Beijer ELECTRONICS MQTT Client JSON – iX Developer Script Module User Guide

Beijer ELECTRONICS MQTT Client JSON – iX Developer Script Module User Guide

Contents

Function and area of use

This document explains how to subscribe to topics of a MQTT broker and also publish data to these topics. This example utilizes a JSON object “MachineData” containing 10 Tags, DeviceId and TimeStamp to be exchanged with the MQTT broker.

About this document

This quick start document should not be considered as a complete manual. It is an aid to be able to startup a normal application quickly and easily.

Use the following hardware, software, drivers and utilities in order to obtain a stable application:

In this document we have used following software and hardware

- iX Developer 2.40 SP5 / SP6
- X2 baseV2/pro/marine/control/extreme series, C2, PC devices

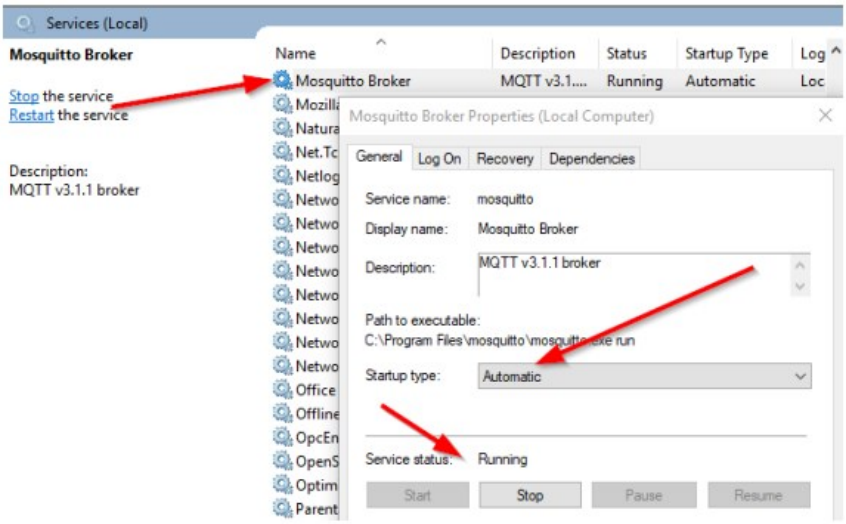
For further information refer to

- iX Developer Reference Manual (MAxx831)
- iX Developer User’s Guide (MAxx832)
- Beijer Electronics knowledge database, HelpOnline

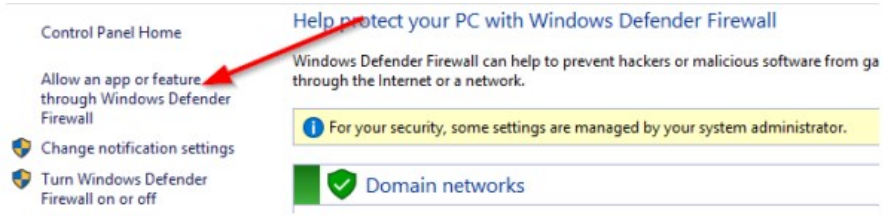
  This document and other quick start documents can be obtained from our homepage. Please use the address support.**europe@beijerelectronics.com** for feedback.

Eclipse Mosquitto™

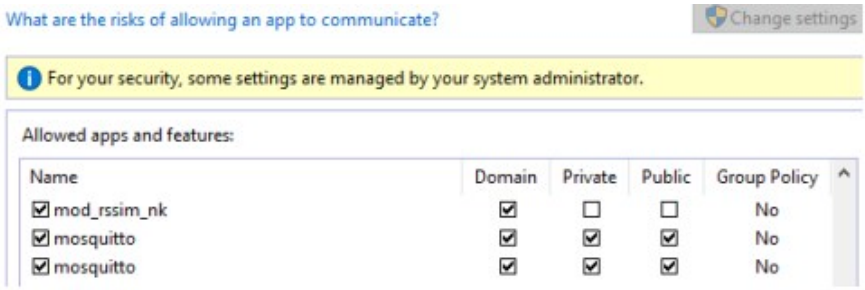
Eclipse Mosquitto™ is an open source MQTT broker. **https://mosquitto.org/download/** After installation go to services and check whether Mosquitto´s Startup type is set to “Automatic”, which means that it is started automatically with Windows. After the installation you either have to restart Windows or start Mosquitto manually.



