Azure IoT Hub Connector

Device compatible with the MQTT protocol

- Name: MQTTAzure
- Version: 1.0.0.7
- Protocol: MQTT
- Interface: TCP/IP
- Runtime: .NET 2.0 (Multiplatform)
- Configuration:
    - Devices / Protocols

Azure IoT Hub
Channels Configuration
Nodes Configuration
Points Configuration
Testing the Driver Execution
Troubleshoot
Driver Revision History

Overview

Communication Driver Information	
Driver name	MQTTAzure
Assembly Name	T.ProtocolDriver.MQTTAzure
Assembly Version	1.0.0.7
Available for Linux	True
Devices supported	Any device compatible with the MQTT protocol
Manufacturer	Microsoft
Protocol	MQTT (Message Queuing Telemetry Transport) is a machine-to-machine (M2M)/"Internet of Things" connectivity protocol.
Interface	TCP/IP
PC Requirements	Ethernet port
Implemented Methods	Connect, Disconnect, Subscribe, UnSubscribe and Publish.

Azure IoT Hub

Overview

Azure IoT Hub is a managed cloud service that acts as a central message hub for bi-directional communication between IoT applications and their connected devices. It facilitates secure, reliable connectivity for millions of devices, supporting messaging patterns such as device-to-cloud, cloud-to-device, and file upload from devices.

The software platform integrates with Azure IoT Hub and other Azure services for enhanced IoT solutions, enabling real-time analytics, machine learning, business process automation, and event-driven responses.

For more information on Azure IoT topics, see **IoT concepts and Azure IoT Hub**.

System Requirements

The requirements for a successful configuration of MQTT and Azure are listed below:

- MQTT Driver version 0.2.1
- Microsoft Azure IoT Hub Account
- Visual Studio Code with an Azure IoT Hub

Azure Subscription and Tokens

Configuration Settings, using Visual Studio Code

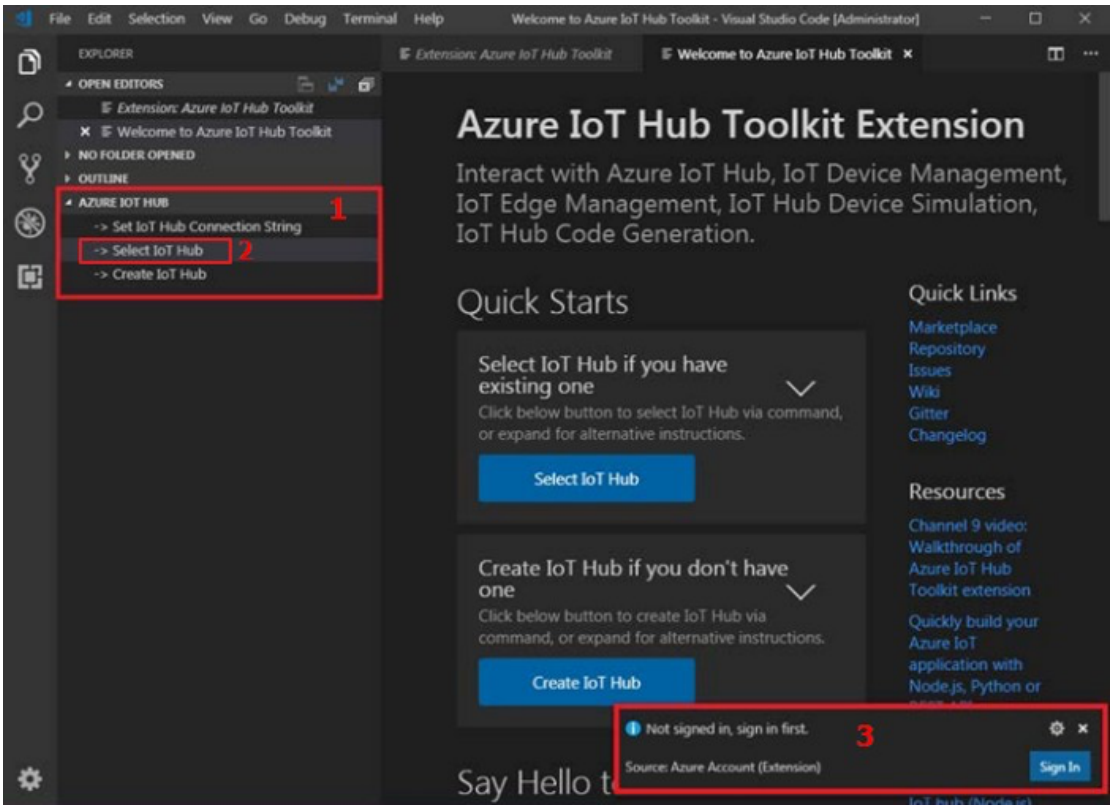
Visual Studio Code is an open-source, streamlined code editor with support for development operations like debugging, task running, and version control. Visual Studio Code can be downloaded here.

