EDM MQTT PROTOCOL

November 8, 2022

Document ver. 1.4

© Crystal Instruments Corporation

Contents

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 © 2005-2022 Crystal Instruments Corporation. All rights reserved.
All trademarks and registered trademarks used herein are the property of their respective holders.

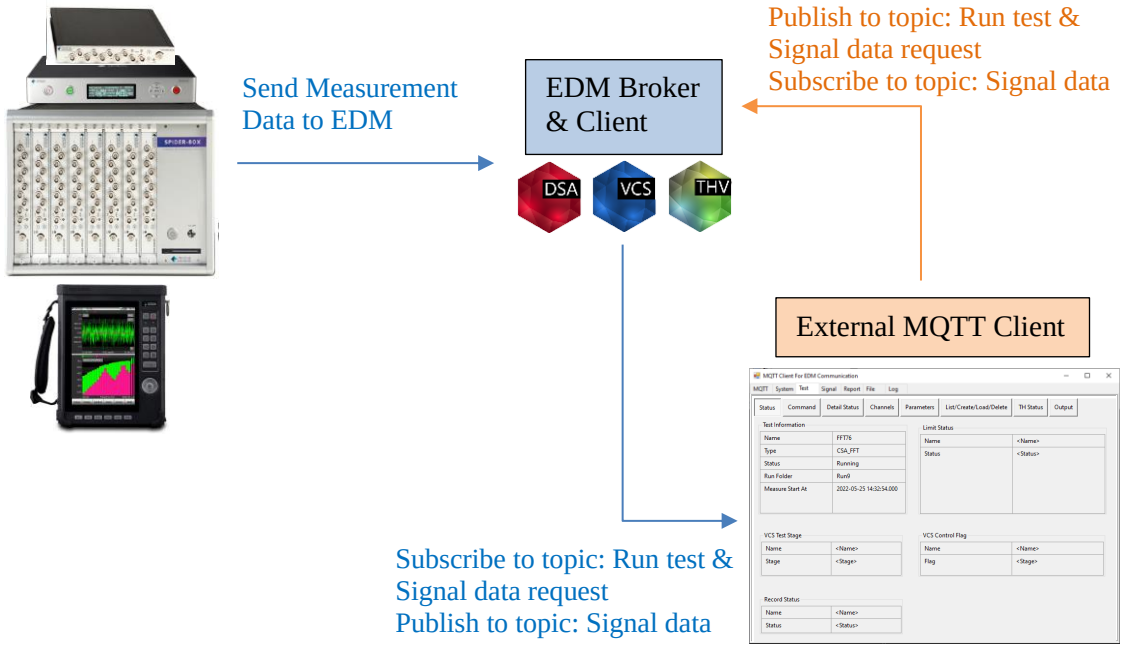
EDM MQTT

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 or send command to an EDM MQTT client

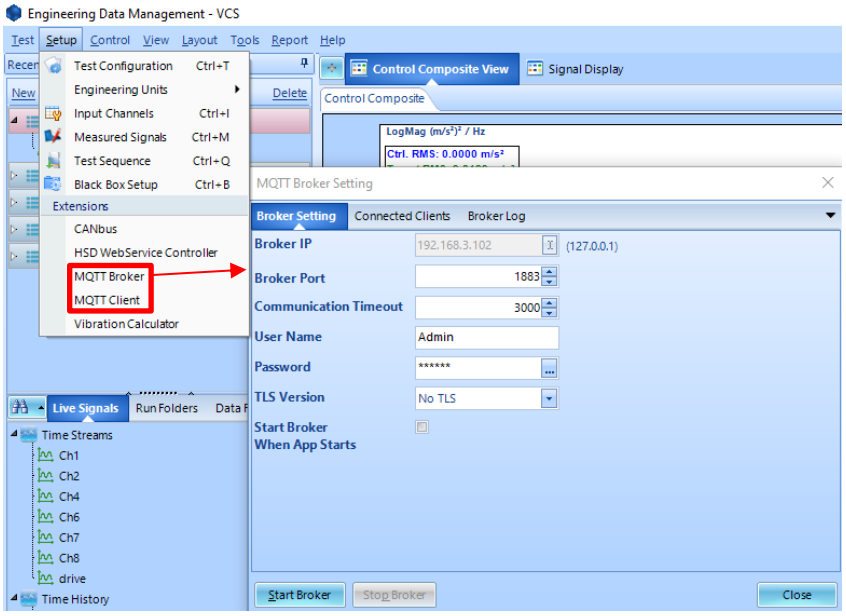
MQTT broker can be

- 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 built-in MQTT Client and MQTT Broker at the same time, available in EDM VCS, EDM DSA, and EDM THV. 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.**

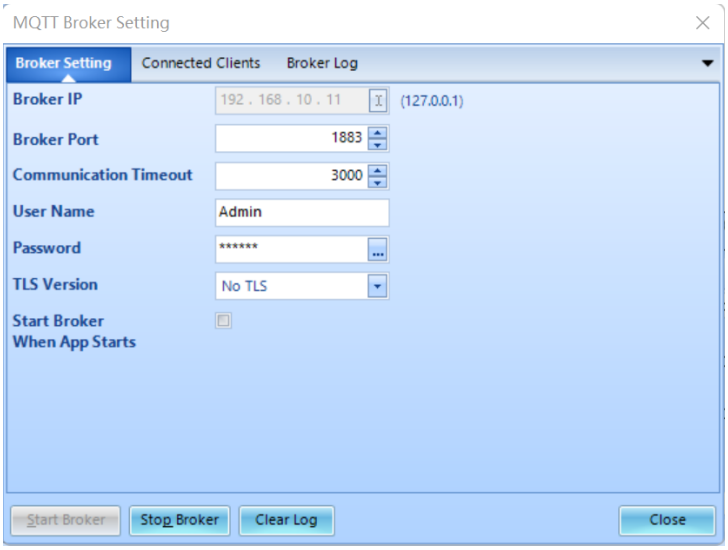


EDM MQTT Broker

When all MQTT clients are in the same **local area network**, users can enable the built-in MQTT Broker of EDM, and let the built-in MQTT Client of EDM connect to the Broker for data publishing, and the MQTT Clients of other applications connect to the Broker to subscribe to obtain data, thus achieve the purpose of communication or system integration. Only one broker is needed in LAN.

When some MQTT clients are out of the LAN, local and remote clients must connect to a MQTT broker on the **Internet**. The built-in Broker should be disabled and the built-in MQTT Client of EDM connects to the external MQTT Broker for data publishing.

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.

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.

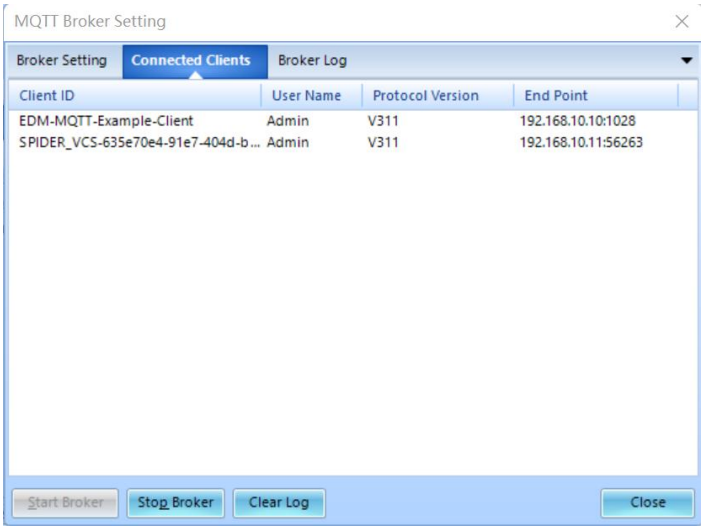
Password – The MQTT Broker Password that clients use to login.

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.

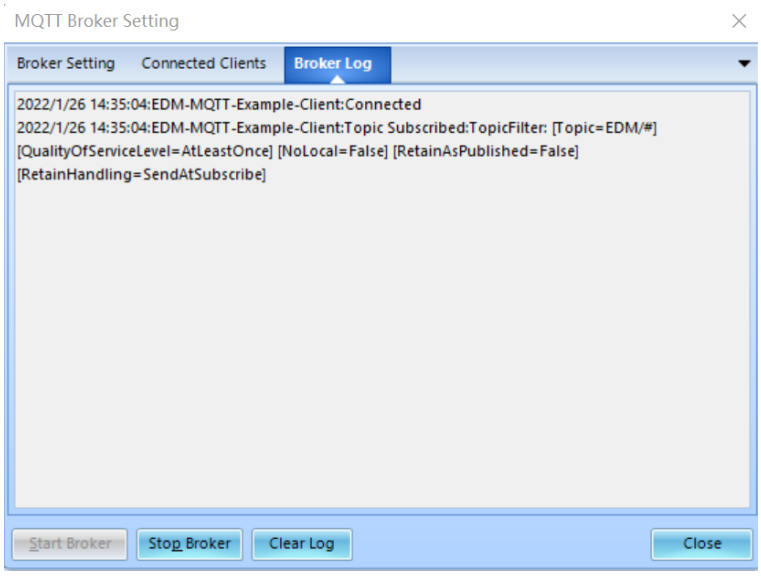
Connect Clients

Show all MQTT clients connected to this Broker



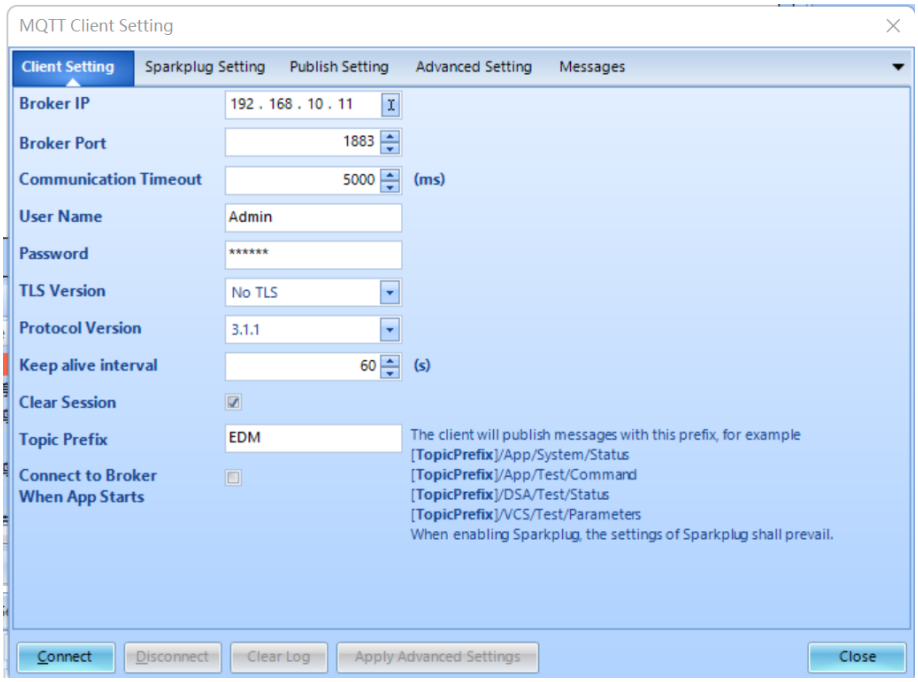
Broker Log

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.

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.

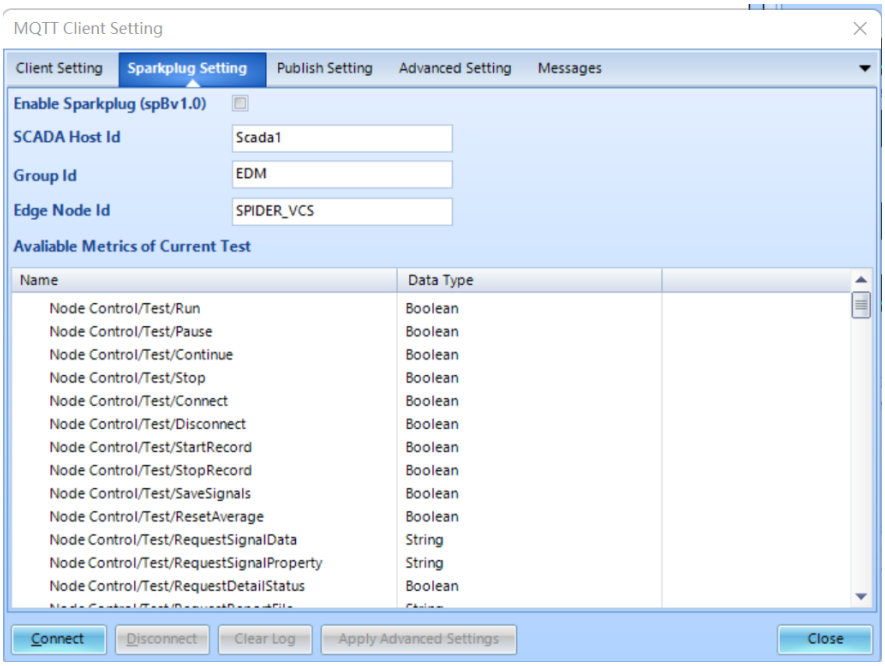
Clear Session – Determines if the connection should subscribe to topics that retain QoS 1 and 2 messages, otherwise treats the connection as clean.

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.

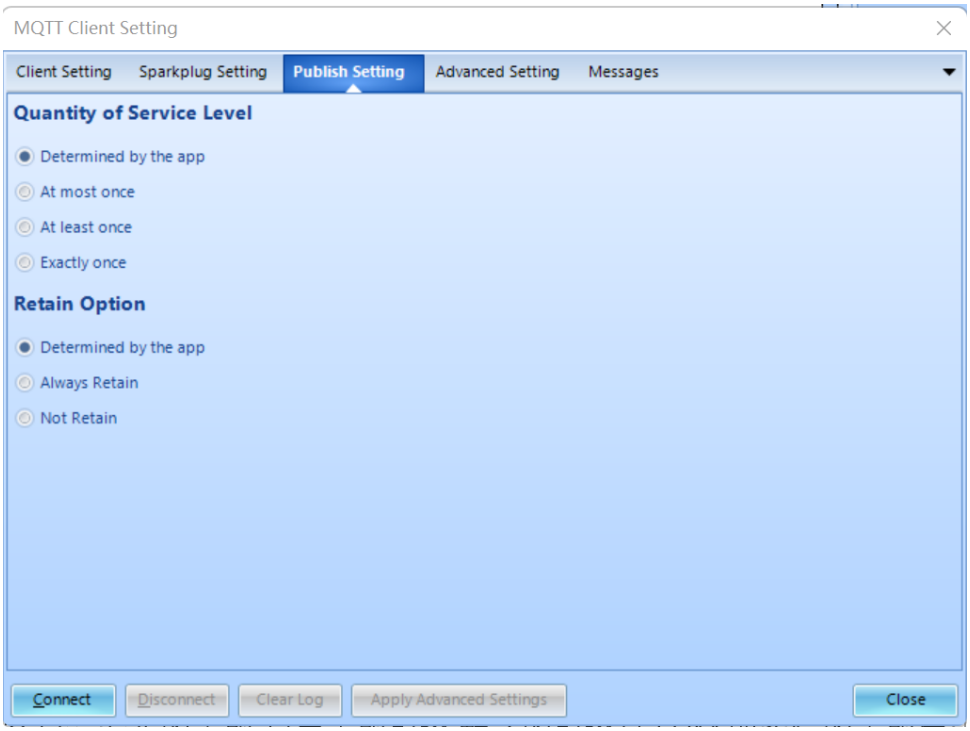
Sparkplug Setting

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.



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 acknowledgement and communication.

Retain has 3 options:

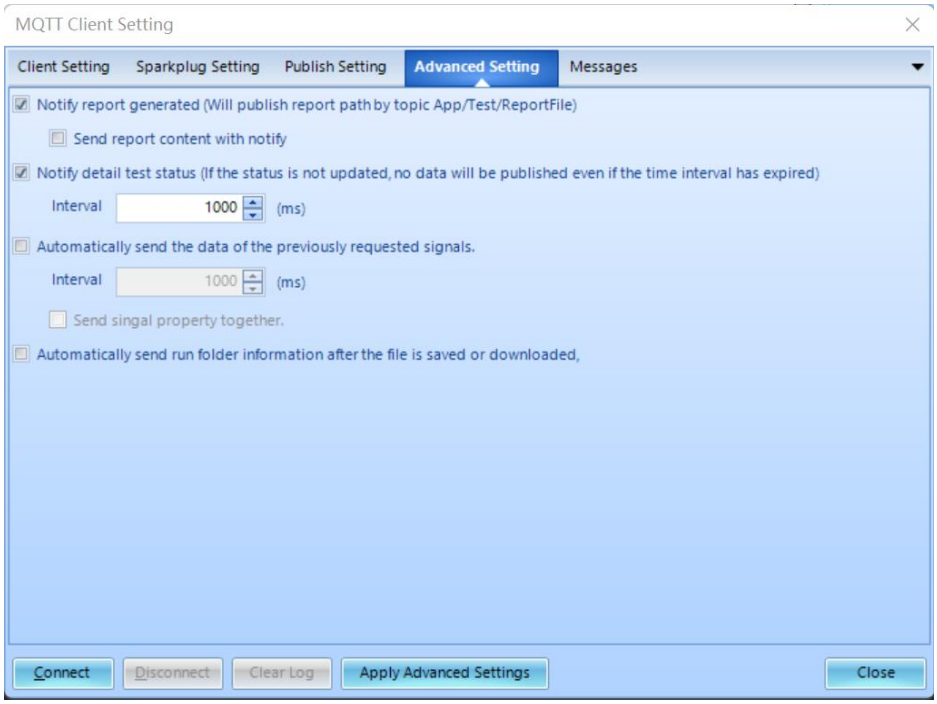
Determined by the app – Lets EDM determine the retained message option.

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.

Advanced Settings

Here are some special states set on whether to publish as well as time intervals.



Notify report generated – EDM's Broker will notify 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.

