Connect to AWS Cloud Through MQTT with the MGate 5105 Industrial Protocol Gateway

Moxa Technical Support Team
support@moxa.com

Contents

Released on March 30, 2019

About Moxa

Moxa is a leading provider of edge connectivity, industrial networking, and network infrastructure solutions for enabling connectivity for the Industrial Internet of Things. With over 30 years of industry experience, Moxa has connected more than 50 million devices worldwide and has a distribution and service network that reaches customers in more than 70 countries. Moxa delivers lasting business value by empowering industry with reliable networks and sincere service for industrial communications infrastructures. Information about Moxa's solutions is available at www.moxa.com.

How to Contact Moxa

Tel: +886-2-8919-1230
Fax: +886-2-8919-1231

MOXA®
Reliable Networks ▲ Sincere Service

1. Introduction

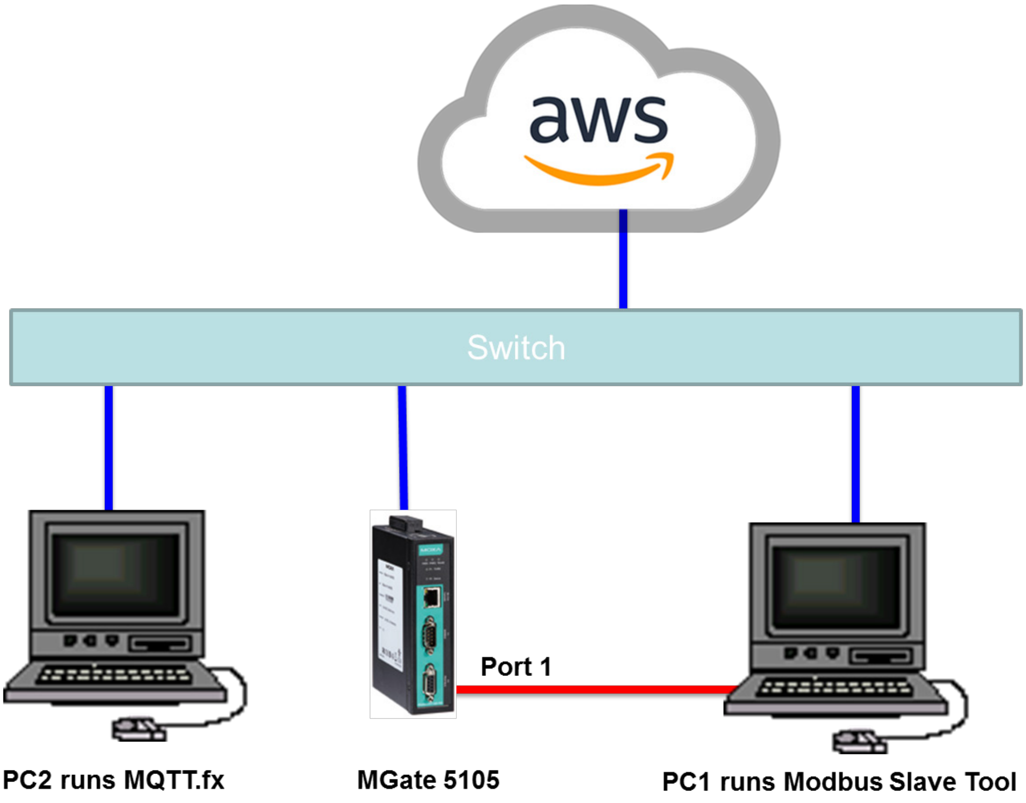
The MGate 5105 performs easy protocol conversions between Modbus RTU/ASCII, Modbus TCP, and EtherNet/IP protocols. From Firmware Versions 4.0 upwards support publishing the time stamps of the fieldbus devices to cloud servers. The cloud server include Microsoft Azure, Alibaba Cloud, or MQTT Broker.

This document demonstrates how to connect the MGate 5105 to AWS IoT via Generic MQTT mode. We also demonstrate how to publish fieldbus data messages and subscribe to message from AWS IoT.

2. System Topology

Figure 1 illustrates the system topology. PC1 runs Modbus Slave tool to act as a Modbus RTU device. It connects to MGate 5105's Port 1. The MGate 5105 acts as a MQTT Client device and connects to AWS IoT. PC2 runs MQTT.fx, which is an MQTT Client. We use MQTT.fx to publish messages in AWS IoT and subscribe to MGate 5105's topic in it.