1. To download the extension that allows interactions to Visual Studio, Azure IoT Hub, and IoT Device Management, click **here**.

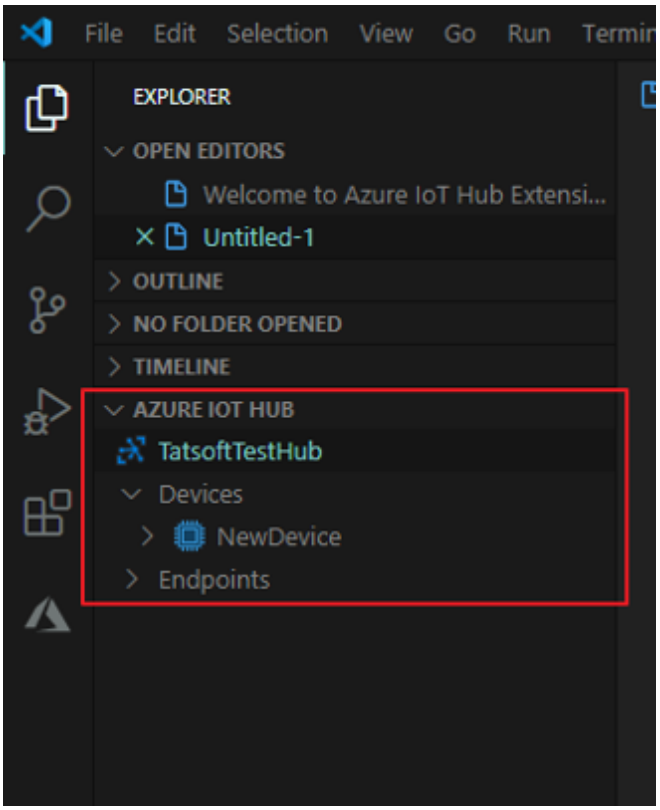
> ⚠ This guide assumes that an IoT Hub already exists within the Azure Portal account.

2. After installing the extension, open the VSCode application. In the explorer pane of the VS Code, click the ”**Azure IoT Hub**” tab in the bottom left corner (1), and click ”**Select IoT Hub**” (2) in the context menu.