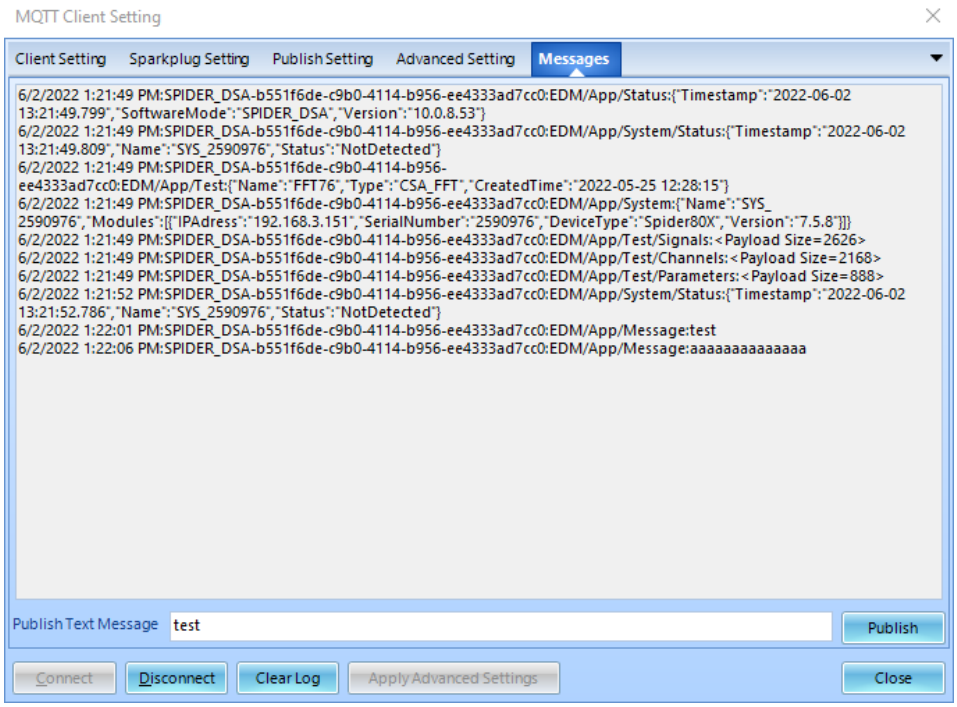
Notify detail test status – EDM's Broker will notify clients of the test detail status within a set interval and if there any data to update.

Automatically send data of previously requested signals – EDM's Broker will send data of previously requested signals every set interval as well as the signal's property if selected.

Automatically send run folder information – EDM's Broker will send run folder information if the files have been saved or downloaded.

Messages

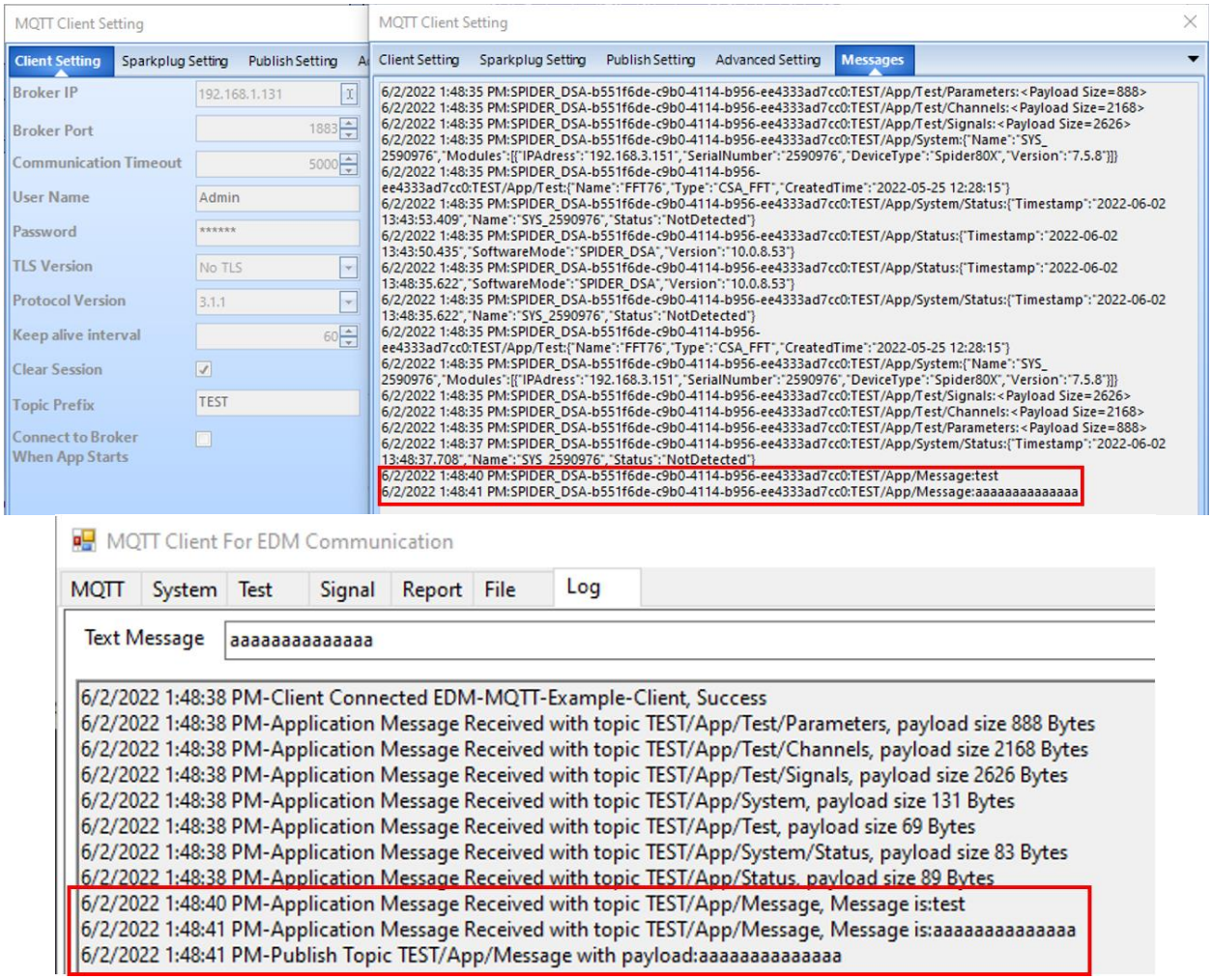
Shows published and received messages between the MQTT clients. It can also send simple text messages between clients.



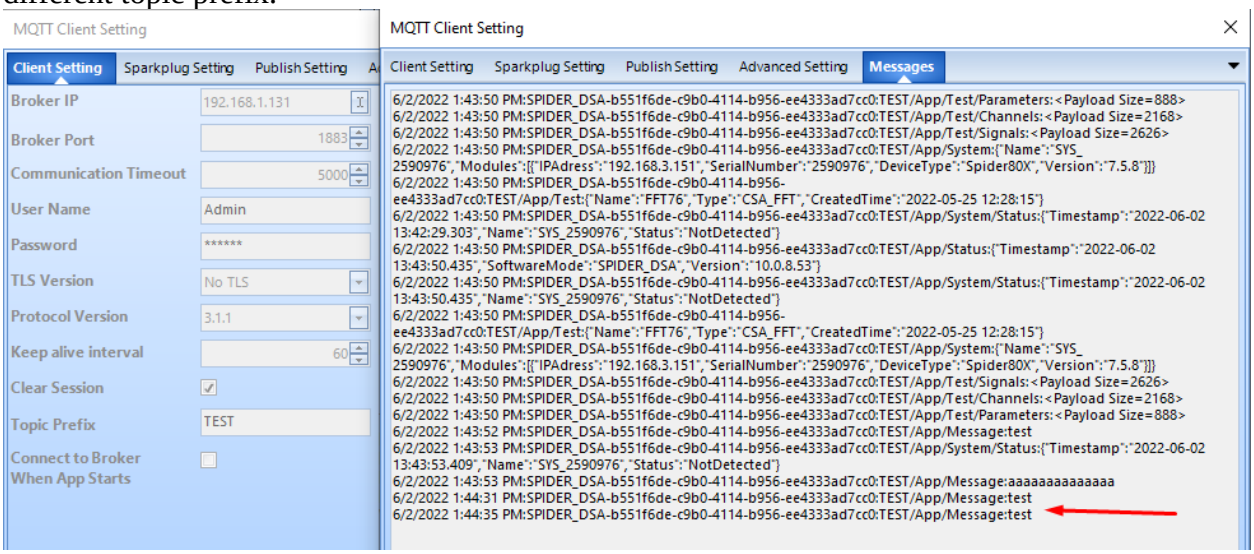
Subject Prefix

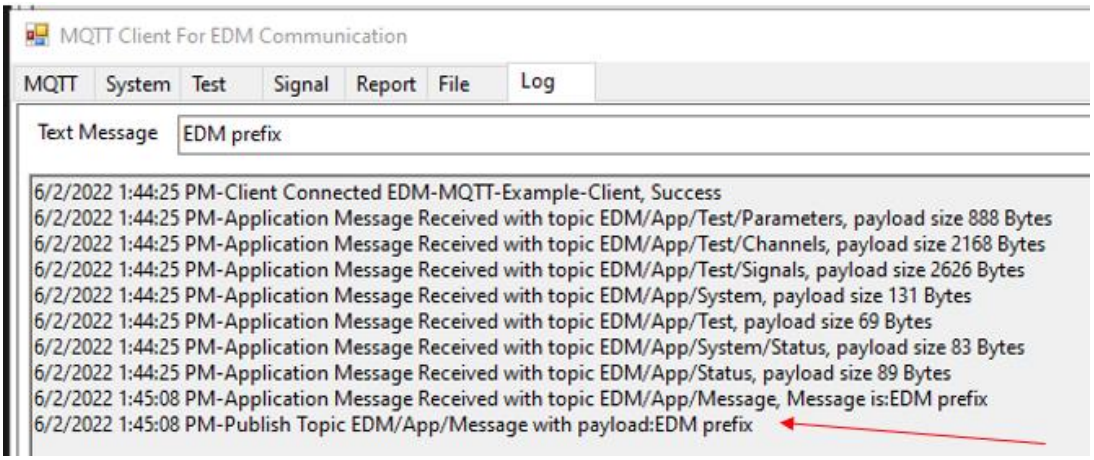
To distinguish the topics published by different clients, enable topic prefix.
The **default prefix** is **EDM**, and the actual topic is EDM/App/Message, EDM/App/Test, etc.
The prefix can be customized.
(The prefix is omitted in the following topic descriptions)
If there are multiple EDM clients that 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 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.





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

Publishing Payload

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

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

RequestDetailStatus
Requests the live status data of the current running test.
Returns certain topics such as **VCS/Test/RandomStatus**, **DSA/Test/DSAStatus**, etc., and data that are similar to these:

Name	Value
Timestamp	2022-02-23 15:07:12.083
AccUnit	m/s²
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.

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

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
EvalutionStop	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/Parameters

When the test is loaded or the **parameters** are **changed**, the latest status of all parameters is released with Retain Message.
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;
}
```

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;
}
```

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/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;
}
```

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.

OutputOff
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	Amplitdue	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.

DSA/Test/DSAStatus

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;
}
```

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.

VCS/Test/Stage

VCS tests a specific **Stage state**, and 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_DisplayFinished
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/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 MQTTTWRTestStatus
{
    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 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 MQTTTHStatus
{
    public string Timestamp;
    public double TotalTime;
    public double RemainTime;
    public double TargetTemperature;
    public double ControlTemperature;
    public double TargetHumdity;
    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/VHStatus

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.

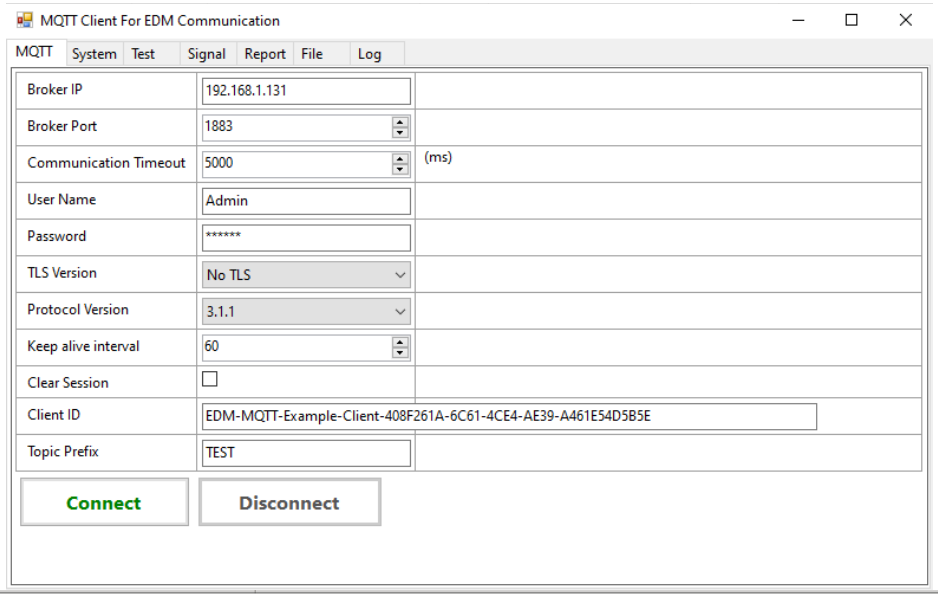
```
public struct MQTTVTStatus
{
    public string Timestamp;
    public double TotalTime;
    public double RemainTime;
    public double TargetTemperature;
    public double ControlTemperature;
    public double TargetVibration;
    public double ControlVibration;
    public List<MQTTTemperatureStatus> LatestTemperatures;
    public List<MQTTVibrationStatus> LatestVibrations;
}
```

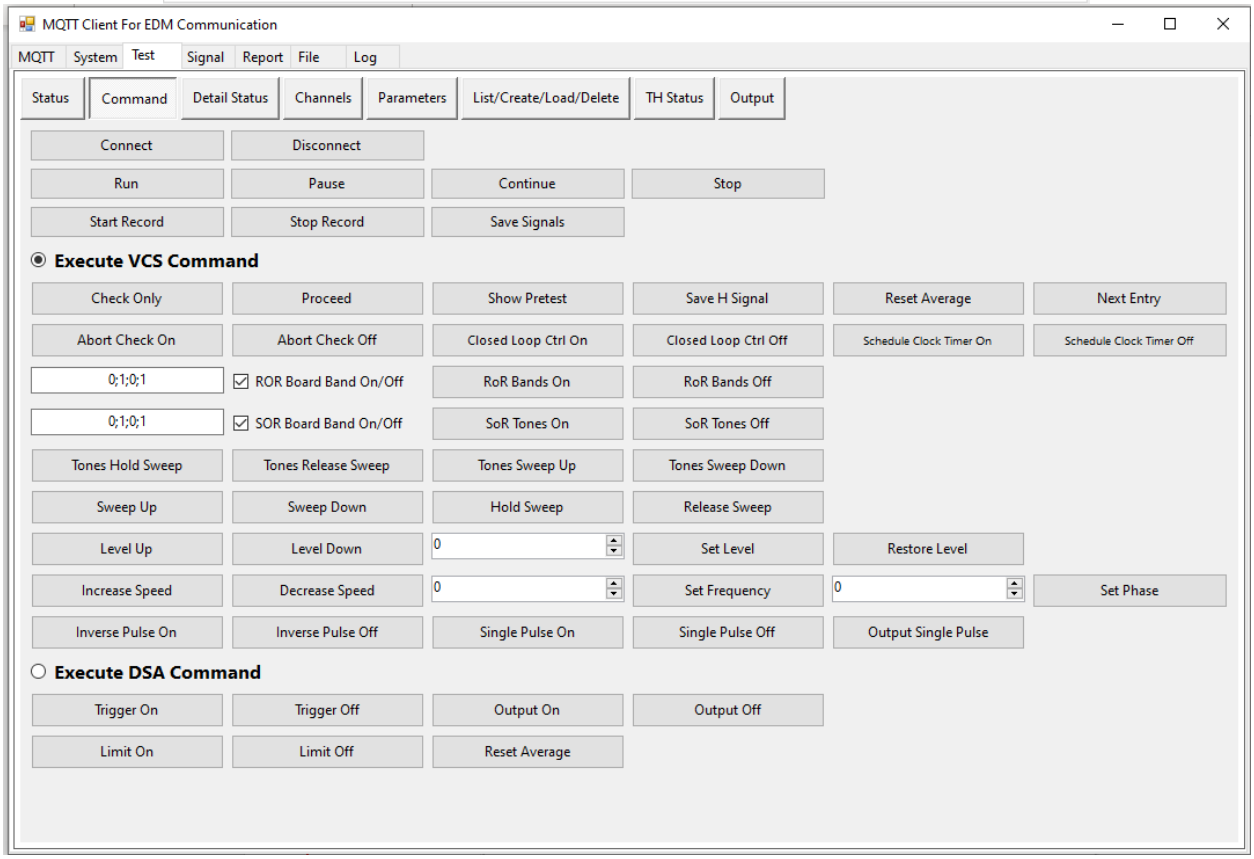
```
public struct MQTTVibrationStatus
{
    public string Name;
    public double Vibration;
    public string Unit;
}
```

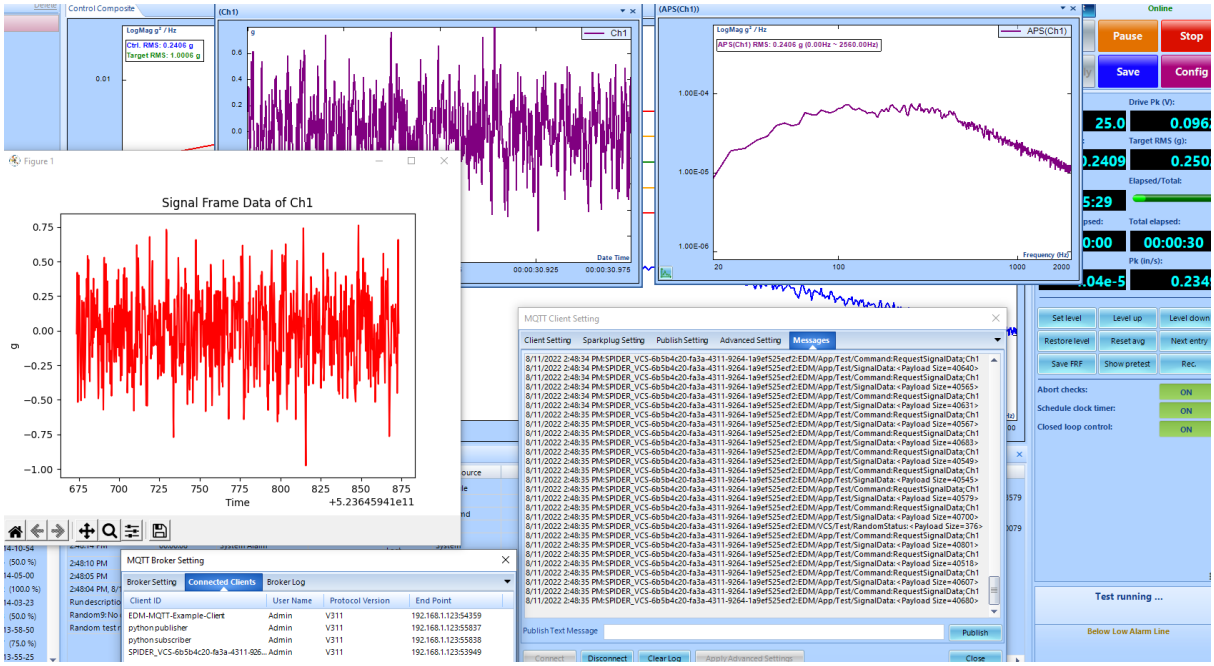
EDM MQTT Client Demo Program

Crystal Instrument developed a MQTT Client Demo Program written in C# and Python that showcase how the MQTT protocol work and how to use 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.