< Figure 1. System Topology >



3. Prerequisites

3.1 Modbus Slave Tool

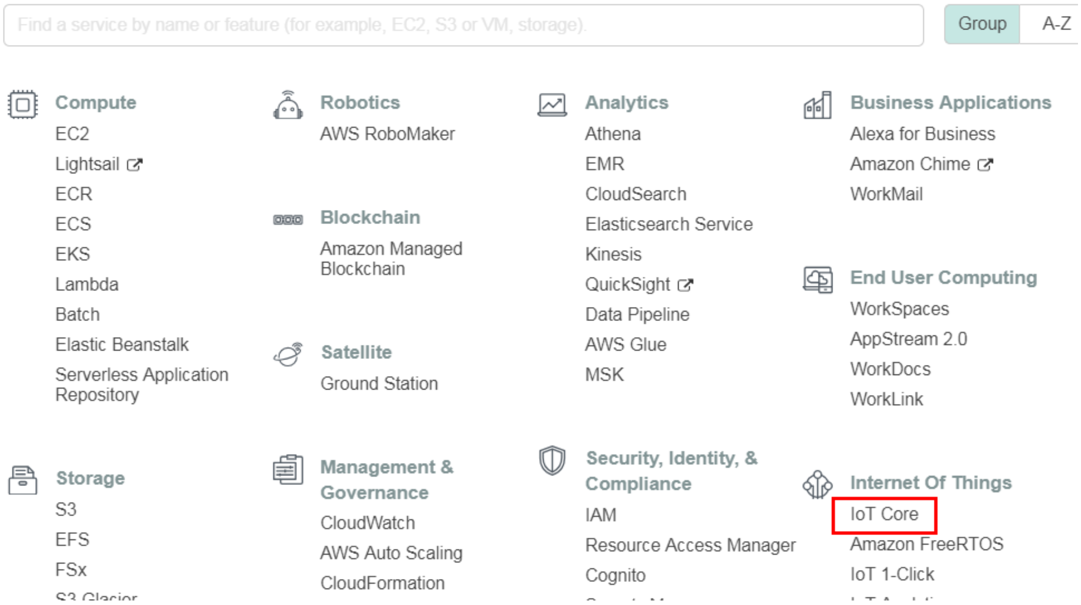
Modbus Slave is a very popular Modbus slave simulator for testing and debugging of your modbus devices, which support Modbus RTU/ASCII and Modbus TCP/IP.

Download from website: http://www.modbustools.com/download.html

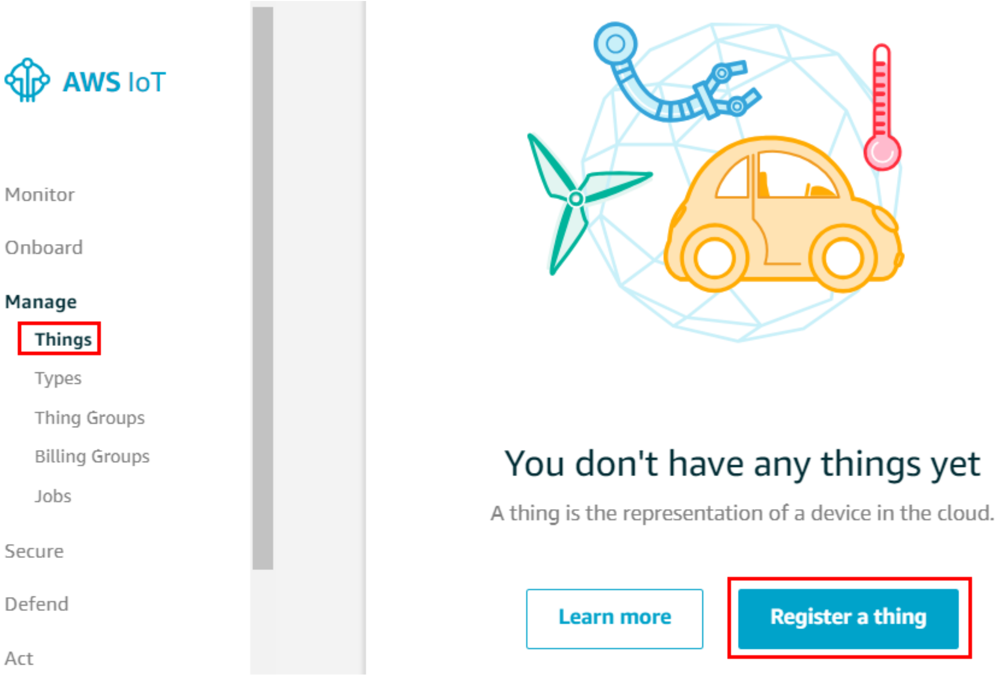
3.2 Create AWS IoT and Thing

1. Use AWS user account to log in to AWS Console.
   Website: https://aws.amazon.com/console/

2. Find the Internet Of Things → IoT Core service:



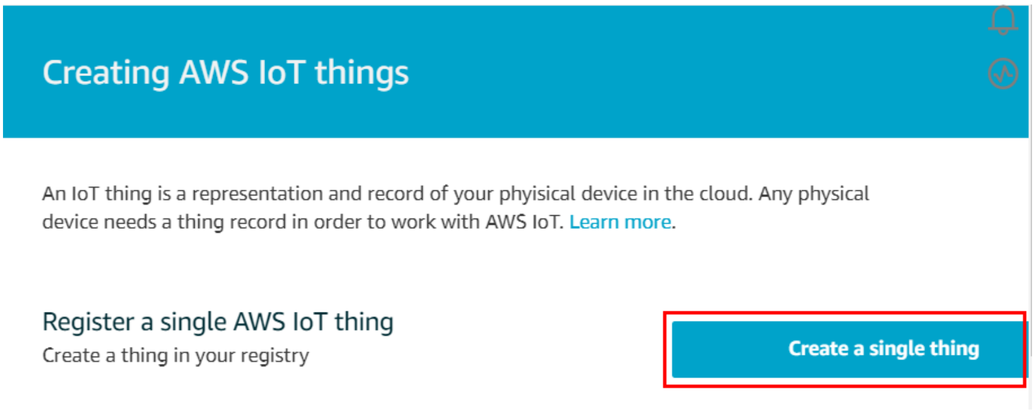
3. Create a Thing:
   a. Register a thing: **Manage** → **Things**.