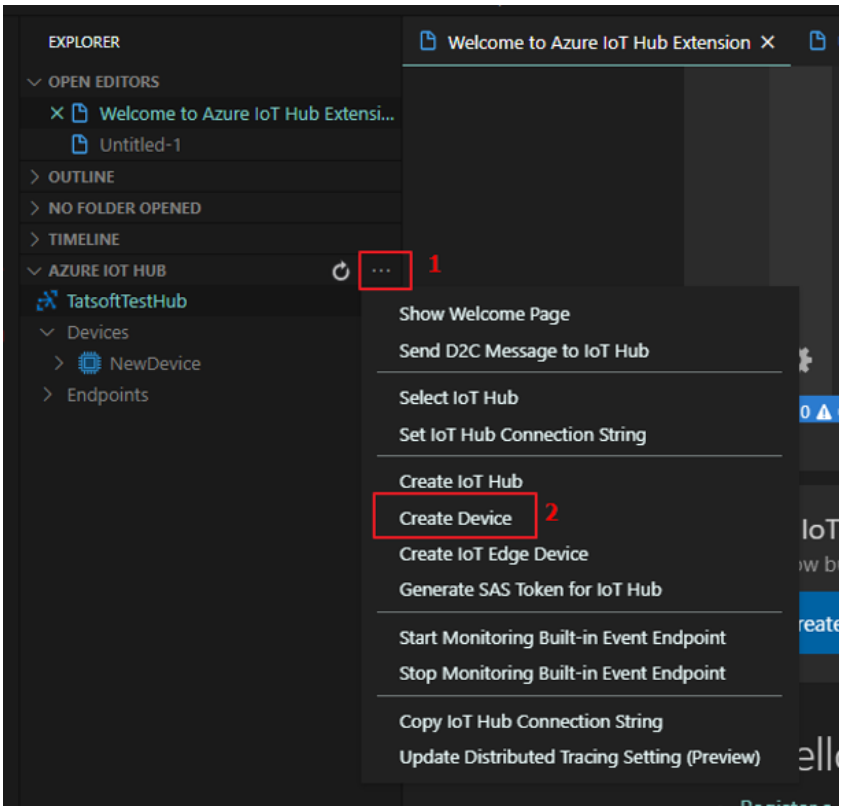
If you are not signed into Azure, a pop-up will appear in the bottom right corner to prompt you sign in to Azure (3).



3. Your Azure Subscription list will appear after you sign in. Select the Azure Subscription and IoT Hub. After a few seconds, devices and endpoints will appear in the Azure IoT Hub tab.