EDM MQTT Client Demo Package

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

Demo Package Content

Crystal Instrument will provide a zip file or software installer exe 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 Manual

How to Install the EDM MQTT Client Demo Program

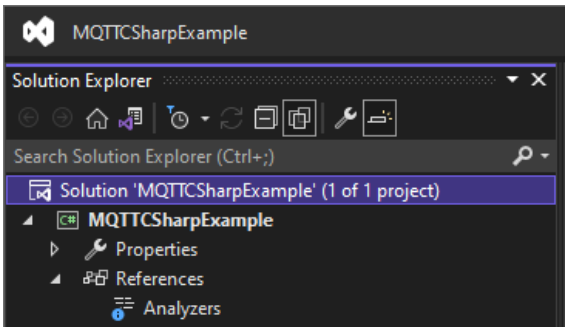
If the distributed MQTT Client Demo Program comes in a zip file, then extract the zip file into a suitable location on the computer. Or if it comes in a software installer exe file, then run it and follow the steps to download the MQTT Client Demo Program. After the MQTT Client Demo Program has been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

There are three 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 the MQTTnet reference to the MQTTnet packaged that should be placed in the bin folder of the C# demo program.

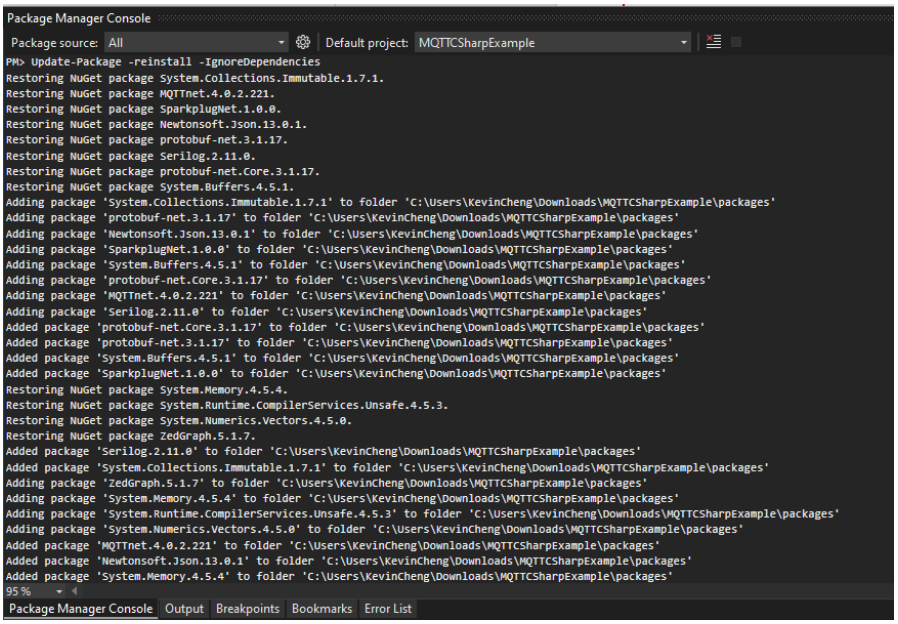
EDM MQTT Client C# Demo Showcase

How to Build the MQTT Client C# Demo

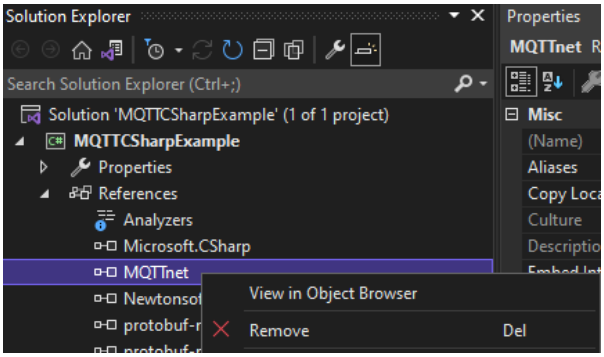
The MQTT Client C# Demo folder comes with a .csproj that can be opened with visual studio. The highlighted solution file in the Solution Explorer should be saved for the nuget package manager to work.



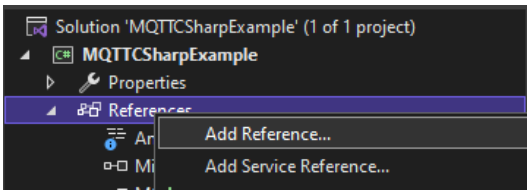
After the solution has been saved, go to Tools > Nuget Package Manager > Package Manager Console. Then enter Update-Package -reinstall -IgnoreDependencies.

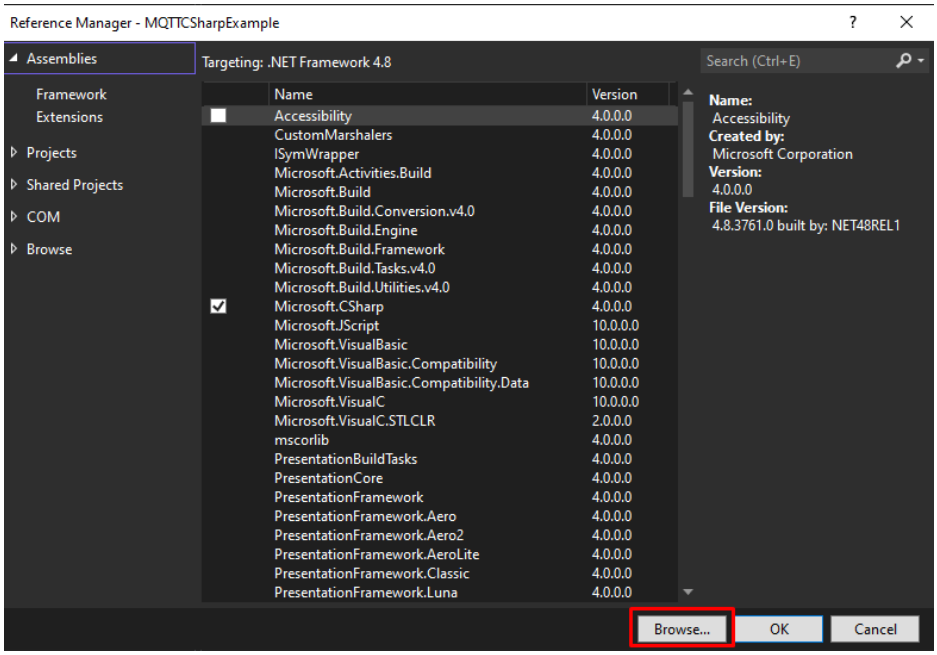


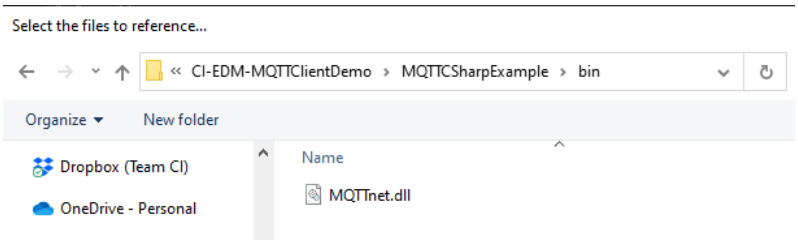
The package manager will proceed to reinstall all nuget packages used in the MQTT C# demo. Then once the manager is finished, go to Solution Explorer > Expand Reference > Remove MQTTnet.

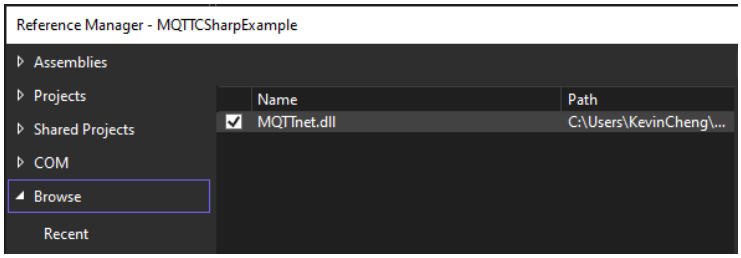


Then right click References > Add Reference > Browse… > Select MQTTnet.dll in the C# demo bin folder > OK.





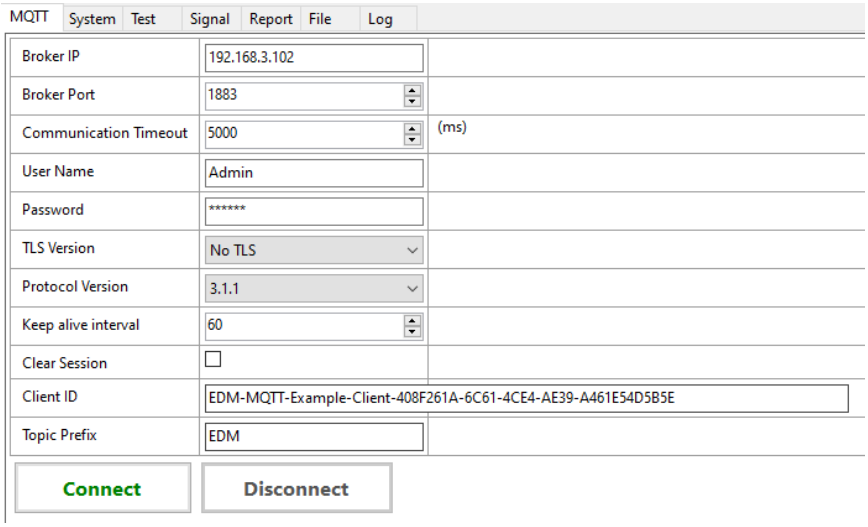




Now the project can be built.

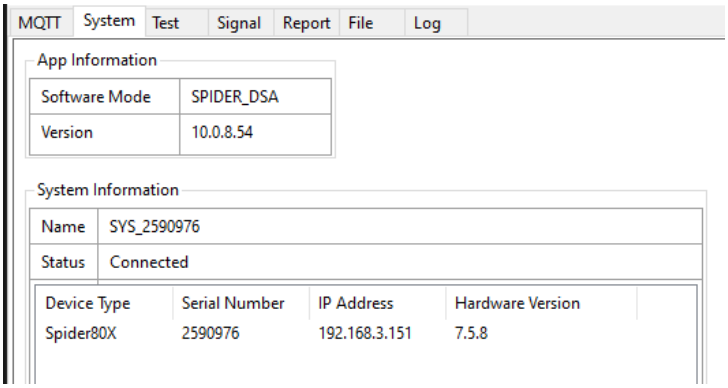
C# Program GUI

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



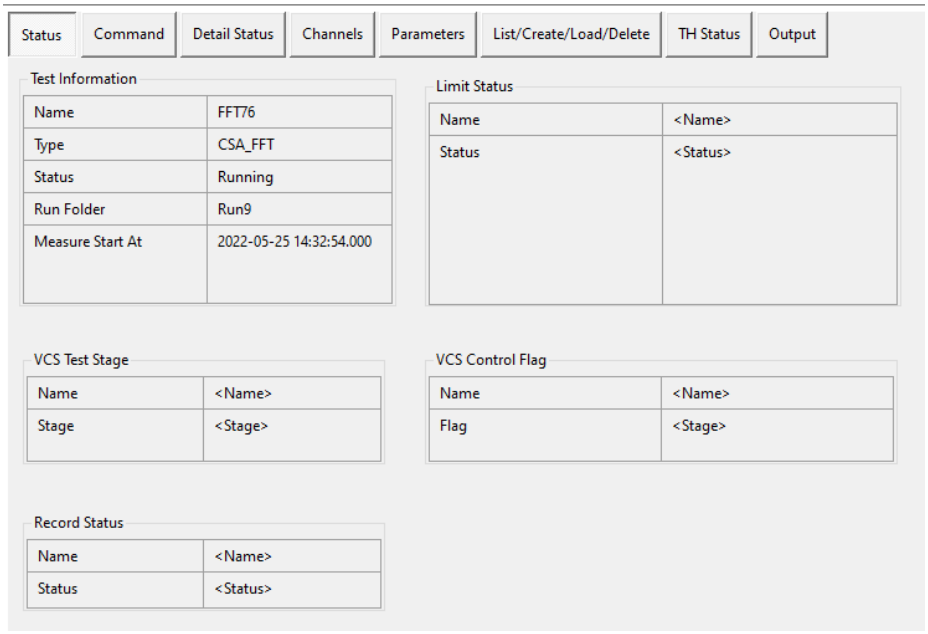
System

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



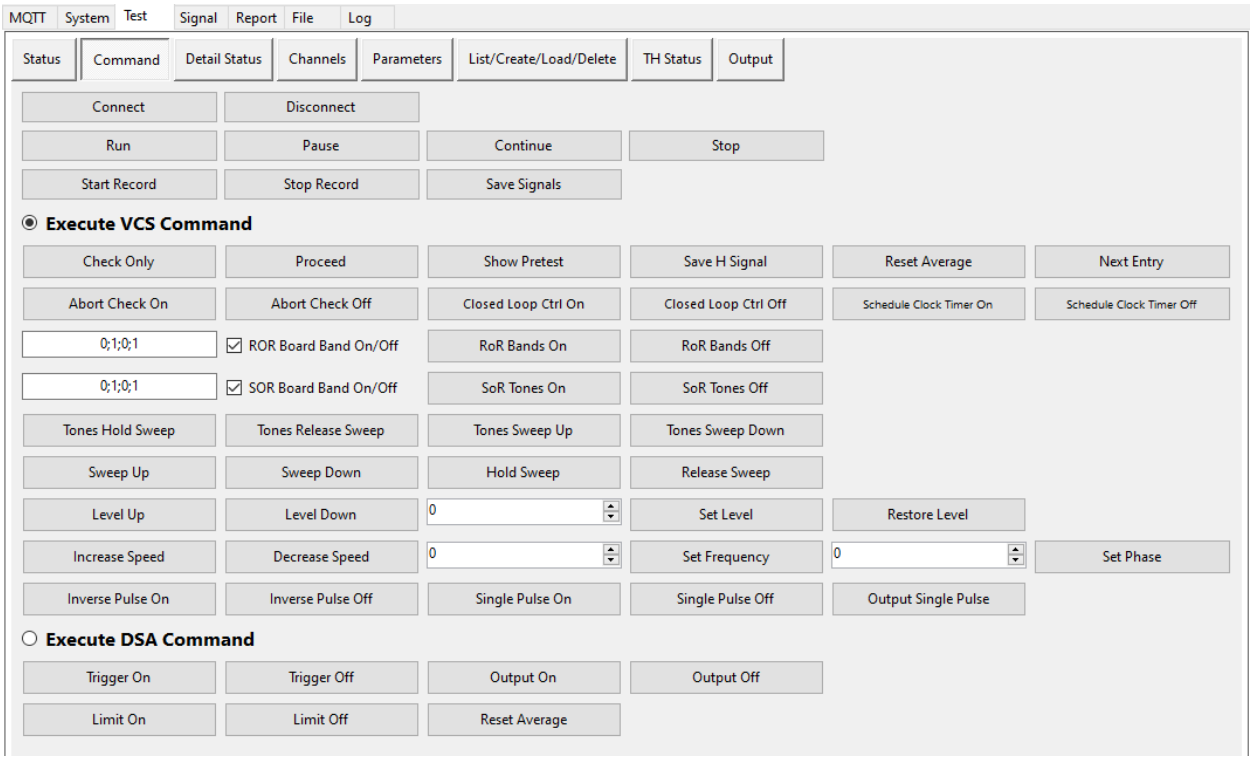
Test

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.