You don't have any things yet

A thing is the representation of a device in the cloud.

   b. Create a single thing:



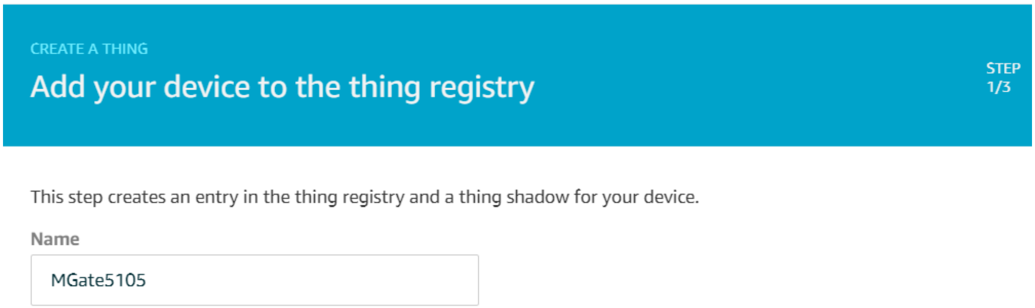
Creating AWS IoT things

An IoT thing is a representation and record of your phyisical device in the cloud. Any physical device needs a thing record in order to work with AWS IoT. Learn more.

Register a single AWS IoT thing

Create a thing in your registry

   c. Give a thing's name:



CREATE A THING

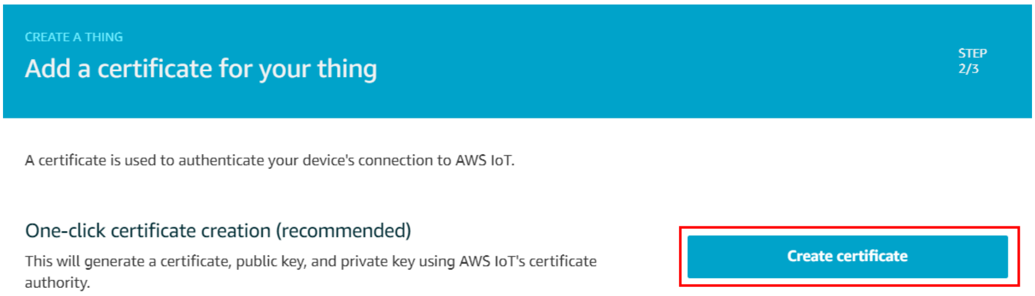
Add your device to the thing registry

STEP 1/3

This step creates an entry in the thing registry and a thing shadow for your device.

Name: MGate5105

d. Execute **Create certificate**:



CREATE A THING

Add a certificate for your thing

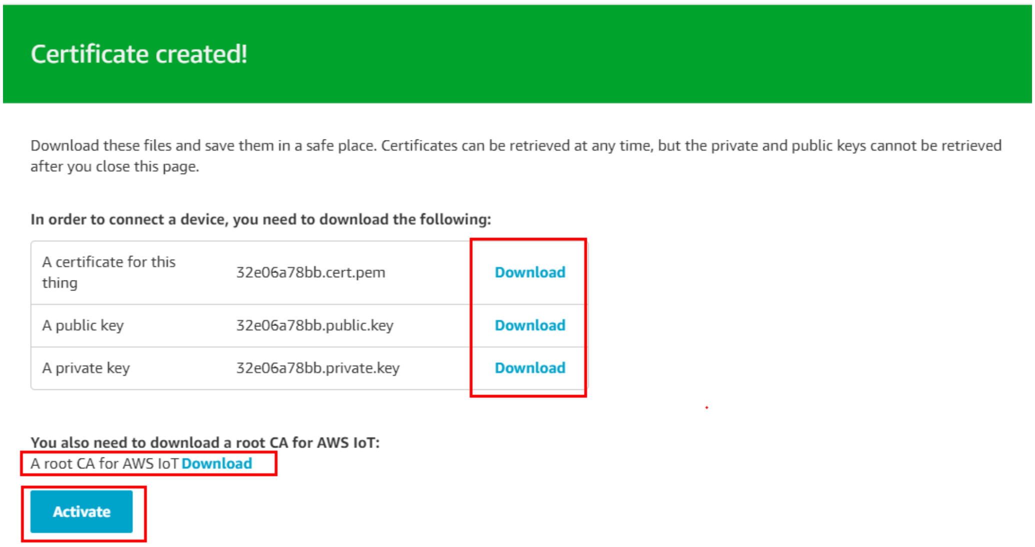
STEP 2/3

A certificate is used to authenticate your device's connection to AWS IoT.

One-click certificate creation (recommended)

This will generate a certificate, public key, and private key using AWS IoT's certificate authority.

e. After creating a certificate, download the thing's certificate, private key, and root CA certificate. Then click **Activate**.



Certificate created!

