get; set; }
    public MqttQualityOfServiceLevel QualityOfServiceLevel { get; set; }
    public bool Retain { get; set; }
    public bool Dup { get; set; }
    public List<MqttUserProperty> UserProperties { get; set; }
    public string ContentType { get; set; }
    public string ResponseTopic { get; set; }
    public MqttPayloadFormatIndicator? PayloadFormatIndicator { get; set; }
    public uint? MessageExpiryInterval { get; set; }
    public ushort? TopicAlias { get; set; }
    public byte[] CorrelationData { get; set; }
    public List<uint> SubscriptionIdentifiers { get; set; }
}
```

App Universal Topics

Messages are returned in JSON format.

App/Message

Subscribe to this topic to receive **text messages** and publish text messages through this topic.

App/Status

Publish **App status**, automatically publish with Retain Message after Client connects, including App type and version.

For example:

```
{
    "Timestamp": "2021-12-21 01:13:11",
    "SoftwareMode": "SPIDER_VCS",
    "Version": "10.0.8.10"
}
public struct MQTTAppStatus
{
    public string Timestamp;
    public string SoftwareMode;
    public string Version;
}
```

App/Error

Subscribe to this topic to receive string error messages that may occurred from EDM.

App/System

Publish **System status**, **Client connection** or **test connection switch**, is automatically published as Retain Message, including System name and Module contained in System.

For example:

```
{
    "Name": "SYS_2592064",
    "Modules":
```

```
[{
    "IPAdress": "192.168.10.14",
    "SerialNumber": "2592064",
    "DeviceType": "Spider80X",
    "Version": "7.5.8"
}]}
```

```
public struct MQTTSystem
{
    public string Name;
    public List<MQTTDeviceModule> Modules;
}

3 references
public struct MQTTDeviceModule
{
    public string IPAdress;
    public string SerialNumber;
    public string DeviceType;
    public string Version;
}
```

App/System/Status

Publish the **status of the current system**, whether it is **connected**, **disconnected**, **detected**, etc., including the System name and status.

For example:

```
{
    "Timestamp": "2021-12-21 01:13:11",
    "Name": "SYS_2592064",
    "Status": "Detected"
}
```

```
public struct MQTTSystemStatus
{
    public string Timestamp;
    public string Name;
    public string Status;
}
```

Status
NotDetected
Detected
Connected
Disconnected

App/System/TimeStatus

Payload message sent back via the **SetNTP** command that contains the spider device sync time to a NTP server.

```
public struct MQTTDeviceTimeStatus
{
    public string SerialNumber;
    public int Year;
    public int Month;
    public int Day;
    public int Hour;
    public int Minute;
    public int Second;
    public int Millisecond;
    public int Microsecond;
}
```

App/Test

After the test is created or loaded, it will actively publish the test topic message, including the test name and type.

For example:

```
{
    "Name": "Random31",
    "Type": "VCS_Random",
    "CreatedTime": "2021-12-21 01:13:11"
}
public struct MQTTTest
{
    public string Name;
    public string Type;
    public string CreatedTime;
}
```

App/Test/Status

When the test status changes, publish the **latest test status** with Retain Message.

For example:

```
{
    "Timestamp": "2021-12-21 01:13:11",
    "Name": "Random1",
    "RunFolder": "RunFolder36",
    "MeasureStartAt": "2021-12-21 01:12:35",
    "Status": "Running"
}
public struct MQTTTestStatus
{
    public string Timestamp;
    public string Name;
    public string Status;
    public string RunFolder;
    public string MeasureStartAt;
}
```

App/Test/Command

General test command, the built-in Client will automatically subscribe, and other Clients issue test commands.

The command format is plain text and the payload is encoded in UTF8 bytes with: cmd;params.
If there are multiple parameters, use a semicolon (;) to separate. The command is case-sensitive.

For specific software version commands such as VCS or DSA, they are listed in their respective topics.

Common Commands

Commands here should be identical to using the control panel in the EDM software.

Connect

Connects to the spider system specified in the current test.

Disconnect

Disconnects the connected spider system in the current test.

Run

Runs the current test. It will make EDM connect to the spider system if disconnected.

Pause

Pauses the current test.

Continue

Continues the current test if paused.

Stop

Stops the current test if running.

StartRecord

Starts recording the current test for all enabled input channels.

StopRecord

Stops the recording if it had started.

SaveSignals

Saves the current frame of signal.

ResetAverage

Resets the control waveform.

RequestTestStatus

Request the test status sent via **App/Test/Status**.

RequestDetailStatus

Requests the live status data of the current running test.

Returns certain topics such as **VCS/Test/RandomStatus**, **DSA/Test/DSASatus**, etc., and data that are similar to these:



Name	Value
Timestamp	2022-02-23 15:07:12.083
AccUnit	m/s ²
VelUnit	in/s
DisplUnit	in
EntryIndex	5
DrivePeak	0.395815789699554
TargetRMS	9.8128662109375
ControlRMS	9.73551273345947
Level	1
TotalTime	153.625
RemainTime	202.074996948242
FullLevelTime	97.9250030517578
VelPeak	1.0033996086421
DisplPeakPeak	0.00721887514860494
Kurtosis	3

RequestSignalData

Requests the live signal frame data.

Ex. CMD; sig[N];

“RequestSignalData;Ch1;Ch2;”

Returns **App/Test/SignalData** topic and the signal requested.

Parameter	Description
sig[N]	The Live Signal, where N is the number of signals that is in EDM. Multiple signal data can be obtained at the same time, each signal name, separated by semicolon. Ex: sig1, sig2, sig3, sigN sig1 = Time Stream Ch1 sig2 = Time Stream Ch2 etc.

RequestSignalDataWithFormat

Requests the live signal frame data scaled according to a display format with X axis, Y axis and spectrum type.

RequestSignalProperty

Requests a property from a live signal.

Ex. CMD; Property; sig[N];

“RequestSignalProperty;RMS;Ch1;Ch2;”

Returns **App/Test/SignalProperty** topic and the property and value requested.

Parameter	Description
prop	Property (prop) is: rms, peak, pkpk, min, One of max, mean, case-insensitive, prop followed by one or more corresponding value and unit
sig[N]	Signal names to request the property.

RequestReportFile

Requests the report file in the saved directory.

Ex. CMD; File Name Path;

“RequestReportFile;C:\[\...]\Documents\Random53 Default 15-33-59.docx;”

Parameter	Description
filename	The full path can be found from: Obtained from the information returned by App/test/ReportFile

RequestRunFolder

Requests the current run folder.

Returns **App/Test/RunFolder** topic and the run folder name, file path and list of file names in the folder and their file paths.

LoadTest

Loads a known test in EDM

Ex. CMD; Test Name;

“LoadTest;Random53;”

Parameter	Description
testname	The name of the test that users wants to load.

CreateTest

Creates a new test with name and type in EDM.

Only supports VCS test.

Ex. CMD; Test Name; Test Type;

“CreateTest;Random54;VCS_Random;”

Parameter	Description
name	The name of the test that users wants to create.
type	The type of the test that users wants to create.

DeleteTest

Deletes the known test in EDM

Ex. CMD; Test Name;

“DeleteTest;Random53;”

Parameter	Description
name	The name of the test that users wants to delete.

ListTest

Returns App/Test/List topic and the list of test names from EDM.

GenerateReport

Generates a report of the current test in EDM.

Ex. CMD; Report Template Name;

“GenerateReport;Input Channels;”

Parameter	Description
reportTemplateName	The report template name specified in EDM Toolbar -> Report

StartTestSequence

Starts the test sequence.

PauseTestSequence

Pause the test sequence.

ResumeTestSequence

Resume the test sequence.

StopTestSequence

Stops the test sequence.

NextTestSequence

Goes to the next test in the test sequence.

ListReportNotes

Get a list of report notes in a report template

SetReportNotes

Set the list of report notes in a report template

Ex. CMD;[{"Name": "Content"}, {"Name": "Content"}]

"SetReportNotes;[{"Name": "DUT", "Content": ""}, {"Name": "Serial number", "Content": "89056"}, {"Name": "Notes", "Content": "test"}]"

Parameter	Description
jsonStringArray	A array of name and content serialize in JSON.

App/Test/CommandEx

Similar to App/Test/Command, but this is for payloads serialized in JSON.

Specific software suites, such as VCS or DSA have their own CommandEx topics.

The payload is encoded in UTF8 bytes after serializing the following format:

```
public struct MQTTCommand
{
    public string Type;
    public Dictionary<string, string> Parameters;
}
```

For how to assign values to Parameters, please refer to PublishWithValue(object val) in the MQTT C# demo code.

Currently there are no general commands that uses App/Test/CommandEx. Otherwise, please refer to App/Test/Command for any commands.

App/Test/RecordStatus

When the test records or stops recording (when the record state changes), the **latest state of the record** is published in Retain Message, including the record name and record state.

For example:

```
{
    "Timestamp": "2021-12-21 01:13:11",
    "Name": "REC0055",
    "Status": "Started"
}
```

```
public struct MQTTRecordStatus
{
    public string Timestamp;
    public string Name;
    public string Status;
}
```

Status
Started
Stopped

App/Test/LimitStatus

When the test limit setting is **exceeded** or there is a system state that needs to be **alarmed**, the latest state of the record is released, and the alarm-related information.

For example:

```
{
    "Timestamp": "2021-12-21 01:13:11",
    "Name": "Random36",
    "LimitStatus": " RMS Higher than Abort"
}
```

```
public struct MQTTLimitStatus
{
    public string Timestamp;
    public string Name;
    public string Status;
}
```

Status Message

Status Type	Message Meaning
UserAbort	User Abort
SchAbort	Schedule Finished
Timeout	Time out
Overload	Overload
Paused	Paused

ICPDrop	ICP Disconnected
DSPResourceError	DSP Resource Error
SlaveError	Communication Error
NOTCHING	Notching Limit
HighAbortLine	Exceeds High Abort Line
LowAbortLine	Below Low Abort Line
RMSHighAbort	RMS Higher than Abort
RMSLowAbort	RMS Lower than Abort
HighAlarmLine	Exceeds High Alarm Line
LowAlarmLine	Below Low Alarm Line
RMSHighAlarm	RMS Higher than Alarm
RMSLowAlarm	RMS Lower than Alarm
HighControlLoss	High Control Level
LowControlLoss	Low Control Level
HighRMSChange	High RMS Change
LowRMSChange	Channel Lost
ExceedDriveLimit	Exceeds Drive Limit
SigmaClipping	Sigma clipping
ChannelLimit	Abort Limit
OpenLoop	Open Loop
SysClockFailed	System Clock Error
EvaluationStop	Evaluation Time Out
DispLimit	Approaching Displacement Limit
AlarmLimit	Alarm Limit
ControlChannelLost	Ctrl. Chnl. Lost
MonitorChnLost	Monitor Chnl. Lost
ResonanceLostSweep	Resonance Lost Sweep
EventStop	Event Stop
TimeLimit_Raw	Time signal limit exceeded
TimeLimit_RMS	Time signal RMS limit exceeded

App/Test/Channels

When the test is loaded or the **channel table** is changed, the latest status of the channel is published with the Retain Message.

For example (with only one channel):

```
[{
    "Module": "Ch1#SN: 2580672",
    "Enable": true,
    "LocationId": "Ch1",
    "Quantity": "Acceleration",
    "Unit": "g",
    "Sensitivity": 100.0,
    "InputMode": "AC-Single End",
    "ChannelType": "Control",
    "InputRange": "Auto",
    "HighPassFrequency": 0.5,
```

```
"Integration":"No Integration",
"ControlWeighting":"N/A"
},
{Ch2..},{Ch3..}]
```

```
public struct MQTTChannel
{
    public string Module;
    public bool Enable;
    public string LocationId;
    public string Quantity;
    public string Unit;
    public double Sensitivity;
    public string InputMode;
    public string ChannelType;
    public string InputRange;
    public double HighPassFrequency;
    public string Integration;
    public string ControlWeighting;
}
```

App/Test/ChannelStatus

At the request command of **RequestChannelStatus**, EDM sends data to this topic that contains the channel status. It can be automatically sent via EDM MQTT Client Advanced setting.

```
[{
    "Min": "-0.107002266",
    "Max": "0.137734428",
    "Peak": "0.137734428",
    ...
},
{Ch2..},{Ch3..}]
```

```

public struct MQTTChannelStatus
{
    public float Min;
    public float Max;
    public float Peak;
    public float RMS;
    public float Average;
    public float MaxPeak;
    public float MaxRms;
    public float VelPeak;
    public float DispPeak;
    public int MaxInputRange; //Max Input Range

    public bool IsIEPE; //IEPE enabled
    public bool IsOverload; //Channel over load
    public bool IsIEPEWithSensor; //IEPE enabled with sensor
    public bool IsStrainGageNotConnected;
    public bool IsSensorOverload; //Sensor over load
    public bool IsFailed; //Channel Failed

    public string Unit;
    public string LocationId;
    public string VelUnit;
    public string DispUnit;

    public string ChannelLost; //Channel lost
    public string LimitExceed; //Abort/Alarm limit exceed
    public string Notching; //Notching (For IEPE)
}

```

App/Test/Parameters

When the test is loaded or the **parameters** are **changed**, the latest status of all parameters is released with Retain Message. This can also be received via **ListParameters** command.

For example (only some sample parameters are included):

```
{
    "Vcs_Sine_Block": 3,
    "Vcs_Sine_Control": 0,
    "Vcs_Sine_DriveInit": 0.005,
    "Vcs_Sine_DriveLimit": 2.0,
    "Vcs_Sine_AdjustLevelStep": 10.0,
    ...
}
```

App/Test/Signals

When the test is loaded or the **signal configuration** is **changed**, the latest signal list is published with Retain Message.

For example (only Ch1 is included):

```
[{
    "Timestamp": "2022-01-26 16:32:52.520",
    "Name": "Ch1",
    "Type": "Equidistant",
    "UnitX": "ms",
    "UnitY": "g",
    "UnitZ": "",
    "BlockSize": 1024,
```

```

    "SamplingRate":0.0
},
{...},{...}]

```

```

public struct MQTTSignal
{
    public string Timestamp;
    public string Name;
    public string Type;
    public string UnitX;
    public string UnitY;
    public string UnitZ;
    public ulong BlockSize;
    public double SamplingRate;
    public string WindowType;
    public string DisplayFormat;
}

```

App/Test/List

After the client requests the test list data through **ListTest** Command, publishes the test list through this topic, and returns the test list (name, type).

For example:

```

[{
    "Name":"Random31",
    "Type":"VCS_Random",
    "CreatedTime": "2021-12-21 01:13:11"
},
{
    "Name":"Random51",
    "Type":"VCS_Random",
    "CreatedTime": "2021-12-21 01:13:11"
}]

```

App/Test/SignalData

After the client requests the **signal data** through the **RequestSignalData** Command, the signal data is published through the topic.

For example (only Ch1 is included, the data part is omitted for display, only the ending value is included):

```

[{
    "Signal":
    {
        "Timestamp":"2022-01-26 16:40:30.345",
        "Name":"Ch1",
        "Type":"Equidistant",
        "UnitX":"Time (ms)",
        "UnitY":"g",
        "UnitZ":"Label2",
        "BlockSize":1024,
        "SamplingRate":20479.999694824222
    }
}]

```

```

},
"ValueX": [507112829749.90686, {...}, 507112829799.85803],
"ValueY": [0.05019712820649147, {...}, 0.090758346021175385],
"ValueZ": [507112829.74990684]
}
]
```

```

public struct MQTTSignalFrameData
{
    public MQTTSignal Signal;

    public double[] ValueX;
    public double[] ValueY;
    public double[] ValueZ;

    public int XSequenceType; // 0-->linear, 1->log
    public double XStart;
    public double XDelta;
    public int XLength;
}
```

App/Test/SingleSignalData

The signal data with single precision.

```

public struct MQTTSignalFrameDataSingle
{
    public MQTTSignal Signal;

    public float[] ValueX;
    public float[] ValueY;
    public float[] ValueZ;

    public int XSequenceType; // 0-->linear, 1->log
    public float XStart;
    public float XDelta;
    public int XLength;
}
```

App/Test/CompressedSignalData

Compressed signal data that is published through this topic.

Shares the same struct as App/Test/SignalData topic.

When compressed data is sent, the X-axis data will no longer be sent in array form, but will be calculated based on the parameters XStart/XDelta/XLength/XSequenceType

App/Test/CompressedSingleSignalData

Compressed single precision signal data.

Shares the same struct as App/Test/SingleSignalData topic.

App/Test/ReportFile

After the client requests the **report data** through the **RequestReportFile** Command, the report data is published through the topic, and can also return the actual report content. The client needs to actively save the report.

For example:

```
{
    "ReportName": "D:\\SystemFolders\\Documents\\Sine2 Run2 16-45-29.docx",
    "ReportContent": "<base64 String>"
}
```

ReportName is the full path. If ReportContent is not empty, it is a string in Base64 format, which needs to be converted into binary for storage.

```
public struct MQTTReportFile
{
    public string ReportName;
    public byte[] ReportContent;
}
```

App/Test/ReportTemplates

An array of report templates that clients can subscribe to when connected to EDM and after test changes.

```
1 reference
public struct MQTTReportTemplates
{
    public string[] Templates;
}
```

App/Test/ReportNotes

The list of report notes that can be requested via **ListReportNotes** command.

```
1 reference
public struct MQTTReportNote
{
    public string Name;
    public string Content;
}
```

App/Test/RecordFile

After the client requests the **record data** through the **RequestRecordFile** Command, the record data is published through the topic and the client needs to actively save the record.

```
{
    "FileName": "SIG0002.atfx",
    "AtfxFileContent": "<base64 String>",
    "DataFiles": ["C:\\Users\\KevinCheng\\Documents\\EDM\\importapssignals2\\Random9\\Run1 Aug 03, 2022 17-02-38\\SIG0002-2590976.dat"],
    "DataFileContents": ["<base64 String>"]
}
```

FileName is the full path. If AtfxFileContent is not empty, it is a string in Base64 format, which needs to be converted into binary for storage and stored in atfx format. There may be one or more DataFiles, each of which needs to be stored in dat format.

```
public class MQTTRecordFile
{
    public string FileName;
    public byte[] AtfxFileContent;
    public List<string> DataFiles;
    public List<byte[]> DataFileContents;
}
```

App/Test/SignalProperty

After the client requests the **signal property** through the **RequestSignalProperty** Command, the signal property is published through the topic.

For example:

```
[{
    "SignalName": "Block(Ch1)",
    "PropertyName": "RMS",
    "Value": 0.11243738979101181,
    "Unit": "g"
}]
```

```
public struct MQTTSignalProperty
{
    public string SignalName;
    public string PropertyName;
    public double Value;
    public string Unit;
}
```

App/Test/RunFolder

After the client requests the run folder through **RequestRunFolder** Command, the run folder path and file name are published through the topic.

For example:

```
{
    "RunName": "Run2",
    "RunPath": "C:\\\\Users\\\\KevinCheng\\\\Documents\\\\EDM\\\\test\\\\Random53\\\\Run2 Feb 23, 2022 15-51-20",
    "RunFiles": {
        [
            [0]: "TimeHistory0122.atfx",
            [1]: "SIG0001.atfx"
        ],
    }
}
```

```
public class MQTTRunFolder
{
    public string RunName;
    public string RunPath;
    public List<string> RunFiles;
}
```

App/Test/AdvancedStatus

A topic that is returned from sending a command of RequestPeakFrequency or RequestPeakValue.

```
public struct MQTTAdvancedStatus
{
    public string Timestamp;
    public string Name;
    public string Type;
    public string Value;
    public string Unit;
}
```

Global Parameter Topics

GlobalParameter/Request

A request topic that the client can send commands to EDM to receive a list of global parameters or the global parameter data through the **GlobalParameter/Response** topic.

ListGlobalParameters

Request a list of known global parameters in the current EDM test.

[Global Parameter Key]

Request data regarding the global parameter from the current EDM test.

It is best to get an exact global parameter key for this command from **ListGlobalParameter** command.

GlobalParameter/Response

After the client requests the **global parameters** through the **ListGlobalParameters** Command, a list of the global parameters from EDM are published through the topic.

For example:

```
{
    "Item1": "ListGlobalParameters",
    "Item2":
    [
        "Test.CreatedAt",
        "Test.CreatedVersion",
        "Test.Description",
        ...
    ]
}
```

Here is the list of known global parameters that EDM can send.

For a more complete list of known global parameters for a specific test, please use the **GlobalParameter/Request** topic and the **ListGlobalParameter** command.

The client can also use the global parameter as commands through the **GlobalParameter/Request** topic to receive data from EDM about that parameter and follow a specific format:

Test.CreatedAt

Test.Run.Parameter.AnalysisFreq

Test.Run.Signal.APS(Ch1)

Etc.

Test

This contains basic data regarding a test creation, from creation date, spider system and so on.

Global Parameter	Parameter Meaning
CreatedAt	Test creation date Ex. 11/2/2022 3:47:23 PM
CreatedVersion	Test creation version Ex. 10.1.0.0
Description	Test description
LastRunTime	Test last run time
LastRunVersion	Test last run version
ModifiedAt	Test date last modification to configuration Ex. 11/7/2022 2:30:35 PM
Name	Test name
SpiderSystem	Test spider system
Type	Test type

Test.Run

This contains data regarding the run folders of the test.

Global Parameter	Parameter Meaning
Name	Test latest run folder name

Test.Run.Parameter

This contains data regarding the test configuration.

Some parameters may appear only for certain tests, such as TestSweepType for Sine tests.

Global Parameter	Parameter Meaning
ShakerAccelerationRMS	The following Shaker [...] parameters are from the test shaker settings in the test configuration.
ShakerArmatureMass	
ShakerForceRMS	
ShakerManufacturer	
ShakerMaxDriveFrequency	
ShakerMaxDriveVoltagePeak	
ShakerMaxNegativeDisplacement	
ShakerMaxPositiveDisplacement	
ShakerMaxVelocity	
ShakerMinDriveFrequency	
ShakerName	
ShakerShakerOrientation	
TestAverage	Test average
TestControlStrategy	Test input channel control strategy
TestDeltaFrequency	Test delta frequency
TestDOF	Test degree of freedom

TestDriveLimit	Test drive limit
TestFrequencyRange	Test frequency range
TestLines	Test signal plot point lines
TestOverlapRatio	Test overlap ratio
TestSigmaClipping	Test sigma clipping
VibrationSchedule	Test run schedule
TestShockAbortSensitivity	The following TestShock[...] parameters are from shock type tests and contain data from the test configuration.
TestShockAdjustLevelStep	
TestShockAverage	
TestShockDriveLimit	
TestShockPulseInternal	
TestBandwidth	Test bandwidth
TestMeasurementStrategy	Test measurement strategy
TestInitialDrive	Test initial drive
TestMaximumDriveDuringRampUpAndPreTest	Test max drive during ramp up and pretest
TestSignalPlotPoints	Test signal frame size
TestSweepType	Sine type test sweep type
AnalysisFreq	Test analysis frequency Ex. 12 kHz
Average Mode	Test average mode Ex. Exponential
Average Number	Test average number
Block Size	Test block size Ex. 1024 / 450
OverlapRatio	Test overlap ratio
TestFFTAverageOnOff	FFT test average on or off value Ex. Average off
TestFFTLPFCutoffFreq	FFT test LPF cutoff frequency
Window	Test window Ex. Hanning

Test.Run.NumericalValue

This contains data regarding data gathered during or after a test run.

Some parameters may appear only for certain tests, such as SweepCount for Sine tests.

Global Parameter	Parameter Meaning
CtrlRMS	Control RMS
DrivePk	Drive Peak
FullLevelElapsed	Time since full level elapsed
Level	Run level
Pk	Peak
PkPk	Peak Peak
Remaining	Remaining time left in test run
TargetRMS	Target RMS

TotalElapsed	Time total elapsed when test run
ControlPeak	Control Peak
LevelError	Test level error
NextDrivePk	Next drive peak
NextLevel	Next level
PulseWidth	Pulse width
RMS	RMS
TargetPeak	Target peak
Frequency	Test run frequency
SweepCount	Sine test sweep count
SweepSpeed	Sine test sweep speed

Test.Run.Signal

This contains data regarding the test signals.

There may be additional signals that a test may have depending if it is enabled in the test's measured signals config.

Global Parameter	Parameter Meaning
APS(Ch#)	APS channel signals
APS(drive)	APS drive channel signal
Block(Ch#)	Time block channel signals
Block(drive)	Time block drive channel signal
Ch#	Time stream channel signals
control(f)	Control signal
drive	Drive channel signal
H(f)	H signal
HighAbort(f)	High abort signal
HighAlarm(f)	High alarm signal
Hist_Control_RMS	History control RMS signal
Hist_Profile_RMS	History profile RMS signal
LowAbort(f)	Low abort signal
LowAlarm(f)	Low alarm signal
noise(f)	Noise signal
profile(f)	Profile signal

DSA Topics

DSA/Test/Command

DSA test specific commands, the command format is the same as the topic App/Test/Command.

TriggerOn

Turns on the Trigger.

TriggerOff

Turns off the Trigger.

OutputOn

Turns on the Output.

OutputOn

Turns on the Output.

SetOutputIndex

Sets the output channel to then send other related commands such as OutputOn, OutputOff and SetOutputParameters.

Ex. CMD; index;

“SetOutputIndex;0;”

Parameter	Description
index	Index of the output channel

SetOutputParameters

Sets the output channel parameters.

Ex. CMD; OutputType; parameter[N];

“SetOutputParameters;Sine;1;200;”

The supported output formats and corresponding parameters are as follows:

OutputType	P1	P2	P3	P4	P5
Sine	Amplitude	Frequency	Offset		
Triangle	Amplitude	Frequency			
Square	Amplitude	Frequency			
WhiteNoise	RMS				
PinkNoise	RMS				
DC	Amplitude				
Chirp	Amplitude	LowFrequency	HighFrequency	Percent	Period
SweptSine	Amplitude	LowFrequency	HighFrequency	Period	

If you set Sine, 200Hz, 1V, the payload is SetOutputParameters; Sine; 1; 200

If DC, 1V is set, the payload is SetOutputParameters;DC;1

Parameters that are not required at the end can be omitted, but cannot be separated by a parameter.

Parameter	Description
type	The output channel type
parameter	The various parameters related to the output channel type

LimitOn

Turns on the Limit.

LimitOff

Turns on the Limit.

SetParameter

Sets a value to a configurable parameter in EDM

Ex. CMD; Parameter Name; Value;

“SetParameter;Block Size;2048;”

Parameter	Description
name	The name of the parameter that users wants to change.
value	The value of the parameter that users wants to set.

RequestTriggerSettings

Request the trigger settings from DSA.

SetTriggerSettings

Set the trigger settings to DSA.

Ex. CMD; List of {Parameter name: Value} in JSON

SetTriggerSettings; {"TriggerMode":0,"TriggerSource":"Ch1","TriggerType":0,"DelayPoints":0,
"LowLevel":0.0,"HighLevel":0.0}

TriggerAccept

Accept the current trigger frame.

TriggerNext

Proceed to the next trigger frame.

TriggerReject

Reject the current trigger frame.

DSA/Test/CommandEx

DSA specific commands serialized in JSON. For further details, refer to App/Test/CommandEx.
For the list of commands that can be used for this topic, refer to DSA/Test/Command.

DSA/Test/DSASatus

After the client requests the **test data** through the **RequestDetailStatus** Command, if the test type is **DSA**, the test detailed status will be published through this topic. The detailed test status structure returned by different test types is different.