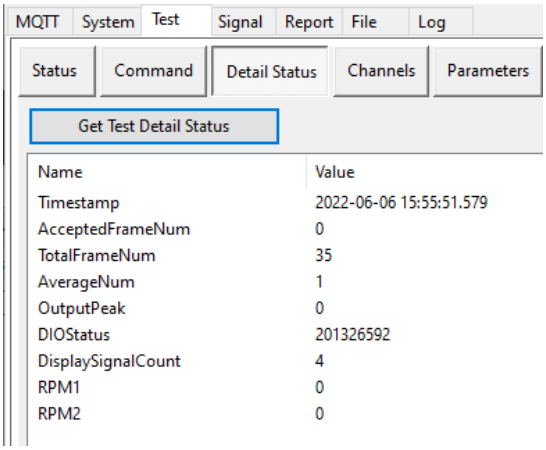
Test's Command

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



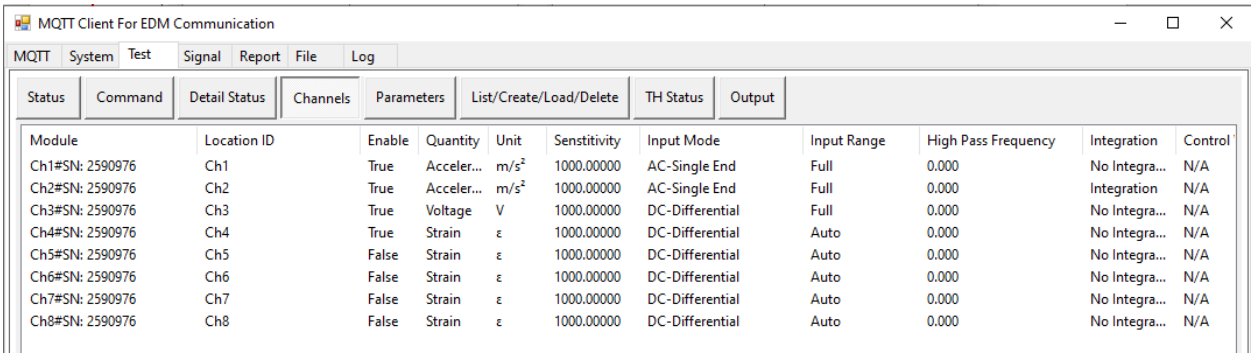
Test's Detail Status

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



Test's Channels

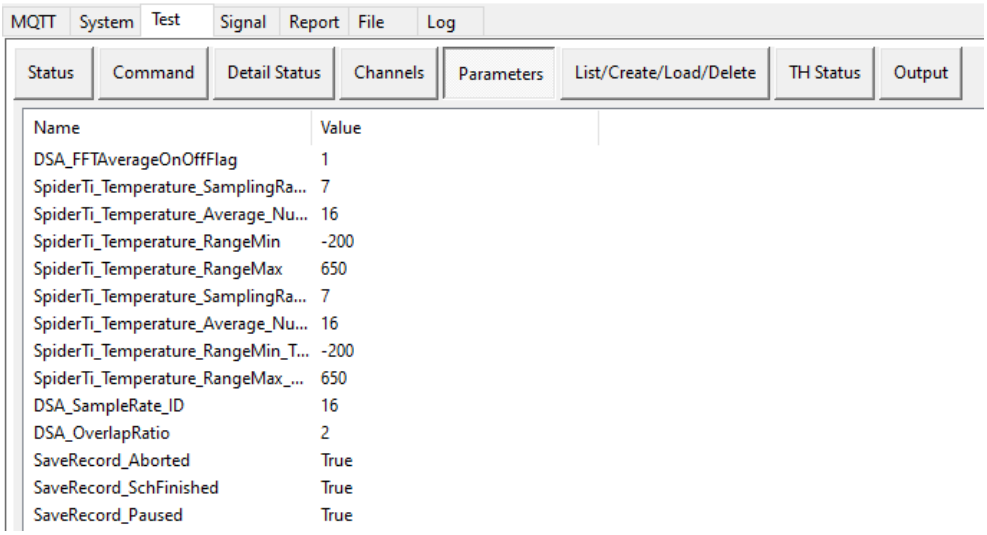
Where it shows the Input Channel Settings data.



Module	Location ID	Enable	Quantity	Unit	Senstitivity	Input Mode	Input Range	High Pass Frequency	Integration	Control
Ch1#SN: 2590976	Ch1	True	Acceler...	m/s²	1000.00000	AC-Single End	Full	0.000	No Integra...	N/A
Ch2#SN: 2590976	Ch2	True	Acceler...	m/s²	1000.00000	AC-Single End	Full	0.000	Integration	N/A
Ch3#SN: 2590976	Ch3	True	Voltage	V	1000.00000	DC-Differential	Full	0.000	No Integra...	N/A
Ch4#SN: 2590976	Ch4	True	Strain	ε	1000.00000	DC-Differential	Auto	0.000	No Integra...	N/A
Ch5#SN: 2590976	Ch5	False	Strain	ε	1000.00000	DC-Differential	Auto	0.000	No Integra...	N/A
Ch6#SN: 2590976	Ch6	False	Strain	ε	1000.00000	DC-Differential	Auto	0.000	No Integra...	N/A
Ch7#SN: 2590976	Ch7	False	Strain	ε	1000.00000	DC-Differential	Auto	0.000	No Integra...	N/A
Ch8#SN: 2590976	Ch8	False	Strain	ε	1000.00000	DC-Differential	Auto	0.000	No Integra...	N/A

Test's Parameters

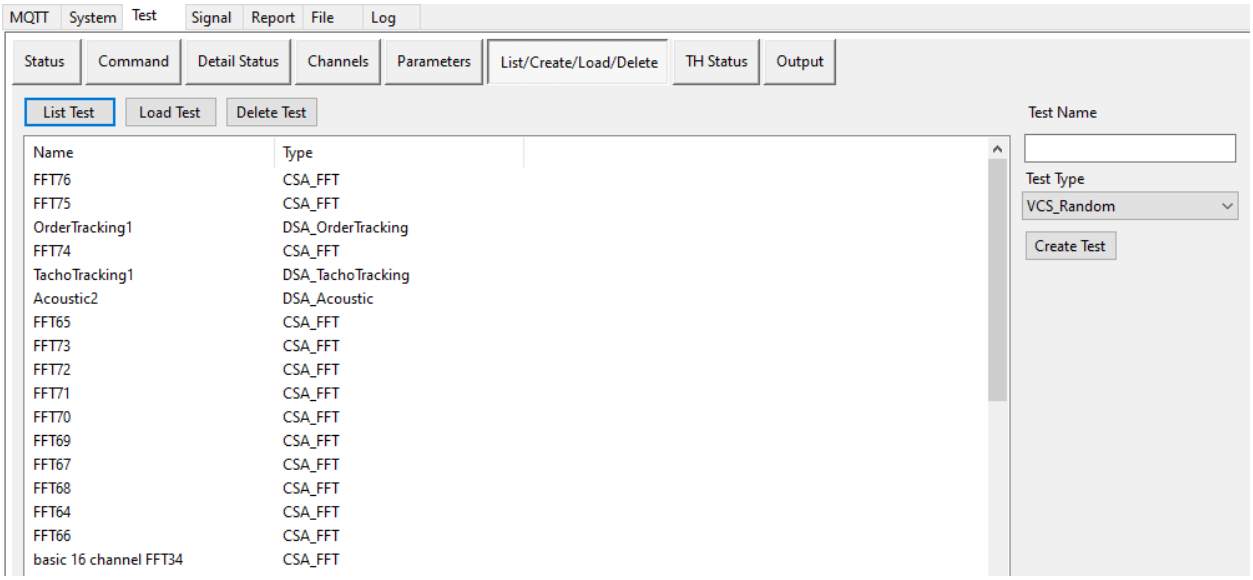
Where it shows the various test configuration parameters.



Name	Value
DSA_FFTAverageOnOffFlag	1
SpiderTi_Temperature_SamplingRa...	7
SpiderTi_Temperature_Average_Nu...	16
SpiderTi_Temperature_RangeMin	-200
SpiderTi_Temperature_RangeMax	650
SpiderTi_Temperature_SamplingRa...	7
SpiderTi_Temperature_Average_Nu...	16
SpiderTi_Temperature_RangeMin_T...	-200
SpiderTi_Temperature_RangeMax_...	650
DSA_SampleRate_ID	16
DSA_OverlapRatio	2
SaveRecord_Aborted	True
SaveRecord_SchFinished	True
SaveRecord_Paused	True

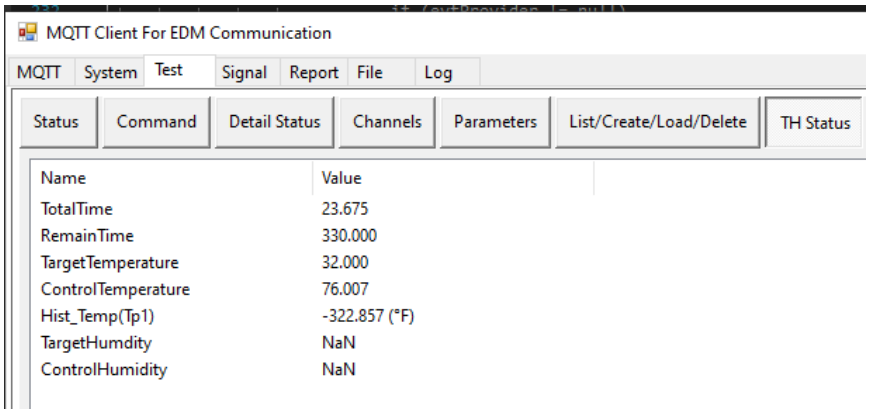
Test's Test Database

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



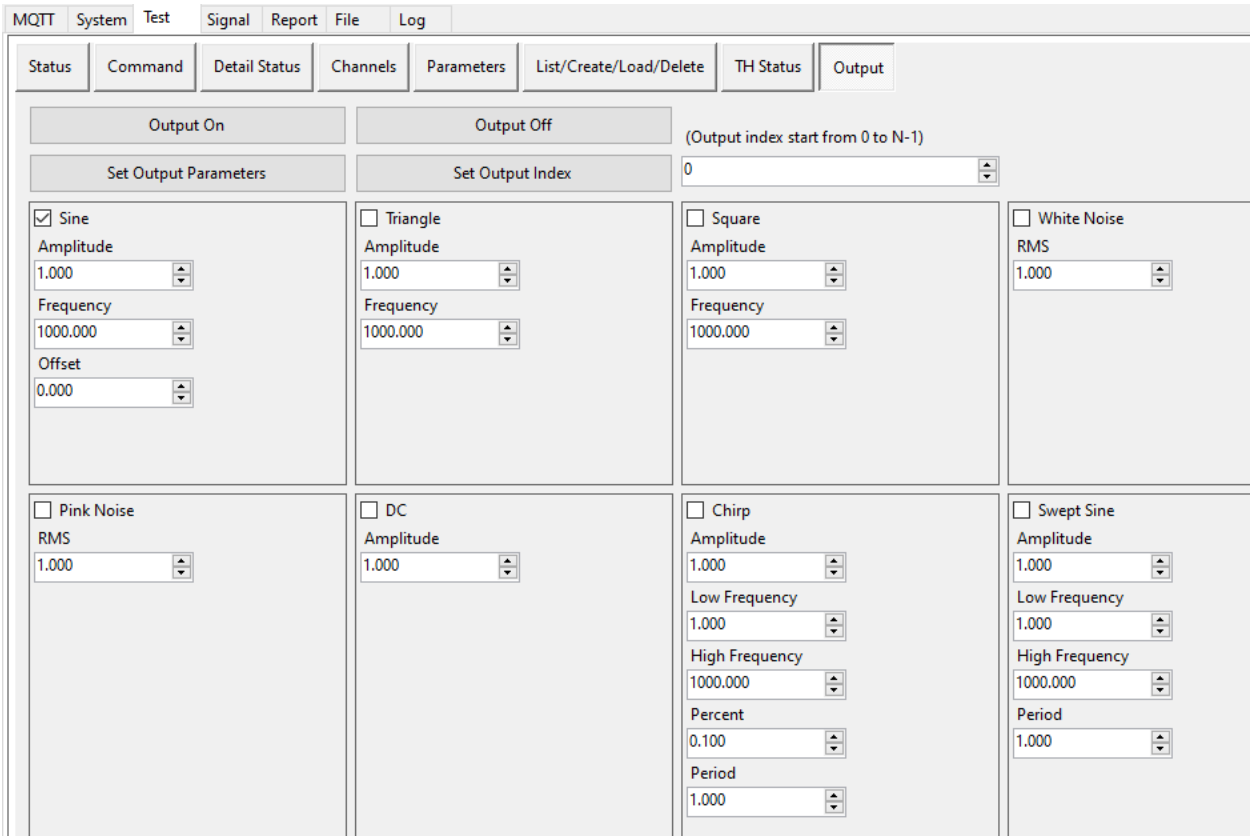
Test's TH Status

Where it shows data relating to temperature and humidity control.



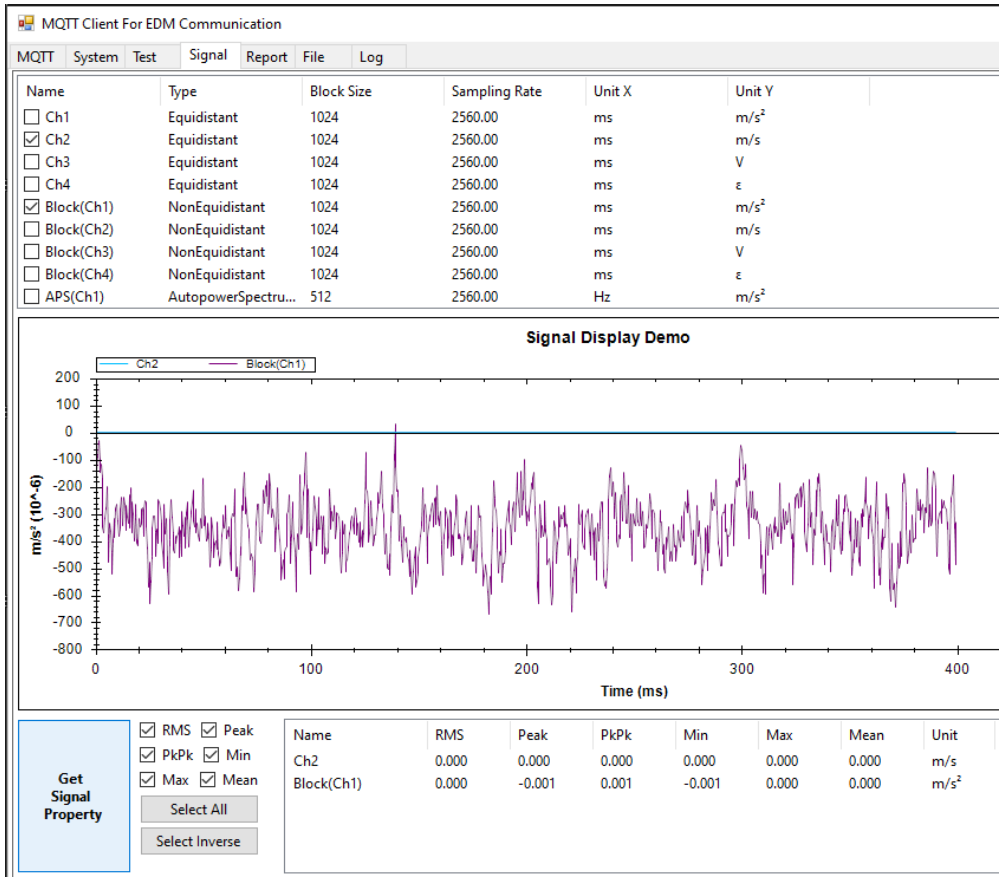
Test's Output

Where it shows more commands relating to the EDM control panel output channel.



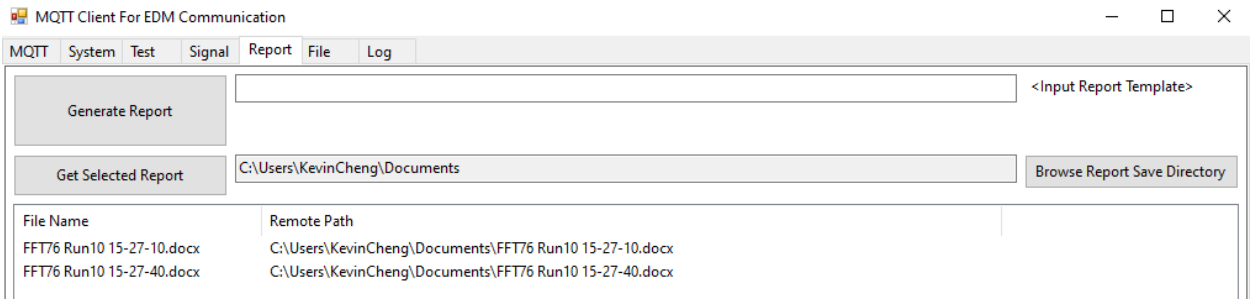
Signal

Where it display 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.



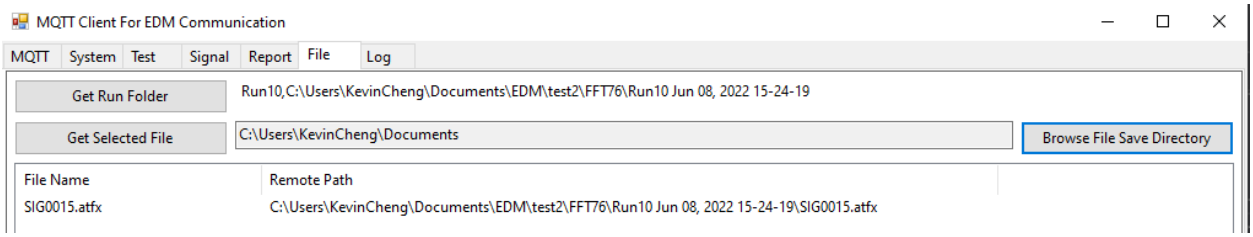
Report

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



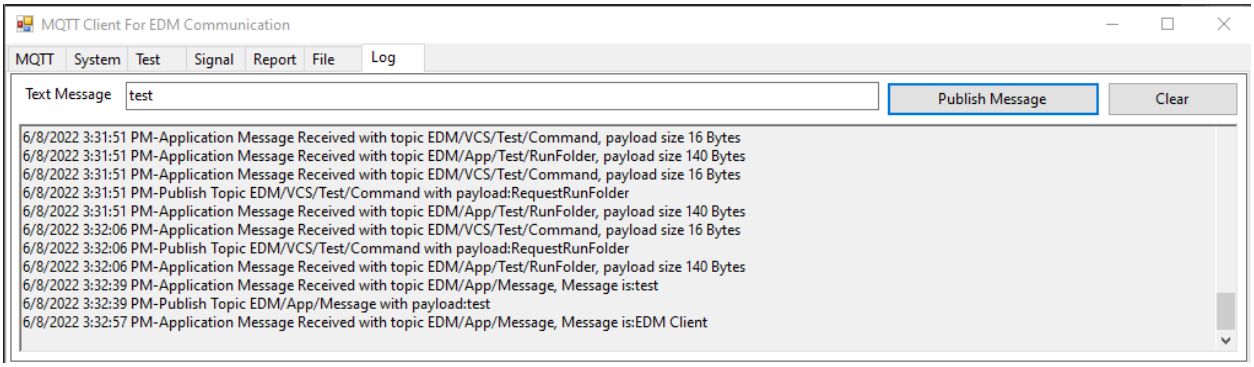
File

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

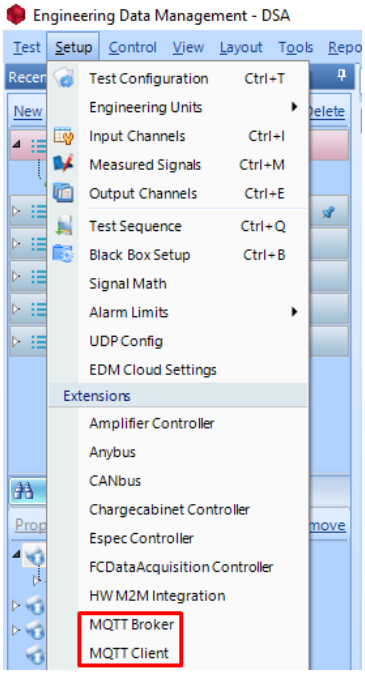
Where message communication between MQTT Clients is stored. It is possible to send a text message between clients as well.