Download these files and save them in a safe place. Certificates can be retrieved at any time, but the private and public keys cannot be retrieved after you close this page.

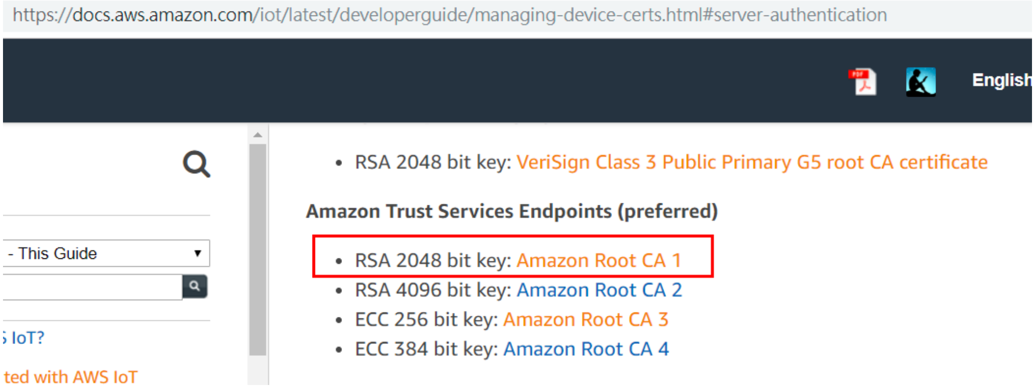
In order to connect a device, you need to download the following:

A certificate for this thing	32e06a78bb.cert.pem	Download
A public key	32e06a78bb.public.key	Download
A private key	32e06a78bb.private.key	Download

You also need to download a root CA for AWS IoT:

A root CA for AWS IoT Download

When you click **Download** of root CA, a new web page pops up as below:



https://docs.aws.amazon.com/iot/latest/developerguide/managing-device-certs.html#server-authentication

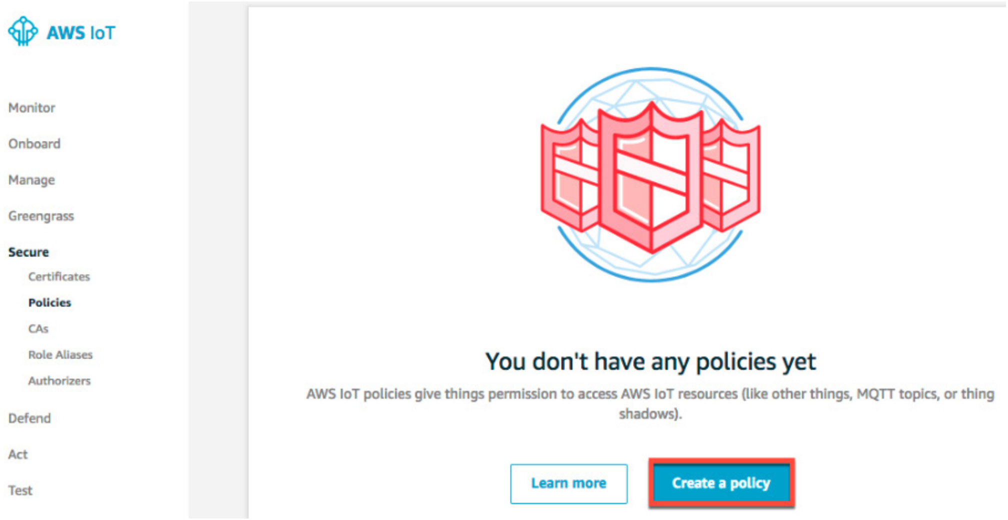
- RSA 2048 bit key: VeriSign Class 3 Public Primary G5 root CA certificate

Amazon Trust Services Endpoints (preferred)

- RSA 2048 bit key: Amazon Root CA 1
- RSA 4096 bit key: Amazon Root CA 2
- ECC 256 bit key: Amazon Root CA 3
- ECC 384 bit key: Amazon Root CA 4

Select **Amazon Root CA 1**. A new web page that shows an Amazon Root CA certificate in PEM format will pop up. Save the content as a *.pem file.

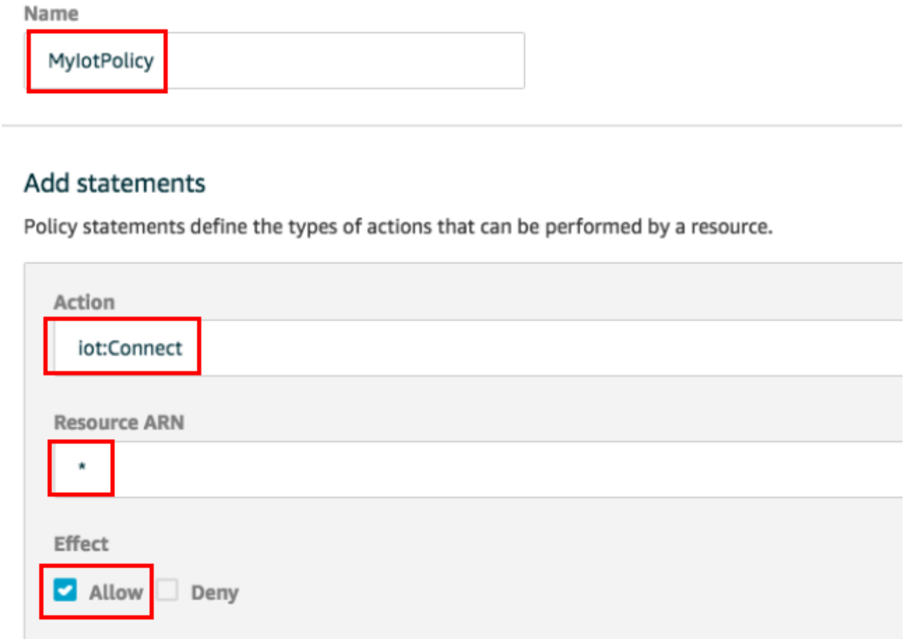
4. Create a Policy
   a. In Secure → Policies, click **Create a policy**:



You don't have any policies yet

AWS IoT policies give things permission to access AWS IoT resources (like other things, MQTT topics, or thing shadows).

   b. In the **Action** field, type **iot:Connect**. In the **Resource ARN** field, type *****. Select the **Allow** check box. This allows all clients to connect to AWS IoT.



Name: MyIotPolicy

Add statements

Policy statements define the types of actions that can be performed by a resource.

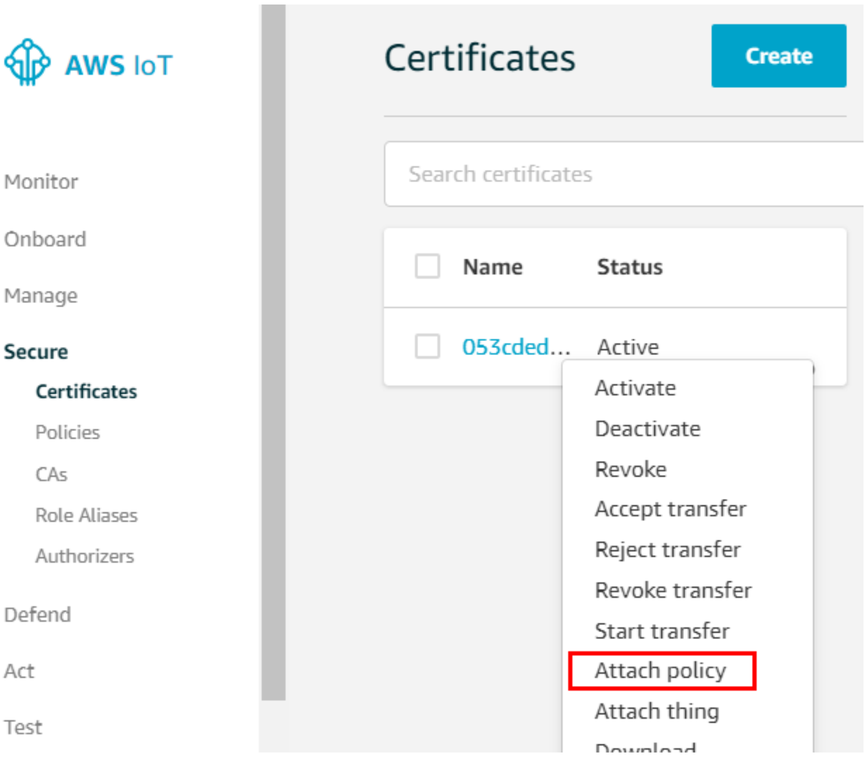
Action: iot:Connect

Resource ARN: *

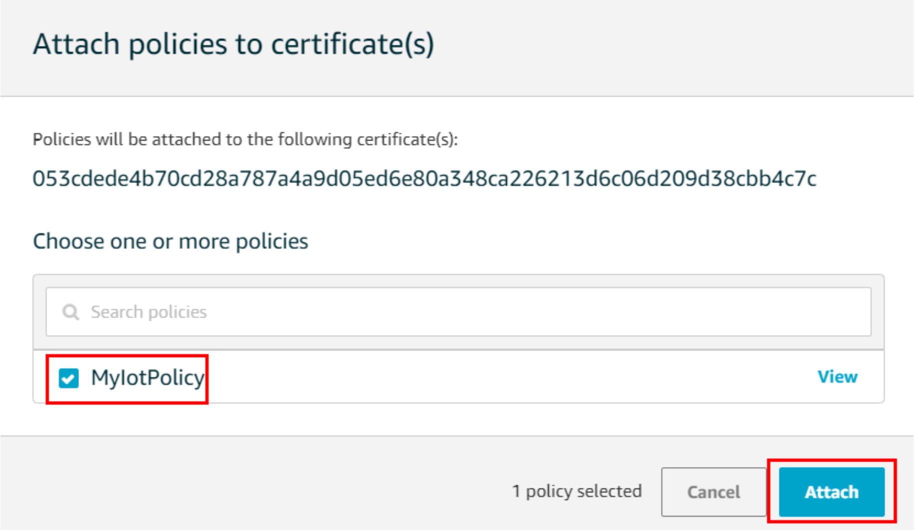
Effect: ☑ Allow □ Deny

After you have entered the information for your policy, choose **Create**.