```
public struct MQTTDSATestStatus
{
    public string Timestamp;
    public int AcceptedFrameNum;
    public int TotalFrameNum;
    public int AverageNum;
    public double OutputPeak;
    public int DIOStatus;
    public int DisplaySignalCount;
    public int RPM1;
    public int RPM2;
}
```

DSA/Test/TriggerSettings

The trigger settings for requesting or setting DSA triggers.

```
public struct MQTTTriggerSettings
{
    public int TriggerMode;
    public string TriggerSource;
    public int TriggerType;
    public int DelayPoints;
    public float LowLevel;
    public float HighLevel;
}
```

DSA/Test/TriggerFrame

The requested trigger signal frame data from EDM when the trigger hits. The data is sent from EDM as **MQTTSignalFrameData** shown in [App/Test/SignalData](#).

VCS Topics

VCS/Test/Command

VCS test specific commands, the command format is the same as the topic App/Test/Command.

These commands work depending on the type of tests.

CheckOnly

Runs the Check Only test.

Proceed

Proceeds running the test if the pre-test works.

SaveHSignal

Saves the current signal when the command is sent.

SetLevel

Sets the level of the test.

Ex. CMD; Level Value;

“SetLevel;10;”

Parameter	Description
level	The level of the test that users wants to set.

LevelUp

Brings up the current level up by 5% or 1dB

LevelDown

Brings down the current level down by 5% or 1dB

RestoreLevel

Restores the level according to the run schedule.

NextEntry

Goes to the next entry of the run schedule.

ShowPretest
AbortChecksOn

Turns on the Abort Checks.

AbortChecksOff

Turns off the Abort Checks.

ScheduleClockTimerOn

Turns on the Run Schedule Clock.

ScheduleClockTimerOff

Turns off the Run Schedule Clock.

ClosedLoopControlOn

Turns on the Closed Loop Control.

ClosedLoopControlOff

Turns off the Closed Loop Control.

SetFrequency

Sets the frequency of the test.

Ex. CMD; Frequency Value;

“SetFrequency;10;”

Parameter	Description
freq	The frequency of the test that users wants to set.

SetPhase

Sets the phase of the test.

Ex. CMD; Phase Value;
“SetPhase;10;”

Parameter	Description
phase	The phase of the test that users wants to set.

HoldSweep

Holds the sweep level.

SweepUp

Increases the sweep level.

SweepDown

Decreases the sweep level.

ReleaseSweep

Resumes the sweep level.

IncreaseSpeed

Speeds up the sweep.

DecreaseSpeed

Slows down the sweep.

RoRBandsOn

Turns on certain amount of RoR Bands.

Ex. CMD; bit[N];

“RoRBandsOn;1;1;1;1;”

Parameter	Description
bit1; bit2; bit3; ...; bitN	1->open, 0 ->off, that is, 0;1;1;1 is similar to a string

RoRBandsOff

Turns off all RoR Bands.

SoRTonesOn

Turns on certain amount of SoR Tones.

Ex. CMD; bit[N];

“SoRBandsOn;1;1;1;1;”

Parameter	Description
bit1; bit2; bit3; ...; bitN	1->open, 0 ->off, that is, 0;1;1;1 is similar to a string

SoRTonesOff

Turns off all SoR Tones.

SoRTonesHoldSweep

Holds the sweep level.

SoRTonesReleaseSweep

Resumes the sweep level.

SoRTonesSweepUp

Increases the sweep level.

SoRTonesSweepDown

Decreases the sweep level.

InversePulseOn

Turns on the Inverse Pulse.

InversePulseOff

Turns off the Inverse Pulse.

SinglePulseOn

Turns on the Single Pulse.

SinglePulseOff

Turns off the Single Pulse.

OutputSinglePulse

Outputs the Single Pulse.

SetParameter

Sets a value to a configurable parameter in EDM

Ex. CMD; Parameter Name; Value;

“SetParameter;VCSGeneral_BlockSize;2048;”

Parameter	Description
name	The name of the parameter that users wants to change.
value	The value of the parameter that users wants to set.

ListParameters

Get the list of parameters in EDM

SetChannelTable

Sets the channel table with an imported csv file. The client must send the command and file content as bytes to EDM MQTT network.

Ex. CMD; File content;

Encoding.UTF8.GetBytes("SetChannelTable;{File.ReadAllText(C:\Users\test\Downloads\channel_table.csv")}")

SetRandomProfile

Sets the random profile table with an imported csv file. The client must send the command and file content as bytes to EDM MQTT network.

Ex. CMD; File content;

Encoding.UTF8.GetBytes("SetRandomProfile;{File.ReadAllText(C:\Users\test\Downloads\random_profile.csv")}")

SetSineProfile

Sets the sine profile table with an imported csv file. The client must send the command and file content as bytes to EDM MQTT network.

Ex. CMD; File content;

Encoding.UTF8.GetBytes("SetSineProfile;{File.ReadAllText(C:\Users\test\Downloads\sine_profile.csv")}")

SetShockProfile

Sets the shock profile table with an imported csv file. The client must send the command and file content as bytes to EDM MQTT network.

Ex. CMD; File content;

Encoding.UTF8.GetBytes("SetShockProfile;{File.ReadAllText(C:\Users\test\Downloads\shock_profile.csv")}")

SetSchedule

Sets the run schedule with an imported csv file. The client must send the command and file content as bytes to EDM MQTT network.

Ex. CMD; File content;

Encoding.UTF8.GetBytes("SetSchedule;{File.ReadAllText(C:\Users\test\Downloads\schedule.sch")}")

RequestPeakFrequency

Request peak frequency data from a source signal.

Ex. CMD; On or Off; Signal;

"RequestPeakFrequency;On;Spectrum(Ch1)"

RequestPeakValue

Request peak value data from a source signal.

Ex. CMD; On or Off; Signal;

"RequestPeakValue;On;Hist_Peak(Ch2)"

ShutdownPC

Request the connected PC where EDM MQTT Client is on to shutdown.

Please note that any EDM MQTT connected to the MQTT Server that will receive this command from a MQTT Client that issue it will shut down the PC.

Ex. CMD; Delay time in seconds

“ShutdownPC;10”

SetNTP

Set the NTP server that EDM uses to synchronize spider devices date and time.

Ex. CMD; NTP server; NTP Port; Sync interval;

“SetNTP;time.windows.com;123;10;”

SetInputRange

Set all of the listed input channel’s range to a specific range.

Ex. CMD; Range;

“SetInputRange;AutoRange;”

RequestChannelStatus

Get the channel status that returns **App/Test/ChannelStatus**.

SetSimpleRandomProfile

Set a Random profile for a VCS Random test.

Ex. CMD; List of {Parameter name: Value, List of entry row data {}} in JSON

```
SetSimpleRandomProfile;{"AccUnit":"g","Entries": [{"Frequency":20.0,"Profile":0.00273233,"HighAbort":6.0,"HighAlarm":3.0,"LowAbort":-6.0,"LowAlarm":-3.0}, {"Frequency":40.0,"Profile":0.00273233,"HighAbort":6.0,"HighAlarm":3.0,"LowAbort":-6.0,"LowAlarm":-3.0}]}]
```

```
public struct MQTTRandomSimpleProfile
{
    public string AccUnit;
    public List<MQTTRandomProfileEntry> Entries;
}

public struct MQTTRandomProfileEntry
{
    public float Frequency;
    public float Profile;

    public float HighAbort;
    public float HighAlarm;
    public float LowAbort;
    public float LowAlarm;
}
```

SetSimpleSineProfile

Set a Sine profile for a VCS Sine test.

Ex. CMD; List of {Parameter name: Value, List of entry row data {}} in JSON

```
SetSimpleSineProfile;{"AccUnit":"g","Entries": [{"Frequency":50.0,"Profile":1.0,"SegmentType":9,"HighAbort":6.0,"HighAlarm":3.0,"LowAbort":-6.0,"LowAlarm":-3.0}, {"Frequency":100.0,"Profile":1.0,"SegmentType":9,"HighAbort":6.0,"HighAlarm":3.0,"LowAbort":-6.0,"LowAlarm":-3.0}]}]
```

```

public struct MQTTSineSimpleProfile
{
    public string AccUnit;
    public List<MQTTSineProfileEntry> Entries;
}

public struct MQTTSineProfileEntry
{
    public float Frequency;
    public float Profile;

    public int SegmentType;

    public float HighAbort;
    public float HighAlarm;
    public float LowAbort;
    public float LowAlarm;
}

```

SetSimpleShockProfile

Set a Shock profile for a VCS Shock test.

Ex. CMD; List of {Parameter name: Value} in JSON

SetSimpleShockProfile;{"MainPulseType":0,"LimitType":0,"Amplitude":0.0,"PulseWidth":0.0,"CompensationType":0,"CompensationPulseType":5,"PrePulseHeight":0.0,"PostPulseHeight":0.0}

```

public struct MQTTShockSimpleProfile
{
    public int MainPulseType;
    public int LimitType;

    public float Amplitude;
    public float PulseWidth;

    public int CompensationType;
    public int CompensationPulseType;

    public float PrePulseHeight;
    public float PostPulseHeight;
}

```

RequestShakerParameters

Request the current test shaker parameters, which is received in **VCS/Test/Shaker**.

SetShakerParameters

Set the current test shaker parameters.

Ex. CMD; List of {Parameter name: Value} in JSON

SetShakerParameters;{"RandomForceRMS":444.92,"RandomMaxAcc":163.44416666667,"SineForcePeak":9812.176696,"SineMaxAcc":735.49875,"ShockForcePeak":444.92,"ShockMaxAcc":490.3325,"MaxPosDispl":0.00635,"MaxNegDispl":0.00635,"Orientation":"Vertical","MaxVelocity":1.778,"MaxDriveVolt":10.0,"MinDriveFreq":1.0,"MaxDriveFreq":2500.0,"Measurement

Noisy":false,"ArmatureDiameter":1.5,"ArmatureMass":0.2,"FixtureMass":0.0,"HeaderExpanderMass":0.0,"SlipTableMass":0.0,"DriveBarMass":0.0}

RequestMassParameters

Request the current test payload and fixture mass, which is received in **VCS/Test/Mass**.

SetMassParameters

Set the current test payload and fixture mass parameters.

Ex. CMD; { Payload mass, Fixture Mass } in JSON

SetMassParameters;{"UUTMass":0.0999989738902,"FixtureMass":0.0}

VCS/Test/CommandEx

VCS specific commands serialized in JSON. For further details, refer to App/Test/CommandEx. For the list of commands that can be used for this topic, refer to VCS/Test/Command.

VCS/Test/Stage

The VCS test's specific **Stage state**, that EDM publishes the latest state with Retain Message when the test state changes.

For example:

```
{
    "Timestamp": "2021-12-21 01:13:11",
    "Name": "Random1",
    "Stage": "Norm_Run"
}
```

```
public struct MQTTVCSTestStage
{
    public string Timestamp;
    public string Name;
    public string Stage;
}
```

Stage
BeforeStart
Pre_MeasureNoise
Pre_IncreaseDrive
Pre_AdjustDrive
Pre_CloseLoop
Pre_Finished
Pre_Failed
Pre_DisplayFinished
Pre_DisplayFailed
Norm_Run
Norm_Pause
Norm_Stop

VCS/Test/ControlUpdated

When the command in VCS/Test/Command is executed and the **status update** needs to be returned, the built-in Client proxy actively publishes the message of the topic, and the client can subscribe to the message to get the push of the command execution result.

For example:

```
{
    "Timestamp": "2021-12-21 01:13:11",
    "Name": "Random1",
    "Flag": "VCS_AbortCheckOff"
}
```

```
public struct MQTTVCSControlFlag
{
    public string Timestamp;
    public string Name;
    public string Flag;
}
```

VCS/Test/Shaker

The test shaker data can be requested through this topic.

```
public struct MQTTShakerData
{
    public double RandomForceRMS;
    public double RandomMaxAcc;
    public double SineForcePeak;
    public double SineMaxAcc;
    public double ShockForcePeak;
    public double ShockMaxAcc;

    public double MaxPosDispl;
    public double MaxNegDispl;

    public String Orientation;
    public double MaxVelocity;
    public double MaxDriveVolt;
    public double MinDriveFreq;
    public double MaxDriveFreq;

    public bool MeasurementNoisy;
    public double ArmatureDiameter;
    public double ArmatureMass;
    public double FixtureMass;

    ///<summary>
    /// Header expander mass, vertical only.
    ///</summary>
    public double HeaderExpanderMass;
    ///<summary>
    /// Slip table mass, horizontal only.
    ///</summary>
    public double SlipTableMass;
    ///<summary>
    /// Drive bar mass, horizontal only.
    ///</summary>
    public double DriveBarMass;
}
```

VCS/Test/Mass

The payload and fixture mass parameters that is part of the shaker parameters configuration.

```
public struct MQTTMassParameters
{
    public double UUTMass;
    public double FixtureMass;
}
```

VCS/Test/RandomStatus

After the client requests the **test data** through the **RequestDetailStatus** Command, if the test type is **Random**, the test detailed status will be published through this topic. The detailed test status structure returned by different test types is different.

```
public struct MQTTRandomTestStatus
{
    public string Timestamp;
    public string AccUnit;
    public string VelUnit;
    public string DisplUnit;

    public int EntryIndex;

    public double DrivePeak;

    public double TargetRMS;
    public double ControlRMS;
    public double Level;

    public double TotalTime;
    public double RemainTime;
    public double FullLevelTime;

    public double VelPeak;
    public double DisplPeakPeak;

    public double Kurtosis;
}
```

VCS/Test/SineStatus

After the client requests the **test data** through the **RequestDetailStatus** Command, if the test type is **Sine**, the test detailed status will be published through this topic and returned to Json. The detailed test status structure returned by different test types is different.

```
public struct MQTTsineTestStatus
{
    public string Timestamp;
    public string AccUnit;
    public string VelUnit;
    public string DisplUnit;

    public int EntryIndex;

    public double DrivePeak;

    public double TargetPeak;
    public double ControlPeak;
    public double Level;

    public double TotalTime;
    public double RemainTime;
    public double FullLevelTime;

    public double VelPeak;
    public double DisplPeakPeak;
    public double TotalCycle;
    public double ElapsedCycle;

    public double SampleRate;
    public double Frequency;

    public double SweepRate;
    public int SweepNum;
    public int SweepDirection;
    public int SweepIndex;
}
```

VCS/Test/ShockStatus

After the client requests the **test data** through the **RequestDetailStatus** Command, if the test type is **Shock**, the test detailed status will be published through this topic and returned to Json. The detailed test status structure returned by different test types is different.

```
public struct MQTTShockTestStatus
{
    public string Timestamp;
    public string AccUnit;
    public string VelUnit;
    public string DisplUnit;

    public int EntryIndex;

    public double DrivePeak;

    public double TargetPeak;
    public double ControlPeak;
    public double RMS;

    public double Level;

    public int TotalPulse;
    public int RemainPulse;
    public int FullLevelPulse;

    public double VelPeak;
    public double DisplPeakPeak;
}
```

VCS/Test/TWRStatus

After the client requests the **test data** through the **RequestDetailStatus** Command, if the test type is **TWR**, the test detailed status will be published through this topic and returned to Json. The detailed test status structure returned by different test types is different.

```
public struct MQTWTWRTTestStatus
{
    public string Timestamp;
    public string AccUnit;
    public string VelUnit;
    public string DisplUnit;

    public int EntryIndex;

    public double DrivePeak;

    public double TargetRMS;
    public double ControlRMS;

    public double Level;

    public double TotalTime;
    public double RemainTime;
    public double FullLevelTime;

    public double VelPeak;
    public double DispPeakPeak;

    public int OutputIndex;

    public int TotalRepeat;
    public int CurRepeat;
}
```

VCS/Test/MIMORandomStatus

After the client requests the **test data** through the **RequestDetailStatus** Command, if the test type is **MIMO Random**, the test detailed status will be published through this topic and returned to Json. The detailed test status structure returned by different test types is different.

```
public struct MQTTMIMORandomTestStatus
{
    public string Timestamp;
    public string AccUnit;
    public string VelUnit;
    public string DisplUnit;

    public int EntryIndex;

    public double[] DrivePeaks;

    public double[] TargetRMSS;
    public double[] ControlRMSS;
    public double Level;

    public double TotalTime;
    public double RemainTime;
    public double FullLevelTime;

    public double[] VelPeaks;
    public double[] DisplPeakPeaks;

    public double Kurtosis;
}
```

VCS/Test/MIMOSineStatus

After the client requests the **test data** through the **RequestDetailStatus** Command, if the test type is **MIMO Sine**, the test detailed status will be published through this topic and returned to Json. The detailed test status structure returned by different test types is different.

```
public struct MQTTMIMOSineTestStatus
{
    public string Timestamp;
    public string AccUnit;
    public string VelUnit;
    public string DisplUnit;

    public int EntryIndex;

    public double[] DrivePeaks;

    public double[] TargetPeaks;
    public double[] ControlPeaks;
    public double Level;

    public double TotalTime;
    public double RemainTime;
    public double FullLevelTime;

    public double[] VelPeaks;
    public double[] DisplPeakPeaks;
    public double TotalCycle;
    public double ElapsedCycle;

    public double SampleRate;
    public double Frequency;

    public double SweepRate;
    public int SweepNum;
    public int SweepDirection;
    public int SweepIndex;
}
```

VCS/Test/MIMOSHockStatus

After the client requests the **test data** through the **RequestDetailStatus** Command, if the test type is **MIMO Shock**, the test detailed status will be published through this topic and returned to Json. The detailed test status structure returned by different test types is different.

```

public struct MQTTMIMOShockTestStatus
{
    public string Timestamp;
    public string AccUnit;
    public string VelUnit;
    public string DisplUnit;

    public int EntryIndex;

    public double[] DrivePeaks;

    public double[] TargetPeaks;
    public double[] ControlPeaks;
    public double RMS;

    public double Level;

    public int TotalPulse;
    public int RemainPulse;
    public int FullLevelPulse;

    public double[] VelPeaks;
    public double[] DisplPeakPeaks;
}

```

VCS/Test/MIMOTWRStatus

After the client requests the **test data** through the **RequestDetailStatus** Command, if the test type is **MIMO TWR**, the test detailed status will be published through this topic and returned to Json. The detailed test status structure returned by different test types is different.

```

public struct MQTTMIMOTWRTestStatus
{
    public string Timestamp;
    public string AccUnit;
    public string VelUnit;
    public string DisplUnit;

    public int EntryIndex;

    public double[] DrivePeaks;

    public double[] TargetRMSS;
    public double[] ControlRMSS;

    public double Level;

    public double TotalTime;
    public double RemainTime;
    public double FullLevelTime;

    public double[] VelPeaks;
    public double[] DisplPeakPeaks;

    public int OutputIndex;

    public int TotalRepeat;
    public int CurRepeat;
}

```

THV Topic

THV/Test/THStatus

After the client requests the **test data** through the **RequestDetailStatus** Command, if the test type is **VCS Test + TH**, the test detailed status will be published through this topic and returned to Json. The detailed test status structure returned by different test types is different.

```
public struct MQTTHStatus
{
    public string Timestamp;
    public double TotalTime;
    public double RemainTime;
    public double TargetTemperature;
    public double ControlTemperature;
    public double TargetHumidity;
    public double ControlHumidity;
    public List<MQTTTemperatureStatus> LatestTemperatures;
    public List<MQTTHumditiyStatus> LatestHumdities;
}

public struct MQTTTemperatureStatus
{
    public string Name;
    public double Temperature;
    public string Unit;
}

2 references
public struct MQTTHumditiyStatus
{
    public string Name;
    public double Humdity;
    public string Unit;
}
```

THV/Test/VibStatus

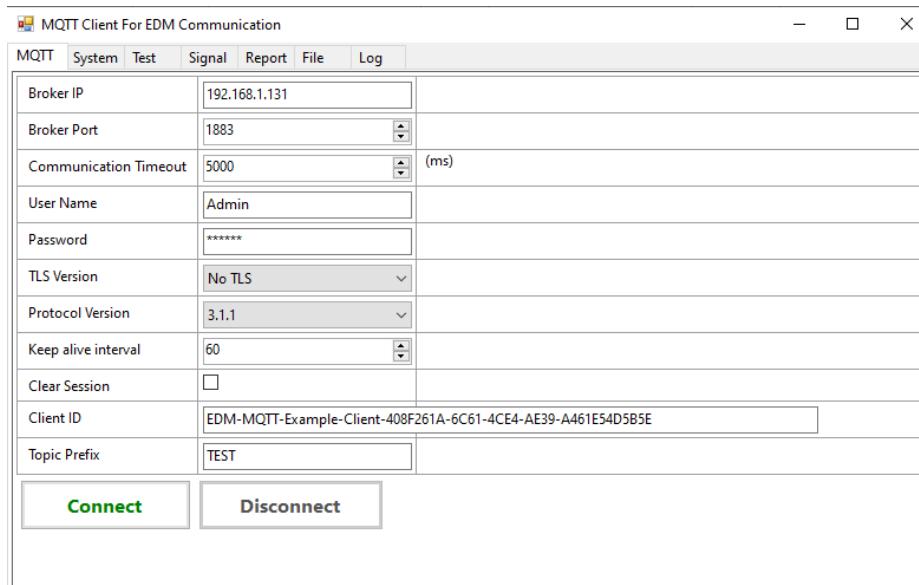
After the client requests the **test data** through the **RequestDetailStatus** Command, if the test type is **HH**, the test detailed status will be published through this topic and returned to Json. The detailed test status structure returned by different test types is different.

```
public struct MQTTVibStatus
{
    public string Timestamp;
    public double TotalTime;
    public double RemainTime;
    public double TargetVibration;
    public double ControlVibration;
    public List<MQTTVibrationStatus> LatestVibrations;
}

public struct MQTTVibrationStatus
{
    public string Name;
    public double Vibration;
    public string Unit;
}
```

EDM MQTT Client Demo Program

Crystal Instrument developed a MQTT Client Demo Program written in various coding languages that showcase how the MQTT protocol works and using the various EDM Topics. The demo program can be used with an EDM application, such as VCS or DSA, to control the test status and view the test data.



MQTT Client For EDM Communication

MQTT System Test Signal Report File Log

Status Command Detail Status Channels Parameters List/Create/Load/Delete TH Status Output

Connect Disconnect

Run Pause Continue Stop

Start Record Stop Record Save Signals

Execute VCS Command

Check Only	Proceed	Show Pretest	Save H Signal	Reset Average	Next Entry
Abort Check On	Abort Check Off	Closed Loop Ctrl On	Closed Loop Ctrl Off	Schedule Clock Timer On	Schedule Clock Timer Off
0;1;0;1	<input checked="" type="checkbox"/> ROR Board Band On/Off	RoR Bands On	RoR Bands Off		
0;1;0;1	<input checked="" type="checkbox"/> SOR Board Band On/Off	SoR Tones On	SoR Tones Off		
Tones Hold Sweep	Tones Release Sweep	Tones Sweep Up	Tones Sweep Down		
Sweep Up	Sweep Down	Hold Sweep	Release Sweep		
Level Up	Level Down	0	Set Level	Restore Level	
Increase Speed	Decrease Speed	0	Set Frequency	0	Set Phase
Inverse Pulse On	Inverse Pulse Off	Single Pulse On	Single Pulse Off	Output Single Pulse	

Execute DSA Command

Trigger On	Trigger Off	Output On	Output Off
Limit On	Limit Off	Reset Average	

Figure 1

Control Composite (Ch1)

LogMag g²/Hz Crt. RMS: 0.2406 g Target RMS: 1.0096 g

Signal Frame Data of Ch1

Time 675 700 725 750 775 800 825 850 875

(APS(Ch1))

LogMag g²/Hz APS(Ch1) RMS: 0.2406 g (0.00Hz ~ 2560.00Hz)

Drive Pk (V): 25.0 Target RMS (g): 0.2409 Elapsed/Total: 5:29 Total elapsed: 0:00 00:00:30 Pk (m/s): 0.234

MQTT Client Setting

Client Setting Sparkplug Setting Publish Setting Advanced Setting Messages

Source Node ID

Broker Setting Connected Clients Broker Log

MQTT Broker Setting

Broker ID Client ID User Name Protocol Version End Point

Random test Random No Python-MQTT-Example-Client Admin 192.168.1.123:43399 192.168.1.123:53337

python subscriber Admin V311 192.168.1.123:53338

SPIDER_VCS-6b5b4c20-9a3a-4311-925.. Admin V311 192.168.1.123:53949

Publish Text Message Connect Disconnect Clear Log Apply Advanced Settings Close

Test running ... Below Low Alarm Line

EDM MQTT Client Demo Package

The demo program, along with its source code can be distributed from our sales team or GitHub page.

<https://github.com/Crystal-Instruments/CI-EDM-MQTTClientDemo>

Demo Package Content

Crystal Instrument will provide a zip file that contains the following:

1. EDM MQTT Client Demo C# Program & Code
2. EDM MQTT Client Demo Python Scripts
3. EDM MQTT Client Demo LabVIEW
4. EDM MQTT Client Demo Java code
5. EDM MQTT Manual

How to Install the EDM MQTT Client Demo Program

If the distributed MQTT Client Demo Program comes in a zip file, then extract the zip file into a suitable location on the computer. Or if it comes in a software installer exe file, then run it and follow the steps to download the MQTT Client Demo Program. After the MQTT Client Demo Program has been installed, there should be four folders following the Demo Package Content section.

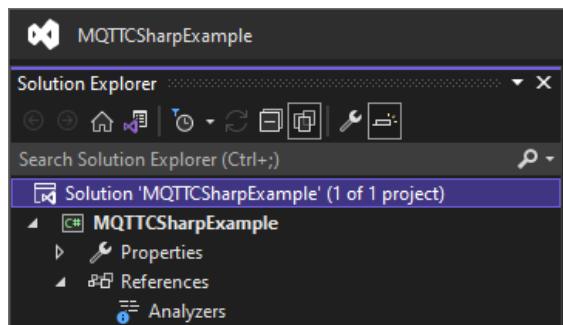
There are several demo coding languages that import the corresponding MQTT package and the manual folder.

For C#, it uses a modified MQTTnet package while the other packages can be reinstalled via nuget. Thus, please refer to the MQTTnet reference to the MQTTnet packaged that should be placed in the bin folder of the C# demo program.

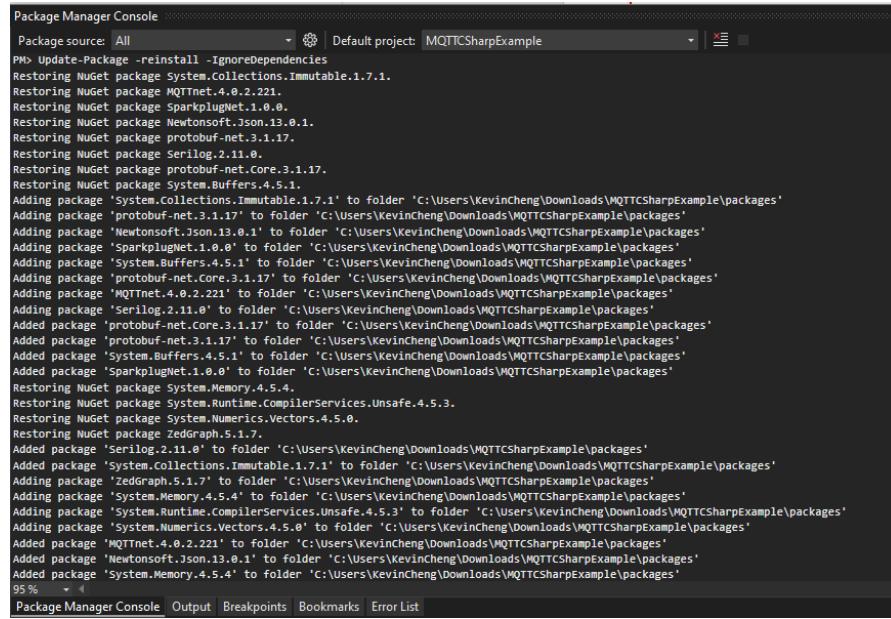
EDM MQTT Client C# Demo Showcase

How to Build the MQTT Client C# Demo

The MQTT Client C# Demo folder comes with a .csproj that can be opened with visual studio. The highlighted solution file in the Solution Explorer should be saved for the nuget package manager to work.



After the solution has been saved, go to Tools > Nuget Package Manager > Package Manager Console. Then enter Update-Package -reinstall -IgnoreDependencies.

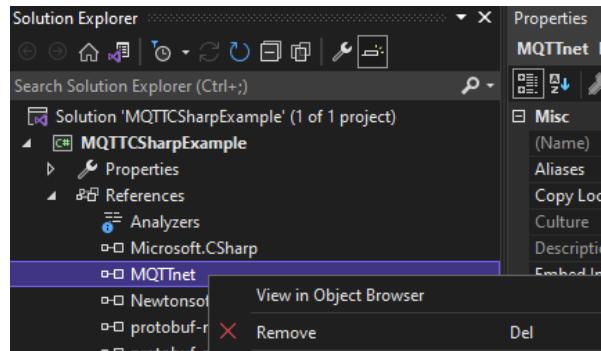