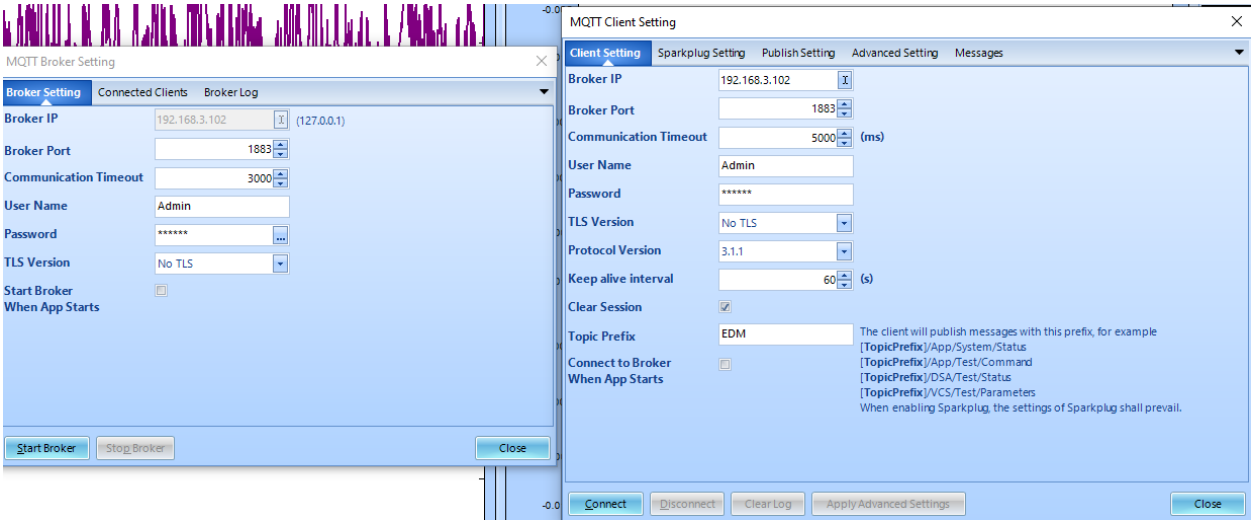
Connecting and Running a Test

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.

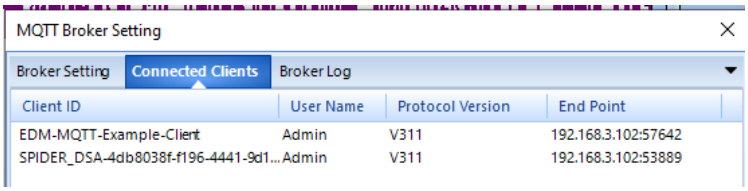
Once EDM is open, go to the toolbar Setup menu and click both MQTT Broker and MQTT Client.

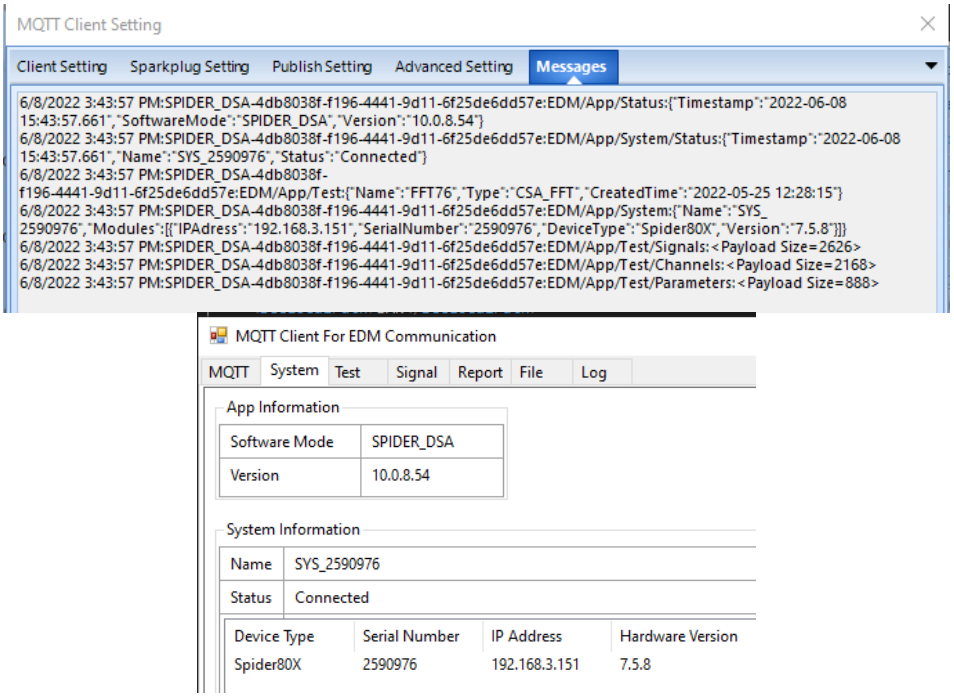


Two windows should appear starting at the connection parameter setting tab. Make sure the parameters are the same between the EDM MQTT Broker, EDM MQTT Client and MQTT Client Demo Program. Once they are, click Start Broker and Connect for all windows.

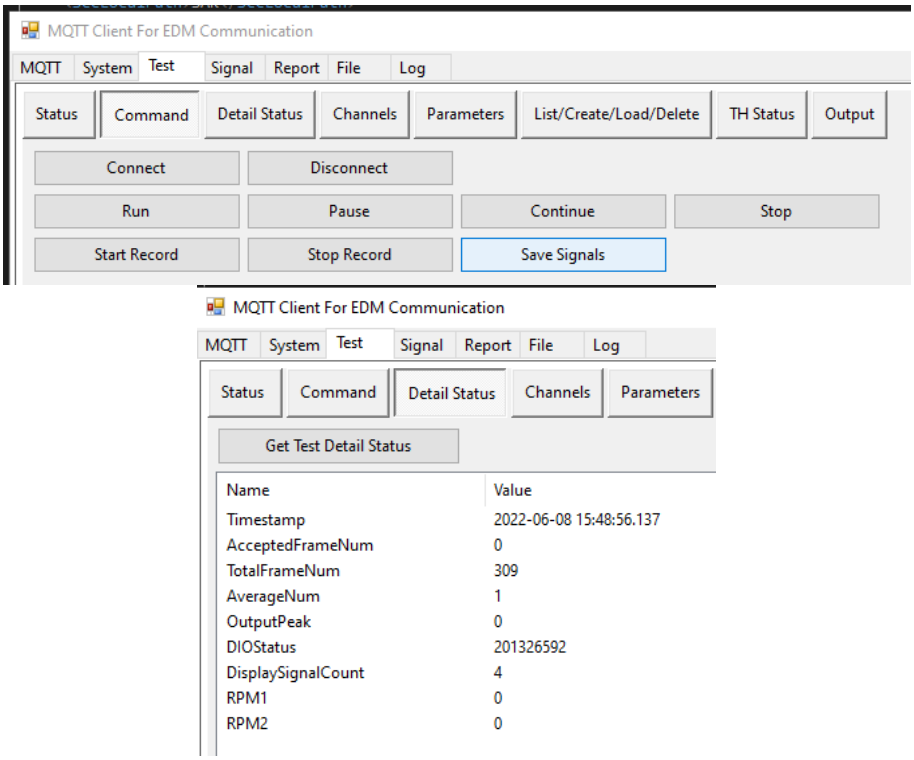


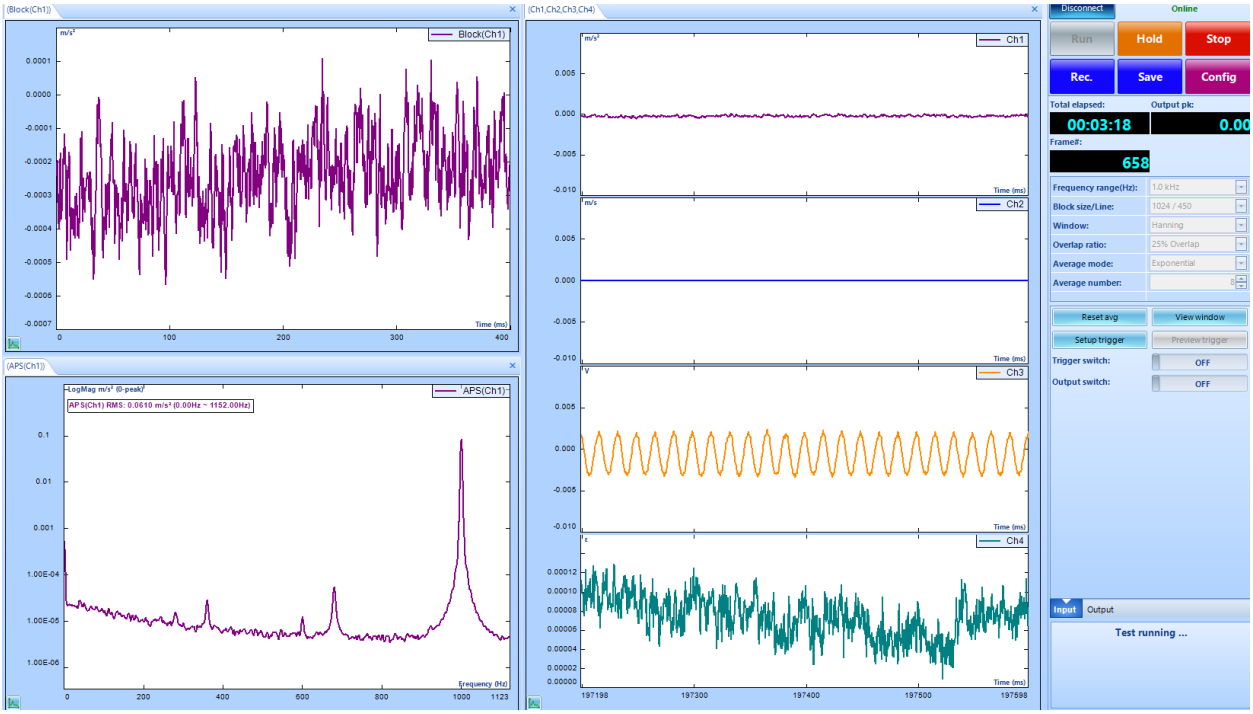
Once connected, the MQTT network should start communicating with each other and send off some topic messages regarding each client information and the test data.



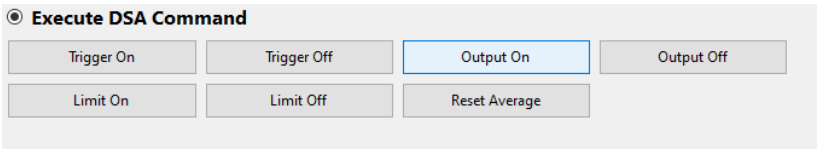


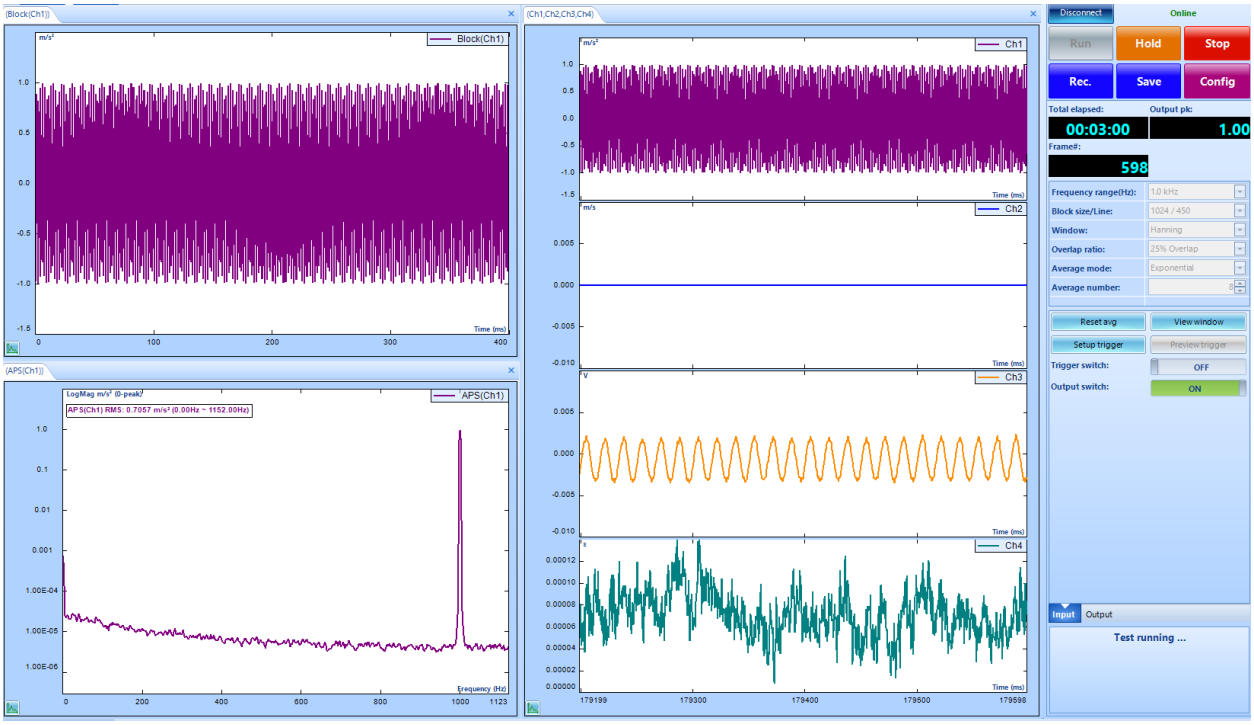
Then, go to the Test's Command tab and click Connect then Run. From here, EDM MQTT Client will send the test detail status updates to the MQTT Client Demo Program.





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.





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:

pip install paho-mqtt

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 do 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, below is the following imports:

```
from paho.mqtt import client as mqtt_client
import numpy as np
```

Numpy can be installed by the following command:

pip install numpy

In the initial part of script is 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
__init__	brokerIP=”127.0.0.1”, port=1883, username=’Admin’, password=’123456’	Initialize method Set up variables, publisher, subscriber, topics, connect clients
connect_mqtt	client_id	Connects MQTT broker / client
publish	pubtopic, msg	Publishes a topic to EDM broker
on_message	client, userdata, msg	Handles receiving a message from EDM
subscribe	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 MQTTExample_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")
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(",")[0][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].s
plit(",")[0][3:-1]
        signalUnitX =
mqttClient.LUT['EDM/App/Test/SignalData'].split("ValueX")[0].split("UnitX")[1].
split(",")[0][3:-1]
        signalUnitY =
mqttClient.LUT['EDM/App/Test/SignalData'].split("ValueX")[0].split("UnitY")[1].
split(",")[0][3:-1]

        Xvalues =
mqttClient.LUT['EDM/App/Test/SignalData'].split("ValueX")[1].split("ValueY")[0]
.split("ValueZ")[0][3:-3]
        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.

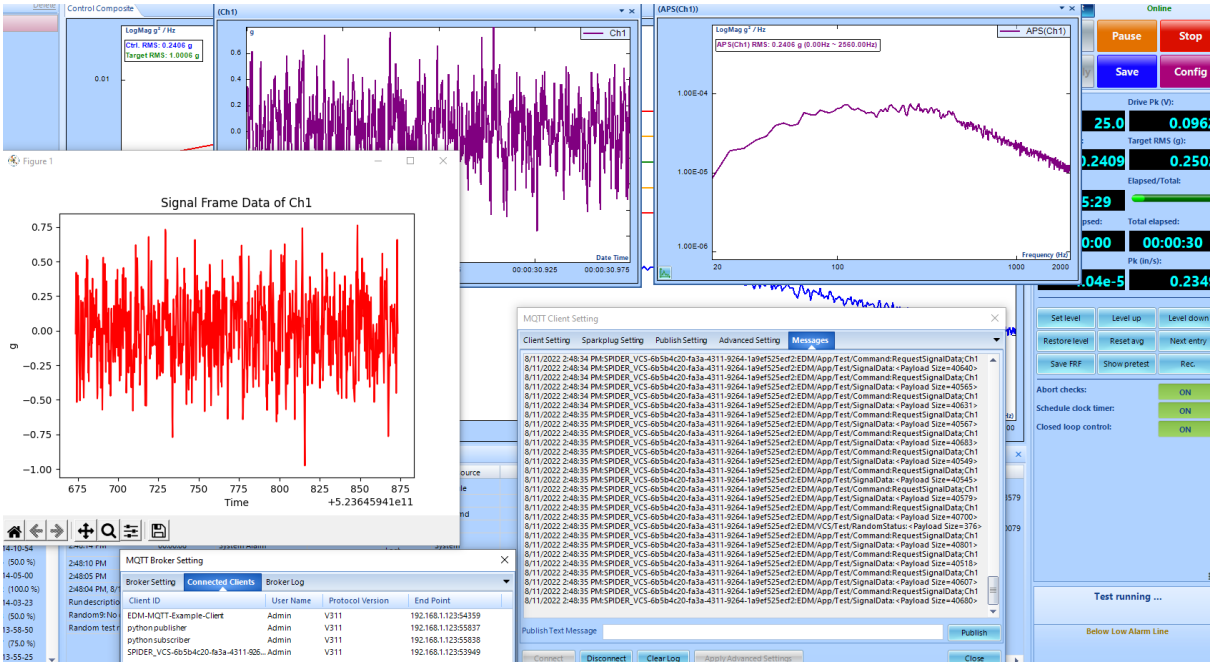
```
        ## Print Statements for displaying points of the signal frame ##
        # columnlabelstring = "X Values || Y Values || Z Value"
        # print("\n" + " " * (len(columnlabelstring)//4) + "Signal Frame
Data\n" + "-" * len(columnlabelstring))
        # print(columnlabelstring + "\n")

        # 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.

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]
```