5. Attach an AWS IoT Policy to a Device Certificate
   a. Choose **Secure → Certificates**. In the box you created for the certificate, choose ... to open a drop-down menu, and then choose **Attach policy**.



   b. In the **Attach policies to certificate(s)** dialog box, select the check box next to the policy you created in the previous step, and then choose **Attach**.


Attach policies to certificate(s)

Policies will be attached to the following certificate(s):

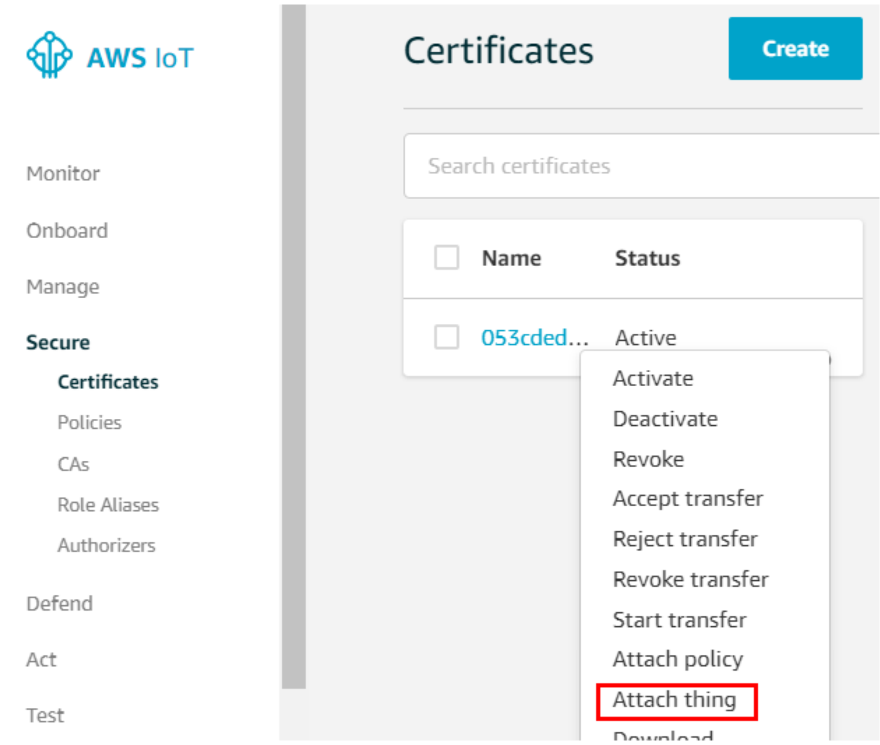
053cdede4b70cd28a787a4a9d05ed6e80a348ca226213d6c06d209d38cbb4c7c

Choose one or more policies

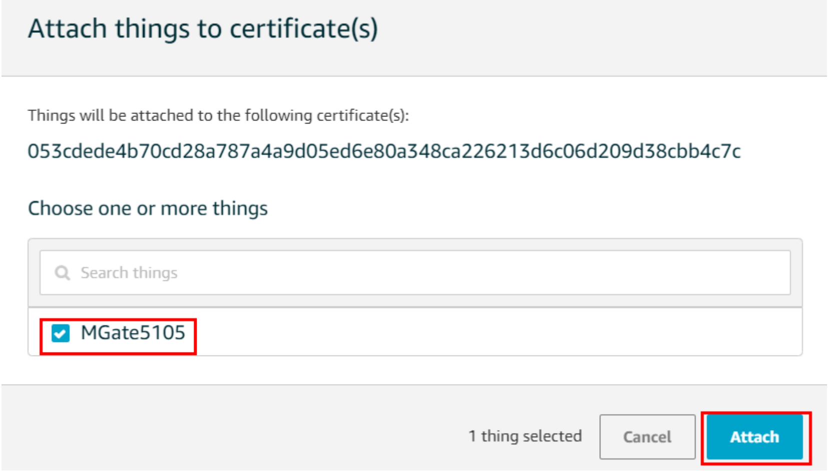
6. Attach a Certificate to a Thing:

   A device must have a certificate, private key, and root CA certificate to authenticate with AWS IoT.

   a. In the box created for the certificate, choose ... to open a drop-down menu, and then choose **Attach thing**.



   b. In the **Attach things to certificate(s)** dialog box, select the check box next to the thing you registered, and then choose **Attach**.


Attach things to certificate(s)

Things will be attached to the following certificate(s):

053cdede4b70cd28a787a4a9d05ed6e80a348ca226213d6c06d209d38cbb4c7c

Choose one or more things

3.3 MQTT.fx Tool

MQTT.fx is a MQTT Client written in Java, based on Eclipse Paho. You can download thr latest version at the following website:https://mqttfx.jensd.de/index.php/download.