In Windows control panel go to “Allow an app…”

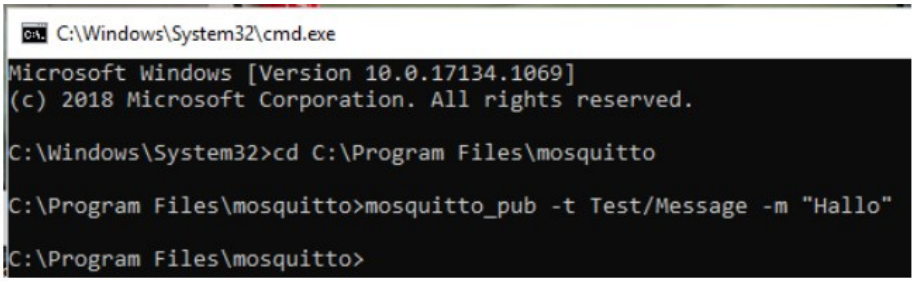


Allow access to mosquitto.exe for all network types



Testing Mosquitto
To publish a message start CMD and change to the Mosquitto directory.

- Use the mosquitto_pub.exe to publish a message to a certain topic.
- The parameter -t specifies the topic Test/Message
- The parameter -m specifies the message “Hallo”



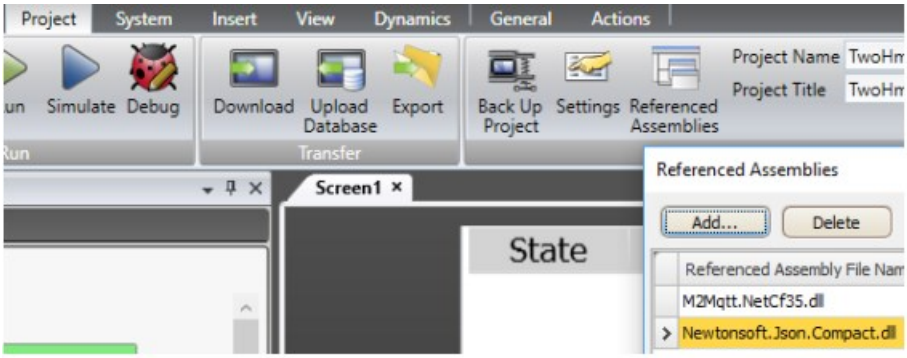
- To subscribe to one or multiple topics use the mosquitto_sub.exe

- The parameter specifies the host(broker) that you want to connect to -h 127.0.0.1
- The parameter specifies the port -p 1883
- The parameter specifies the topic(s) -t Test/#
- The # means that you subscribe to all subtopics of the topic Test

M2Mqtt and Json Library

Since iX 2.40 SP4 the M2Mqtt.NetCf35.dll as well as the Newtonsoft.Json.Compact.dll are included in the iX installation.
They are located in the subfolder ..\Bin\Extensions of your iX installation.

Include the libraries into your project
Add the M2Mqtt.NetCf35.dll and the Newtonsoft.Json.Compact.dll as Referenced Assemblies to your project.