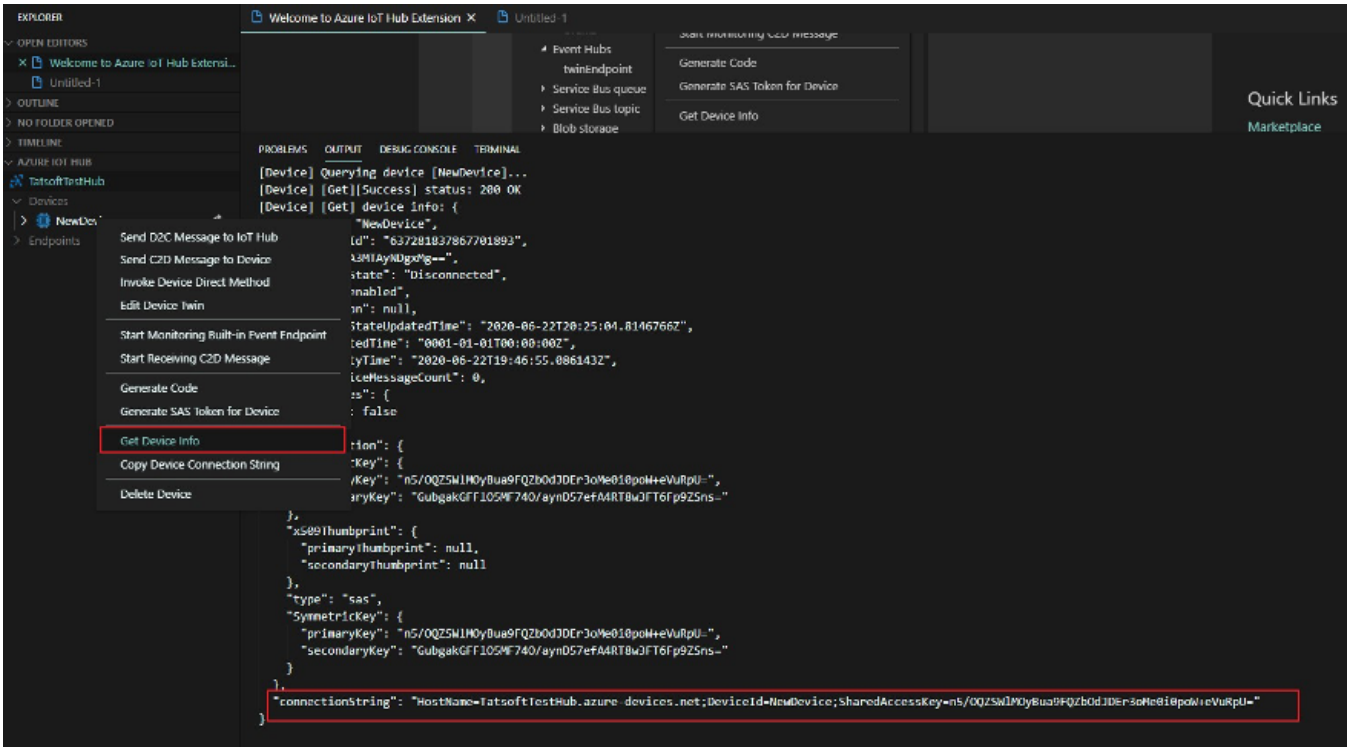


4. New IoT Devices can be created through the Azure IoT Hub extension. Select the context menu (1), click **Create Device** (2), and enter a Device ID for the new IoT device.



5. Now that your device is created, you need to setup your Connection String and SAS Token.

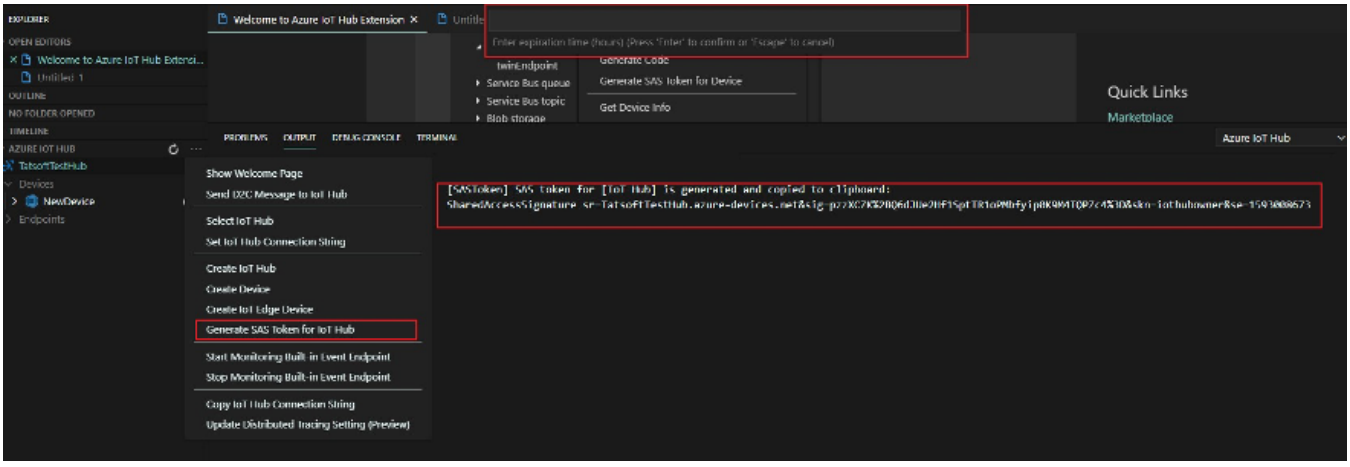
Right-click on your device and select **Get Device Info**. You should see information displayed in the output window.