```

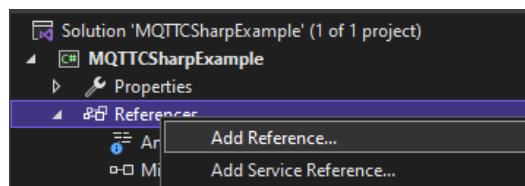
Package Manager Console
PM> Update-Package -reinstall -IgnoreDependencies
Restoring NuGet package System.collections.Immutable.1.7.1...
Restoring NuGet package MQTTnet.4.0.2.221...
Restoring NuGet package SparkplugNet.1.0.0...
Restoring NuGet package Newtonsoft.Json.13.0.1...
Restoring NuGet package protobuf-net.3.1.17...
Restoring NuGet package Serilog.2.11.0...
Restoring NuGet package protobuf-net.core.3.1.17...
Restoring NuGet package System.Buffers.4.5.1...
Adding package 'System.Collections.Immutable.1.7.1' to folder 'C:\Users\KevinCheng\Downloads\MQTTCSHarpExample\packages'
Adding package 'protobuf-net.3.1.17' to folder 'C:\Users\KevinCheng\Downloads\MQTTCSHarpExample\packages'
Adding package 'Newtonsoft.Json.13.0.1' to folder 'C:\Users\KevinCheng\Downloads\MQTTCSHarpExample\packages'
Adding package 'SparkplugNet.1.0.0' to folder 'C:\Users\KevinCheng\Downloads\MQTTCSHarpExample\packages'
Adding package 'protobuf-net.core.3.1.17' to folder 'C:\Users\KevinCheng\Downloads\MQTTCSHarpExample\packages'
Adding package 'MQTTnet.4.0.2.221' to folder 'C:\Users\KevinCheng\Downloads\MQTTCSHarpExample\packages'
Adding package 'Serilog.2.11.0' to folder 'C:\Users\KevinCheng\Downloads\MQTTCSHarpExample\packages'
Added package 'protobuf-net.3.1.17' to folder 'C:\Users\KevinCheng\Downloads\MQTTCSHarpExample\packages'
Added package 'protobuf-net.3.1.17' to folder 'C:\Users\KevinCheng\Downloads\MQTTCSHarpExample\packages'
Added package 'System.Buffers.4.5.1' to folder 'C:\Users\KevinCheng\Downloads\MQTTCSHarpExample\packages'
Added package 'SparkplugNet.1.0.0' to folder 'C:\Users\KevinCheng\Downloads\MQTTCSHarpExample\packages'
Restoring NuGet package System.Memory.4.5.4...
Restoring NuGet package System.Runtime.CompilerServices.Unsafe.4.5.3...
Restoring NuGet package System.Numerics.Vectors.4.5.0...
Restoring NuGet package ZedGraph.5.1.7...
Added package 'Serilog.2.11.0' to folder 'C:\Users\KevinCheng\Downloads\MQTTCSHarpExample\packages'
Added package 'System.Collections.Immutable.1.7.1' to folder 'C:\Users\KevinCheng\Downloads\MQTTCSHarpExample\packages'
Adding package 'ZedGraph.5.1.7' to folder 'C:\Users\KevinCheng\Downloads\MQTTCSHarpExample\packages'
Added package 'System.Memory.4.5.4' to folder 'C:\Users\KevinCheng\Downloads\MQTTCSHarpExample\packages'
Adding package 'System.Runtime.CompilerServices.Unsafe.4.5.3' to folder 'C:\Users\KevinCheng\Downloads\MQTTCSHarpExample\packages'
Adding package 'System.Numerics.Vectors.4.5.0' to folder 'C:\Users\KevinCheng\Downloads\MQTTCSHarpExample\packages'
Added package 'MQTTnet.4.0.2.221' to folder 'C:\Users\KevinCheng\Downloads\MQTTCSHarpExample\packages'
Added package 'Newtonsoft.Json.13.0.1' to folder 'C:\Users\KevinCheng\Downloads\MQTTCSHarpExample\packages'
Added package 'System.Memory.4.5.4' to folder 'C:\Users\KevinCheng\Downloads\MQTTCSHarpExample\packages'
95% < 

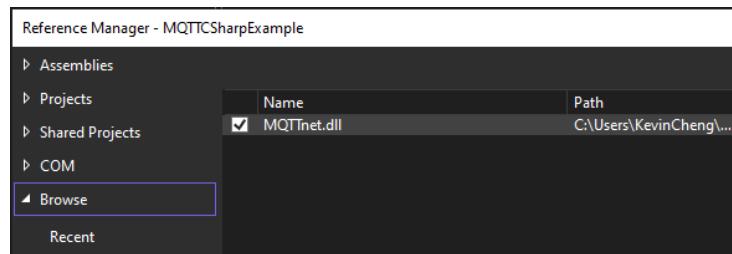
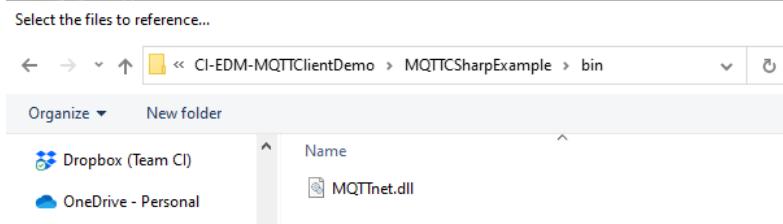
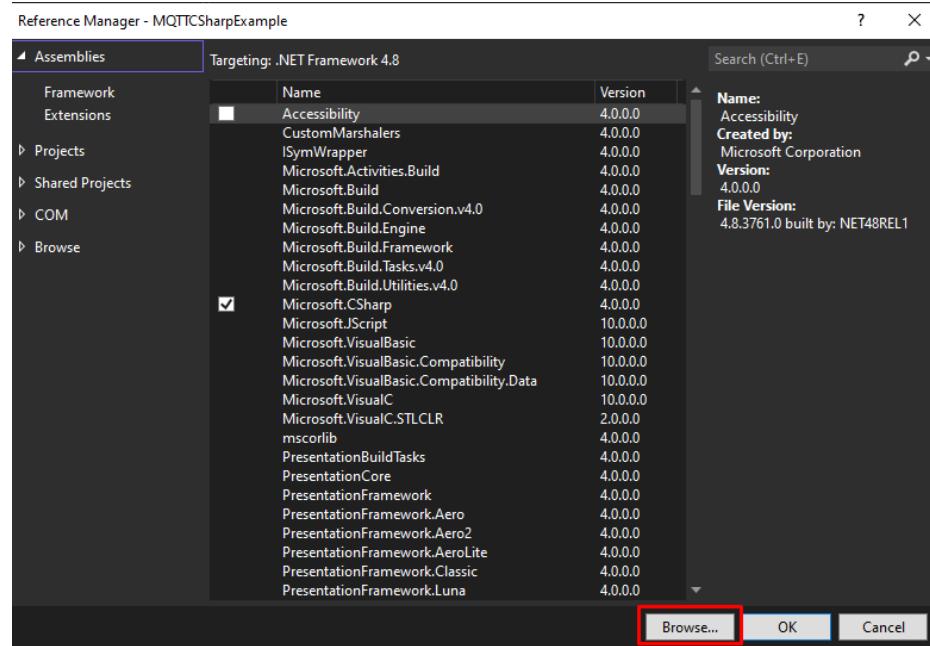
```

The package manager will proceed to reinstall all nuget packages used in the MQTT C# demo. Then once the manager is finished, go to Solution Explorer > Expand Reference > Remove MQTTnet.



Then right click References > Add Reference > Browse... > Select MQTTnet.dll in the C# demo bin folder > OK.





Now the project can be built.

C# Program GUI

Starting screen when the demo program loads up. This is the MQTT Client Demo Program connection parameter settings.

MQTT		System	Test	Signal	Report	File	Log	Debug
Broker IP	192.168.1.23	<u>Select local IP ↓</u>						
Broker Port	1883							
Communication Timeout	5000	(ms)						
User Name	Admin							
Password	*****							
TLS Version	No TLS							
CA certificate file								<input type="button" value="Browse"/>
Client certificate file								<input type="button" value="Browse"/>
Client private key file								<input type="button" value="Browse"/>
Protocol Version	3.1.1							
Keep alive interval	5000							
Clear Session	<input type="checkbox"/>							
Client ID	EDM-MQTT-Example-Client-6C840CEF-67A1-4BEE-A88E-979A0A4DF3E0							
Topic Prefix	EDM							
<input type="button" value="Connect"/>		<input type="button" value="Disconnect"/>						

System

Where it displays data about the EDM application and the connected spider system.

MQTT		System	Test	Signal	Report	File	Log												
App Information <table border="1"> <tr> <td>Software Mode</td> <td>SPIDER_DSA</td> </tr> <tr> <td>Version</td> <td>10.0.8.54</td> </tr> </table>								Software Mode	SPIDER_DSA	Version	10.0.8.54								
Software Mode	SPIDER_DSA																		
Version	10.0.8.54																		
System Information <table border="1"> <tr> <td>Name</td> <td>SYS_2590976</td> </tr> <tr> <td>Status</td> <td>Connected</td> </tr> <tr> <td>Device Type</td> <td>Spider80X</td> <td>Serial Number</td> <td>2590976</td> <td>IP Address</td> <td>192.168.3.151</td> <td>Hardware Version</td> <td>7.5.8</td> </tr> </table>								Name	SYS_2590976	Status	Connected	Device Type	Spider80X	Serial Number	2590976	IP Address	192.168.3.151	Hardware Version	7.5.8
Name	SYS_2590976																		
Status	Connected																		
Device Type	Spider80X	Serial Number	2590976	IP Address	192.168.3.151	Hardware Version	7.5.8												

Test

Where it offers more sub tabs that goes into controlling the test and displaying the data. For the Test's Status tab, it shows the current test information and depending on the application show other type of data.

Test Information		Limit Status	
Name	FFTOffice41	Name	<Name>
Type	CSA_FFT	Status	<Status>
Status	Stopped		
Run Folder	Run(17)		
Measure Start At	2025-04-09 11:01:04.000		

VCS Test Stage		VCS Control Flag	
Name	<Name>	Name	<Name>
Stage	<Stage>	Flag	<Stage>

Record Status	
Name	<Name>
Status	<Status>

Test's Command

Where it offers similar commands to EDM Control Panel. Here it shows all possible commands for various test types.

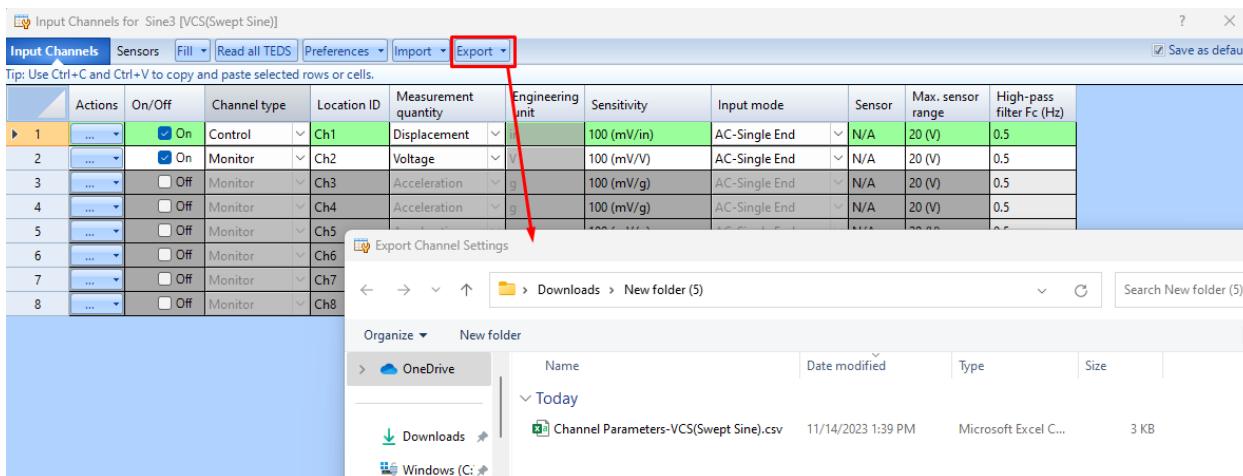
Connect	Disconnect	Command Type	<input checked="" type="radio"/> Plain text	<input type="radio"/> Json																																																						
Run	Pause	Continue	Stop																																																							
Start Record	Stop Record	Save Signals	Shutdown PC																																																							
Start Test Sequence	Next Test Sequence	Pause Test Sequence	Resume Test Sequence																																																							
<input type="radio"/> Execute VCS Command <table border="1"> <tr> <td>Check Only</td> <td>Proceed</td> <td>Show Pretest</td> <td>Save H Signal</td> <td>Reset Average</td> <td>Next Entry</td> </tr> <tr> <td>Abort Check On</td> <td>Abort Check Off</td> <td>Closed Loop Ctrl On</td> <td>Closed Loop Ctrl Off</td> <td>Schedule Clock Timer On</td> <td>Schedule Clock Timer Off</td> </tr> <tr> <td>0;1;0;1</td> <td><input checked="" type="checkbox"/> ROR Board Band On/Off</td> <td>RoR Bands On</td> <td>RoR Bands Off</td> <td colspan="2"></td> </tr> <tr> <td>0;1;0;1</td> <td><input checked="" type="checkbox"/> SOR Board Band On/Off</td> <td>SoR Tones On</td> <td>SoR Tones Off</td> <td colspan="2"></td> </tr> <tr> <td>Tones Hold Sweep</td> <td>Tones Release Sweep</td> <td>Tones Sweep Up</td> <td>Tones Sweep Down</td> <td colspan="2"></td> </tr> <tr> <td>Sweep Up</td> <td>Sweep Down</td> <td>Hold Sweep</td> <td>Release Sweep</td> <td colspan="2"></td> </tr> <tr> <td>Level Up</td> <td>Level Down</td> <td>0</td> <td>Set Level</td> <td>Restore Level</td> <td></td> </tr> <tr> <td>Increase Speed</td> <td>Decrease Speed</td> <td>0</td> <td>Set Frequency</td> <td>0</td> <td>Set Phase</td> </tr> <tr> <td>Inverse Pulse On</td> <td>Inverse Pulse Off</td> <td>Single Pulse On</td> <td>Single Pulse Off</td> <td colspan="2">Output Single Pulse</td> </tr> </table>					Check Only	Proceed	Show Pretest	Save H Signal	Reset Average	Next Entry	Abort Check On	Abort Check Off	Closed Loop Ctrl On	Closed Loop Ctrl Off	Schedule Clock Timer On	Schedule Clock Timer Off	0;1;0;1	<input checked="" type="checkbox"/> ROR Board Band On/Off	RoR Bands On	RoR Bands Off			0;1;0;1	<input checked="" type="checkbox"/> SOR Board Band On/Off	SoR Tones On	SoR Tones Off			Tones Hold Sweep	Tones Release Sweep	Tones Sweep Up	Tones Sweep Down			Sweep Up	Sweep Down	Hold Sweep	Release Sweep			Level Up	Level Down	0	Set Level	Restore Level		Increase Speed	Decrease Speed	0	Set Frequency	0	Set Phase	Inverse Pulse On	Inverse Pulse Off	Single Pulse On	Single Pulse Off	Output Single Pulse	
Check Only	Proceed	Show Pretest	Save H Signal	Reset Average	Next Entry																																																					
Abort Check On	Abort Check Off	Closed Loop Ctrl On	Closed Loop Ctrl Off	Schedule Clock Timer On	Schedule Clock Timer Off																																																					
0;1;0;1	<input checked="" type="checkbox"/> ROR Board Band On/Off	RoR Bands On	RoR Bands Off																																																							
0;1;0;1	<input checked="" type="checkbox"/> SOR Board Band On/Off	SoR Tones On	SoR Tones Off																																																							
Tones Hold Sweep	Tones Release Sweep	Tones Sweep Up	Tones Sweep Down																																																							
Sweep Up	Sweep Down	Hold Sweep	Release Sweep																																																							
Level Up	Level Down	0	Set Level	Restore Level																																																						
Increase Speed	Decrease Speed	0	Set Frequency	0	Set Phase																																																					
Inverse Pulse On	Inverse Pulse Off	Single Pulse On	Single Pulse Off	Output Single Pulse																																																						
<input checked="" type="radio"/> Execute DSA Command <table border="1"> <tr> <td>Trigger On</td> <td>Trigger Off</td> <td>Output On</td> <td>Output Off</td> <td colspan="2"></td> </tr> <tr> <td>Limit On</td> <td>Limit Off</td> <td colspan="2">Reset Average</td> <td colspan="2"></td> </tr> </table>					Trigger On	Trigger Off	Output On	Output Off			Limit On	Limit Off	Reset Average																																													
Trigger On	Trigger Off	Output On	Output Off																																																							
Limit On	Limit Off	Reset Average																																																								

Test's Advanced Command

A section for advance commands such as setting the input channels, run schedule and certain test profiles.

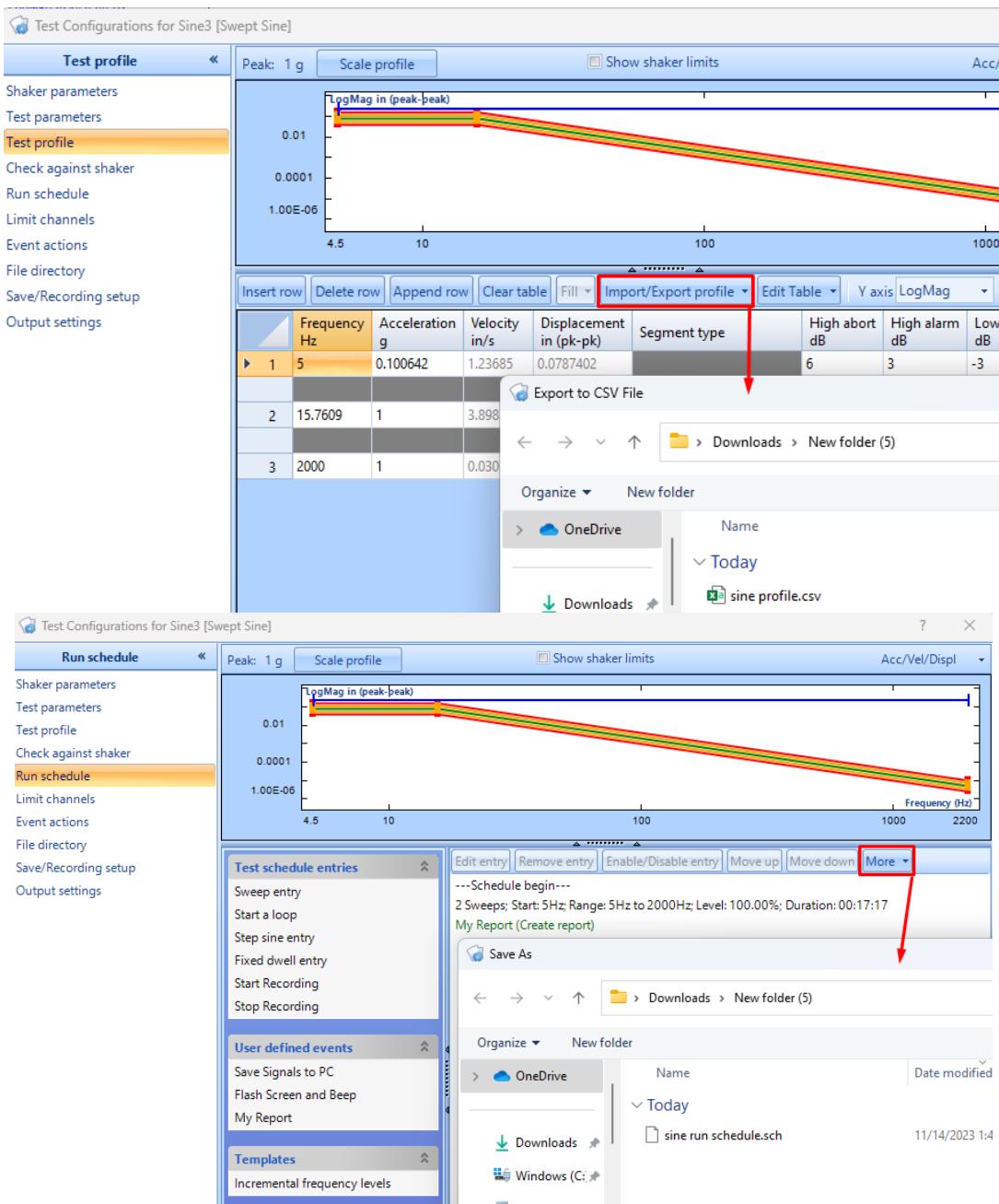
Set Channel Table Advanced Command(Select a CSV file which exported by EDM input channel setup dialog)	
Set Channel Table	<input type="text"/> <input type="button" value="Browse"/> <input type="button" value="Set"/>
Set Schedule Advanced Command(Select a Json file which exported by EDM Schedule setup dialog)	
Set Schedule	<input type="text"/> <input type="button" value="Browse"/> <input type="button" value="Set"/>
VCS Advanced Command(Select a CSV file which exported by VCS profile editor)	
Set Random Profile	<input type="text"/> <input type="button" value="Browse"/> <input type="button" value="Set"/>
Set Sine Profile	<input type="text"/> <input type="button" value="Browse"/> <input type="button" value="Set"/>
Set Shock Profile	<input type="text"/> <input type="button" value="Browse"/> <input type="button" value="Set"/>
Set NTP Server,Port,Sync interval	
NTP Server	<input type="text"/> NPT Port <input type="text" value="123"/> Sync interval <input type="text" value="100"/> (s) <input type="button" value="Set"/>

This require an exported file from EDM to be able to browse and set in the C# MQTT demo.



The screenshot illustrates the process of exporting channel settings from EDM. The top part shows the 'Input Channels' dialog in EDM, where a user has selected several channels (Ch1 to Ch8) and configured them. The 'Export' button is highlighted with a red box. The bottom part shows a file explorer window displaying a CSV file named 'Channel Parameters-VCS(Swept Sine).csv' located in the 'Downloads' folder. This file is the export of the current channel settings.

Actions	On/Off	Channel type	Location ID	Measurement quantity	Engineering unit	Sensitivity	Input mode	Sensor	Max. sensor range	High-pass filter Fc (Hz)
1	On	Control	Ch1	Displacement	mm	100 (mV/in)	AC-Single End	N/A	20 (V)	0.5
2	On	Monitor	Ch2	Voltage	V	100 (mV/V)	AC-Single End	N/A	20 (V)	0.5
3	Off	Monitor	Ch3	Acceleration	g	100 (mV/g)	AC-Single End	N/A	20 (V)	0.5
4	Off	Monitor	Ch4	Acceleration	g	100 (mV/g)	AC-Single End	N/A	20 (V)	0.5
5	Off	Monitor	Ch5							
6	Off	Monitor	Ch6							
7	Off	Monitor	Ch7							
8	Off	Monitor	Ch8							



Test's Detail Status

Where it is actively updated during test run and shows the test advance data.

Get Test Detail Status	
Name	Value
Timestamp	2023-11-14 11:38:50.627
AccUnit	g
VelUnit	in/s
DisplUnit	in
EntryIndex	3
DrivePeak	0.192360997200012
TargetRMS	0.500316917896271
ControlRMS	0.502462923526764
Level	0.5
TotalTime	132.225006103516
RemainTime	317.975006103516
FullLevelTime	0
VelPeak	0.50073159550588
DisplPeakPeak	0.0035521385487257
Kurtosis	3

Test's Channels

Shows the Input Channel Settings data.

Input Range		Channel Index	0	Set Input Range							
Module	Location ID	Enable	Quantity	Unit	Sensitivity	Input Mode	Input Range	High Pass Frequency	Integration	Control Weighting	
(M) 2590976	Apple5	True	Acceler.	m/s ²	101.97162	DC-Single End	Full	0.000	No Integra...	N/A	
(M) 2590976	Cata	True	Acceler.	m/s ²	101.97162	DC-Single End	Full	0.000	No Integra...	N/A	
(M) 2590976	Ca	True	Voltage	V	1000.00000	DC-Single End	Full	0.000	No Integra...	N/A	
(M) 2590976	Pa	True	Voltage	V	1000.00000	DC-Single End	Full	0.000	No Integra...	N/A	
(M) 2590976	B8675B	True	Voltage	V	1000.00000	DC-Single End	Full	0.000	No Integra...	N/A	
(M) 2590976	ABCD	True	Voltage	V	1000.00000	DC-Single End	Full	0.000	No Integra...	N/A	
(M) 2590976	ABC9	True	Voltage	V	1000.00000	DC-Single End	Full	0.000	No Intgra...	N/A	
(M) 2590976	QYU2	True	Voltage	V	1000.00000	DC-Single End	Full	0.000	No Intgra...	N/A	

Test's Channel Status

Get the running test channel status.

Get Channel Status									
Location ID	Min	Max	Peak	RMS	Average	Is IEPE	Is Overload	Is SG Not Connected	
Apple5	-0.02401	-0.01611	-0.02401	0.02027	-0.02006	False	False	False	
Cata	-0.02226	-0.01358	-0.02226	0.01799	-0.01792	False	False	False	
Ca	-0.00421	-0.00340	-0.00421	0.00379	-0.00381	False	False	False	
Pa	-0.00385	-0.00306	-0.00385	0.00347	-0.00346	False	False	False	
B8675B	-0.00391	-0.00311	-0.00391	0.00354	-0.00351	False	False	False	
ABCD	-0.00709	-0.00627	-0.00709	0.00663	-0.00668	False	False	False	
ABC9	-0.00186	-0.00100	-0.00186	0.00144	-0.00143	False	False	False	
QYU2	-0.00900	-0.00814	-0.00900	0.00859	-0.00857	False	False	False	

Test's Parameters

Shows the various test configuration parameters.

<ParameterName>			Set Parameter	List Parameters
Name	Value	Description		
Block Size	1024	256,512,1024,2048,4096,8192,16484,32768,65536		
Average Mode	1	0-->Linear, 1-->Exponential, 2-->Peak hold		
Average Number	8			
Window Type	0	0-->Hanning, 1-->Hamming, 2-->Flattop, 3-->Uniform, etc.		
Weighting Type	0			
DSA_SampleRate_ID	6	0-->102.4KHz, 1-->81.92KHz, 2-->64KHz, 3--> 51.2KHz, etc.		
DSA_OverlapRatio	0	0-->No overlap, 1-->As high as possible, 2-->25%, 3-->50%, etc.		
DSA_LowPassCutoffFreq	0	0~Fa		
DSA_FFTAverageOnOffFlag	1	0-->ON, 1-->OFF		
DSA_WindowParam0				
DSA_WindowParam1				
DSA_WindowParam2				

Test's Profile

Set up EDM VCS Random, Sine or Shock profile.

Acceleration Unit	g										
Random Profile		Frequency	Acceleration	High Abort	High Alarm	Low Alarm	Low Abort			<input type="button" value="Set Random Profile"/> <input type="button" value="Add Entry"/> <input type="button" value="Delete Entry"/>	
Sine Profile		Frequency	Acceleration	Segment Type	High Abort	High Alarm	Low Alarm	Low Abort		<input type="button" value="Set Sine Profile"/> <input type="button" value="Add Entry"/> <input type="button" value="Delete Entry"/>	
Shock Profile	Main Pulse Type	HalfSine = 0	Limit Type	MIL_STD810 = 0	Pulse Width	0.000	Compensation Type	PrePulseOnly = 0	Compensation Pulse Type	Double rectangular = 5	<input type="button" value="Set Shock Profile"/>
	Amplitude	0.000	Post Pulse Height	0.000	Pre Pulse Height	0.000					
										<pre>Segment Type: ConstAccAmp1 = 0 ConstVelAmp1 = 1 ConstDispAmp1 = 2 AccLogLog = 3 VelLogLog = 4 DispLogLog = 5 AccLinLin = 6 VelLinLin = 7 DispLinLin = 8 ConstAmp1 = 9 (default) AccLinLog = 10</pre>	

Test's Shaker

Shows the current test's shaker data.

<Parameter Name>		Update	Get Shaker Parameters	Set Shaker Parameters
Name	Value	Description		
RandomForceRMS	444.92			
RandomMaxAcc	163.4441666666667			
SineForcePeak	9812.176696			
SineMaxAcc	735.49875			
ShockForcePeak	444.92			
ShockMaxAcc	490.3325			
MaxPosDispl	0.00635			
MaxNegDispl	0.00635			
Orientation	Vertical			
MaxVelocity	1.778			
MaxDriveVolt	10			
MinDriveFreq	1			
MaxDriveFreq	2500			
MeasurementNoisy	False			
ArmatureDiameter	1.5			
ArmatureMass	0.2			
FixtureMass	0			
HeaderExpanderMass	0			

Payload mass (kg) Fixture mass (kg)

Test's Test Manager

Shows the test database that is in EDM. It is possible to delete or create new tests here.

List Test	Load Test	Delete Test	Test Name
Name	Type		
FFTOffice41	CSA_FFT		
FFTMQTTTest47	CSA_FFT		
FFTOfficeDRDNASSingle46	CSA_FFT		
FFTOfficeRegularNAS40	CSA_FFT		
FFTOfficeOutput44	CSA_FFT		
FFTOfficeMulti38	CSA_FFT		
FFTOffice80N45	CSA_FFT		
FFTHome1SpiderNas43	CSA_FFT		
FFTHome2SpiderNas42	CSA_FFT		
FFT36	CSA_FFT		
FFT35	CSA_FFT		
FFTOfficeSingle39	CSA_FFT		
FFTOffice37	CSA_FFT		
FFT34	CSA_FFT		
FFT33	CSA_FFT		
FFT32	CSA_FFT		

Test Type

Test's TH Status

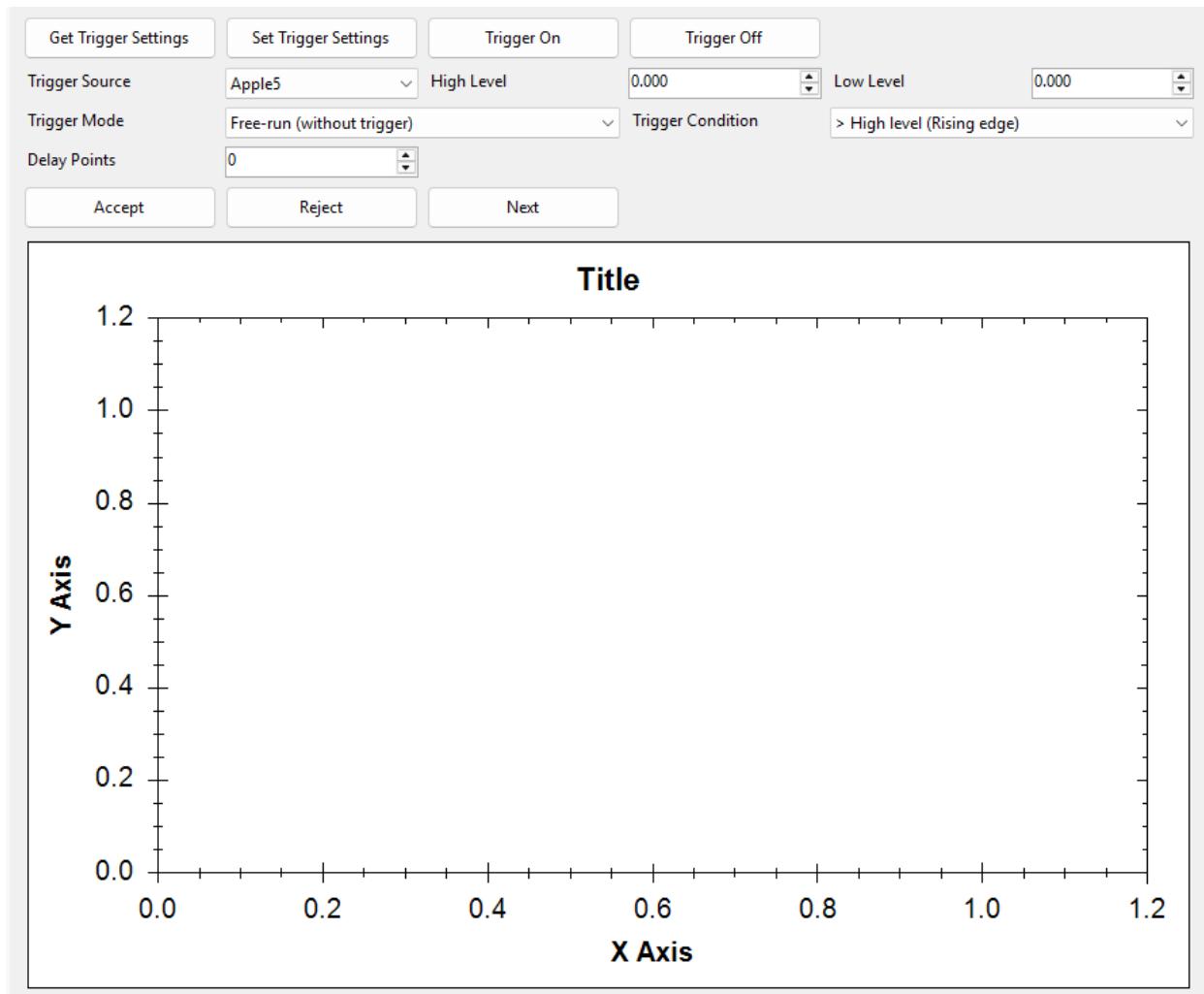
Shows data relating to temperature and humidity control. If the current test if Halt & Hass, the vibration data can also be retrieved.

Name	Value
TotalTime	23.675
RemainTime	330.000
TargetTemperature	32.000
ControlTemperature	76.007
Hist_Temp(Tp1)	-322.857 ("F)
TargetHumidity	NaN
ControlHumidity	NaN

Name	Value
TotalTime	90.540
RemainTime	109.520
TargetVibration	30.000
ControlVibration	0.015
Vib(Ch1)	0.015 (g)

Test's Trigger

Set up triggers and receive signal data.



Test's Output

Shows more commands relating to the EDM control panel output channel.

<input type="button" value="Output On"/>	<input type="button" value="Output Off"/>	(Output index start from 0 to N-1) <input type="text" value="0"/>		
<input type="button" value="Set Output Parameters"/>		<input type="button" value="Set Output Index"/>		
<p><input checked="" type="checkbox"/> Sine</p> <p>Amplitude <input type="text" value="1.000"/></p> <p>Frequency <input type="text" value="1000.000"/></p> <p>Offset <input type="text" value="0.000"/></p>		<p><input type="checkbox"/> Triangle</p> <p>Amplitude <input type="text" value="1.000"/></p> <p>Frequency <input type="text" value="1000.000"/></p>	<p><input type="checkbox"/> Square</p> <p>Amplitude <input type="text" value="1.000"/></p> <p>Frequency <input type="text" value="1000.000"/></p>	<p><input type="checkbox"/> White Noise</p> <p>RMS <input type="text" value="1.000"/></p>
<p><input type="checkbox"/> Pink Noise</p> <p>RMS <input type="text" value="1.000"/></p>		<p><input type="checkbox"/> DC</p> <p>Amplitude <input type="text" value="1.000"/></p>	<p><input type="checkbox"/> Chirp</p> <p>Amplitude <input type="text" value="1.000"/></p> <p>Low Frequency <input type="text" value="1.000"/></p> <p>High Frequency <input type="text" value="1000.000"/></p> <p>Percent <input type="text" value="0.100"/></p> <p>Period <input type="text" value="1.000"/></p>	<p><input type="checkbox"/> Swept Sine</p> <p>Amplitude <input type="text" value="1.000"/></p> <p>Low Frequency <input type="text" value="1.000"/></p> <p>High Frequency <input type="text" value="1000.000"/></p> <p>Period <input type="text" value="1.000"/></p>

Test's Global Parameters

Here the client can request a list global parameters from EDM and their corresponding data values in Json format. There are two more UI elements that extract the Json data in readable format.

List available Global Parameters
Get Global Parameter
Copy Json content

Name