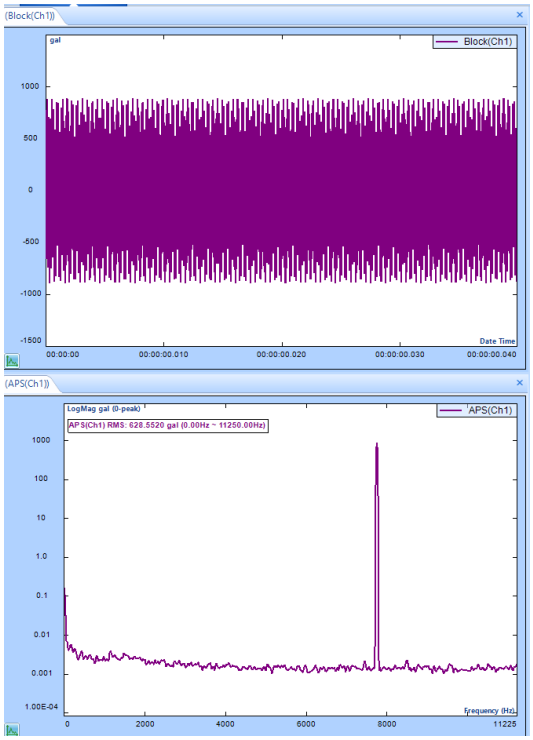
Other utilities help to document the script, including making a directory for the results, labeled with the current date as well as the serial number and device (Spider-80X, Spider-81, etc.)

```
# 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.

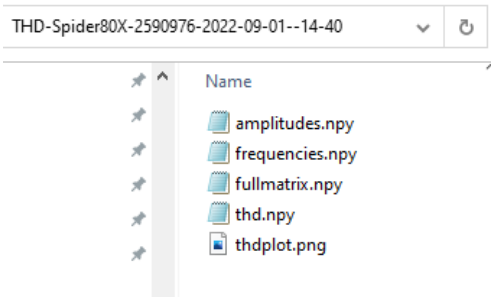
```
for freq in tqdm(freqs):
    print() # make a new line after tqdm progress bar
    for amp in amps:
        print(str(amp), "Volts at", str(freq), "Hz")
        mqttClient.output_sine(amp,freq)
        time.sleep(5)
        mqttClient.reset_average()
        time.sleep(5)
        # After inner loop 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)] )
```



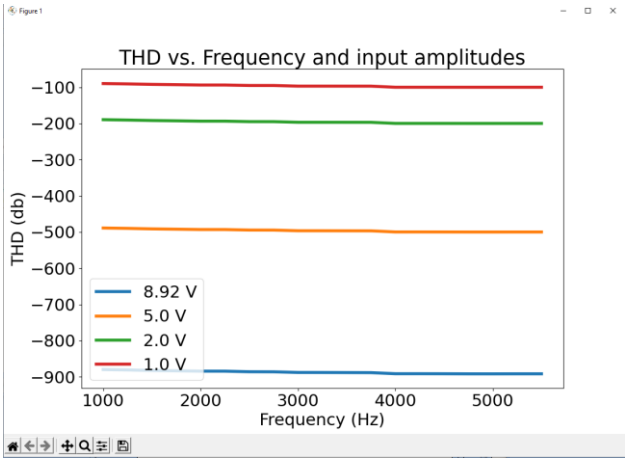
The data is saved to the directory to in case it is desired to inspect it after the script is complete, to avoid having to rerun the entire script.

```
## 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)
```



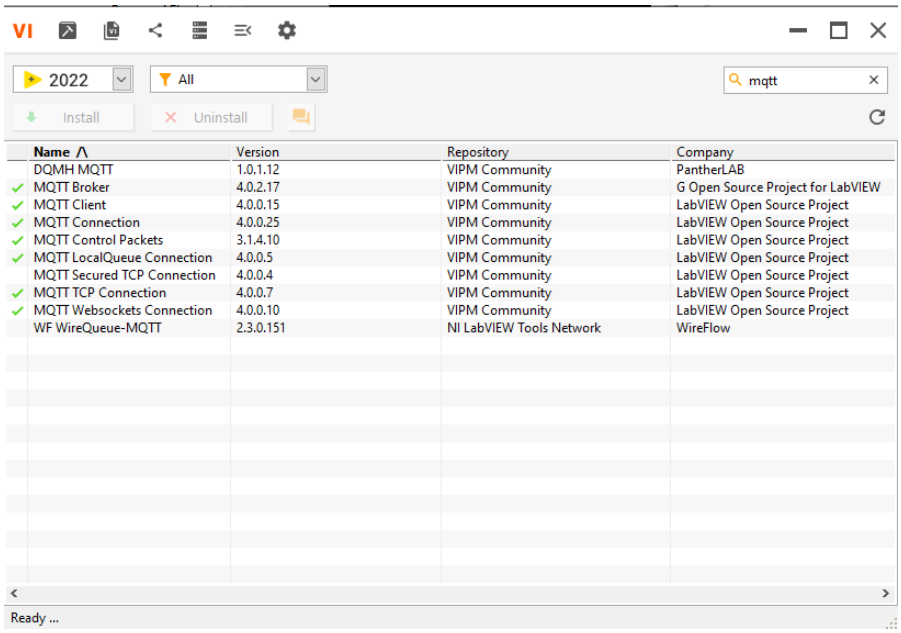
This part of the script plots the data and saves the plot to the directory.