6. Review the Connection String that was created for the device and record the following pieces of information from this string:

```
HostName = TatsoftTestHub.azure-devices.net DeviceId = NewDevice
```

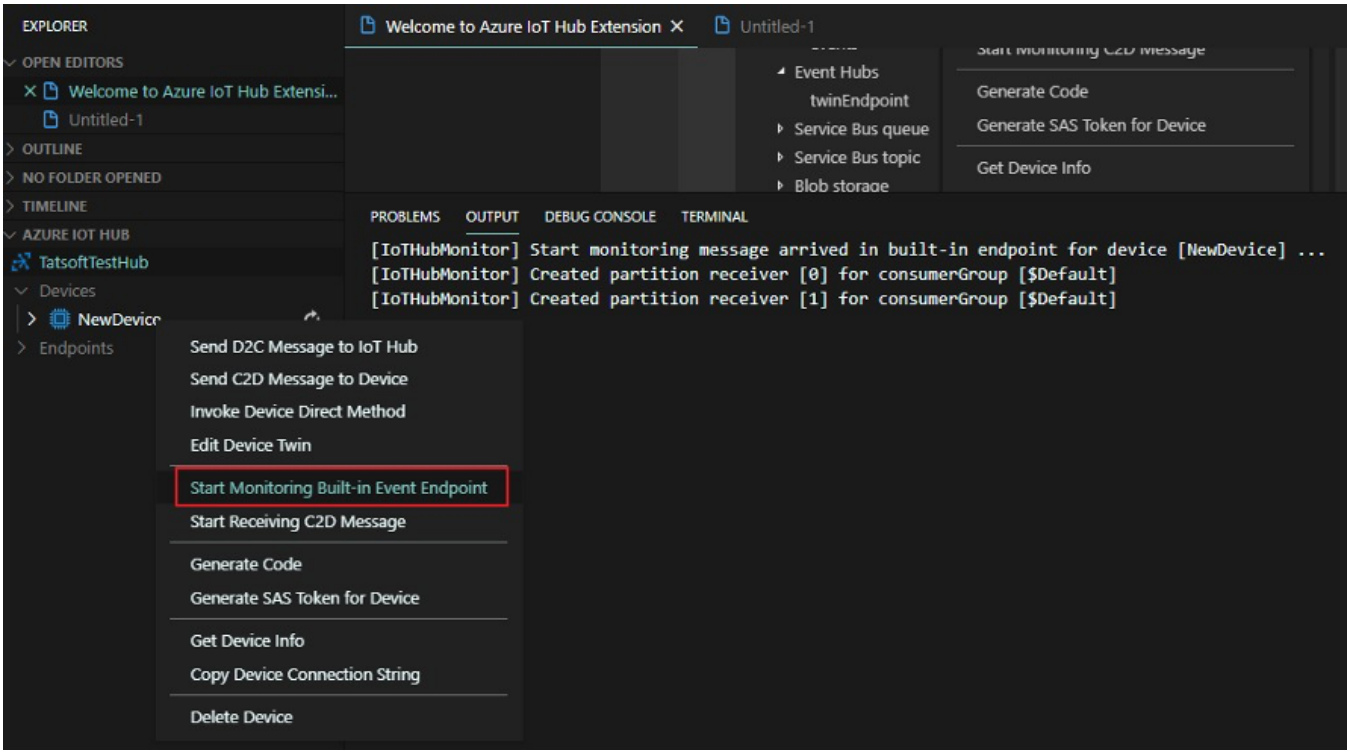
7. Right-click on the device, select **Generate SAS Token for Iot Hub**, and enter the expiration time. You should see the information displayed in the output window.



8. Record the following pieces of information from this string:

```
Password = SharedAccessSignature sr=TatsoftTestHub.azure-devices.net&sig={signature}
&skn=iothubowner&se=1593008673
```

9. To verify that data is flowing from the MQTT Driver to the cloud-based Azure IoT Hub, right-click on the device and select **Start Monitoring Built-In Event Endpoint**.