```
Test.Run.NumericalValue.OutputPk
Test.Run.NumericalValue.SpiderString_Frame
Test.Run.NumericalValue.TotalElapsed
Test.Run.Signal.Apple5
Test.Run.Signal.Cata
Test.Run.Signal.Ca
Test.Run.Signal.Pa
Test.Run.Signal.B8675B
Test.Run.Signal.ABCD
Test.Run.Signal.ABC9
Test.Run.Signal.QYU2
Test.Run.Signal.Block(Apple5)
Test.Run.Signal.Block(Cata)
Test.Run.Signal.Block(Ca)
Test.Run.Signal.Block(Pa)
Test.Run.Signal.Block(B8675B)
Test.Run.Signal.Block(ABCD)
Test.Run.Signal.Block(ABC9)
Test.Run.Signal.Block(QYU2)
Test.Run.Signal.APS(Apple5)
Test.Run.Signal.APS(Cata)
Test.Run.Signal.APS(Ca)
Test.Run.Signal.APS(Pa)
Test.Run.Signal.APS(B8675B)
Test.Run.Signal.APS(ABCD)
Test.Run.Signal.APS(ABC9)
Test.Run.Signal.APS(QYU2)
Test.Run.Table.ChannelStatus
```

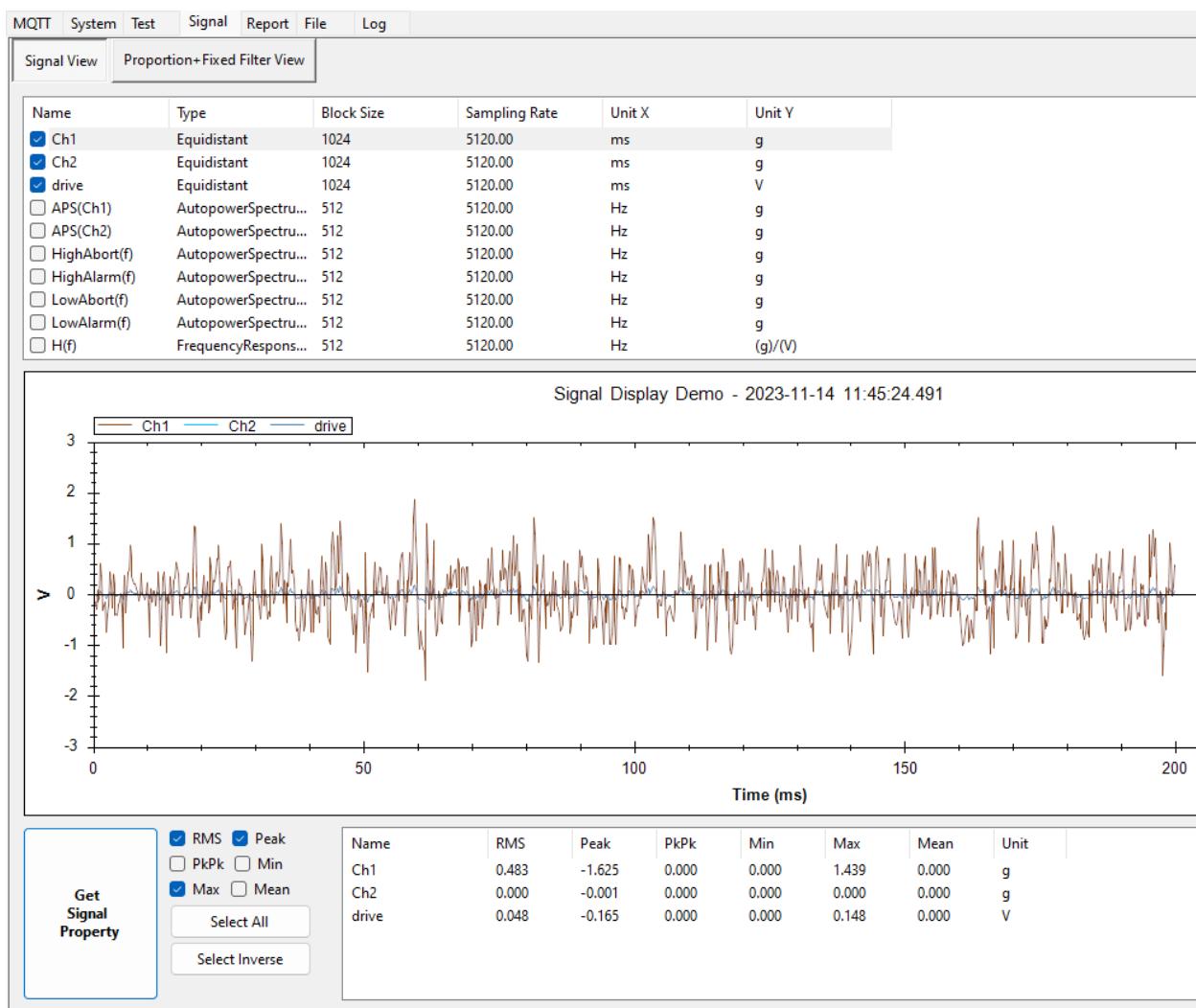
```
{
  "SignalId": 216,
  "Name": "APS(Apple5)",
  "Type": "Frequency",
  "NvhType": "AutopowerSpectrum",
  "FirstTime": 615934255142.56482,
  "Group": {
    "Index": null,
    "Value": {
      "Languages": [
        {
          "Key": "en-US",
          "Value": "Auto-Power Spectra"
        }
      ]
    }
  },
  "Data": [
    {
      "Values": [
        0.0,
        25.0
      ]
    }
  ]
}
```

Data	(Collection)
Mean	1.6340160300387652E-06
Name	APS(Apple5)
NvhType	AutopowerSpectrum
OverallRMS	0.02361517059509317
Pk	0.0006673340103588998
PkPk	0.0006673335446976125
RMS	2.4825914806569926E-05
RPM	0
Type	Frequency

	Frequency (Hz)	Acceleration (m/s ²)	Z axis
▶	0	0.000667334010...	615934255.1425...
*			

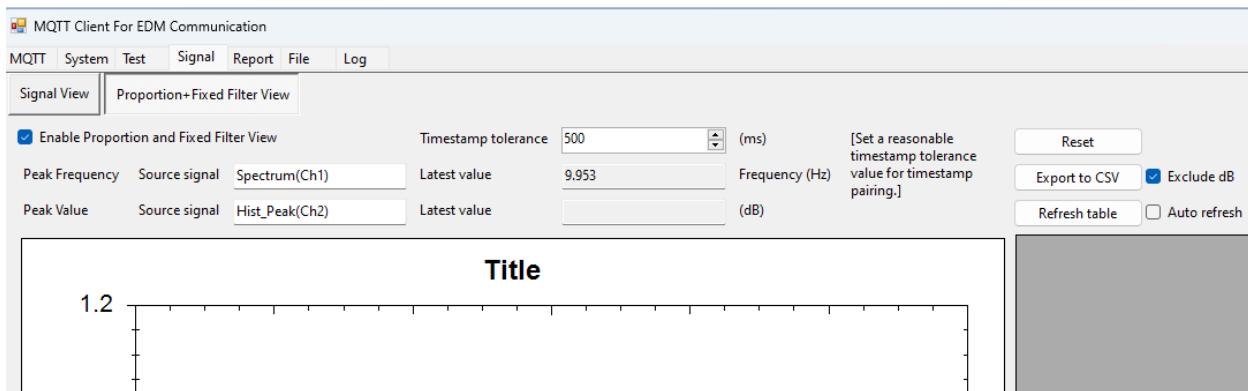
Signal

Where it displays a list of available signals in EDM, a graph for displaying each signal frame data and the signal's property table. Each signal can be selected in the list and it will populated the graph if the test is running. The same can be applied to the Get Signal Property button.



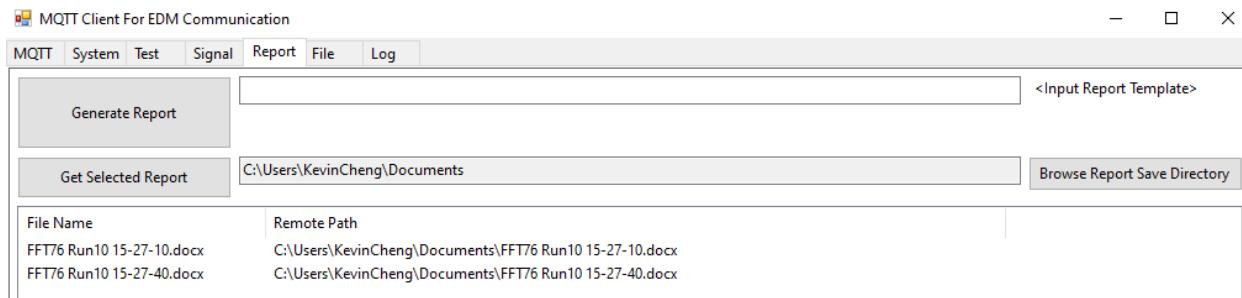
Signal's Proportion+Fixed Filter View

Show the signal peak values. Used with Shutdown Protection System, also known as Sine Reduction in VCS.

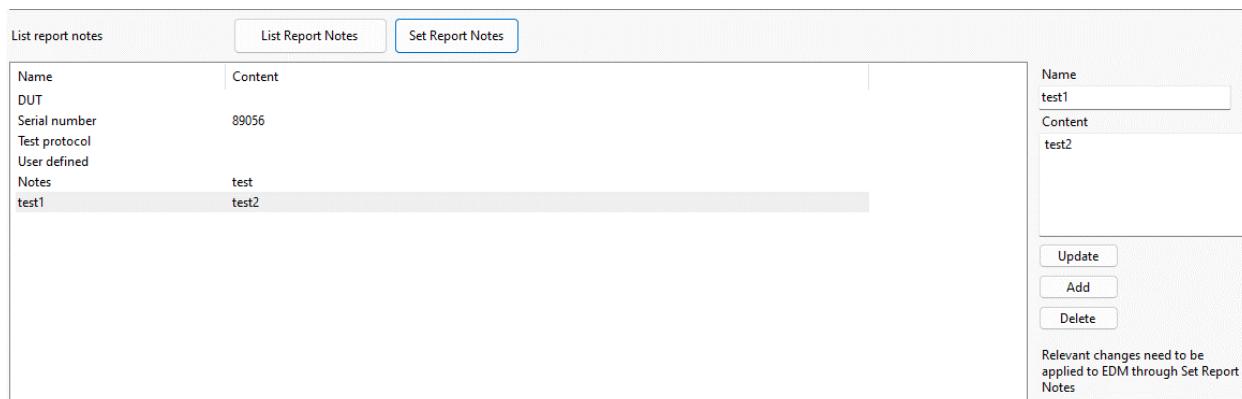


Report

Where it can send a command to EDM to generate a report and save it in the Save Directory.

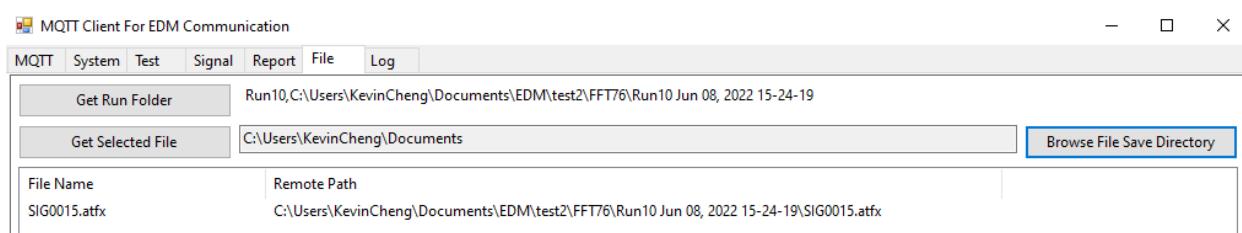


It can also get a list of report notes in a report template and set specific content for those notes.



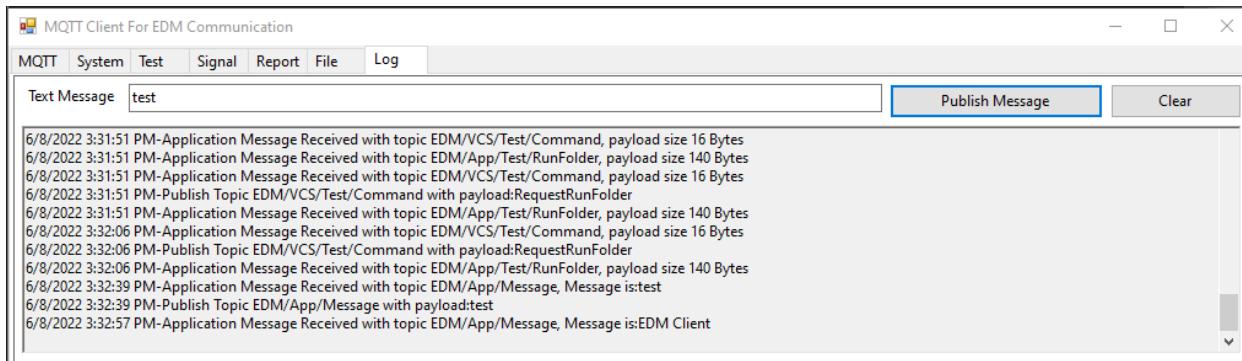
File

Where it can send a command to EDM to get the run folder file location and save a ATFX file from EDM in a Save Directory location.



Log

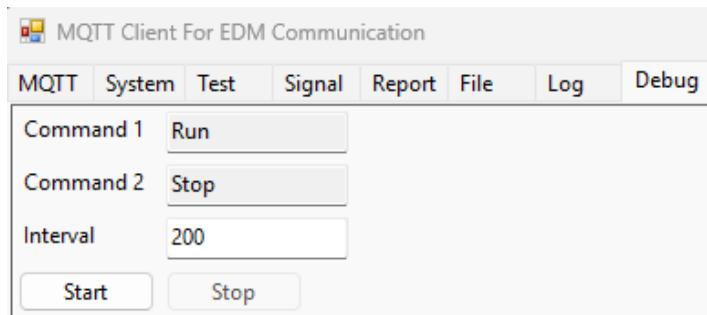
Where message communication between MQTT Clients is stored. It is possible to send a text message between clients as well.



Debug

The debug tab is a simple publish MQTT commands based on a interval. The commands can be changed in code.

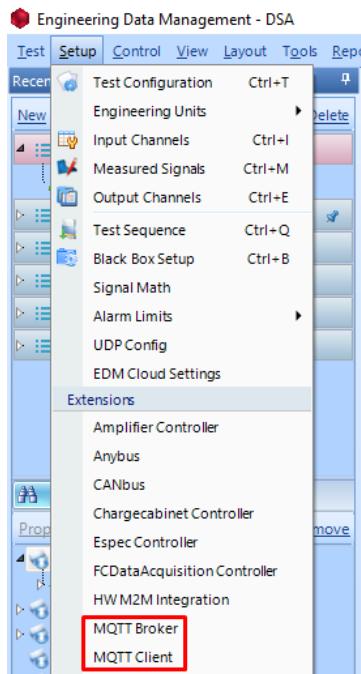
This page is only visible if building and running the debug version. Otherwise the release version will not show this tab.



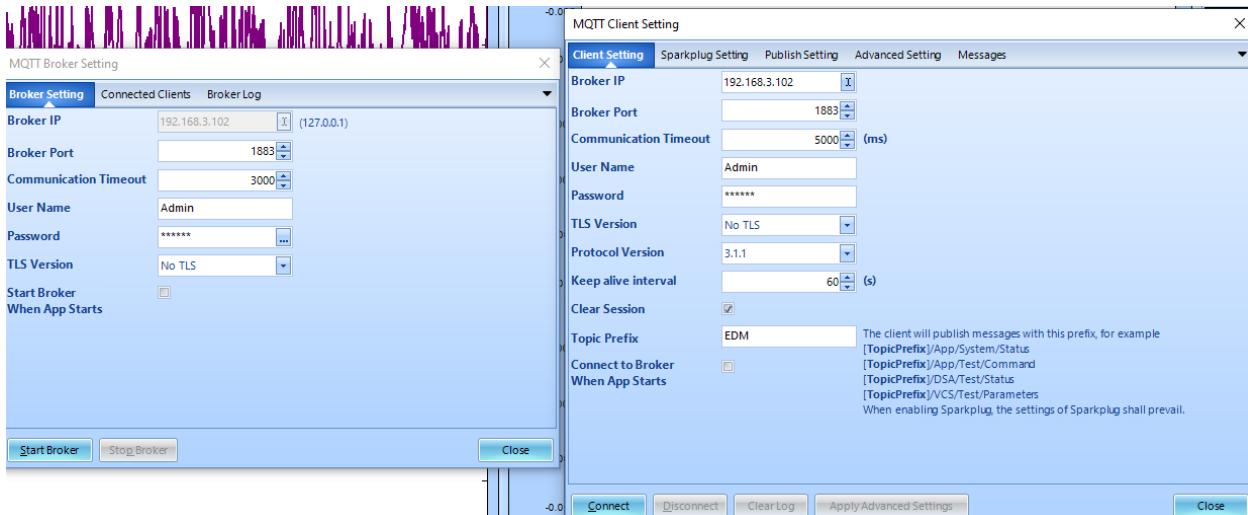
Connecting and Running a Test

To connect and run a test with the MQTT Client Demo Program, first have both the demo program and a EDM application capable of communicating with MQTT opened, such as EDM VCS, DSA and THV.

Once EDM is open, go to the toolbar Setup menu and click both MQTT Broker and MQTT Client.

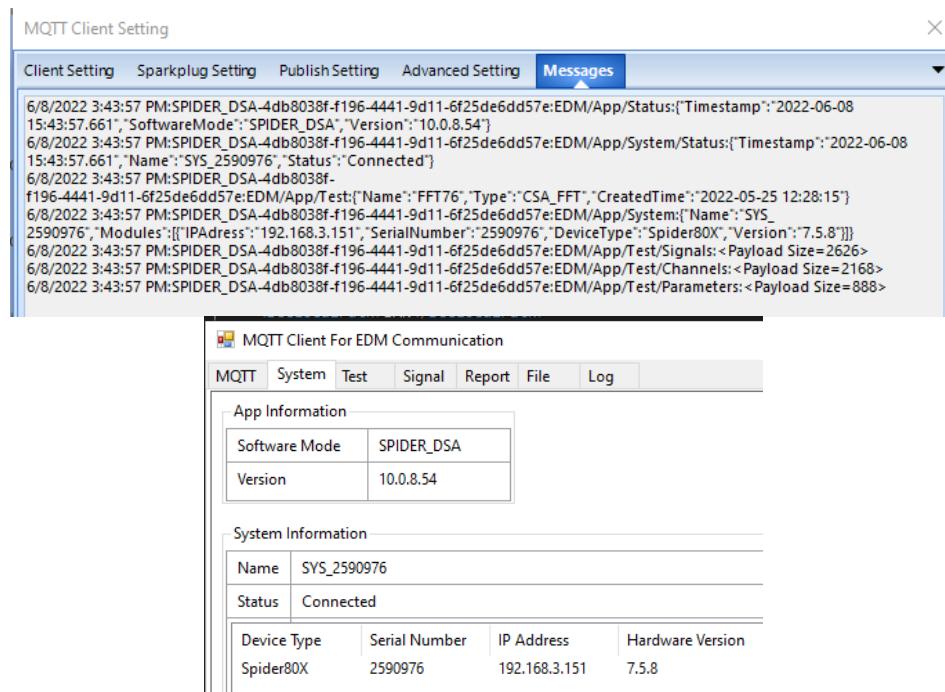


Two windows should appear starting at the connection parameter setting tab. Make sure the parameters are the same between the EDM MQTT Broker, EDM MQTT Client and MQTT Client Demo Program. Once they are, click Start Broker and Connect for all windows.

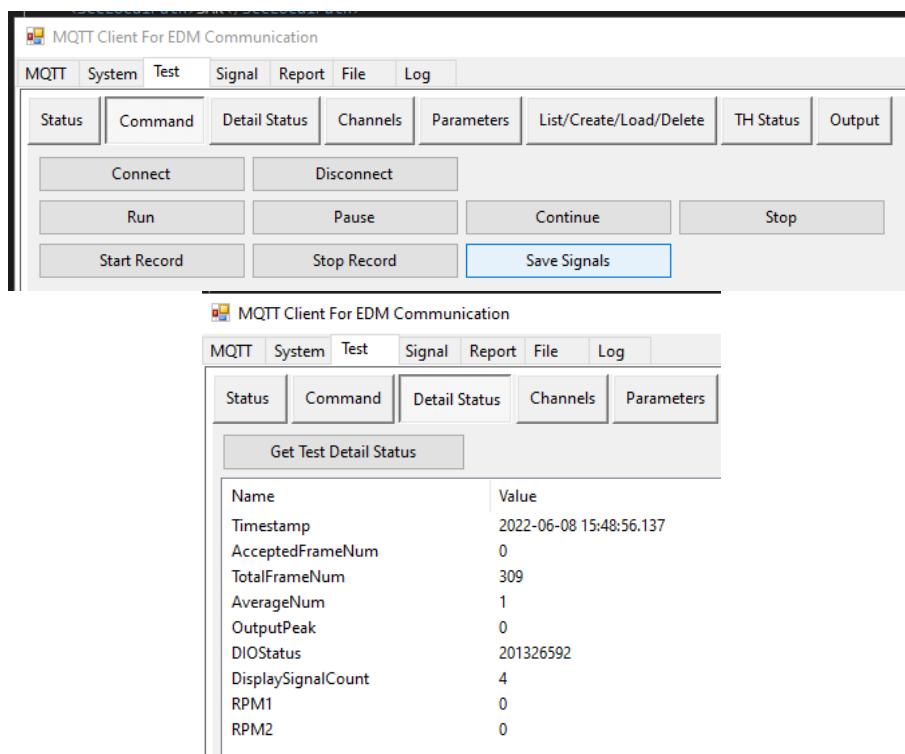


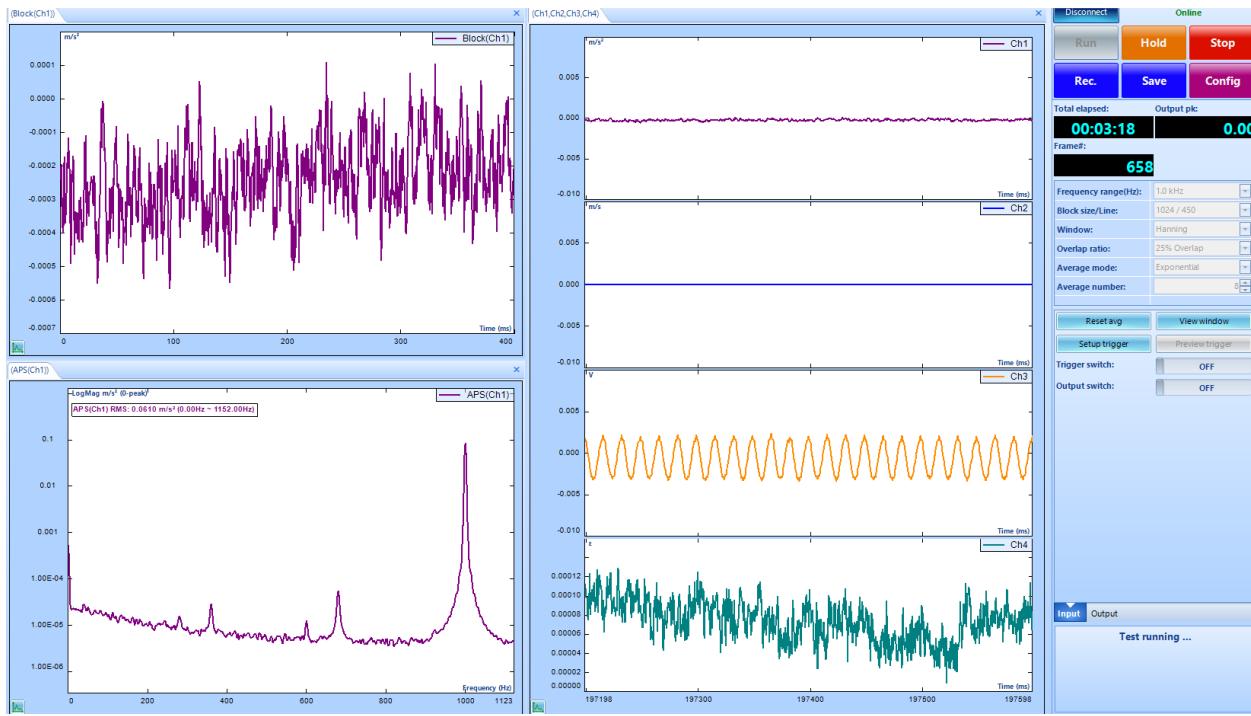
Once connected, the MQTT network should start communicating with each other and send off some topic messages regarding each client information and the test data.





Then, go to the Test's Command tab and click Connect then Run. From here, EDM MQTT Client will send the test detail status updates to the MQTT Client Demo Program.

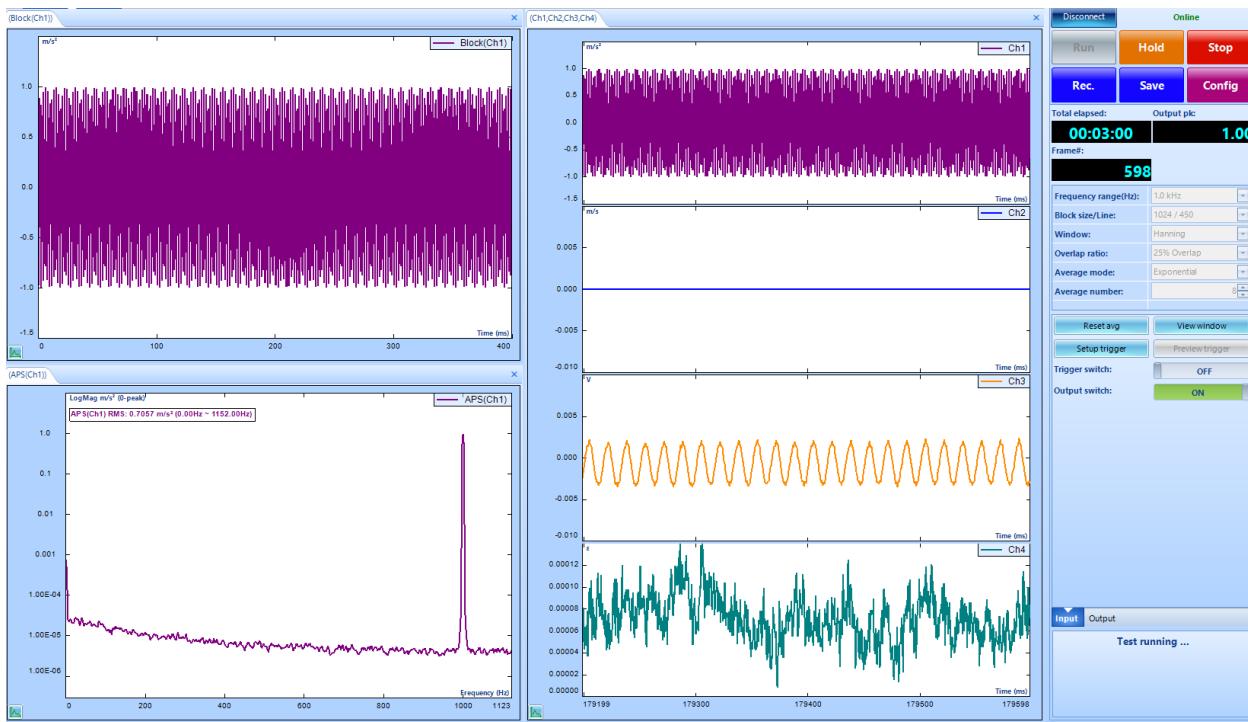




And depending on the EDM application and test type, it is possible to send other type of commands, such as turning on an output channel in DSA.

④ Execute DSA Command

Trigger On	Trigger Off	Output On	Output Off
Limit On	Limit Off	Reset Average	



EDM MQTT Client Python Scripts

An API has been created in Python to abstract away the specific commands that MQTT needs to talk to EDM. Our Python API provides utilities and allows for creating readable scripts for automating tests in EDM. Instead of requiring the user to be familiar with how to setup publishers and subscribers, the user can simply import our Python module and call functions. This API script uses a package called **Paho MQTT** that can be installed using the following command, depending on the version in the **MQTTPythonExample** folder:

pip install paho-mqtt==1.6.1

or

pip install paho-mqtt==2.1.0

There are some changes from 1.6.1 to 2.0.0 ~ 2.1.0 that make scripts created in 1.6.1 incompatible with paho-mqtt 2.0.0+.

The Python side is segmented broadly into two sections – the module/API and the user-level scripts. The user-level scripts are very customizable, allowing the user to execute any number of functions in any sequence they want. They will also have access to the API code, so they can modify it if they want.

There are some scripts showing how to run a test and get a signal frame data. And how to get Total Harmonic Distortion (THD) measurement.

MQTT Client API Script

In the API script, there will be various imports depending on the folder version. Some imports are already installed via python, but there may be additional packages that need to be installed, such as **numpy** for paho-mqtt v1.6.1 folder. Running the script first will lead to various import package errors in the console that will point out missing packages that need to be installed.

In the initial part of script are the various methods that initialize the MQTT client variables, connecting to other clients and publishing and subscribing topics. To view the actual code, please refer to the mqtt.py script.

Method Name	Parameters	Description
<code>__init__</code>	Refer to mqtt.py due to different version scripts	Initialize method Set up variables, publisher, subscriber, topics, connect clients
<code>connect_mqtt</code>	Refer to mqtt.py due to different version scripts	Connects MQTT broker / client
<code>publish</code>	pubtopic, msg	Publishes a topic to EDM broker
<code>on_message</code>	client, userdata, msg	Handles receiving a message from EDM
<code>subscribe</code>	topics	Subscribe to EDM topics

The rest of the script are topic commands that use the publish method to send out the publish topic prefix, command and any parameters if needed.

There are three publish topic prefix depending on which applications for sending commands:

- EDM/App/Test/Command
- EDM/VCS/Test/Command
- EDM/DSA/Test/Command

Running a Test and Plotting Signal Data

In the **Paho-MQTT v1.6.1/MQTTEExample_RunningATest.py** script, it shows how to use the Python mqtt.py script to connect to EDM MQTT network, run a test and receive a signal frame data.

The following are the imports:

```
import mqtt
import time
import numpy as np
import datetime
import os
import matplotlib.pyplot as plt
```

Matplotlib can be installed by the following command:

pip install matplotlib

In the first part of the script is initializing the Python MQTT client.

```
# Connect to MQTT Broker
mqttClient = mqtt pubsub(brokerIP = "192.168.1.123", prefix=prefix)
mqttClient.subclient.loop_start()

# initial sleep is needed to get system status messages
time.sleep(2)
```

After that is creating a save folder that can be used to save signal data and plot images if needed.

```
# Get date to use it for save folder name
now = datetime.datetime.now()
dt_string = now.strftime("%Y-%m-%d--%H-%M")

softwareMode =
mqttClient.LUT['EDM/App/Status'].split("SoftwareMode")[1].split(",")[0][3:-1]
serialnumber =
mqttClient.LUT['EDM/App/System'].split("SerialNumber")[1].split(",")[0][3:-1]
devicetype =
mqttClient.LUT['EDM/App/System'].split("DeviceType")[1].split(",")[0][3:-1]

savedirectory = softwareMode + "-" + devicetype + "-" + serialnumber + "-" +
dt_string

print(savedirectory)

# Create save folder
# 'dir' is windows equivalent of 'ls' on linux, lists files in the directory
r = os.system("dir " + savedirectory)
# if the directory does not exist, r will be 1 - create it before continuing
if r == 1:
    os.system("mkdir " + savedirectory)
```

The script can now connect to EDM and start running a test. And the current EDM is VCS, proceed after the pre-test is done.

```
# Connect and run a EDM test
mqttClient.connect()
mqttClient.run()

r = input("\nPress enter once pre-test is done")
# any other key followed by enter will exit, or ctrl-c will exit
```

```

if r != '':
    exit()

# Start test after pre-test
mqttClient.proceed()
time.sleep(2)

```

After the test successfully runs, the script will grab the status which should be “Running”, set up a couple of variables and enable plot interactivity.

```

testStatus =
mqttClient.LUT['EDM/App/Test/Status'].split("Status")[1].split(",")[-1][3:-1]

signalFrame = []
count = 0
plt.ion()

```

If the test is running, the script will start to get block(Ch1) data and process the data into a array.

```

try:
    while(testStatus == "Running"):
        # Request signal data, such as Channel 1
        mqttClient.get_channel_data(1)
        # Wait for receiving and parsing message
        # Change time to receive data in real time or occasional updates
        time.sleep(0.01)
        # Refresh graph to view the current signal frame
        # or comment it out to have a time history graph of the signal
        plt.clf()

        ## Parsing the received message ##
        signalName =
        mqttClient.LUT['EDM/App/Test/SignalData'].split("ValueX")[0].split("Name")[1].split(",")[-1]
        signalUnitX =
        mqttClient.LUT['EDM/App/Test/SignalData'].split("ValueX")[0].split("UnitX")[1].split(",")[-1]
        signalUnitY =
        mqttClient.LUT['EDM/App/Test/SignalData'].split("ValueX")[0].split("UnitY")[1].split(",")[-1]

        Xvalues =
        mqttClient.LUT['EDM/App/Test/SignalData'].split("ValueX")[1].split("ValueY")[0].split("ValueZ")[-1]
        X = Xvalues.split(",")

```

```

X = np.fromiter(X, float)

Yvalues =
mqttClient.LUT['EDM/App/Test/SignalData'].split("ValueX")[1].split("ValueY")[1]
.split("ValueZ")[0][3:-3]
Y = Yvalues.split(",")
Y = np.fromiter(Y, float)

Zvalues =
mqttClient.LUT['EDM/App/Test/SignalData'].split("ValueX")[1].split("ValueY")[1]
.split("ValueZ")[1][3:-3]
Z = Zvalues.split(",")
Z = np.fromiter(Z, float)

signalFrame.append(X)
signalFrame.append(Y)
signalFrame.append(Z)

thd = np.array(signalFrame)

```

After the data has been processed, it can be used in the following code blocks, such as printing out the data into the console, saving the data to .npy files or plotting out the signal frame data.

```

## Print Statements for displaying points of the signal frame ##
# columnlabelstring = "X Values || Y Values || Z Value"
# print("\n" + " " * (len(columnlabelstring)//4) + "Signal Frame
Data\n" + "-" * len(columnlabelstring))
# print(columnlabelstring + "\n")

# nspace1 = 5
# nspace2 = 11
# nspace3 = 13

# for x in range(len(signalFrame[0] / 4)):
#     print(" "*nspace1, signalFrame[0][x], " "*nspace2,
signalFrame[1][x]," "*nspace3, signalFrame[2][0])

## Save data to .npy files in the created save folder above ##
# np.save(savedirectory + "/fullSignalFrame"+str(count)+".npy", thd)
# # Save individual slices too
# np.save(savedirectory + "/X"+str(count)+".npy", thd[0])
# np.save(savedirectory + "/Y"+str(count)+".npy", thd[1])
# np.save(savedirectory + "/Z"+str(count)+".npy", thd[2])

```

```

# print("\nSaved .npy files to ", savedirectory)

## Generate plot ##
# Adjust the X values for time domain signals
# comment out for frequency domain signals
thd[0] = thd[0] - thd[2][0]

plt.plot(thd[0], thd[1], 'r', label=signalName)
plt.title("Signal Frame Data of " + signalName)
plt.xlabel(signalUnitX)
plt.ylabel(signalUnitY)
plt.draw()
# plt.savefig(savedirectory + "/" + signalName + "-plot" + str(count) + ".png")
plt.pause(0.001)

count += 1
signalFrame = []
testStatus =
mqttClient.LUT['EDM/App/Test/Status'].split("Status")[1].split(",")[0][3:-1]

except KeyboardInterrupt:
    print('Keyboard Interrupt received -- exiting main loop')

```

The test can be stopped at any time or after the test is done, the script will close the client loop.

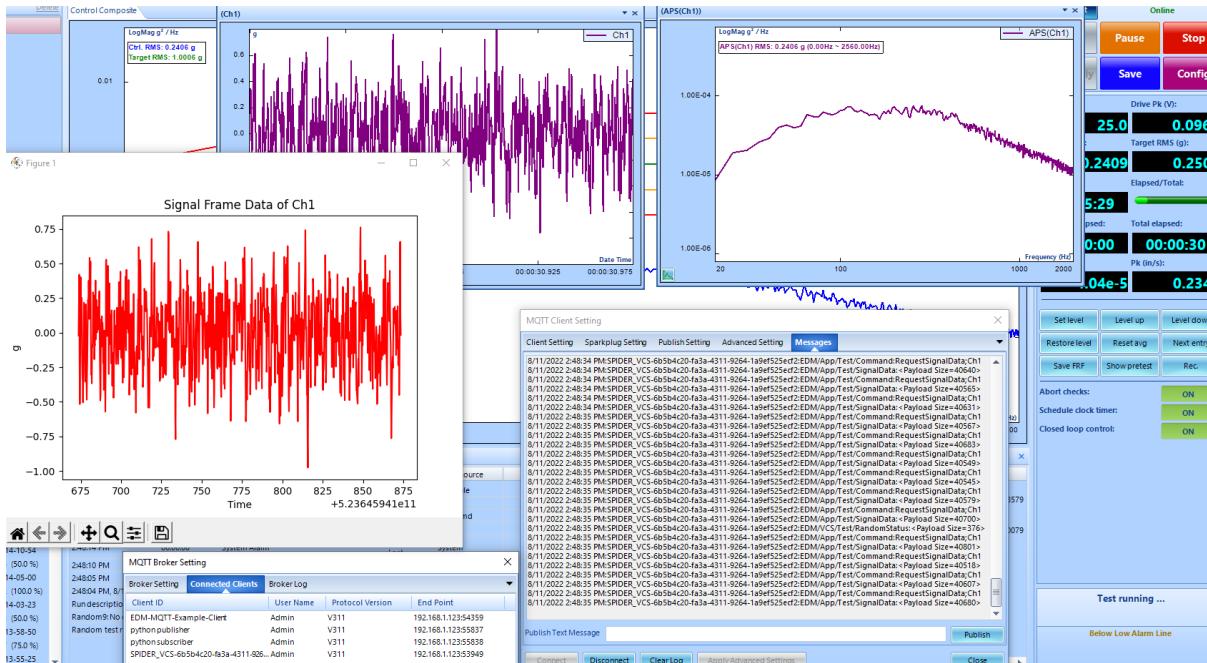
```

# For any interruption from the python script, stop the test
mqttClient.stop()

try:
    mqttClient.subclient.loop_stop()
    print("MQTT loop stopped successfully")
except:
    print("Failed to stop subscriber loop")

```

The below screenshot shows a matplotlib plot of the current running Random test in EDM VCS with the EDM broker showing the connected python pub / sub and EDM Client log showing the python script requesting data.



Automating THD Measurement

To demonstrate some of the capabilities, an example of measuring THD at multiple amplitudes and frequencies was made. The changing of amplitudes and frequencies is automated so that the user does not need to manually enter each one. This section is based on **Paho-MQTT v1.6.1**.

Setup the publisher and subscriber class. The mqtt module contains functions for communicating with both DSA and VCS.

```
mqttClient = mqtt pubsub(brokerIP = "192.168.1.123")
mqttClient.subclient.loop_start()
# initial sleep is needed to get system status messages
time.sleep(2)
```

Multiple amplitudes and frequencies are defined to be looped over. More amplitudes and frequencies will require more time for the script to complete.