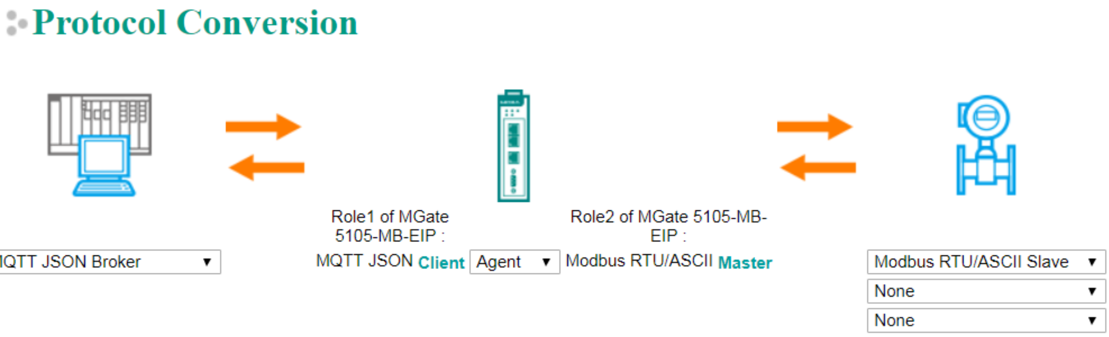
4. MGate 5105 Settings

Log in to MGate 5105's web console, then do the following settings:

4.1 Protocol Conversion

The MGate 5105 supports two kinds of MQTT data message formats: JSON and RAW. In this demonstration, we use the JSON format. In Protocol Conversion Settings, choose **MQTT JSON Client** as Role1. In the fieldbus site, choose the following protocols: Modbus RTU/ASCII Slave, Modbus TCP Server, or EtherNet/IP Adapter. Note that multiple combinations are allowed for settings in Role2. In this demonstration, we choose Modbus RTU/ASCII Slave.

Set as below:



Protocol Conversion

MQTT JSON Broker ▼

Role1 of MGate 5105-MB-EIP : MQTT JSON Client Agent ▼

Role2 of MGate 5105-MB-EIP : Modbus RTU/ASCII Master

Modbus RTU/ASCII Slave ▼

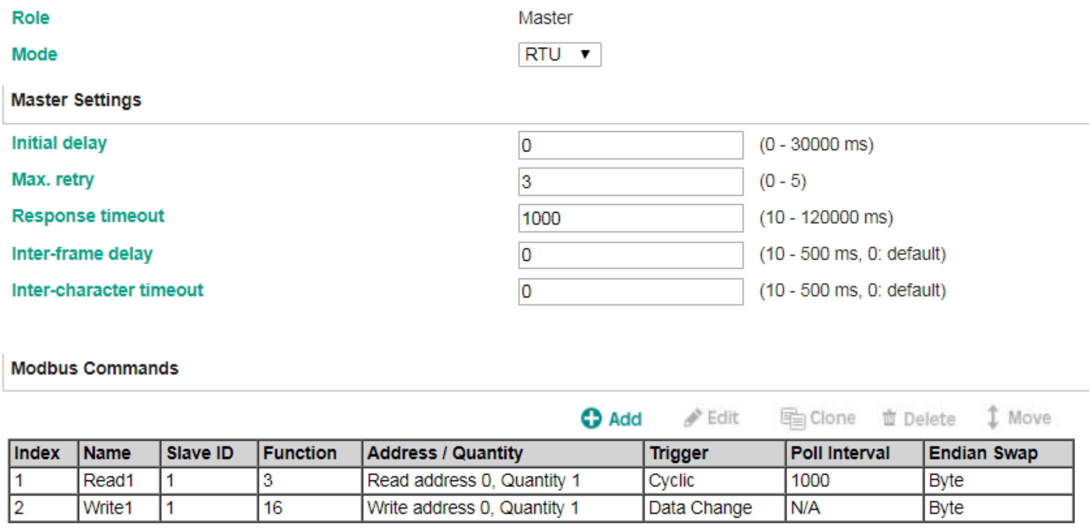
None ▼

None ▼

4.2 Modbus RTU Master Settings

1. In the **Modbus RTU/ASCII Master** Settings web page, we choose **RTU** for Mode and keep **Master Settings** as the default setting.
2. Add a **Read1** Modbus command to send a function code 03 and a command for quantity as 1, and Endian Swap as Byte. Poll interval is 1000 ms.
3. Add a **Write1** Modbus command to send a function code 06 command, and Endian Swap as Byte. Its **Trigger** command is **Data Change**.

Set as below:



Role: Master

Mode: RTU ▼

Master Settings

Initial delay	0	(0 - 30000 ms)
Max. retry	3	(0 - 5)
Response timeout	1000	(10 - 120000 ms)
Inter-frame delay	0	(10 - 500 ms, 0: default)
Inter-character timeout	0	(10 - 500 ms, 0: default)

Modbus Commands

Index	Name	Slave ID	Function	Address / Quantity	Trigger	Poll Interval	Endian Swap
1	Read1	1	3	Read address 0, Quantity 1	Cyclic	1000	Byte
2	Write1	1	16	Write address 0, Quantity 1	Data Change	N/A	Byte