```
# 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()
```

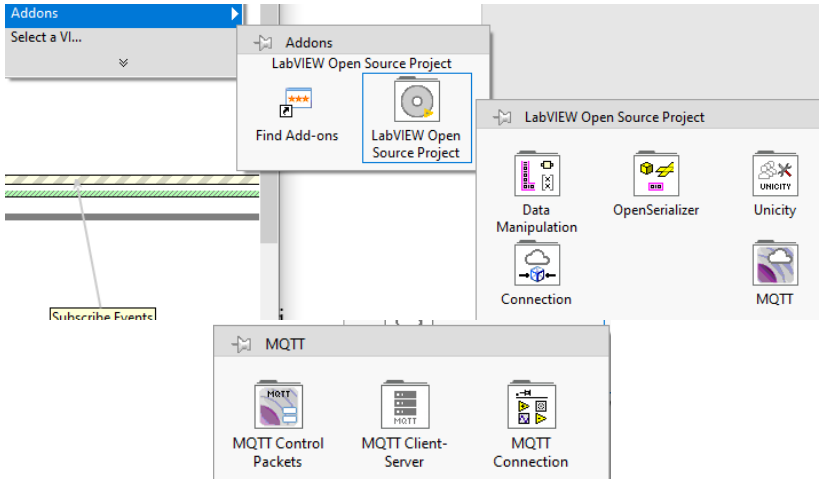


EDM MQTT Client LabVIEW Demo

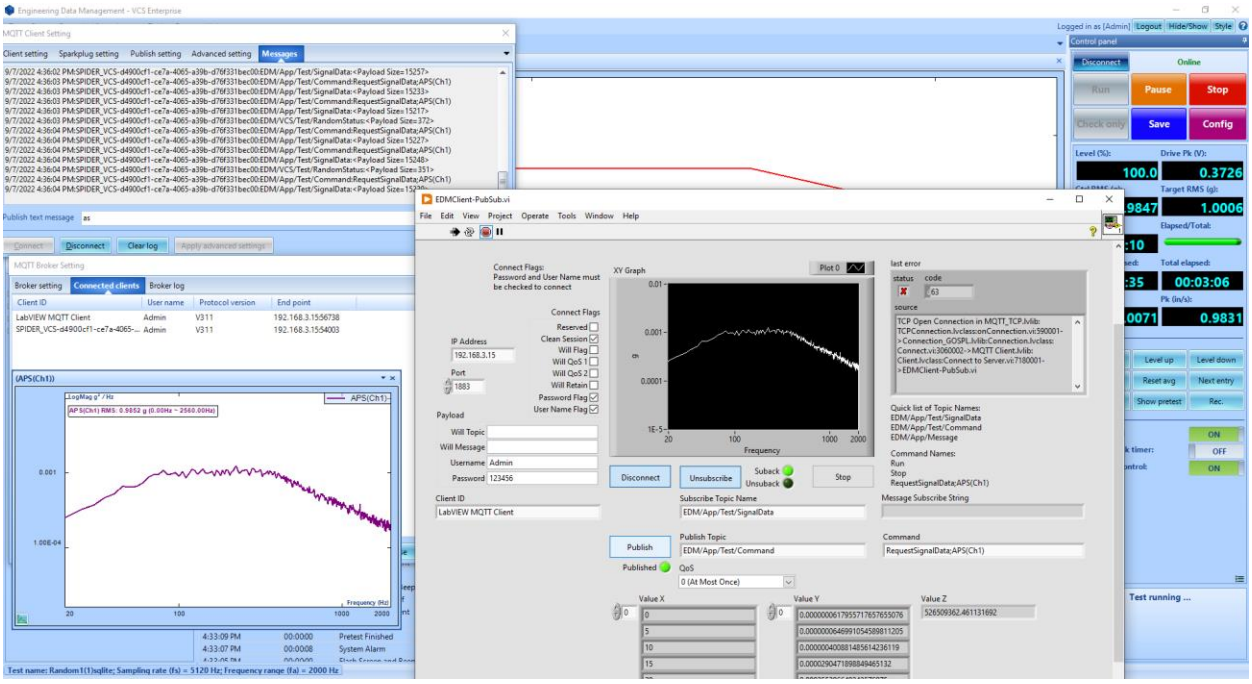
The EDM MQTT Client LabVIEW Demo utilizes the LabVIEW Open Source Project MQTT packages available via the LabVIEW **VI Package Manager**.
To download the MQTT packages, go to the search bar in the VI Package Manager and type in **MQTT**. It will list the various MQTT packages that LabVIEW has available.
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.



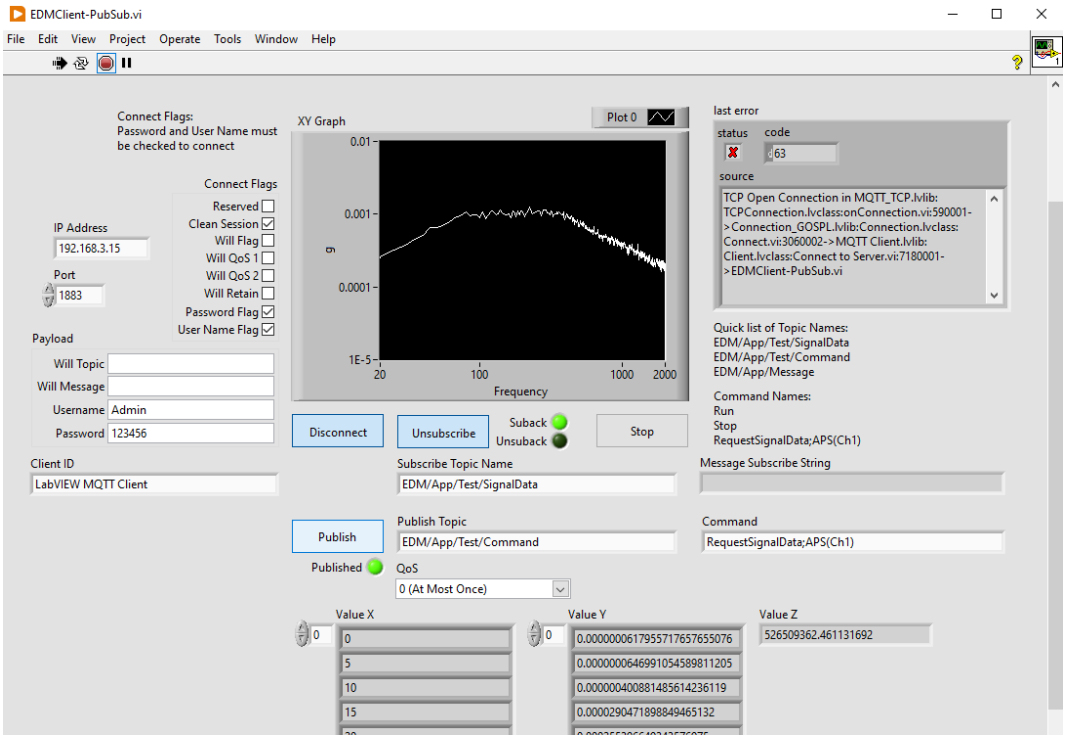
After downloading the MQTT packages, right clicking in the Block Diagram of a new vi file will open the functions menu. Then go to Addons > LabVIEW Open Source Project > MQTT and it will reveal the MQTT functions and some MQTT example vi nodes.



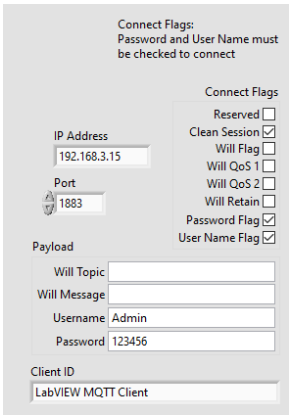
In this EDM MQTT Client LabVIEW Demo vi example, it has the basic capabilities of being able to connect to the EDM MQTT network, subscribe to topics and publish topics with commands. As shown below with a EDM VCS Random test running, MQTT network up and the LabVIEW MQTT Client Demo connected and requesting 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.



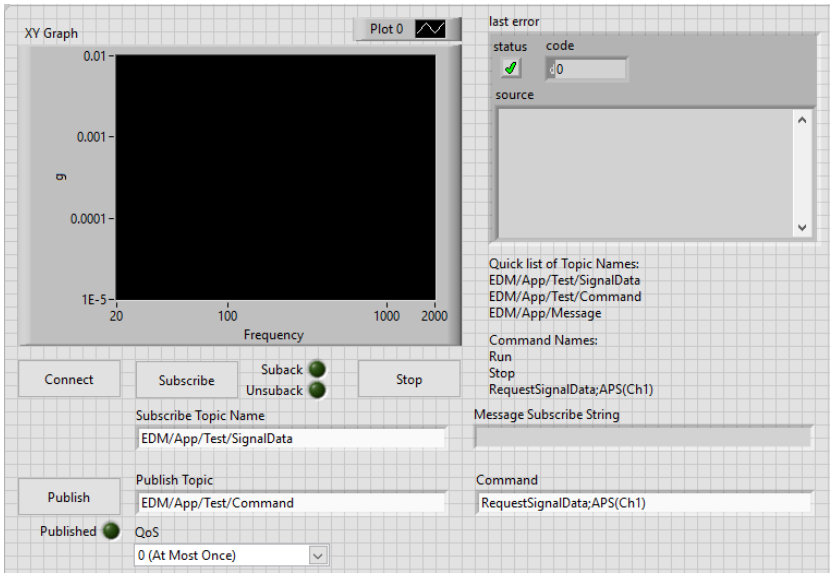
LabVIEW Front Panel



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.

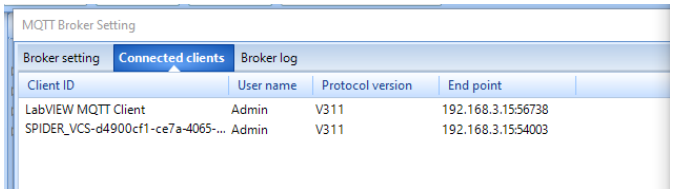


The connection parameter settings have the payload, connection flags, client ID and TCP configuration that are required to connect to the EDM MQTT broker. It is noted that the connect flags for password and username must be checked when sending a payload with a username and password to connect to the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



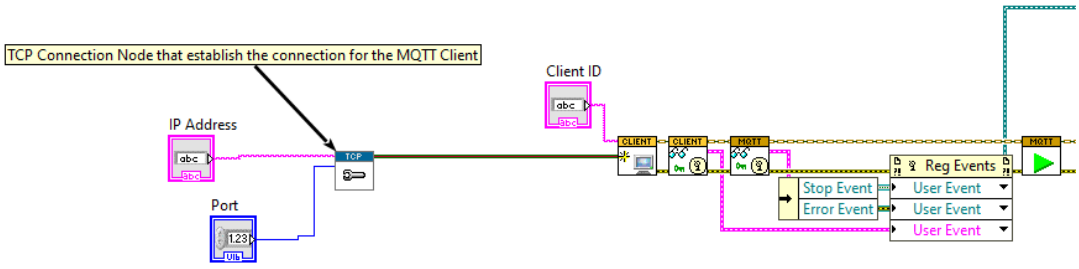
After inputting the correct connection parameter settings, clicking the Connect button will then attempt to connect the LabVIEW MQTT Client to the EDM MQTT broker. It may be necessary to reconnect if it doesn't succeed the first time.

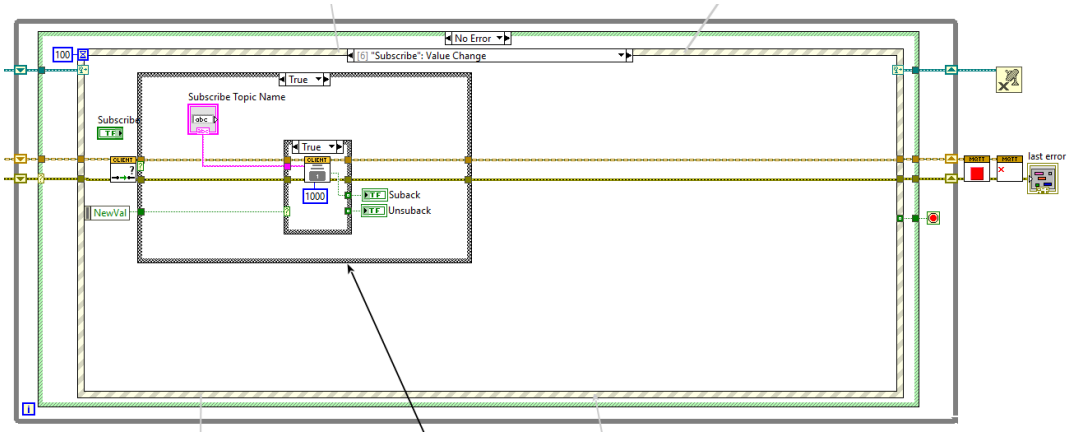
Once the LabVIEW MQTT Client connects to the EDM MQTT broker, the client ID should show up in the EDM MQTT broker Connected clients tab.



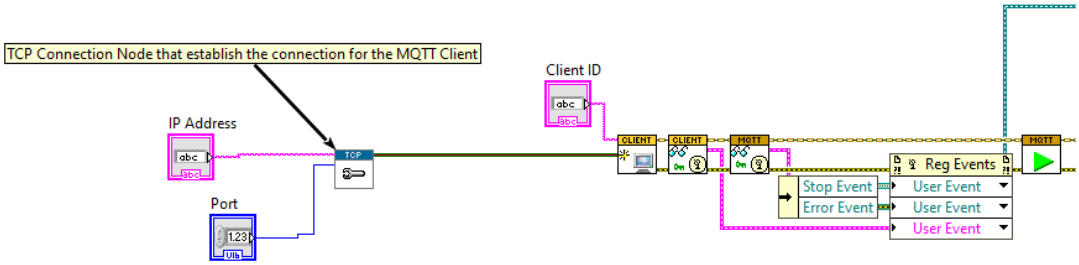
Then the LabVIEW MQTT Client is ready to subscribe or publish topics to EDM. Please note that since this is a basic LabVIEW MQTT Client Demo, the user may need to add additional displays to the front panel when requesting data such as the test status or input channel information.

LabVIEW Block Diagram

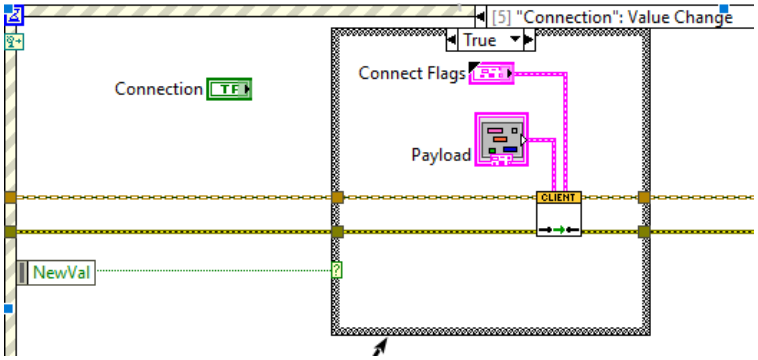




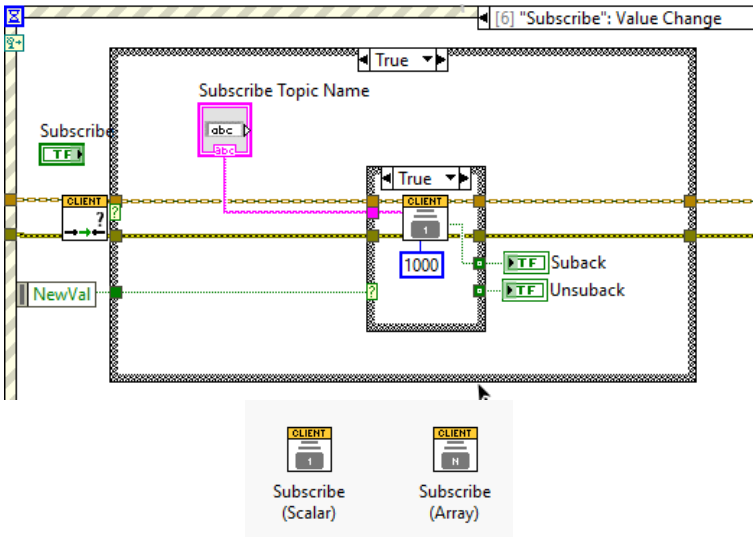
The above shows a brief overview the entire block diagram used in making the LabVIEW MQTT Client Demo, from the start of the MQTT client, the main function loop of every event to the disconnection and stop function of the MQTT client with any error messages if they occurred. The entire block diagram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



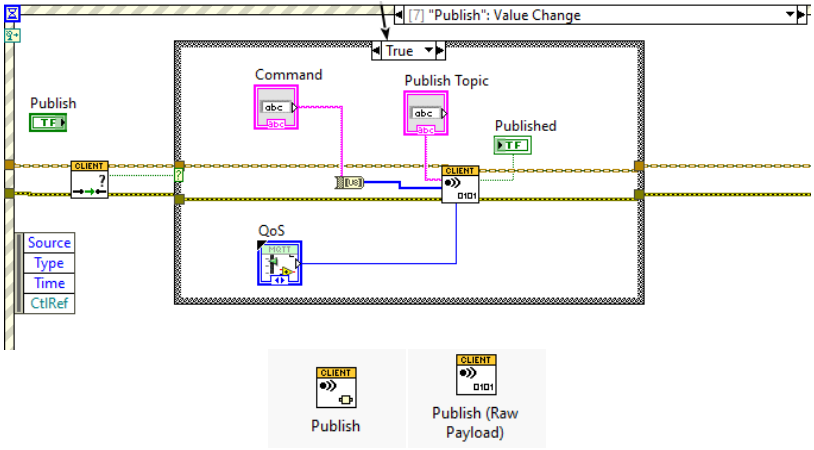
Starting with the left side of the block diagram is the TCP configuration and client ID control nodes for the Create MQTT Client node. At the top of each MQTT node is the MQTT Client in and out and at the bottom of each MQTT node is the error in and out. Both the MQTT Client and Error goes through each 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.



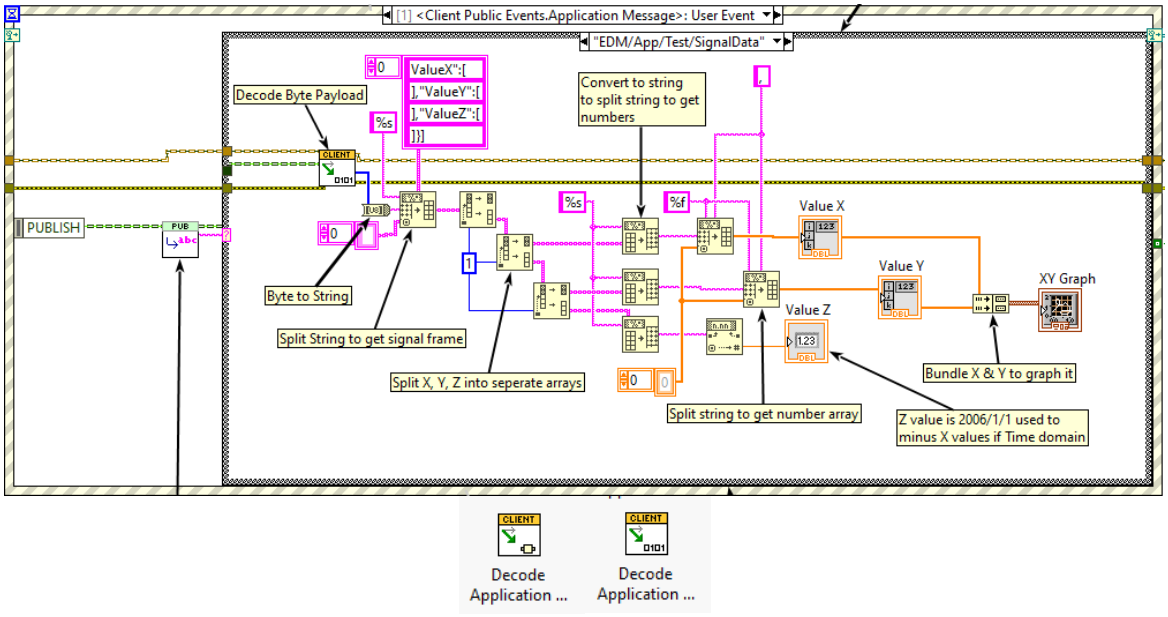
Next is the event structure after the Start node that encompasses all the functions of the LabVIEW MQTT Client Demo from subscribe, publish, connect, stop, public events and so on. Here is the connect button event structure where it takes in the payload and connect flags to connect to the EDM MQTT broker.



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.

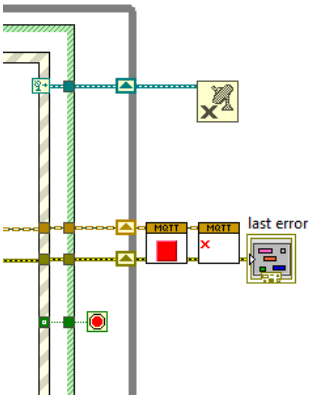


Here is the publish event structure where the Publish (Raw Payload) node takes in the topic name, command in bytes and quality of service. The publish node has to be raw payload in order for EDM to read in the command that comes from the LabVIEW MQTT Client Demo. Once the payload has been sent to EDM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



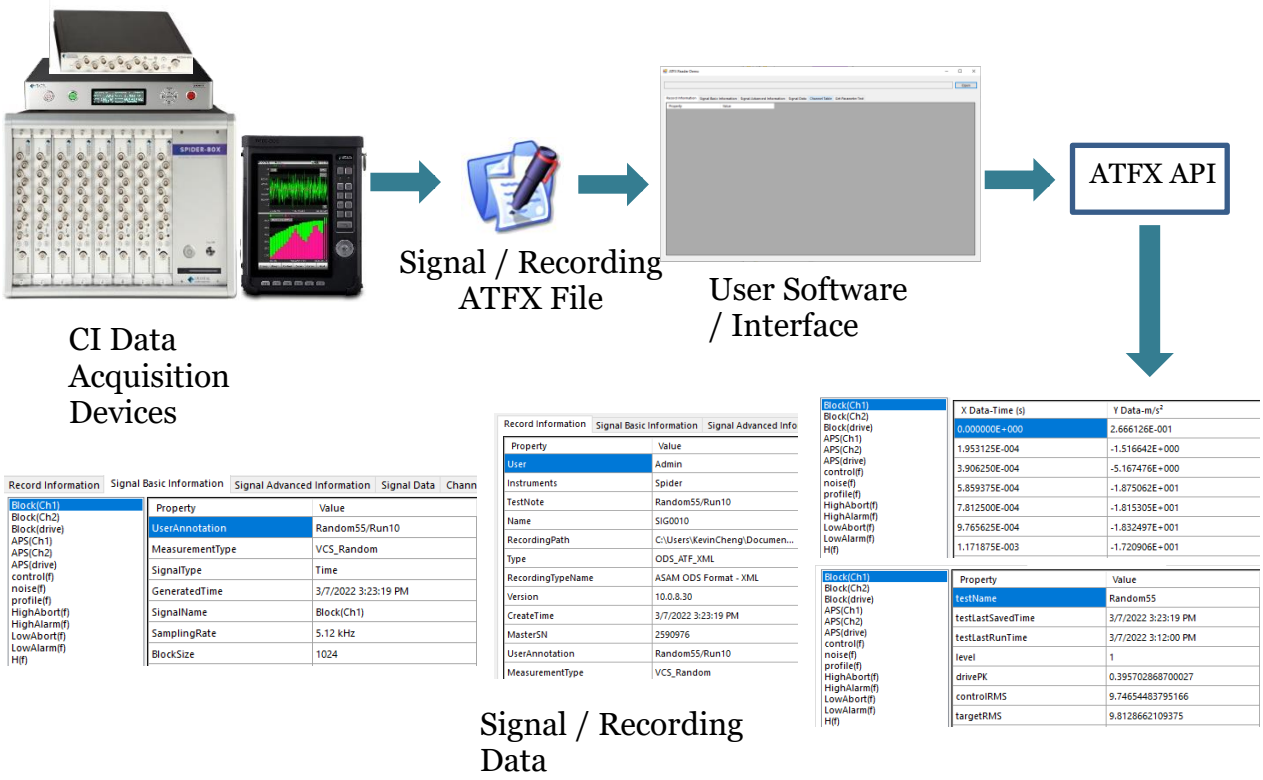
Once the subscribe topic sent by EDM has been received, it will then decode the payload in the public event structure. First, in this structure, it must determine the type of topic name to handle how to display the data from the getTopicName node on the left side. Once it determines the topic, it then 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.



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.

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

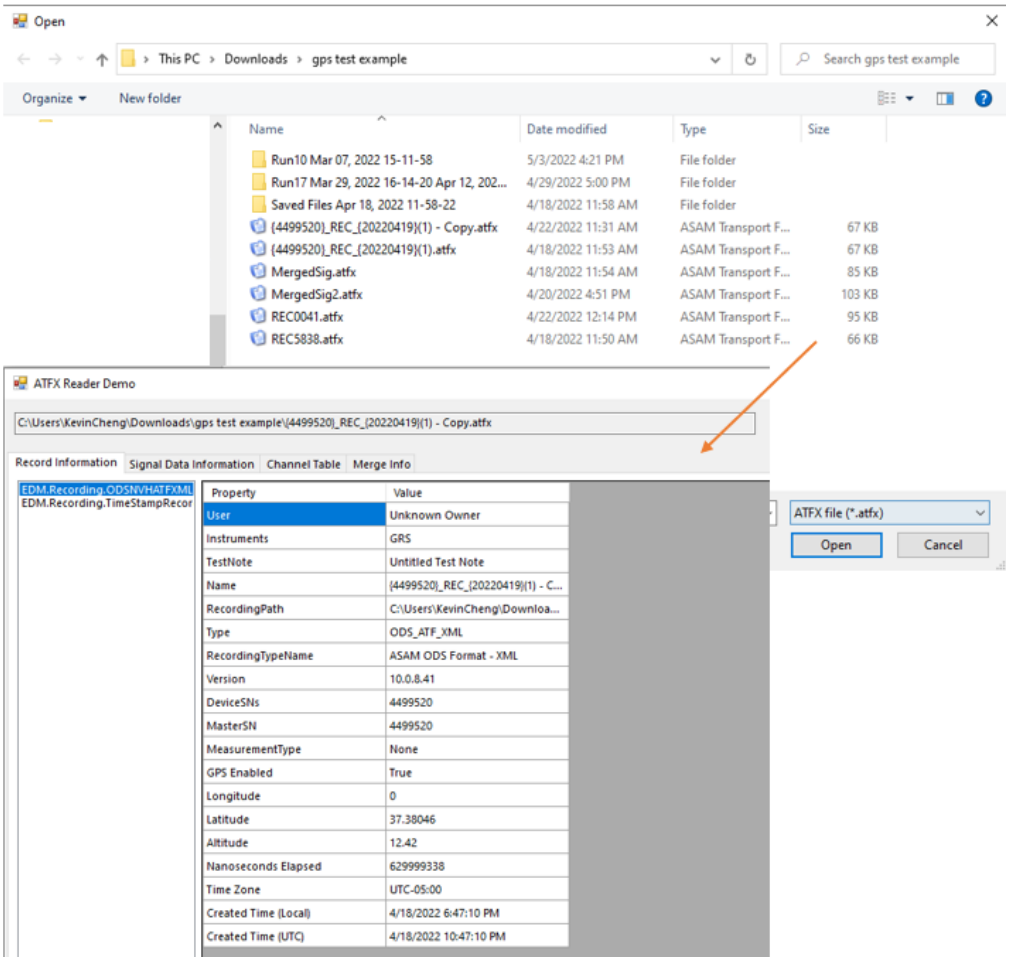


The Crystal Instruments (CI) ATFX ODS Signal Reader Application Programming Interface (API) consists of 2 Windows Dynamic-Linked Libraries (DLL) providing third-party applications an interface to access the signal data stored in the ASAM Transport Format XML (ATFX) files.
ATFX files are formatted according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

END USER LICENSE AGREEMENT FOR CRYSTAL INSTRUMENTS SOFTWARE

--- Updated May 11, 2022

IMPORTANT – READ CAREFULLY. This End User License Agreement ("the Agreement") is a legally binding agreement between you ("the Licensee") and Crystal Instruments Corporation ("Crystal Instruments") for the Crystal Instruments EDM (Engineering Data Management) software, PA (Post Analyzer), EDM Cloud, CI Store, EDC (Embedded Device Control), various API, or the embedded software installed in CoCo, Spider and other series hardware, which includes software components and tools and written documentation ("Software") that accompanies this Agreement. This Agreement contains **WARRANTY AND LIABILITY DISCLAIMERS**.

1. SCOPE OF THE LICENSE RIGHT
1.1 By installing, copying, or using the Software, the Licensee agrees to be bound by the terms of this Agreement.
1.2 Subject to the terms and conditions of this Agreement, Crystal Instruments hereby grants to the Licensee a non-exclusive, non-transferable, right to use the Software, as ordered by the Licensee, solely for the Licensee's own use and solely with the Crystal Instruments hardware for which it is intended.
1.3. The Licensee shall not be entitled to copy or distribute the Software or parts thereof; publish the Software for others to copy; sell, rent, lease, or lend the Software; or transfer or assign the Software or the license rights to the Software to a third party in any other way whatsoever.
1.4 The Licensee shall, however, be entitled to make back-up copies of the Software to the extent that applicable law expressly permits. The use of the back-up copy shall be subject to the terms of this Agreement.
1.5 The Licensee shall ensure that the Software is stored in such a manner that third parties do not have access to it and that a third party does not come into possession of the Software in any other way. The Licensee shall make all employees who have access to the Software fully aware of this obligation.

2. CHANGES TO THE SOFTWARE
2.1 The Licensee shall not be entitled to make any changes to the Software, or reverse engineer, decompile, or disassemble the Software, except and only to the extent that applicable law expressly permits.
2.2 In the event of the Licensee or a third party interfering with or making any changes to the Software, Crystal Instruments may terminate the Agreement with immediate effect, and Crystal Instruments hereby disclaims any liability for the consequences of such interference or change.

3. INTELLECTUAL PROPERTY RIGHTS
3.1 The Software is protected by copyright law and other intellectual property laws. Crystal Instruments or its suppliers own all copyright and any other intellectual property rights in the Software. The Licensee shall respect Crystal Instruments' and its suppliers' rights and the Licensee shall be fully liable in the event of any violation of these rights, including unauthorized passing on of the Software or any part of it to a third party.
3.2 The Licensee shall not be entitled to break, change or delete any security codes or license keys, nor shall the Licensee be entitled to change or remove statements in the Software or on the media on which the Software is delivered regarding copyrights, trademarks, or any other proprietary notices.
3.3 Information and data supplied by Crystal Instruments with the Software, such as, but not limited to, user manuals and documentation, are proprietary to Crystal Instruments or its suppliers. Such information is furnished solely to assist the Licensee in the installation, operation and use of the Software and the Licensee agrees not to reproduce or copy such information, except as is reasonably necessary for proper use of the Software.

4. TRADEMARKS
4.1 The Licensee acknowledges Crystal Instruments' and its suppliers' sole ownership of any trademarks including service marks, logos and other proprietary marks submitted with the Software, and all associated goodwill. This Agreement does not grant the Licensee any rights to the trademarks of Crystal Instruments and its suppliers.
4.2 The Licensee agrees not to use the trademarks in any manner that will diminish or otherwise damage Crystal Instruments' or its suppliers' goodwill in the trademarks. The Licensee agrees not to adopt, use, or register any corporate name, trade name, trademark, domain name, service mark, certification mark, or other designation similar to, or containing in whole or in part, the trademarks of Crystal Instruments.

5. CLOUD SERVICE PROVIDED BY CRYSTAL INSTRUMENTS
5.1 Data Location When cloud service is enabled, Crystal Instruments Corporation may process and store the customer data anywhere Crystal Instruments Corporation or its agents maintain facilities and services.
5.1.1 Facilities All facilities used to store and process an application and customer data will adhere to reasonable security standards no less protective than the security standards at facilities where Crystal Instruments Corporation processes and stores its own information of a similar type.

5.2 Data Processing and Security
5.2.1 Scope of Processing By entering into this agreement, customer instructs Crystal Instruments Corporation to process customer personal data and other data related to its services only in accordance with applicable law: (a) to provide the cloud services; (b) as further specified by customer via customer's use of the cloud services (including the admin console and other functionality of the services); (c) as documented in the form of this agreement, including these terms; and (d) as further documented in any other written instructions given by customer and acknowledged by Crystal Instruments Corporation as constituting instructions for purposes of these Terms.
5.2.2 Data Security Crystal Instruments Corporation will use third party technical measures to protect customer data against accidental or unlawful destruction, loss, alteration, unauthorized disclosure or access. Crystal Instruments Corporation is not responsible or liable for the deletion of or failure to store any customer data and other communications maintained or transmitted through use of the services. In addition, Crystal Instruments is not responsible or liable for unauthorized access of the customer data. Customer is solely responsible for securing and backing up data. Crystal

Instruments Corporation does not warrant that the operation of the software or the services will be error-free or uninterrupted. Neither the software nor the services are designed, manufactured, or intended for high risk activities.

5.2.3 Data Deletion

Deletion by Customer: Crystal Instruments Corporation will enable Customer to delete Customer Data during the Term in a manner consistent with the functionality of the Services.

Deletion on Termination. On expiry of the Term, Crystal Instruments would delete all Customer Data. Customer acknowledges and agrees that Customer will be responsible for exporting, before the Term expires, any Customer Data it wishes to retain afterwards.

5.3 Accounts Customer must have an account to use the services, and is responsible for the information it provides to create the account, the security of passwords for the account, and for any use of its account. If customer becomes aware of any unauthorized use of its password or its account, Customer will notify Crystal Instruments Corporation as promptly as possible. Crystal Instruments Corporation has no obligation to provide customer multiple accounts.

5.4 Payment Terms for Cloud Service

5.4.1 Free Quota Certain services are provided to customer without charge up to the fee threshold, as applicable.

5.4.2 Online Billing At the end of the applicable fee accrual period, Crystal Instruments Corporation will issue an electronic bill to customer for all charges accrued above the fee threshold based on (i) Customer's use of the Services during the previous fee accrual period; (ii) any additional units added; (iii) any committed purchases selected; and/or (iv) any package purchases selected. For use above the fee threshold, customer will be responsible for all fees up to the amount set in the account and will pay all fees in the currency set forth in the invoice. If customer elects to pay by credit card, debit card, or other non-invoiced form of payment, Crystal Instruments Corporation will charge (and customer will pay) all fees immediately at the end of the fee accrual period. If customer elects to pay by invoice (and Crystal Instruments Corporation agrees), all fees are due as set forth in the invoice. Customer's obligation to pay all fees is non-cancellable. Crystal Instruments Corporation's measurement of Customer's use of the services is final. Crystal Instruments Corporation has no obligation to provide multiple bills. Payments made via wire transfer must include the bank information provided by Crystal Instruments Corporation.

5.4.3 Payment Information Crystal Instruments Corporation will not store any payment related information on its facilities. All payment information, including recurring payments are stored at a third party facility. Crystal Instruments will not be responsible or liable for unauthorised access to this information.

5.4.4 Taxes for Cloud Services

(a) Customer is responsible for any taxes, and customer will pay Crystal Instruments Corporation for the services without any reduction for taxes. If Crystal Instruments Corporation is obligated to collect or pay taxes, the taxes will be invoiced to customer, unless customer provides Crystal Instruments Corporation with a timely and valid tax exemption certificate authorized by the appropriate taxing authority. In some states the sales tax is due on the total purchase price at the time of sale and must be invoiced and collected at the time of the sale. If customer is required by law to withhold any taxes from its payments to Crystal Instruments Corporation, customer must provide Crystal Instruments Corporation with an official tax receipt or other appropriate documentation to support such withholding. If under the applicable tax legislation the services are subject to local VAT and the customer is required to make a withholding of local VAT from amounts payable to Crystal Instruments Corporation, the value of services calculated in accordance with the above procedure will be increased (grossed up) by the customer for the respective amount of local VAT and the grossed up amount will be regarded as a VAT inclusive price. Local VAT amount withheld from the VAT-inclusive price will be remitted to the applicable local tax entity by the customer and customer will ensure that Crystal Instruments Corporation will receives payment for its services for the net amount as would otherwise be due (the VAT inclusive price less the local VAT withheld and remitted to applicable tax authority).

(b) If required under applicable law, customer will provide Crystal Instruments Corporation with applicable tax identification information that Crystal Instruments Corporation may require to ensure its compliance with applicable tax regulations and authorities in applicable jurisdictions. Customer will be liable to pay (or reimburse Crystal Instruments Corporation for any taxes, interest, penalties or fines arising out of any mis-declaration by the Customer.