```
# define ranges for frequencies and amplitudes as list
freqs = [freq*1000 for freq in range(1,4)]
amps = [amp*0.1 for amp in range(1,11)]
# or define custom ranges
freqstep = 250
minfreq = 1000
maxfreq = 10000
freqs = np.arange(minfreq,maxfreq + freqstep,freqstep)
amps = [8.92, 5, 2, 1]
```

Other utilities help to document the script, including making a directory for the results, labeled with the current date as well as the serial number and device (Spider-80X, Spider-81, etc.)

```
# Get date to use it for folder name
now = datetime.datetime.now()
dt_string = now.strftime("%Y-%m-%d--%H-%M")

serialnumber =
mqttClient.LUT['EDM/App/System'].split("SerialNumber")[1].split(",")[0][3:-1]
devicetype =
mqttClient.LUT['EDM/App/System'].split("DeviceType")[1].split(",")[0][3:-1]

savedirectory = "THD-" + devicetype + "-" + serialnumber + "-" + dt_string

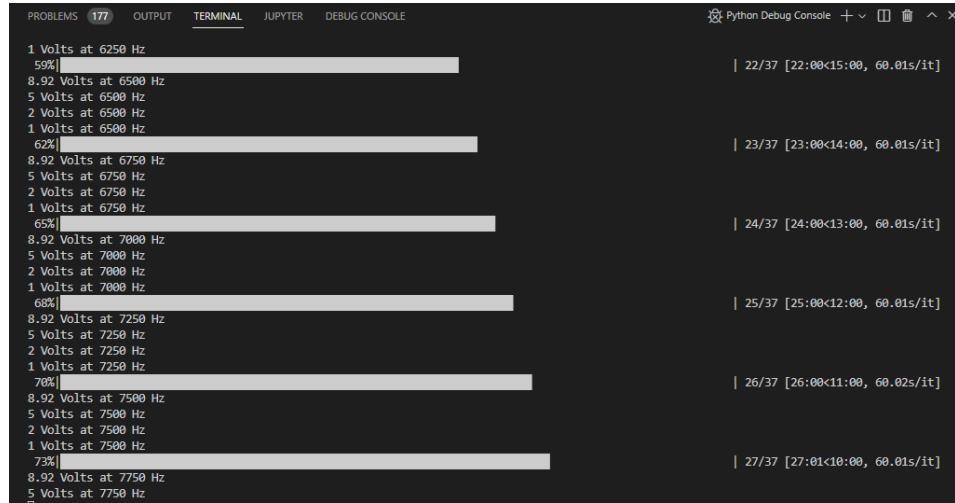
# 'dir' is windows equivalent of 'ls' on linux, lists files in the directory
r = os.system("dir " + savedirectory)
# if the directory does not exist, r will be 1 - create it before continuing
if r == 1:
    os.system("mkdir " + savedirectory)
```

The main loop iterates over a list of amplitudes and frequencies. The amplitude and frequency are set, the APS averaging is reset, the APS is retrieved from EDM, then the data is parsed to get the amplitudes and frequencies. Delays/sleeps are necessary in between some of these operations to account for some of the limitations of MQTT.

```
for freq in tqdm(freqs):
    print() # make a new line after tqdm progress bar
    for amp in amps:
        print(str(amp), "Volts at", str(freq), "Hz")
        mqttClient.output_sine(amp,freq)
        time.sleep(5)
        mqttClient.reset_average()
        time.sleep(5)
        # After inner loop has settled/paused, get APS
        mqttClient.get_APS(1) # Publisher message
        # Need another sleep between requesting message and receiving
        time.sleep(5)

        # parsing the received message
        Yvalues =
        mqttClient.LUT['EDM/App/Test/SignalData'].split("ValueX")[1].split("ValueY")[1]
        .split("ValueZ")[0][3:-3]
        Y = Yvalues.split(",")
        harms = []
```

```
# go up to numharmonics or less, depending on frequency
for harmonic in range(min(int(mqttClient.freqrange / freq),
numharmonics)):
    i = int(freq / mqttClient.freqresolution) * (harmonic + 1)
    x = float(Y[i])
    harms.append(x)
thd.append( [freq, amp, computeTHD(harms)] )
```



PROBLEMS 177 OUTPUT TERMINAL JUPYTER DEBUG CONSOLE

1 Volts at 6250 Hz
59% | 22/37 [22:00<15:00, 60.01s/it]

8.92 Volts at 6500 Hz
5 Volts at 6500 Hz
2 Volts at 6500 Hz
1 Volts at 6500 Hz
62% | 23/37 [23:00<14:00, 60.01s/it]

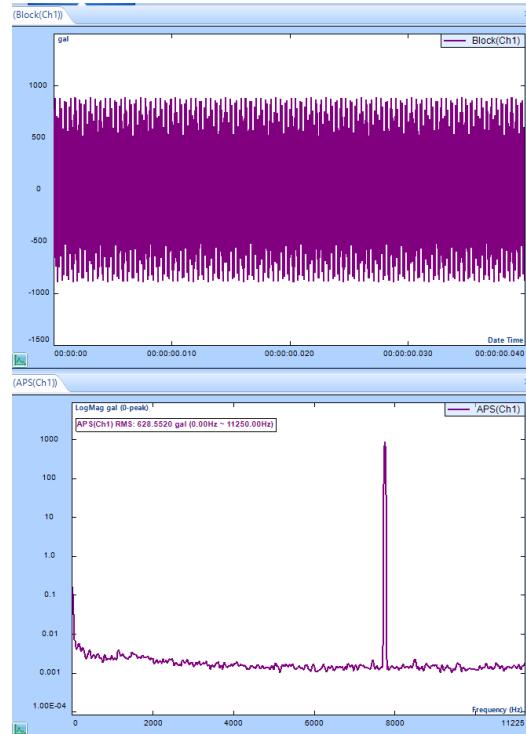
8.92 Volts at 6750 Hz
5 Volts at 6750 Hz
2 Volts at 6750 Hz
1 Volts at 6750 Hz
65% | 24/37 [24:00<13:00, 60.01s/it]

8.92 Volts at 7000 Hz
5 Volts at 7000 Hz
2 Volts at 7000 Hz
1 Volts at 7000 Hz
68% | 25/37 [25:00<12:00, 60.01s/it]

8.92 Volts at 7250 Hz
5 Volts at 7250 Hz
2 Volts at 7250 Hz
1 Volts at 7250 Hz
70% | 26/37 [26:00<11:00, 60.02s/it]

8.92 Volts at 7500 Hz
5 Volts at 7500 Hz
2 Volts at 7500 Hz
1 Volts at 7500 Hz
73% | 27/37 [27:01<10:00, 60.01s/it]

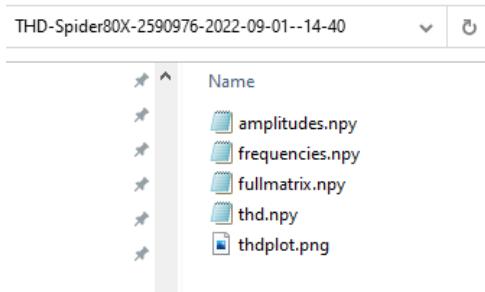
8.92 Volts at 7750 Hz
5 Volts at 7750 Hz



The data is saved to the directory to in case it is desired to inspect it after the script is complete, to avoid having to rerun the entire script.

```
## Save data to .npy files
thd = np.array(thd)
np.save(savedirectory + "/fullmatrix.npy", thd)
# Save individual slices too
np.save(savedirectory + "/frequencies.npy", thd[:,0])
np.save(savedirectory + "/amplitudes.npy", thd[:,1])
np.save(savedirectory + "/thd.npy", thd[:,2])

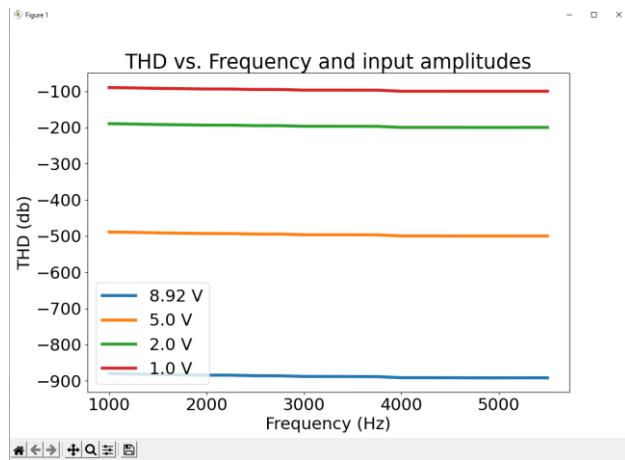
print("-" * len(columnlabelstring) + "\nSaved .npy files to", savedirectory)
```



This part of the script plots the data and saves the plot to the directory.

```
# Generate plot
# horizontal axis is frequency, vertical axis is thd
# multiple lines drawn for each input amplitude
plt.rcParams.update({'font.size': 22})
plt.figure(figsize=(12,8))
numamps = len(amps)
legend = []
for i in range(numamps):
    freqs    = thd[:,0][i::numamps]
    dbthd   = thd[:,2][i::numamps]
    inputamp = thd[:,1][i::numamps][0]
    legend.append(str(inputamp) + " V")
    plt.plot(freqs, dbthd, linewidth=4)