Connect to a broker
Create a new instance of the MqttClient class: static string sMqttBroker = “127.0.0.1”; // “192.168.99.123”; // “BE526KFN” // “localhost” static MqttClient mqttClient;
The simplest overload of the MqttClient class constructor needs only the MQTT broker URL as parameter without specifying the protocol (it’s TCP) and the port (the default is 1883); it can be an host name or an IP address.
There are other overloads that you can use to specify:

- A different port (not standard 1883 or 8883 for TLS/SSL connection)
- The X509 certificate needed for a TLS/SSL connection to the broker (as we’ll see in the following sections)
- Optional callbacks for above certificate validation

Create a new MqttClient instance inside the Created method of the script modul “SCM_MQTT_Client”.

void SCM_MQTT_Client_Created(System.Object sender, System.EventArgs e){ mqttClient = new MqttClient(sMqttBroker);
// The below overload has to be used for X2 series devices with .Net compact framework when using a certificate
// byte[] certBytes = ReadFileToByteArray(@”C:\Users\310208195\root.crt”);
// X509Certificate certificate = new X509Certificate(certBytes);
// mqttClient = new MqttClient(sMqttBroker, 8883, true, certificate, certificate, MqttSslProtocols.TLSv1_2);
// The below overload can be used for C2 series or PC Runtime with .Net full framework when using a certificate
// mqttClient = new MqttClient(sMqttBroker, 8883, true, new X509Certificate(@”C:\Users\310208195\root.crt”), new X509Certificate(@”C:\Users\310208195\root.crt”), MqttSslProtocols.TLSv1_2);

// Register to message received
mqttClient.MqttMsgPublishReceived += client_receivedMessage;
m_Timer= new Timer();
m_Timer.Enabled = true;

m_Timer.Interval = 2500; //ms
m_Timer.Tick += OnTimerTick;

Subscribe to one or multiple topics and connect to the broker
string[] sTopics = new string[] {“Machine1”}; byte[] QOSbytes = new byte[] {
MqttMsgBase.QOS_LEVEL_EXACTLY_ONCE};
Specify a QOS level for each topic – the array size of “QOSbytes” has to fit the array size of “sTopics”!

public void mqttConnect(){ if (mqttClient.IsConnected) return; string clientId = Guid.NewGuid().ToString();
mqttClient.Connect(clientId); // Subscribe to topic mqttClient.Subscribe(sTopics, QOSbytes);

Configuration of the Json object
The configuration as well as serialization/deserialization of the Json object are done inside the script module “SCM_JSON”. In the definition section the Tag Array is specified. private static GlobalDataItem[] Machine1Tags = new GlobalDataItem[{

- Globals.Tags.pubTemperature,
- Globals.Tags.pubProducedPCS,
- Globals.Tags.pubVoltage,
- Globals.Tags.pubCurrent,
- Globals.Tags.pubSpeed,
- Globals.Tags.pubBatchName,
- Globals.Tags.pubStatus,
- Globals.Tags.pubPower,
- Globals.Tags.pubOperatingHours,
- Globals.Tags.pubAlarmsActive};

The data structure is defined inside the class “MachineData”.
public class MachineData{
public string DeviceId { get; set; }
public string TimeStamp { get; set; }
public object[] Data { get; set; }
public MachineData(){}
public MachineData(string deviceId, DateTime timeStamp, object[] data){
DeviceId = deviceId;
TimeStamp = timeStamp.ToString();
Data = data;
}
}

Publish the Json object to a topic
private bool mqttPublish(string sTopic, string sMessage){
bool bSuccess = false;
System.Text.ASCIIEncoding enc = new System.Text.ASCIIEncoding();
byte[] MessageBytes = enc.GetBytes(sMessage);
if (mqttClient.IsConnected){
mqttClient.Publish(sTopic, MessageBytes,
MqttMsgBase.QOS_LEVEL_EXACTLY_ONCE, false);
bSuccess = true;
}
return bSuccess;
}

In this example the JSON object is published every 2500ms utilizing a timer.

```
private void OnTimerTick(System.Object Sender, EventArgs e){
Globals.Tags.mqttConnected.Value = mqttClient.IsConnected;
Globals.Tags.mqttBrokerTopics.Value = mqttClient.IsConnected? sMqttBroker +"
connected – Topics: " + string.Join(";", sTopics): "not connected";
if (mqttClient.IsConnected)
publishMessage(Globals.SCM_JSON.Create_JSON());
}
private void publishMessage(string sJSON){
if (Globals.Tags.EncryptMessage.Value)
sJSON = encryptMessage(sJSON);
mqttPublish(sTopics[0], sJSON);
}
```