Channels Configuration

Protocol Options

- `DrvParams[0]` = **ClientType** (string): MQTT client type — `"ApplicationNode"` or `"EoNNode"` (Edge of Network Node). Determines SparkplugB topic structure. Default: `"EoNNode"`.
- `DrvParams[1]` = **TagProperties** (string): Comma-separated list of tag properties to include in MQTT messages (e.g., `"Quality, Description"`). Default: `"Quality"`.
- `DrvParams[2]` = **TimePublishRate** (integer, milliseconds): Minimum publish rate. Minimum: 50ms. Default: 500ms.

Nodes Configuration

Station Configuration

Station syntax: ***<URL> ; <Port> ; <ClientID> ; <Username> ; <Password> ; <SslProtocol> ; [X509Certificate] ; <PayloadFormat>***

Where:

- ***<URL>**:* MQTT Broker (Server) name. It must be the same name configured in host Name parameter in Azure IoT Hub.
- ***<Port>**:* MQTT Broker port. It must be the same *port* configured in the Broker to be listening. Default value is **8883**.
- **<DeviceID>**: Device Id configured in Azure IoT Hub.
- ***<Username>**:* Username defined on MQTT Broker. Is a concatenation of Host Name and Device ID. Syntax: ***<URL>/<DeviceID>***.
- ***<Password>**:* Password defined on MQTT Broker. It is the SAS Token defined for the Device ID.
- ***<SslProtocols>**:* IoT Hub uses Transport Layer Security (TLS) to secure connections from IoT devices and services. Three versions of the TLS protocol are currently supported, namely versions 1.0, 1.1, and 1.2. TLS 1.0 and 1.1 are considered legacy and are planned for deprecation. For more information, see Deprecating TLS 1.0 and 1.1 in IoT Hub by Microsoft. To avoid future issues, use TLS 1.2 as the only TLS version when connecting to IoT Hub.
- ***[X509Certificate]***: Optional. Path of X509 Certificate using TLS v1.0. This must be the complete path of X509 certificate in the client computer. The certificate must be installed in the computer.
- ***<PayloadFormat>***: Format of the message sent to the Broker. By default, the payload is set to follow JSON/SparkplugB format.

Long authentication tokens (over 32 characters)

Azure IoT Hub SAS tokens are long — typically 150 characters or more. When an authentication token is longer than 32 characters, store it with **Security / Secrets** and reference the secret in the **Password** field instead of typing the token directly into the station configuration. This keeps long tokens managed and stored securely.

```
URL = <Host Name> Port = <Port Number>
Client ID = <Device Id>
Username = <Host Name>/<Device Id> Password = <SAS Token>
TLS Version = <TLSv1.0>
X509 Certificate = <Certificate thumbprint>  (optional)  QoS = <Quality of Service>
Keep Alive = <Message sent to Broker to prevent the link from being broken> (in seconds)
```

For the IoT Hub and the device configured in VSCode, the parameters follows:

```
URL = TatsoftTestHub.azure-devices.net Port = 8883
Client ID = NewDevice
Username = TatsoftTestHub.azure-devices.net/NewDevice
Password = SharedAccessSignature sr=TatsoftTestHub.azure-devices.net&sig={signature}
&skn=iothubowner&se=1593008673

TLS Version = <TLSv1.0>X509 Certificate =
QoS = AtMostOnce Keep Alive = 1
```

NOTE

*One of the easiest ways to install the certificate on client computer is to use the wizard to import the certificates is through the "**Internet Options**" to import the certificate in the "**Trusted Root Certification Authorities**". You need to import the certificate in DER format.*

Station Examples

URL = CustomHub-MQTT.azure-devices.net

Port = 8883

Client ID = MQTT_Device

Username = CustomHub-MQTT.azure-devices.net/ MQTT_Device

Password = SharedAccessSignature sr=CustomHub-MQTT.azure devices.net&sig={signature}&skn=iothubowner&se=1612 031715

Ssl Protocol = <TLS>

X059 Certificate =

Payload Format = JSON/SparkplugB

Points Configuration

The MQTT for Azure protocol supports Write Commands only.