5.4.5 Invoice Disputes and Refunds Any invoice disputes must be submitted prior to the payment due date. If the parties determine that certain billing inaccuracies are attributable to Crystal Instruments Corporation, Crystal Instruments Corporation will not issue a corrected invoice, but will instead issue a credit memo specifying the incorrect amount in the affected invoice. If the disputed invoice has not yet been paid, Crystal Instruments Corporation will apply the credit memo amount to the disputed invoice and Customer will be responsible for paying the resulting net balance due on that invoice. To the fullest extent permitted by law, customer waives all claims relating to fees unless claimed within thirty days after charged (this does not affect any customer rights with its credit card issuer). Refunds (if any) are at the discretion of Crystal Instruments Corporation and will only be in the form of credit for the services. Nothing in this Agreement obligates Crystal Instruments Corporation to extend credit to any party.

5.4.6 Delinquent Payments; Suspension Late payments may bear interest at the rate of 1.5% per month (or the highest rate permitted by law, if less) from the payment due date until paid in full. customer will be responsible for all reasonable expenses (including attorneys' fees) incurred by Crystal Instruments Corporation in collecting such delinquent amounts. If customer is late on payment for the services, Crystal Instruments Corporation may suspend the services or terminate the account(s) and services(s) for breach

5.5 Account Term & Termination

5.5.1 Account Term The term of the account will begin on the effective date and continue until the agreement is terminated.

5.5.2 Termination for Breach Crystal Instruments Corporation may terminate account for breach if: (i) the account(s) is in material breach of the agreement; or (ii) the customer ceases its business operations or becomes subject to insolvency proceedings and the proceedings are not dismissed within ninety days.

5.5.3 Termination for Convenience Customer may stop using the cloud service at any time. Customer may terminate the account(s) and services for its convenience at any time on prior written notice and upon termination, must cease use of the applicable services.

Crystal Instruments Corporation may terminate the account(s) or services for its convenience at any time without liability to Customer.

5.5.4 Effect of Termination If the account(s) or services(s) are terminated, then: (i) the rights granted by one party to the other will immediately cease; (ii) all fees owed by customer to Crystal Instruments Corporation are immediately due upon receipt of the final electronic bill; (iii) customer will delete the software, any application and any data; and (iv) upon request, each party will use commercially reasonable efforts to return or destroy all confidential information of the other party.

5.6 Customer Obligations for Cloud Services
5.6.1 Compliance Customer is solely responsible for account information and data and for making sure its usage of services is consistent with the terms of the services. Crystal Instruments Corporation reserves the right to review the data for compliance.
5.6.2 Restrictions
Customer will not, and will not allow third parties under its control to: (a) copy, modify, create a derivative work of, reverse engineer, decompile, translate, disassemble, or otherwise attempt to extract any or all of the source code of the services (except to the extent such restriction is expressly prohibited by applicable law); (b) sublicense, resell, or distribute any or all of the services; or (c) create multiple account(s) to simulate or act as a single account or otherwise access the services in a manner intended to avoid incurring fees or exceed usage limits or quotas;
5.6.3 Third Party Components
Third party components (which may include open source software) of the services may be subject to separate license agreements. To the limited extent a third party license expressly supersedes this agreement, that third party license governs customer's use of that third party component.

6. EXPORT RESTRICTIONS
The Software may be subject to the export control laws and regulations of the United States. The Licensee must comply with all domestic and international export control laws and regulations that apply to the Software. These laws include restrictions on destinations, end users, and end use.

7. THE LICENSEE'S CHOICE OF SOFTWARE
The Software is a standard product, which is delivered by Crystal Instruments with the functions that are specified in the accompanying documentation. Any assistance provided by Crystal Instruments in connection with the choice of the Software will be based on the Licensee's information about the Licensee's business provided to Crystal Instruments. The Licensee shall be responsible for both the completeness and the accuracy of such information. Crystal Instruments makes no representations or warranties as to whether the Software meets the functionality or other requirements of the Licensee and assumes no liability therefor.

8. WARRANTIES AND DISCLAIMERS
8.1 The Licensee shall be under obligation to examine and test the Software immediately after installation of the Software.
8.2 On condition that Crystal Instruments is fully paid for the Software that Customer purchased, Crystal Instruments warrants that the Software will be free of material defects for a period of 12 months after the delivery of the Software to Licensee (the "Warranty Period"). A defect in the Software shall be regarded as material if it has a material adverse effect on the functionality of the Software as a whole or if it prevents operation of the Software. Minor bugs or functions that can be improved are not viewed as a defect.
8.3 If the Licensee documents that there is a material defect in the Software, and notifies Crystal Instruments of the defect within the Warranty Period, Crystal Instruments will, at its discretion, without charge: (a) deliver a new version of the Software without the material defect, or (b) remedy the defect, or (c) provide Licensee with instructions for procedures or methods (workarounds) which result in the defect not having a significant effect on the Licensee's use of the Software. If Crystal Instruments fails to do any of the above within 30 days (or such longer period of time as is reasonably necessary given the nature of the defect), the Licensee may terminate this Agreement upon notice to Crystal Instruments, in which event Crystal Instruments will refund to Licensee a pro-rated portion of the license fee paid by Licensee for the Software (based on the portion of the Warranty Period remaining as of the date Licensee notified Crystal Instruments of the defect), provided Licensee returns to Crystal Instruments all the Licensee's versions and copies of the Software, and all manuals and accompanying documentation. This paragraph states the sole obligations of Crystal Instruments, and the sole remedy of Licensee, for defects in the Software, and the parties shall not be entitled to bring further claims against each other.
8.4 EXCEPT FOR THE EXPRESS WARRANTY IN SECTION 7.2 ABOVE, THE SOFTWARE IS PROVIDED "AS IS" AND WITHOUT ANY OTHER WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO WARRANTIES OF ACCURACY, COMPATIBILITY WITH OTHER SOFTWARE OR HARDWARE, MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT. CRYSTAL INSTRUMENTS DOES NOT WARRANT THAT THE OPERATION OF THE SOFTWARE WILL BE WITHOUT INTERRUPTIONS, DEFECT-FREE, OR ERROR-FREE OR THAT PRODUCT DEFECTS OR ERRORS CAN OR WILL BE REMEDIED OR CORRECTED.

9. 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 hardware 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 Software updates 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.

10. LIABILITY AND LIMITATION OF LIABILITY
10.1 CRYSTAL INSTRUMENTS SHALL NOT BE LIABLE FOR ANY INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES (INCLUDING BUT NOT LIMITED TO LOSS OF EXPECTED PROFIT, LOSS OF DATA OR THEIR RECOVERY, LOSS OF GOODWILL OR ANY OTHER SIMILAR DAMAGES), UNDER ANY LEGAL THEORY, IN CONNECTION WITH THE USE OF THE SOFTWARE OR THE INABILITY TO USE THE SOFTWARE, REGARDLESS OF WHETHER CRYSTAL INSTRUMENTS HAS BEEN INFORMED OF THE POSSIBILITY OF SUCH DAMAGES.
10.2 IN NO EVENT SHALL THE TOTAL LIABILITY OF CRYSTAL INSTRUMENTS TO LICENSEE ARISING OUT OF OR RELATING TO THE SOFTWARE EXCEED THE LICENSE FEE PAID BY LICENSEE FOR THE SOFTWARE.
10.3 Crystal Instruments shall not be liable for any errors, defects, or deficiencies which are not related to the Software, nor shall Crystal Instruments be liable for the integration or interaction between the Software and the Licensee's existing hardware and software. Crystal Instruments shall not be liable for the effect of any upgrades on existing hardware, software, or adjustments for the Software regardless of whether such adjustments were developed by Crystal Instruments.
10.4 Crystal Instruments shall have no liability of any nature relating to software or content of third parties that may be included in the Software.
10.5 The limitations in this Section 9 will apply even in the event of failure of essential purpose of any remedy.

11. GOVERNMENT USERS
The Software and related 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.

12. TERM AND TERMINATION
12.1 The term of this Agreement, and Licensee's license rights, which may be referred to the activation period of license, shall be as indicated in Licensee's order. Such term may be perpetual, or may be of limited duration in the event the Software is provided to Licensee for demonstration, evaluation or other similar purposes. Licensee acknowledges that if Licensee's rights are of limited duration, the license key provided to Licensee to enable use of the Software may cease to allow use of the Software after expiration of such activation period.
12.2 Upon termination of the Agreement for any reason, the Licensee is obliged to immediately return or destroy the Software and all copies thereof as directed by Crystal Instruments and, if requested by Crystal Instruments, to certify in writing as to the destruction or return of the Software and all copies thereof.

13. DEFAULTS
If the Licensee is in default of the Agreement, the Licensee's rights under the Agreement shall terminate with immediate effect, and the Licensee shall be under an obligation to return the Software, including any back-up copies and accompanying documentation, without a right to repayment. In addition, Crystal Instruments shall be entitled to damages for any loss, which Crystal Instruments may suffer, in accordance with the general rules of United States law, including all losses, damages, costs, expenses, etc., without any limitations, incurred or suffered by Crystal Instruments as a result of claims from any third party in relation to the Licensee's breach of the Agreement.

14. UPDATES AND RENEW
14.1 For one year after the delivery of the Software, Crystal Instruments will provide Licensee, free of charge, with any updates to the Software that Crystal Instruments makes generally available to its customers. Licensee may renew such right to receive updates, for additional periods of one year each, by paying Crystal Instruments the support renewal fee in effect at the time of such renewal. Licensee acknowledges that if Licensee elects not to renew the right to receive updates, the license key provided to Licensee to enable use of the Software may thereafter cease to allow installation and use of updates. Notwithstanding the above, Crystal Instruments may charge an additional license fee for any optional upgrades Crystal Instruments may release, which include significant new functionality and which Crystal Instruments does not make available without charge to its customers generally.
14.2 Crystal Instruments and the Licensee can agree on the other term about the period of software update after the sales.
14.3 Crystal Instruments has the rights to control the period of software update through various technical means including online activation or certain algorithm embedded in the license keys. The Licensee has no rights to reverse engineer, decompile, or disassemble the algorithm.

15. CHOICE OF LAW AND COURT OF JURISDICTION
15.1 The Agreement shall be governed by the laws of the State of California, and applicable United States federal law.
15.2 Any suit or proceeding arising out of this Agreement shall be brought only in a court located in Santa Clara County, California, and the parties submit to the exclusive jurisdiction and venue of such courts; provided, however, that Crystal Instruments may seek injunctive relief for any breach of this Agreement by Licensee in any court that would otherwise have jurisdiction over Licensee.

16. GENERAL PROVISIONS
16.1 Failure by Crystal Instruments to exercise or enforce any rights hereunder shall not be deemed to be a waiver of any such rights nor affect the exercise or enforcement thereof at any time or times thereafter.
16.2 If any provision or part of this Agreement is or is held by any court of competent jurisdiction to be unenforceable or invalid, such unenforceability or invalidity shall not affect the enforceability of any other provision.
16.3 This Agreement constitutes the entire agreement between the parties with respect to its subject matter and supersedes all prior or contemporaneous understandings regarding that subject matter. No amendment to or modification of this Agreement will be binding unless in writing and signed by an authorized officer of Crystal Instruments.
16.4 Licensee may not transfer or assign Licensee's rights under this Agreement to any third party without the prior written consent of Crystal Instruments, including by operation of law.

17. THIRD PARTY SOFTWARE LICENSE/NOTICES

Crystal Instruments Software uses a number of software products from 3rd parties that are under one of the following licenses, Apache License, GPL License, LGPL License and MIT License. Please contact Crystal Instruments to obtain the most updated list of 3rd party software that are incorporated in the Software.

License Type Definition

***Apache License**
Apache License is a free software license authored by the Apache Software Foundation (ASF). The Apache License requires preservation of the copyright notice and disclaimer. Like any free software license, the Apache License allows the user of the software the freedom to use the software for any purpose, to distribute it, to modify it, and to distribute modified versions of the software, under the terms of the license, without concern for royalties.
The 2.0 version of the Apache License was approved by the ASF in 2004. The goals of this license revision have been to reduce the number of frequently asked questions, to allow the license to be reusable without modification by any project (including non-ASF projects), to allow the license to be included by reference instead of listed in every file, to clarify the license on submission of contributions, to require a patent license

on contributions that necessarily infringe the contributor's own patents, and to move comments regarding Apache and other inherited attribution notices to a location outside the license terms

***GPL License**
The GNU General Public License (GNU GPL or GPL) is the most widely used free software license, which guarantees end users (individuals, organizations, companies) the freedoms to use, study, share (copy), and modify the software. Software that ensures that these rights are retained is called free software. The license was originally written by Richard Stallman of the Free Software Foundation (FSF) for the GNU project.

***LGPL License**
LGPL (formerly the GNU Library General Public License) is a free software license published by the Free Software Foundation (FSF). The LGPL allows developers and companies to use and integrate LGPL software into their own (even proprietary) software without being required (by the terms of a strong copyleft) to release the source code of their own software-parts.

***MIT License**
The MIT License is a permissive free software license originating at the Massachusetts Institute of Technology (MIT)。 The MIT License is compatible with many copyleft licenses, such as the GNU General Public License (GNU GPL). Any software licensed under the terms of the MIT License can be integrated with software licensed under the terms of the GNU GPL.

--- Updated May 11, 2022