Receive and evaluate a received Json object

```
void SCM_MQTT_Client_Created(System.Object sender, System.EventArgs e){
…..
mqttClient.MqttMsgPublishReceived += client_receivedMessage;
…..
private void client_receivedMessage(object sender, MqttMsgPublishEventArgs e){
// Handle message received
string receivedMessage =
System.Text.Encoding.Default.GetString(e.Message,0,e.Message.Length);
Globals.Tags.JSON_String.Value = receivedMessage;
if (checkEncryption(receivedMessage))
receivedMessage = decryptMessage(receivedMessage);
Globals.SCM_JSON.Deserialize_JSON(receivedMessage);
```

Disconnect from a broker

```
public void mqttDisconnect(){
// Unsubscribe topic
mqttClient.Unsubscribe(sTopics);
if (mqttClient.IsConnected)
mqttClient.Disconnect();
```

Note!

- Always make sure to disconnect the client before closing the application!
- This example works well for X2 baseV2/pro/marine/control/extreme series, C2 and PC series devices.
- Please follow below guidelines how to install into your application.

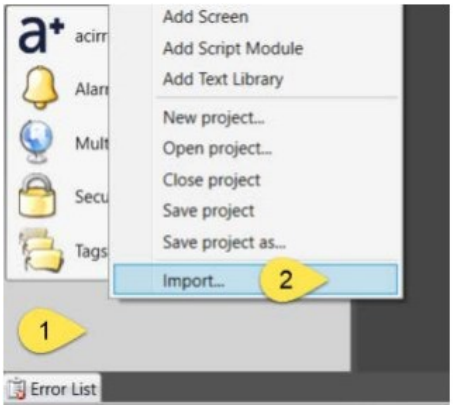
Adding the MQTT client and Json serialisation functionality

Implementation

1. Import the script module "SCM_MQTT_Client" , see example project (iX_MQTT_Client_JSON_V1_0_0).
2. Import the script module "SCM_JSON" , see example project (iX_MQTT_Client_JSON_V1_0_0).
3. Add the M2Mqtt.NetCf35.dll and the Newtonsoft.Json.Compact.dll as referenced assemblies to your project (see 5 and 5.1).
4. Import the screen, see example project (iX_MQTT_Client_JSON_V1_0_0).
5. Add the Tags and/or adjust Tags and Data Types.

6. Adapt the screen to your needs.
7. Transfer the application.
8. Run the application.

Import the project parts
Follow the steps to add the enclosed screen and the script module to your iX project:



1. Unpack the enclosed example ZIP-file to a temporary folder.
2. Start iX Developer and load your project.
3. In the Project Explorer, right-click in the lower left corner (1. in the picture)
4. In the list, select Import… (2. in the picture)
5. Navigate to the temporary folder, where you unpacked the ZIP-file and select SCM_MQTT_Client.neo, click [Open].
6. Navigate to the temporary folder, where you unpacked the ZIP-file and select SCM_JSON.neo, click [Open].
7. Select Screen1.xaml, click [Open].
8. Done!

About Beijer Electronics

Beijer Electronics is a multinational, cross-industry innovator that connects people and technologies to optimize processes for business-critical applications. Our offer includes operator communication, automation solutions, digitalization, display solutions and support. As experts in user-friendly software, hardware and services for the Industrial Internet of Things, we empower you to meet your challenges through leading-edge solutions. Beijer Electronics is a Beijer Group company. Beijer Group has a sale over 1.6 billion SEK in 2019 and is listed on the NASDAQ OMX Nordic Stockholm Small Cap list under the ticker BELE. **www.beijergroup.com**

Contact us
Global offices and distributors

Documents / Resources

	**Beijer ELECTRONICS MQTT Client JSON - iX Developer Script Module** [pdf] User Guide MQTT Client JSON iX Developer Script Module, JSON iX Developer Script Module, iX Developer Script Module, Developer Script Module, Script Module, Module

References

- Download | Eclipse Mosquitto