Address

The Address syntax is: **[Payload] : <Group> ; <Node> ; <Device>**

Where:

- **[Payload] =** A fixed parameter that assists the user in understanding that the address configuration (Group ID, Edge Node ID and Device ID) are the user-defined parameters in the Payload structure (similar to MQTT + SparkplugB protocol).
- <**Group**>: Provides a logical grouping of Edge Nodes.
- <**Node**>: Identifies ID of Edge Node.

- <**Device**>: Identifies ID of Device from Edge Node. This field can be empty when accessing the main Egde Node.

E.g.: **Payload:GroupID;NodeID;DeviceID**

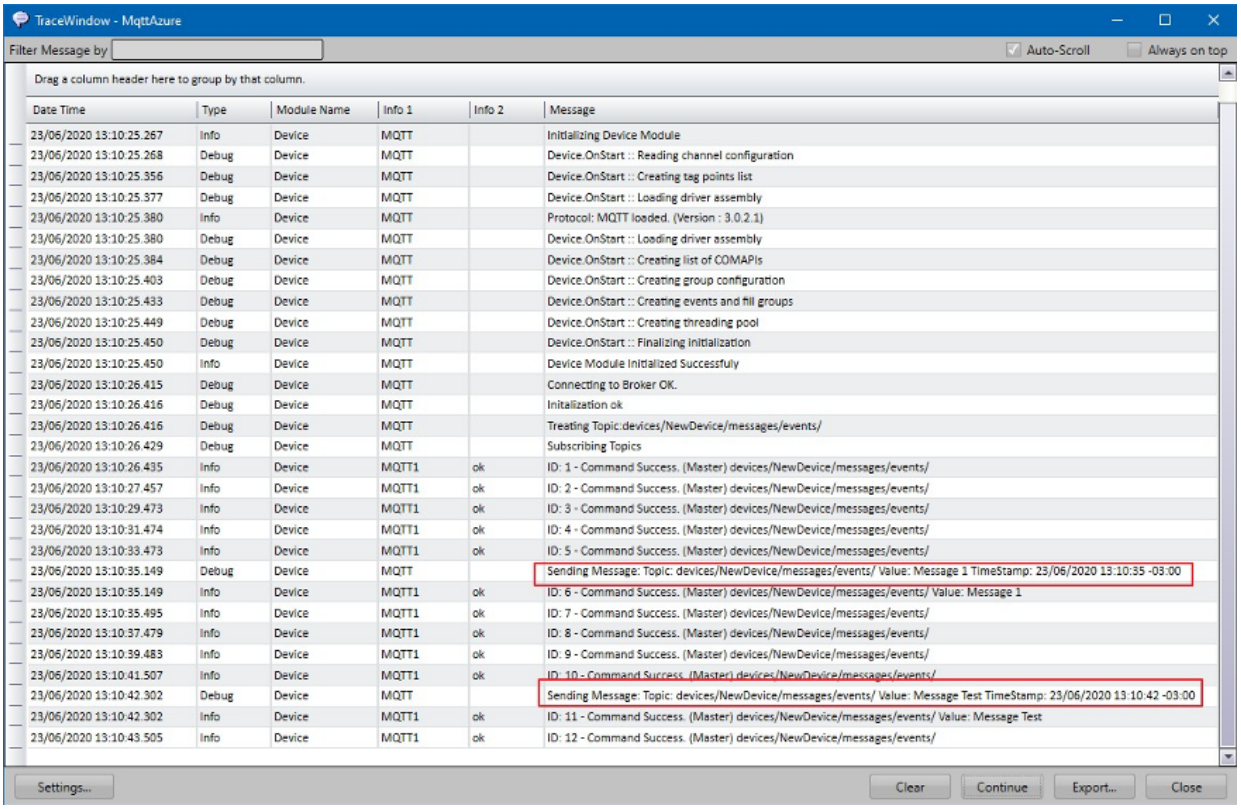
Payload:AzureGroup;Factory1;Motor

For this example device, the correct topic is:

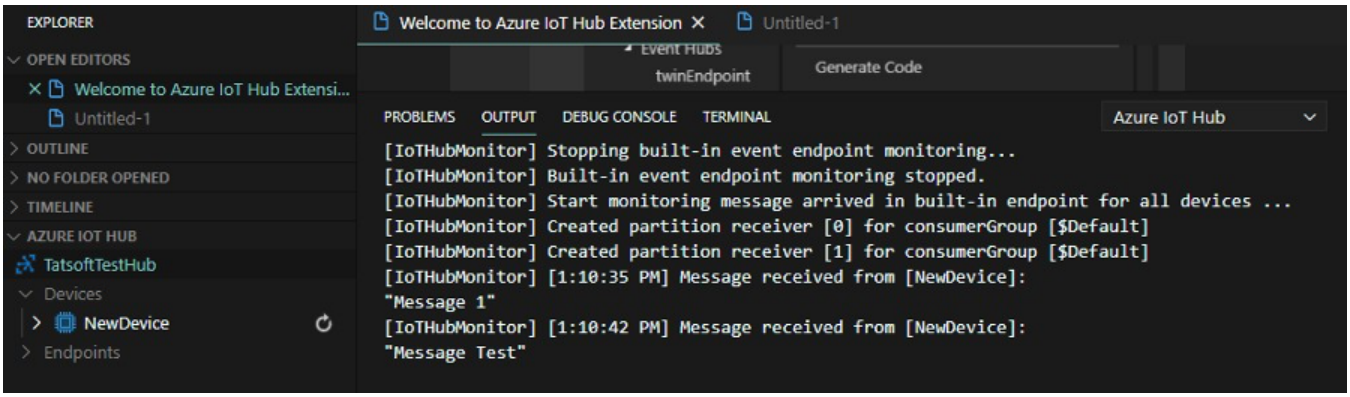
devices/NewDevice/messages/events/

Testing the Driver Execution

Go to **Runtime Startup** and execute the solution. Enable the Debug and Devices options by clicking on settings and selecting the corresponding boxes. Open the TraceWindow and monitor the communication exchange.



Back on VSCode, you should see the received messages in the output window.



Troubleshoot

This section details some errors you might see in your connection to Azure IoT Hub via MQTT protocol and its possible causes.

Error: Exception connecting to the broker

If you see this message on the Trace Window Logs (with modules Debug and Devices enabled), check the port number parameter on **Devices / Nodes / PrimaryStation**.

Device.Node.Status = -1

If you get this error code on `@Devices.Node.<NodeName>.Status`, check if the URL defined on **Devices / Nodes / PrimaryStation** is correct.

Error Connecting to Broker. Status: 5

If you see this message on the Trace Window Logs (with modules Debug and Devices enabled) check the following parameters on **Devices / Nodes / PrimaryStation**:

- **DeviceID**
- **Username**

MQTT driver connection lost: System.EventArgs

If you see this message on the Trace Window Logs (with modules **Debug** and **Devices** enabled), check the **QoS** setting on **Devices / Nodes / PrimaryStation**. As stated in a previous section, IoT Hub will automatically disconnect your client if the QoS value is **2** (Exactly Once).

Another reason is that someone else tried to connect to your IoT Device using the same DeviceID. IoT Hub only supports one active MQTT connection per device. Any new MQTT connection on behalf of the same DeviceID causes IoT Hub to drop the existing connection.

Driver Revision History

MQTTAzure Revision History	
Version	**Notes**
1.0.0.7	New Store and Forward option buffers data locally while Azure IoT Hub is unreachable and forwards it on reconnect; the station password field is now masked.
1.0.0.3	Initial release on new documentation standards.

In this section...