plt.xlabel("Frequency (Hz)")
plt.ylabel("THD (db)")
plt.title("THD vs. Frequency and input amplitudes")
plt.legend(legend)
plt.savefig(savedirectory + "/thdplot.png")
plt.show()
```

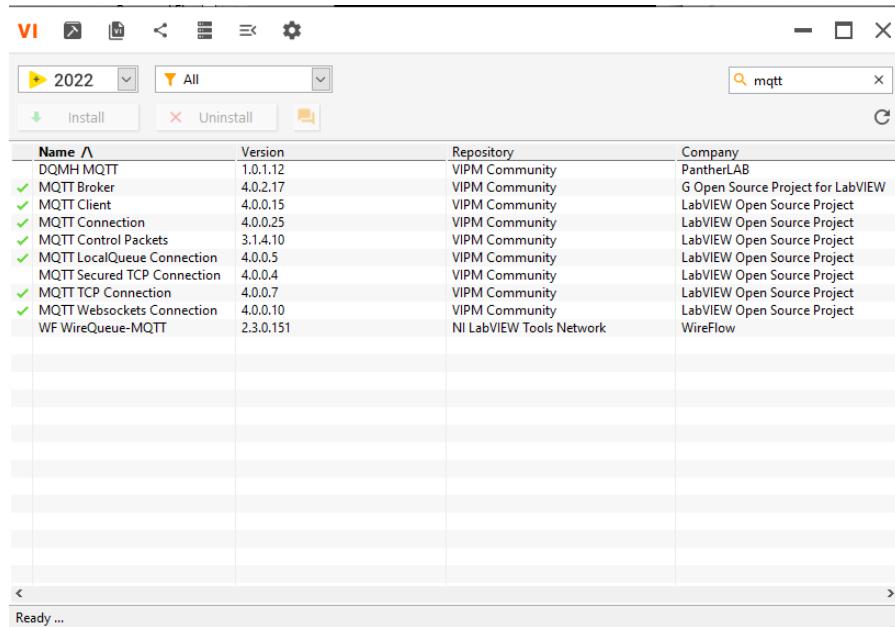


EDM MQTT Client LabVIEW Demo

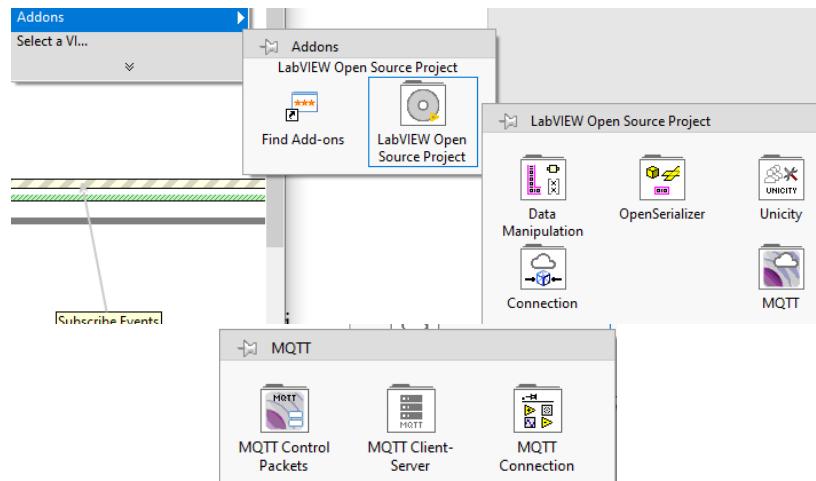
The EDM MQTT Client LabVIEW Demo utilizes the LabVIEW Open Source Project MQTT packages available via the LabVIEW VI Package Manager.

To download the MQTT packages, go to the search bar in the VI Package Manager and type in **MQTT**. It will list the various MQTT packages that LabVIEW has available.

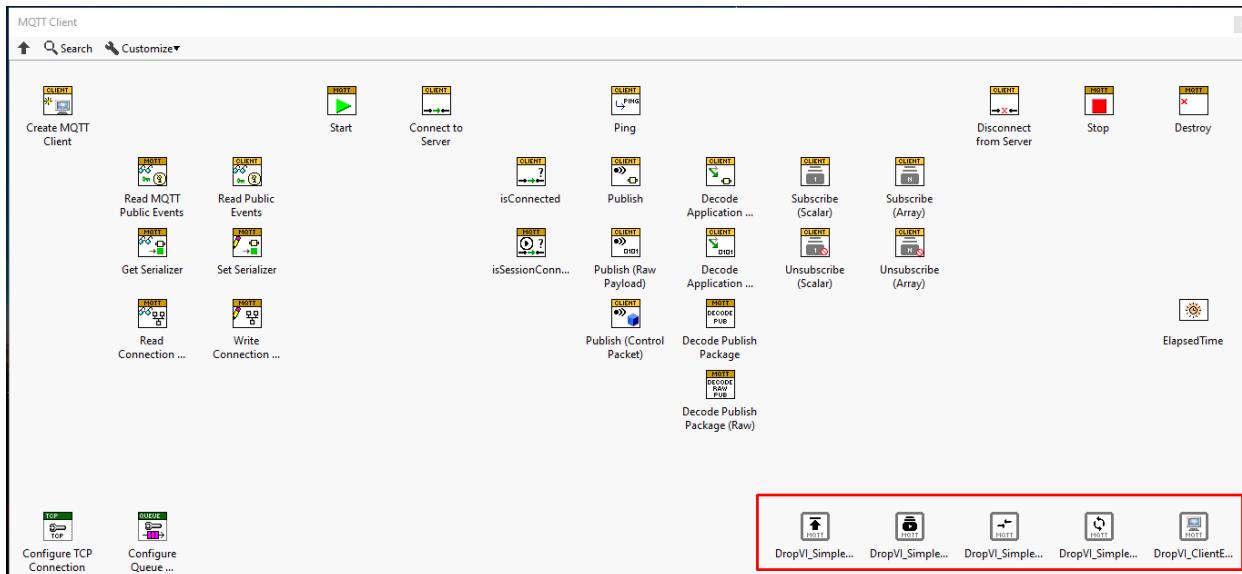
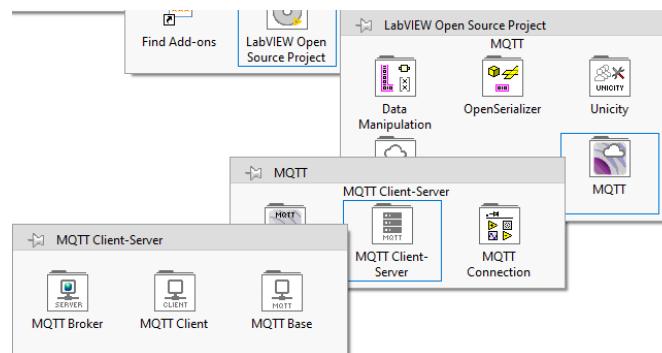
Download **MQTT Broker** should download most of the other MQTT packages in the list. For any other package that doesn't have a checkmark next to its name will also have to be downloaded.



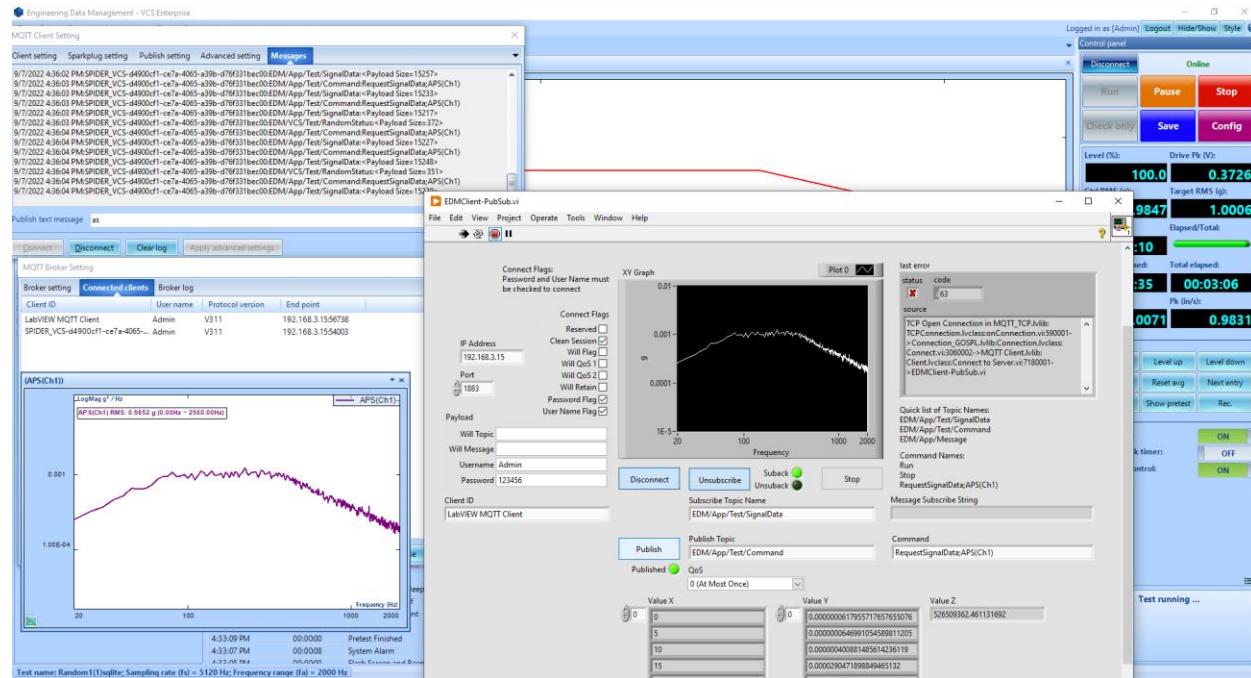
After downloading the MQTT packages, right clicking in the Block Diagram of a new vi file will open the functions menu. Then go to Addons > LabVIEW Open Source Project > MQTT and it will reveal the MQTT functions and some MQTT example vi nodes.



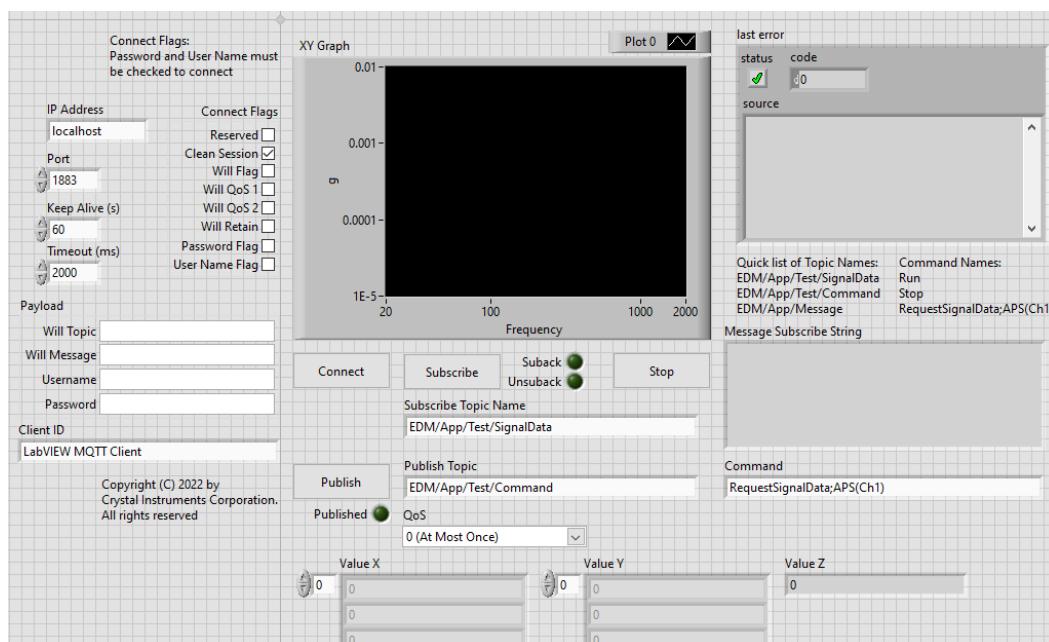
The LabVIEW MQTT addon also contains a couple of example diagrams for users to build off of. These are located in MQTT Client-Server > MQTT Client at the bottom right corner of the window.



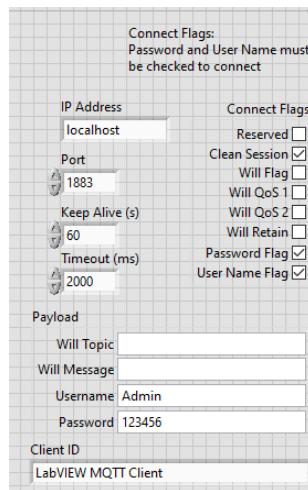
In this EDM MQTT Client LabVIEW Demo vi example, it has the basic capabilities of being able to connect to the EDM MQTT network, subscribe to topics and publish topics with commands. As shown below with a EDM VCS Random test running, MQTT network up and the LabVIEW MQTT Client Demo connected and requesting APS(Ch1) signal data. It should provide enough knowledge to the user of how the MQTT package works in LabVIEW if they wish to expand upon the LabVIEW MQTT Client Demo.



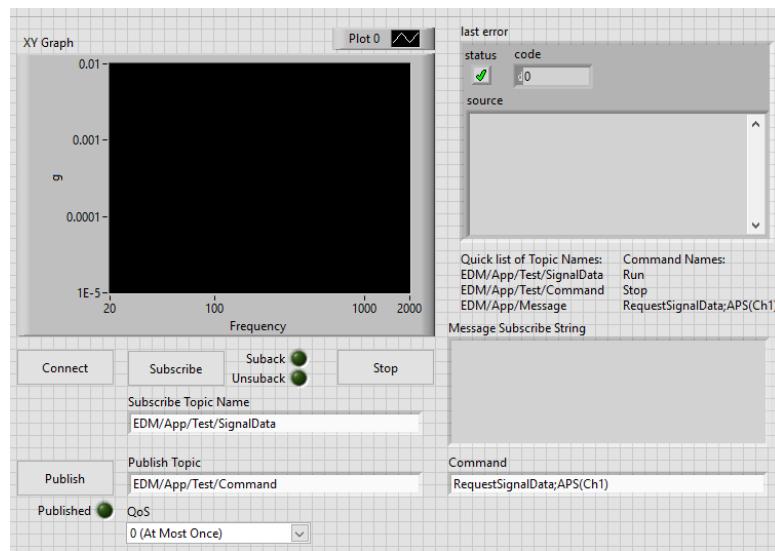
LabVIEW Front Panel



The Front Panel of the LabVIEW MQTT Client Demo has the connection parameter settings, XY graph for plotting signal data, error message box, subscribe and publish topic commands and arrays for X, Y and Z of the signal data.



The connection parameter settings have the payload, connection flags, client ID and TCP configuration that are required to connect to the EDM MQTT broker. It is noted that the connect flags for password and username must be checked when sending a payload with a username and password to connect to the EDM MQTT broker.



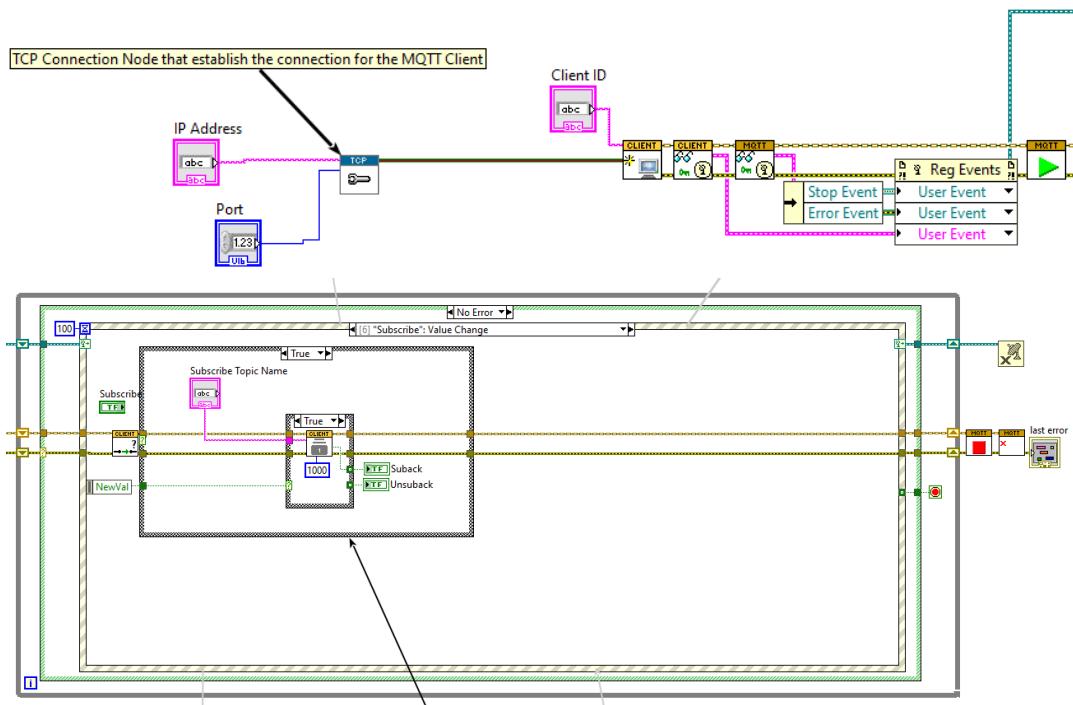
After inputting the correct connection parameter settings, clicking the Connect button will then attempt to connect the LabVIEW MQTT Client to the EDM MQTT broker. It may be necessary to reconnect if it doesn't succeed the first time.

Once the LabVIEW MQTT Client connects to the EDM MQTT broker, the client ID should show up in the EDM MQTT broker Connected clients tab.

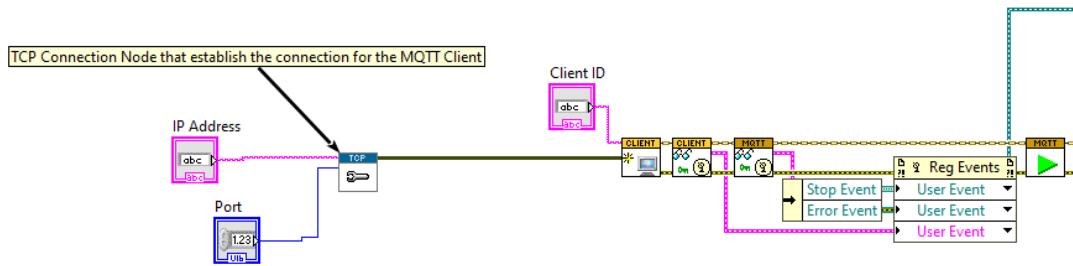
MQTT Broker Setting			
Broker setting	Connected clients	Broker log	
Client ID	User name	Protocol version	End point
LabVIEW MQTT Client	Admin	V311	192.168.3.15:56738
SPIDER_VCS-d4900cf1-ce7a-4065-... Admin		V311	192.168.3.15:54003

Then the LabVIEW MQTT Client is ready to subscribe or publish topics to EDM. Please note that since this is a basic LabVIEW MQTT Client Demo, the user may need to add additional displays to the front panel when requesting data such as the test status or input channel information.

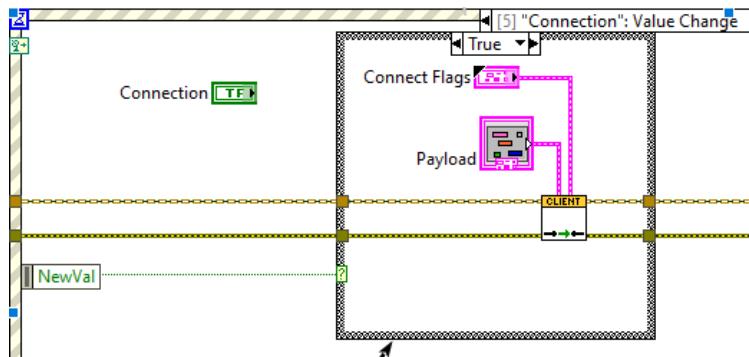
LabVIEW Block Diagram



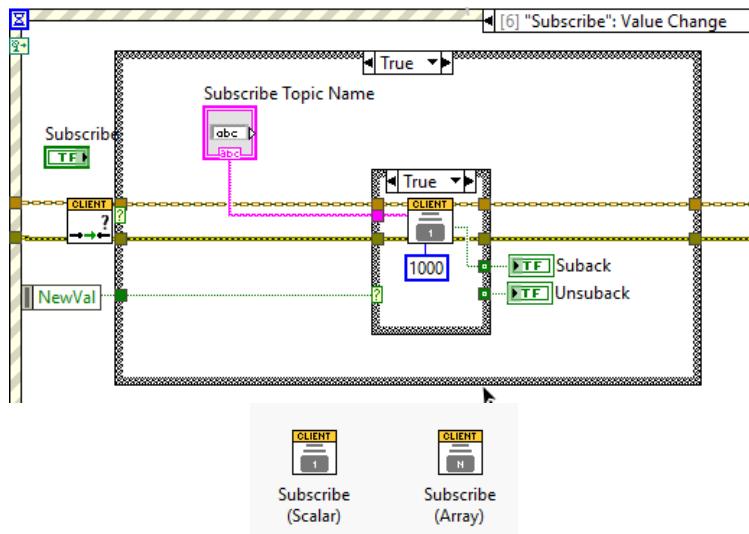
The above shows a brief overview the entire block diagram used in making the LabVIEW MQTT Client Demo, from the start of the MQTT client, the main function loop of every event to the disconnection and stop function of the MQTT client with any error messages if they occurred. The entire block diagram will also have comments pointing to various nodes to help users understand the MQTT package and the various parts used to make the demo.



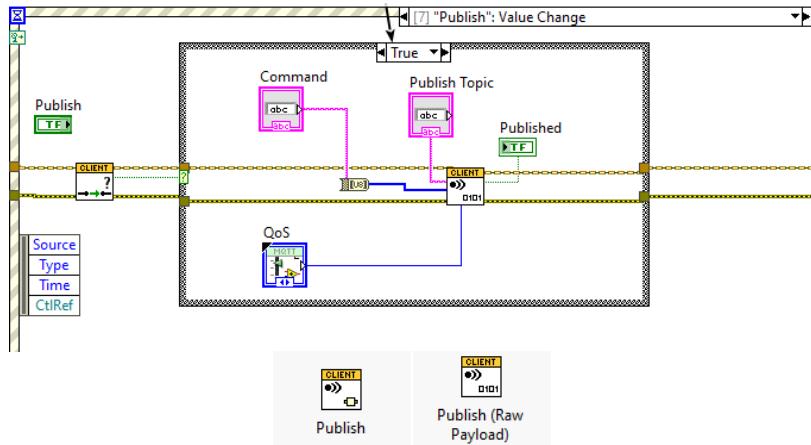
Starting with the left side of the block diagram is the TCP configuration and client ID control nodes for the Create MQTT Client node. At the top of each MQTT node is the MQTT Client in and out and at the bottom of each MQTT node is the error in and out. Both the MQTT Client and Error goes through each MQTT function node until the last MQTT Destroy node. After creating the MQTT client, it goes through the Read Public Event nodes and into the Start node.



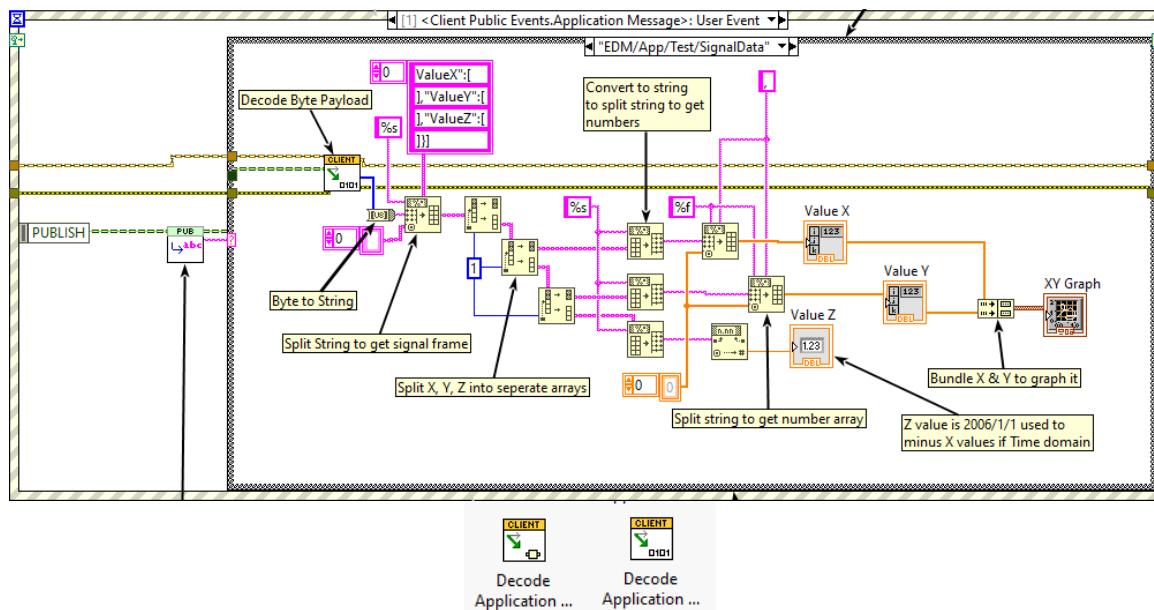
Next is the event structure after the Start node that encompasses all the functions of the LabVIEW MQTT Client Demo from subscribe, publish, connect, stop, public events and so on. Here is the connect button event structure where it takes in the payload and connect flags to connect to the EDM MQTT broker.



Here is the subscribe event structure where the Subscribe (Scalar) node takes in the topic name and timeout. It will stay subscribed to the inputted topic name unless the user press the subscribe button to unsubscribe. It is also possible to have multiple Subscribe (Scalar) nodes or a Subscribe (Array) node to subscribe to multiple topics.



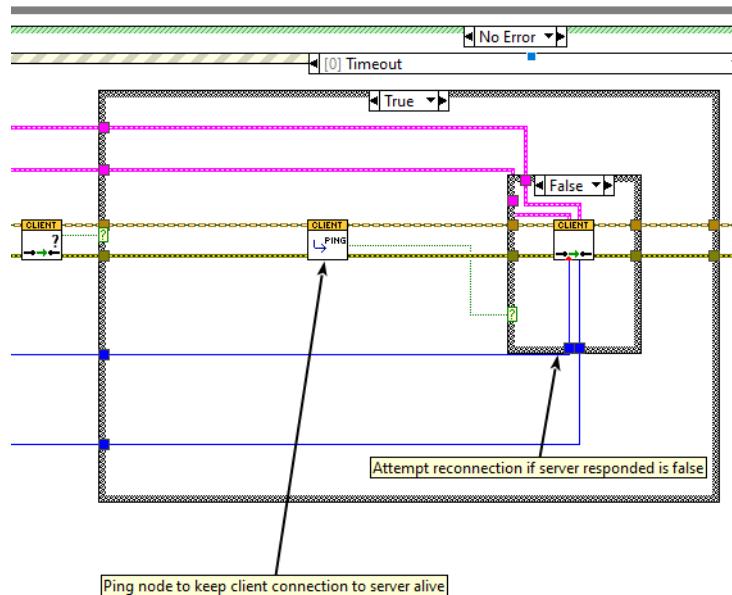
Here is the publish event structure where the Publish (Raw Payload) node takes in the topic name, command in bytes and quality of service. The publish node has to be raw payload in order for EDM to read in the command that comes from the LabVIEW MQTT Client Demo. Once the payload has been sent to EDM and EDM received the command correctly, it will send out the requested topic that the LabVIEW MQTT Client Demo should be subscribed to.



Once the subscribe topic sent by EDM has been received, it will then decode the payload in the public event structure. First, in this structure, it must determine the type of topic name to handle

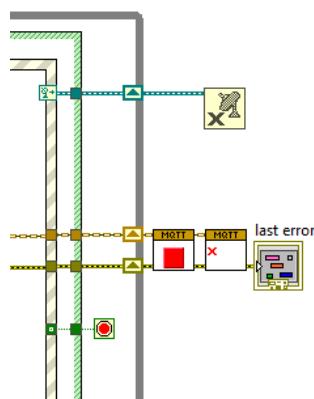
how to display the data from the `getTopicName` node on the left side. Once it determines the topic, it then decodes the payload in bytes.

In the case of a signal data request, the payload is a string that contains data about the signal from its name, signal frame data, create date and so on. For this demo, it extracts the signal frame data and plot the X and Y data points. The block diagram will have comments to explain the process of displaying the X and Y data points of the signal frame data.



In the Timeout event case, there is a client ping node that is needed to keep the client connection to the server alive. Without this node, the client will disconnect from the server after the keep alive time limit is reached.

The ping node also has a server responded boolean where if the server does not respond, the client can attempt to reconnect to the server.



Then the last portion of the block diagram is the Stop and Destroy nodes of the MQTT package, as well as the error out dialog box. This occurs whenever the demo encounters an error or if the user stops the demo.

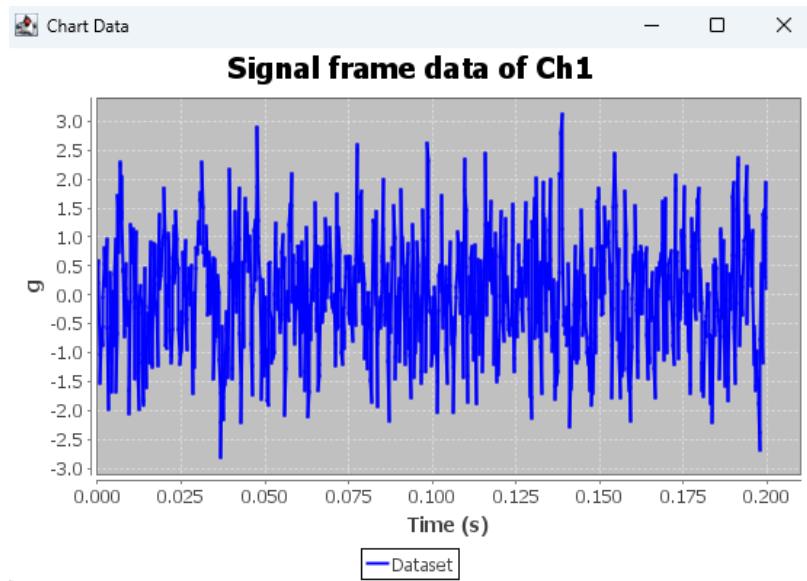
EDM MQTT Client Java Demo

A java code demo with MQTT has been created to showcase how to use MQTT in java to communicate with EDM MQTT broker and client with its topics and commands. The code was created and run from **IntelliJ**.

Please refer to the primary C# or Python demos in coding or understanding more about coding MQTT clients.

Java Scripts

In **RunningTestEx.java**, it shows grabbing a running test signal data and displaying it to a chart.



Java GUI app

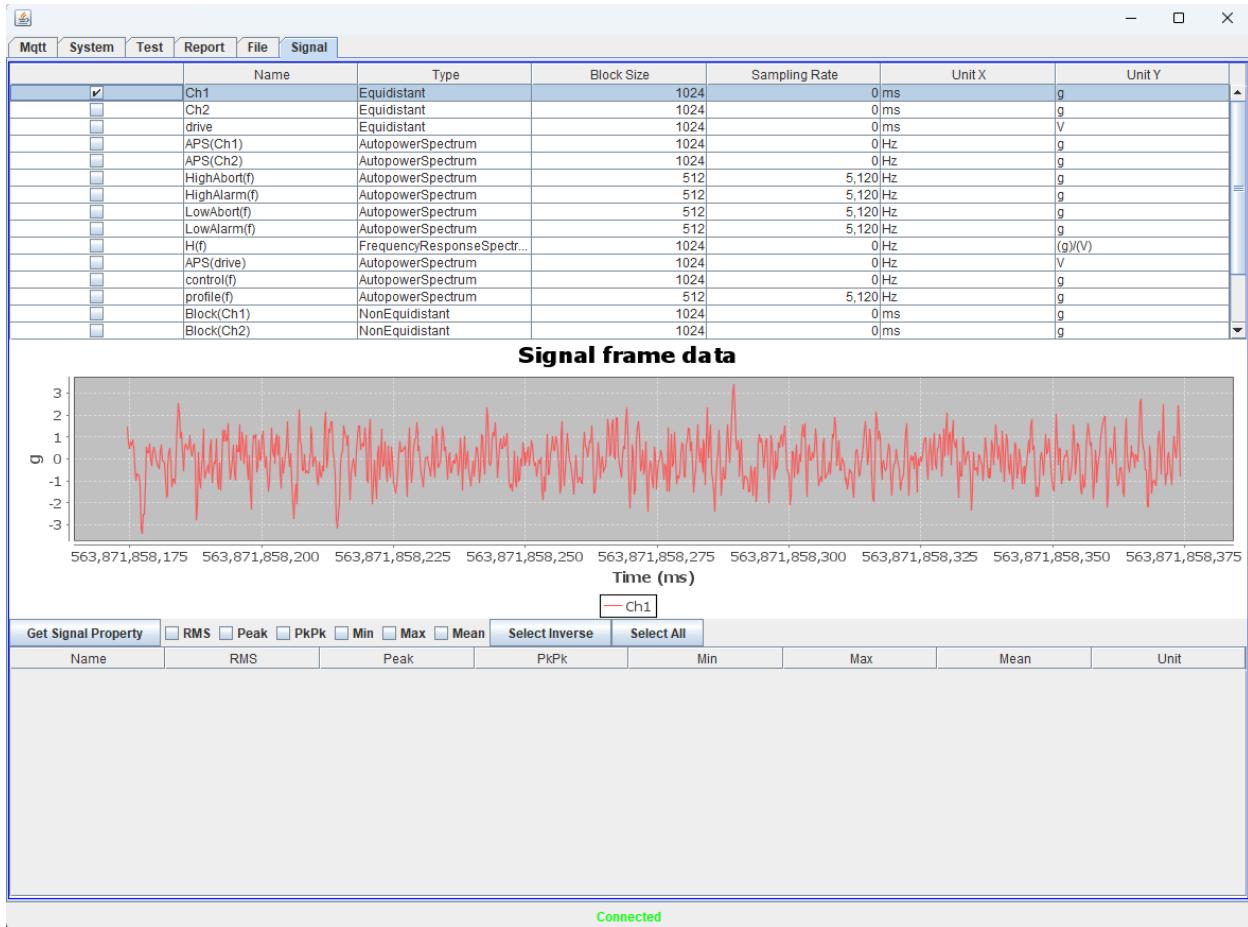
In **\MQTTJavaDemoCode\JavaSwingDemoMqtt** GUI
app\MQTTSharpProj\out\artifacts\MqttDemoCode_jar contains an executable jar file, called **MqttDemoCode.jar**, that is a GUI app that functions the same as the C# GUI demo app. This showcase that other coding languages can replicate the same functionality as the C# GUI demo app, as long as it uses the same EDM MQTT topics and commands.

It requires installing JDK 21 to run.

EDM MQTT Test Application

The application interface includes the following sections:

- Top Navigation Bar:** MQTT, System, Test, Report, File, Signal.
- Command Tab:** Command, Status, Detail Status, Channels, Shaker, List/Create/Load/Delete, Advanced Commands, Output.
- VCS Commands:**
 - Check Only, Proceed, Show Pretest, Save H Signal, Reset Average, Next Entry, Abort Checks On, Abort Checks Off, Closed Loop Co...
 - Closed Loop Co..., Schedule Clock..., Schedule Clock..., 0;1;0;1, RoR Bands On, RoR Bands Off, 0;1;0;1, SoR Tones On, SoR Tones Off
 - Tones Hold Swe..., Tones Release ..., Tones Sweep Up, Tones Sweep D..., Hold Sweep, Release Sweep, Sweep Up, Sweep Down, Level Up
 - Level Down, 0 ▲▼, Set Level, Restore Level, Increase Speed, Decrease Speed, 0 ▲▼, Set Frequency, 0 ▲▼
 - Set Phase, Inverse Pulse On, Inverse Pulse Off, Single Pulse On, Single Pulse Off, Output Single P...
- DSA Commands:**
 - Trigger On, Trigger Off
 - Output On, Output Off
 - Limit On, Limit Off
- Status:** Connected
- System Tab:**
 - App Information:** Software Mode: SPIDER_VCS, Version: 11.1.0.1
 - System Information:** Name: SYS_2605536, Status: Connected, Device Type: Spider80X, Serial Number: 2605536, IP Address: 192.168.1.152, Hardware Version: 7.5.9



EDM MQTT Client C++ Demo

A C++ code demo with MQTT has been created to showcase how to use MQTT in C++ to communicate with EDM MQTT broker and client with its topics and commands. The code was created and run from Visual Studio, but the code can be run on Windows or Linux.

Dependencies:

Windows:

1. Visual Studio 2022

Linux:

1. g++
2. make

C++ CLI app

Windows:

Within \MQTTCPPExample, there is a C++Mqtt.sln file which can be used to build and run the application.

Linux:

Within \MQTTCPPExample\C++Mqt\ there is a Makefile which will compile and create the application executable. Once the executable is created, it can be run.

Once the application is running, the user will need to connect to the application.

Connecting

To connect to the MQTT Broker, the user will enter the following information when prompted:

1. Host: IP address of the MQTT broker
2. Port: Port of the MQTT broker
3. Username: Username specified when initializing the MQTT broker
4. Password: Password specified when initializing the MQTT broker
5. Topic Prefix: Topic prefix specified when initializing the MQTT client

See EDM MQTT Broker and EDM MQTT Client for information on where to find these values.

Once the user is connected, they will be notified, and they can begin writing commands.

```
Please enter host: 10.0.40.41
Please enter port: 1883
Please enter username: Admin
Please enter password: 123456
Please enter topic prefix: EDM
Waiting for connection to host 10.0.40.41:1883
Connected!
```

Commands

Run: Runs the open test

Parameters: None

Reference:

```
The commands you can perform are: Run (Runs Current Test), Stop (Stops Current Test), Get (Gets Signal Data), Publish (Publishes a Topic and Payload), Subscribe (Subscribes to a Topic), and Exit (Exits Program). Please type the action you want to perform.
```

```
run
```

Stop: Stops the running test

Parameters: None

Reference:

```
The commands you can perform are: Run (Runs Current Test), Stop (Stops Current Test), Get (Gets Signal Data), Publish (Publishes a Topic and Payload), Subscribe (Subscribes to a Topic), and Exit (Exits Program). Please type the action you want to perform.
```

```
stop
```

Get: Creates a file containing the APS of Channel 1 of the Spider 80x

Parameters: None

Reference:

```
The commands you can perform are: Run (Runs Current Test), Stop (Stops Current Test), Get (Gets Signal Data), Publish (Publishes a Topic and Payload), Subscribe (Subscribes to a Topic), and Exit (Exits Program). Please type the action you want to perform.
```

```
get
```

Subscribe: Subscribes the MQTT Client to a topic

Parameters:

Topic – the topic that the client will be subscribed to

Reference:

```
The commands you can perform are: Run (Runs Current Test), Stop (Stops Current Test), Get (Gets Signal Data), Publish (Publishes a Topic and Payload), Subscribe (Subscribes to a Topic), and Exit (Exits Program). Please type the action you want to perform.  
subscribe  
Please enter the subscribe topic: GlobalParameter/Response
```

Publish: MQTT Client publishes a payload to a topic**Parameters:**

Topic – the topics that the client will publish to

Payload – the payload that will be published

Reference:

```
Please enter the subscribe topic: GlobalParameter/Response  
The commands you can perform are: Run (Runs Current Test), Stop (Stops Current Test), Get (Gets Signal Data), Publish (Publishes a Topic and Payload), Subscribe (Subscribes to a Topic), and Exit (Exits Program). Please type the action you want to perform.  
publish  
Please enter the publish topic: GlobalParameter/Request  
Please enter the payload: Test.Run.Parameter.ShakerName
```

Exit: Exits the program**Parameters:** None**Reference:**

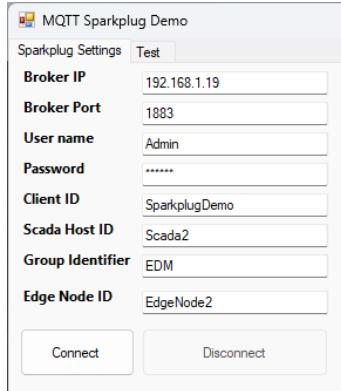
```
The commands you can perform are: Run (Runs Current Test), Stop (Stops Current Test), Get (Gets Signal Data), Publish (Publishes a Topic and Payload), Subscribe (Subscribes to a Topic), and Exit (Exits Program). Please type the action you want to perform.  
exit
```

EDM MQTT C# Sparkplug Demo

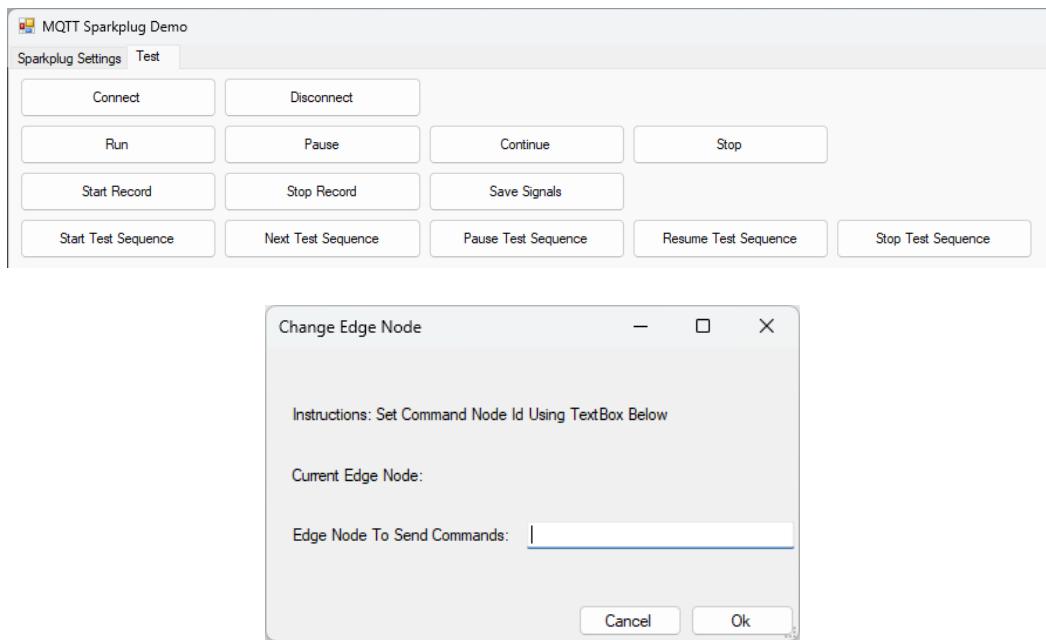
A C# MQTT demo with sparkplug enabled has been created to showcase how a sparkplug application can communicate with EDM. Currently the demo only supports controlling EDM DSA FFT tests. The code has been created with Visual Studio. Please refer to the primary C# or Python demos in coding or understanding more about coding MQTT clients.

C# Demo App

\MQTTSparkplugDemoEXERelease contains the executable for the demo application. Currently the demo contains basic commands, but Sparkplug has the same capabilities as the non-Sparkplug enabled MQTT Client demos.



Upon running the executable, the user can enter in the necessary information and connect to the EDM MQTT broker. Once connected, there are some basic commands for running a test and recording data.



Before the command is sent to the EDM MQTT Broker, the user must specify the **Edge Node** to send the command to. Commonly, this is the EDM MQTT Client, and the edge node id can be found on the EDM MQTT Client Sparkplug tab.

C# Demo Code

The solution file to open the project containing the source code in Visual Studio can be found in **\MQTTSparkplugDemo**

Project Structure

The source code files within the **MQTTSparkplugCSharpExample** project include:

- **MQTTSparkplugExample.cs** – Contains the main command line application.

- **MQTTSparkplugForm.cs** - Contains the GUI for the user to send commands to EDM
- **MQTTSparkplugSupport.cs** - Contains the names of the metrics that the Sparkplug Application expects to be sent
- **ChangeCommandEdgenodeForm.cs** – Contains the GUI and code for the user to change the Edge Node that commands are sent to

Dependencies

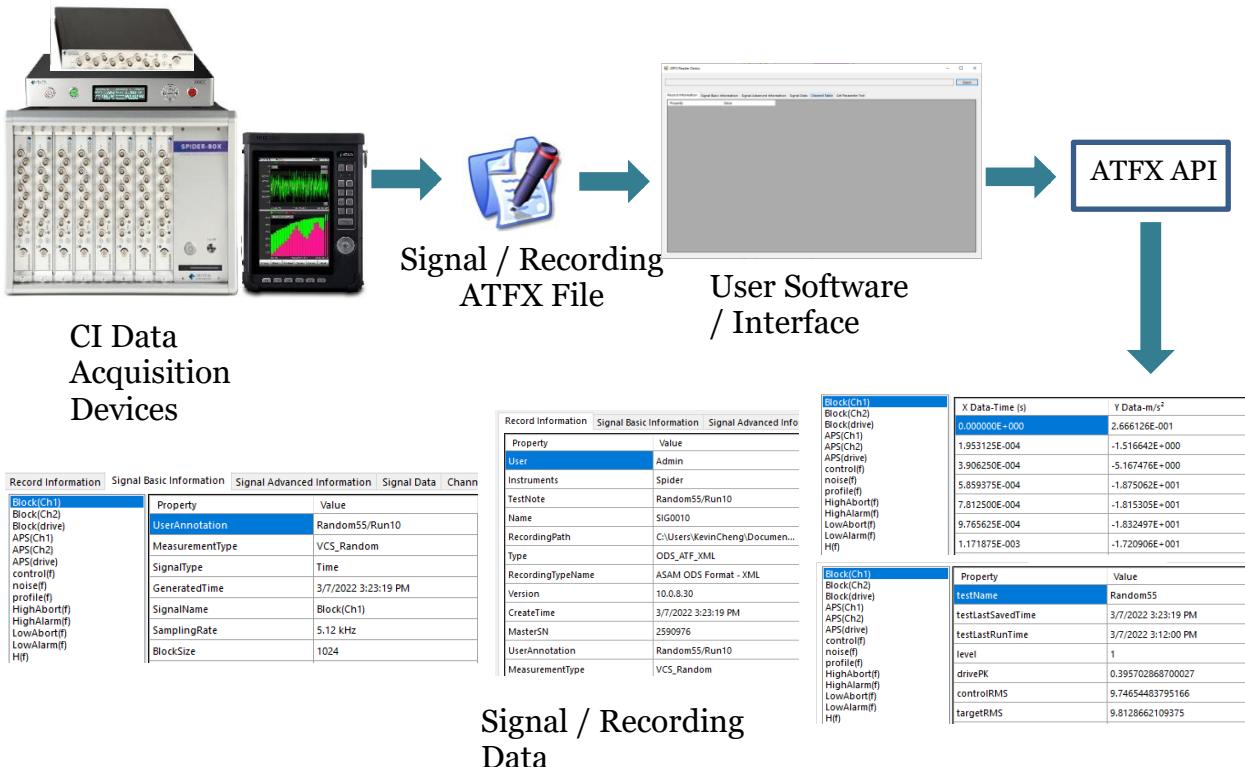
Note: dependencies should already be installed and added to the project

1. Protobuf-net.Core - Version 3.0.0
2. Protobuf-net - Version 3.0.0
3. System.Collections.Immutable - Version 6.0.0
4. System.Memory - Version 4.0.2.0

Build Steps

1. Open the MQTTSparkplugDemo.sln file
2. If there are issues with the SparkplugNet, Serilog, or MQTTnet references, remove them, and readd them using the dlls included in the bin folder
3. If there are issues with the protobuf-net, protobuf-net.Core, System.Memory, or System.Collections.Immutable references, right click on the References node, click **Manage NuGet Packages**, and click **restore** at the top if prompted
 - a. If you are not prompted to restore packages, or if restoring does not fix the issue, ensure the packages listed in the dependencies section are installed
4. Right click the MQTTSparkplugDemo solution and click **Rebuild**

How the User's Software Reads CI Data Files (ATFX API)



The Crystal Instruments (CI) ATFX ODS Signal Reader Application Programming Interface (API) consists of 2 Windows Dynamic-Linked Libraries (DLL) providing third-party applications an interface to access the signal data stored in the ASAM Transport Format XML (ATFX) files.

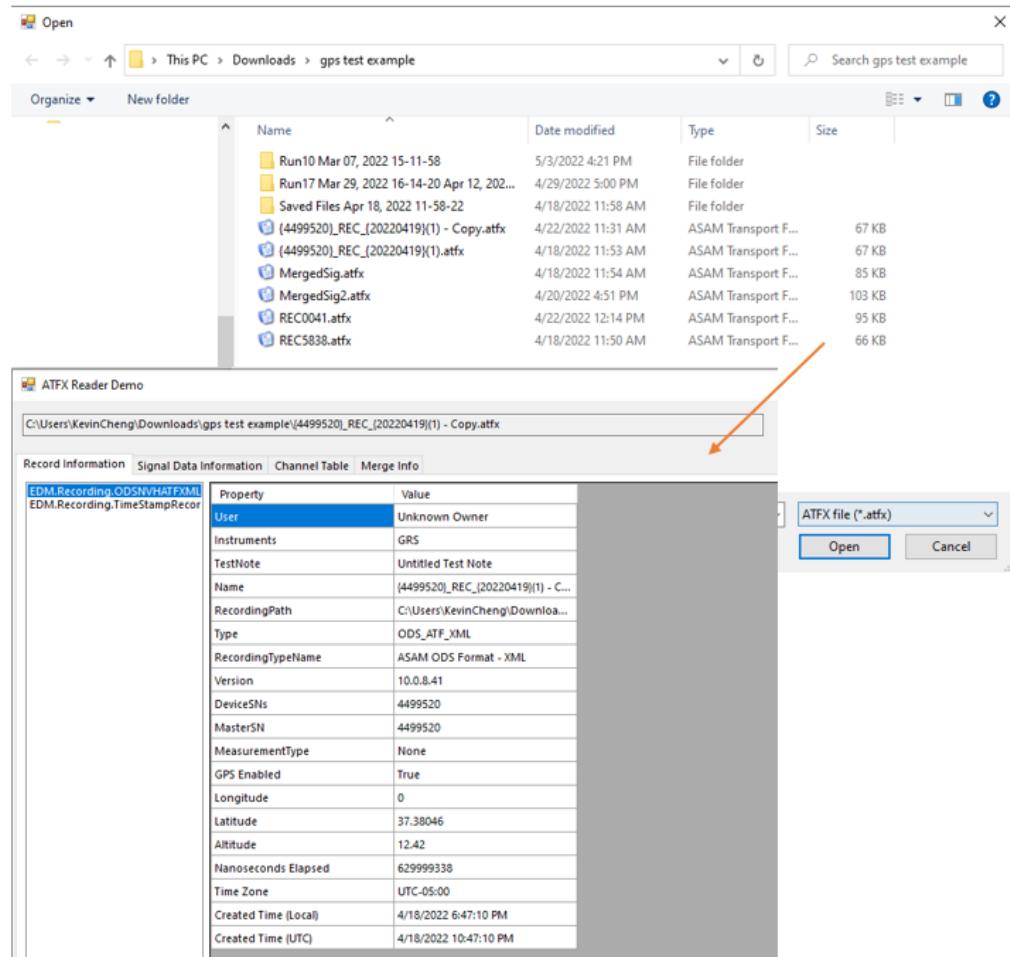
ATFX files are formatted according to the Association for Standardization of Automation and Measuring Systems (ASAM) Open Data Services (ODS) standardization. This is a standard dedicated for storing vibration data and its different forms. CI software natively stores its data using the ATFX format, for both signals and recordings.

For details about the ATFX ODS format please refer to the official website:
<https://www.asam.net/standards/detail/ods/wiki/>

The .atfx files are xml-based files which store the signal data along with all the attributes of the signal data including recording properties and time created, length of recording, number of channels, channel parameters (e.g., input channel sensor and sensitivities), geographic coordinates, sampling rate, high pass filter, etc. The .atfx files contain a reference to a .dat file that are well-defined for storing both raw time data as well as processed spectral data, calculated from methods including Fourier Transform, Frequency Response Functions, Cross-Power Spectrum, Octave Spectrum, etc.

There are 2 additional file types that the .atfx file references that contain raw data: .ts and .gps. The .ts file is a TimeStamp recording that contains an accurate measure of when a recording was

saved with accuracy down to nanoseconds. The .gps file is a GPS recording that contains locational data of where a recording was saved (e.g., latitude, longitude, altitude).



The CI Data File Reader API provides end-users with a streamlined file reading and browsing library to decode ATFX, TS and GPS files. Users can integrate the API with their own custom developed application. Currently, we support Windows-based programs, ideally written in C#. The same API also supports Python, MatLab and LabView.

The API offer direct calls to the ASAM ODS model classes and objects used to store data saved in the ATFX file, such as calling the recording NVHMeasurement and NVHEnvironment to read the DateTime with nano seconds elapsed. The API also provides a Utility class that has functions to return data from the ATFX file without the user needing to understand the complexity of the ASAM ODS model classes. Such as the Utility GetListOfAllSignals that return a list of signals that a ATFX file contains or the Utility GetChannelTable that return a 2D list of strings, where each list is an input channel row.

It is also possible to read any of the signals, time or frequency, in other engineering units (EU), such as Acceleration m/s² to g. As well as reading frequency domain signals in other spectrum types, such as EURms to EUPeak. All done by the signal function GetFrame where users can pass in parameters to return a converted signal frame data saved in the ATFX file.

When the ATFX API read the ATFX file, there may be some differences in the signal frame data, this is due to some display related parameters such as spectrum type not being saved into the ATFX file. By default, the spectrum type is EURms². Engineering units are saved into the ATFX file and should be the the default EU when reading the signal frame.

For more detailed information regarding the ATFX API, please refer the CI Data File Reader API Documentation that can be downloaded via the CI site:

<https://www.crystalinstruments.com/basics-of-test-and-measurement>

<https://www.crystalinstruments.com/programming-corner>

CRYSTAL INSTRUMENTS CORPORATION SOFTWARE LICENSE AGREEMENT

THIS SOFTWARE LICENSE AGREEMENT (“**AGREEMENT**”) IS A LEGALLY BINDING CONTRACT BETWEEN CRYSTAL INSTRUMENTS CORPORATION (“**CRYSTAL INSTRUMENTS**”) AND YOU (“**LICENSEE**”). IF YOU ARE ACCESSING OR USING THE SOFTWARE ON BEHALF OF A CORPORATION OR OTHER ENTITY, THEN REFERENCES TO LICENSEE IN THIS AGREEMENT REFER TO THAT ENTITY. PLEASE READ THIS AGREEMENT CAREFULLY. BY DOWNLOADING, INSTALLING, OR OTHERWISE USING THE CRYSTAL INSTRUMENTS SOFTWARE, LICENSEE ACCEPTS AND AGREES TO BE BOUND BY THIS AGREEMENT. IF LICENSEE DOES NOT AGREE TO ALL OF THE TERMS OF THIS AGREEMENT, LICENSEE IS NOT AUTHORIZED TO USE THE SOFTWARE.

1. **Software.** This Agreement applies to Crystal Instruments software (the “**Software**”) which may include but may not be limited to EDM (Engineering Data Management) software, PA (Post Analyzer), EDM Cloud, CI Store, EDC (Embedded Device Control), and software that is embedded or installed in Crystal Instruments CoCo and Spider series hardware, as any such software may be updated, upgraded or modified from time to time by Crystal Instruments.
2. **License Grant.** Subject to the terms and conditions of this Agreement, Crystal Instruments grants to Licensee a perpetual, non-exclusive, non-transferable license to access, download, install, and use the executable version of the Software for which Licensee has purchased a license, solely for Licensee’s own internal use, solely with the Crystal Instruments hardware for which it is intended (“**Products**”), and solely in accordance with the documentation and user manuals provided by Crystal Instruments relating to the hardware or Software (“**Documentation**”). Notwithstanding the above, if Licensee’s order indicates that the Software license is for a limited term rather than perpetual (e.g., for Software provided for demonstration, evaluation or other similar purposes), then the license shall terminate at the end of the stated term and Licensee acknowledges that the license key provided to Licensee to enable use of the Software may cease to allow use of the Software after expiration of such term.
3. **Restrictions.** Licensee agrees not to directly or indirectly do any of the following, or direct or allow anyone else to do any of the following:
 - (a) disassemble, reverse compile or reverse engineer any part of the Software, attempt to create the source code from the object code for the Software, or otherwise attempt to discover any underlying ideas or algorithms used by the Software, except to the extent that such activities cannot be prohibited by applicable law;
 - (b) copy the Software except as a necessary step in the authorized use of the Software, and except for back-up copies of the Software to the extent that applicable law expressly requires be permitted;
 - (c) copy the Documentation except as reasonably necessary for proper use of the Software;
 - (d) modify, or prepare derivative works based on the Software or Documentation;
 - (e) rent, lease, distribute, publish, host, disclose, or otherwise commercially exploit the Software or Documentation, or make the Software or Documentation available to any third party;
 - (f) remove, alter, or obscure any product identification, copyright, trademark, or other intellectual property notices included in the Software or Documentation, or attempt to break, change or delete any security codes or license keys associated with the Software;
 - (g) exceed the number of users specified in Licensee’s order for the Software, if the number of users or seats is limited.
4. **Ownership.** Licensee acknowledges and agrees that the Software and Documentation are licensed, and not sold, to Licensee. Crystal Instruments and its licensors retain the entire right, title and interest in and to the Software and Documentation and all intellectual property or proprietary rights thereto, including but not limited to all worldwide rights under patent, copyright, trademark and trade secret laws. Licensee does not acquire any ownership interest in the Software or Documentation, or any other rights thereto other than the license rights expressly granted in this Agreement. Any rights not expressly granted by Crystal Instruments in this Agreement are reserved by Crystal Instruments. Licensee acknowledge that the Software and Documentation are confidential information of Crystal Instruments and Licensee agrees to inform its users of the restrictions to which Licensee is subject pursuant to this Agreement.
5. **Software Support.**
 - (a) Error correction services and the provision of updates relating to the Software, as described in this Section (collectively, “**Support**”) is included for the first twelve (12) months after Licensee first accesses the Software, without charge beyond the license fee for the Software. Thereafter, Licensee may purchase Support, for an additional twelve (12) months at a time, by paying the Support renewal fee in effect at the time of the renewal. If Support for Licensee lapses, and Licensee wishes to reinstate Support, Licensee must pay the Support renewal fee then in effect plus all Support renewal fees that would have been paid for the period during which Support had lapsed.

- (b) For the period during which Support is in effect for Licensee, if Licensee notifies Crystal Instruments of a material defect in the Software that prevents the Software from performing a material function described for the Software in the applicable Documentation (a “**Defect**”), Crystal Instruments will, at its discretion, without charge: (a) deliver a new version of the Software in which the Defect is corrected, or (b) deliver a patch to correct the Defect, or (c) provide Licensee with instructions for procedures or methods (workarounds) which result in the Defect not having a significant effect on Licensee’s use of the Software. If Crystal Instruments fails to do any of the above within 30 days of Licensee’s notice (or such longer period of time as is reasonably necessary given the nature of the Defect), then as Licensee’s sole remedies, Licensee may (i) terminate Support upon notice to Crystal Instruments, in which event Crystal Instruments will refund to Licensee a pro-rated portion of the Support fee paid by Licensee for the then-current 12-month Support period, based on the number of days in the Support period elapsed and remaining as of the date of Licensee’s notice, and (ii) if Licensee’s notice of the Defect was given during the first twelve (12) months after Licensee first accessed the Software, then Licensee may terminate this Agreement upon notice to Crystal Instruments at any time before Crystal Instruments corrects the Defect, in which event Crystal Instruments will also refund to Licensee a pro-rated portion of the Software license fee paid by Licensee, pro-rated over such first twelve (12) month period, based on the number of days in such period elapsed and remaining as of the date of Licensee’s notice, provided that Licensee returns to Crystal Instruments all of Licensee’s versions and copies of the Software, and all Documentation. This paragraph states the sole obligations of Crystal Instruments, and the sole remedies of Licensee, for Defects in the Software or Crystal Instruments’ failure to correct Defects.
 - (c) For the period during which Support is in effect for Licensee, Crystal Instruments will provide Licensee, free of additional charge, with any updates to the Software that Crystal Instruments makes generally available to its customers, to correct errors or improve performance of the Software (“**Updates**”). Notwithstanding the above, Crystal Instruments may charge an additional license fee for any optional upgrades Crystal Instruments may release that include significant new functionality and which Crystal Instruments does not make available without charge to its customers generally (“**Upgrades**”). Licensee acknowledges that if Licensee elects not to renew Support, the license key provided to Licensee to enable use of the Software, or other technical means employed by Crystal Instruments, may thereafter cease to allow installation and use of further Updates or Upgrades.
6. **Third-Party Software.** The Software includes certain third-party software, which Crystal Instruments has the right to distribute and license to Licensee. The third-party software may include software that is subject to certain open source licenses (“**Open Source Software**”). The Open Source Software components are licensed to Licensee by the owners of the rights to the Open Source Software, pursuant to license agreements separate from this Agreement. A list of the Open Source Software incorporated in the Software, and the licenses that apply to the Open Source Software, is available to users of the Products upon request. The license agreements relating to the Open Source Software are incorporated by reference in this Agreement and, by using the Software, Licensee is accepting such license agreements. Crystal Instruments shall have no liability of any nature relating to software or content of third parties that may be included in the Software.
7. **Selection of Software.** Licensee is responsible for selecting the Software package that is appropriate for Licensee’s intended use. Any advice or assistance provided by Crystal Instruments in connection with Licensee’s selection of Software is based upon information provided by Licensee, which is necessarily incomplete. Crystal Instruments gives no warranty that the Software selected by Licensee will meet Licensee’s requirements or be compatible with Licensee’s other hardware, software or systems.
8. **Consent to Use of Data.** Licensee agrees that Crystal Instruments and its affiliates may, through Internet connections established by the Software or otherwise, collect technical information related to Licensee’s use of the Software, including but not limited to the serial numbers of Crystal Instruments’ Products with which the Software is used, email addresses of users, and technical information relating to Licensee’s computers, systems, application software, and peripherals. Licensee agrees that Crystal Instruments may use such information to facilitate the provision of Updates to the Software and product support, to improve Crystal Instruments’ products and/or services, or to provide products or services to Licensee. Crystal Instruments will not, however, publish or disclose such information in a form that may personally identify Licensee or any individual user. If Licensee provides any suggestions or feedback to Crystal Instruments regarding the Products, Software or Documentation, Crystal Instruments will have the right to use the suggestions or feedback to improve its Products, Software or Documentation without compensation to Licensee.
9. **API License.** If Licensee has ordered and paid for a license to Crystal Instruments’ application programming interfaces (“**APIs**”), Crystal Instruments grants to Licensee a non-exclusive, non-transferable license (without rights to sublicense) during the term of Licensee’s license to the Software, to access and use the APIs solely for the purpose of integrating Crystal Instruments’ Products purchased by Licensee with Licensee’s systems, to enable such systems to capture data generated by the use of the Products. Licensee acknowledges and agrees that Crystal Instruments retains the entire right, title and interest in and to the APIs and all intellectual property or proprietary rights thereto. Licensee acknowledge that the APIs are confidential information of Crystal Instruments. Licensee agrees not to, and not to direct or allow anyone else to (a) disassemble, reverse compile or reverse engineer the APIs, or otherwise attempt to discover any underlying scripts that support the APIs, (b) modify, or prepare derivative works based on the APIs, (c) rent, lease, distribute, publish, host or disclose the APIs, or otherwise make the APIs available to any third party. Upon termination of Licensee’s license to the Software, the license to the APIs shall immediately terminate, Licensee shall cease using the APIs, and Licensee shall return to Crystal Instruments or destroy all copies of the APIs.
10. **Disclaimer of Warranties.** TO THE MAXIMUM EXTENT PERMITTED BY LAW, CRYSTAL INSTRUMENTS, FOR ITSELF AND ITS LICENSORS, DISCLAIMS ALL WARRANTIES, EXPRESS, IMPLIED, STATUTORY OR OTHERWISE, WITH RESPECT TO THE SOFTWARE AND DOCUMENTATION, INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE AND NON-INFRINGEMENT, AND WARRANTIES THAT MAY ARISE OUT OF COURSE OF DEALING, COURSE OF PERFORMANCE, USAGE OR TRADE PRACTICE.

CRYSTAL INSTRUMENTS DOES NOT WARRANT THAT USE OF THE SOFTWARE WILL BE UNINTERRUPTED OR ERROR-FREE.

11. **Limitation of Liability.** IN NO EVENT WILL CRYSTAL INSTRUMENTS OR ITS LICENSORS BE LIABLE FOR ANY SPECIAL, INCIDENTAL, INDIRECT, EXEMPLARY, PUNITIVE OR CONSEQUENTIAL DAMAGES OF ANY NATURE RELATING TO OR ARISING OUT OF THIS AGREEMENT, THE SOFTWARE, OR THE DOCUMENTATION, INCLUDING BUT NOT LIMITED TO LOSS OF PROFITS, LOSS OF USE, LOSS OF GOODWILL, OR LOSS OF DATA, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE. IN NO EVENT WILL CRYSTAL INSTRUMENTS' AGGREGATE LIABILITY ARISING OUT OF OR RELATED TO THIS AGREEMENT, THE SOFTWARE, OR THE DOCUMENTATION, WHETHER BASED UPON WARRANTY, CONTRACT, TORT, OR OTHERWISE, EXCEED THE AMOUNT LICENSEE HAS PAID FOR THE LICENSE TO USE THE SOFTWARE THAT GAVE RISE TO THE CLAIM. THE LIMITATIONS OF LIABILITY AND WARRANTY DISCLAIMERS SET FORTH IN THIS AGREEMENT ARE FUNDAMENTAL ELEMENTS OF THE BASIS OF THE AGREEMENT BETWEEN LICENSEE AND CRYSTAL INSTRUMENTS AND SHALL APPLY NOTWITHSTANDING ANY FAILURE OF ESSENTIAL PURPOSE OF ANY REMEDY.
12. **Assignment.** Licensee may not transfer, assign, sublicense, pledge, lease, rent, or share its license to the Software, or its other rights under this Agreement, to or with another person or entity, including by operation of law (collectively, an "**Assignment**"), without the prior written consent of Crystal Instruments, which Crystal Instruments shall have the right to grant or withhold in its sole discretion. Any attempted Assignment made without the prior written consent of Crystal Instruments will be void and will constitute a breach of this Agreement. For these purposes, an Assignment will be deemed to include, but not be limited to (a) a transfer or assignment to another person or entity, including in connection with the acquisition of all or substantially all of Licensee's assets, (b) a merger, consolidation or other business combination transaction of Licensee with or into another person or entity, and (c) the transfer of a majority of the outstanding equity interests of Licensee in a transaction or series of related transactions.
13. **Termination.** Crystal Instruments may terminate the licenses granted under this Agreement upon notice to Licensee if Licensee violates this Agreement. Upon any termination of the license granted under this Agreement, Licensee must immediately discontinue use of the Software and Documentation, return (or at Crystal Instruments' option destroy) the Software and Documentation and all copies thereof and, if requested by Crystal Instruments, certify in writing as to such return or destruction. Fees paid for the Software license and Support are not refundable except to the extent otherwise provided in Section 5(b).
14. **Government Users.** The Software and Documentation are "Commercial Items", as that term is defined at 48 C.F.R. §2.101, consisting of "Commercial Computer Software" and "Commercial Computer Software Documentation", as such terms are used in 48 C.F.R. §12.212 or 48 C.F.R. §227.7202, as applicable. The Software and Documentation are being licensed to U.S. Government end users (a) only as Commercial Items and (b) with only those rights as are granted to all other end users pursuant to the terms and conditions herein.
15. **Export.** The Crystal Instruments Products, Software and related technology may be subject to U.S. export control laws and may be subject to export or import regulations in other countries. Licensee agrees to comply fully with all such laws and regulations of the U.S. and other countries ("**Export Laws**") and to ensure that the Products and Software are not exported, directly or indirectly, in violation of Export Laws, either to any countries that are subject to U.S. export restrictions or to any end user who has been prohibited from participating in the U.S. export transactions by any federal agency of the U.S. government. Licensee agrees to indemnify and hold Crystal Instruments harmless from any and all claims, losses, liabilities, damages, fines, penalties, costs and expenses (including attorney's fees) arising from or relating to any breach by Licensee of this paragraph.
16. **Governing Law.** This Agreement will be governed by the laws of the State of California, United States of America, without giving effect to any conflict of laws principles. The United Nations Convention on Contracts for the International Sale of Goods and the Uniform Computer Information Transactions Act shall not apply to this Agreement. Licensee and Crystal Instruments agree that any suit or proceeding arising out of or relating to this Agreement may be brought only in the state court of Santa Clara County, California, U.S.A., or the federal court of the Northern District of California, U.S.A., and they consent to the exclusive personal and subject matter jurisdiction and venue of those courts; provided, however, that nothing shall restrict Crystal Instruments from seeking relief to protect its intellectual property rights in any court of competent jurisdiction.
17. **General Provisions.** This Agreement may not be amended except by a written instrument signed by both parties. No waiver of a breach or default shall be effective unless evidenced by a writing signed by Crystal Instruments. No waiver of a breach or default shall be deemed a waiver of any subsequent breach or default. This Agreement constitutes the entire understanding, agreement, and contract of the parties with respect to its subject matter and supersedes all prior agreements or understandings, written or oral, between the parties with respect thereto. If the application of any provision of this Agreement shall be held to be invalid or unenforceable by any court of competent jurisdiction, then (a) to the extent feasible, that provision will be reformed in a manner that makes it enforceable and which accomplishes the objectives of that provision as nearly as possible, and (b) the validity and enforceability of other provisions of this Agreement will not be affected or impaired thereby. Licensee agrees that this License Agreement will not be construed against Crystal Instruments by virtue of having drafted it.