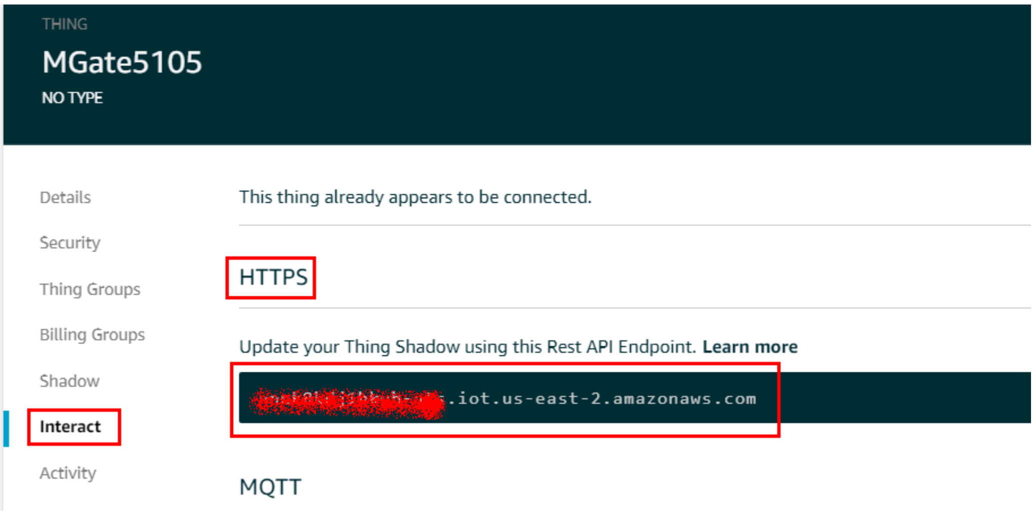
4.3 MQTT JSON Client Settings

1. Basic Settings:

   In **Basic Settings → Remote MQTT broker** string, fill in your MQTT Broker IP address or hostname and broker's listen port.

   Find the broker address for your device (thing) by selecting your device/thing in the AWS IoT console, and then click on **Interact** menu.

   (Things > THING_NAME > Interact)

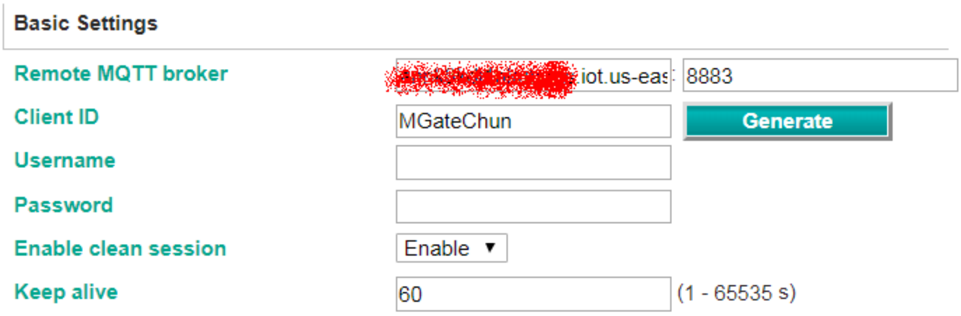


This thing already appears to be connected.

Update your Thing Shadow using this Rest API Endpoint. **Learn more**

```
[redacted].iot.us-east-2.amazonaws.com
```

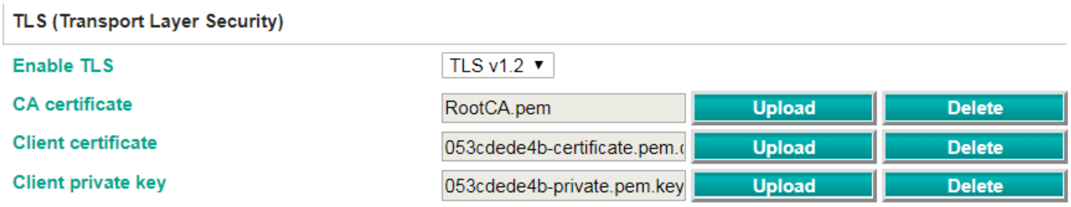
The Rest API endpoint name under HTTPS section is your broker address.
The port number for the secured MQTT connection is “8883”.
Client ID setting is an identity of MQTT session. It must be unique. The broker does not accept the same Client ID connection for a second time. You can fill in an identifiable ID or click the **Generate** button to generate a random ID.
The broker may need the client to provide an username and password to authenticate the client connection. If you need to, fill in the correct username and password.
Set as below:



Basic Settings

Remote MQTT broker	[redacted]iot.us-eas	8883
Client ID	MGateChun	Generate
Username		
Password		
Enable clean session	Enable ▼	
Keep alive	60	(1 - 65535 s)

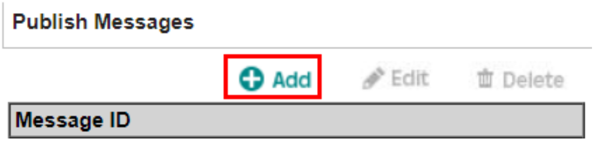
2. TLS (Transport Layer Security) Setting:
The MGate 5105 supports TLS to secure communications between MQTT Broker and Client. Here, we use version 1.2.
To enable a TLS transmission, upload the CA certificate, client certificate, and client keyfile. The certificates and keyfile must be PEM encoded.
Set as below:



TLS (Transport Layer Security)

Enable TLS	TLS v1.2 ▼		
CA certificate	RootCA.pem	Upload	Delete
Client certificate	053cdede4b-certificate.pem.(	Upload	Delete
Client private key	053cdede4b-private.pem.key	Upload	Delete

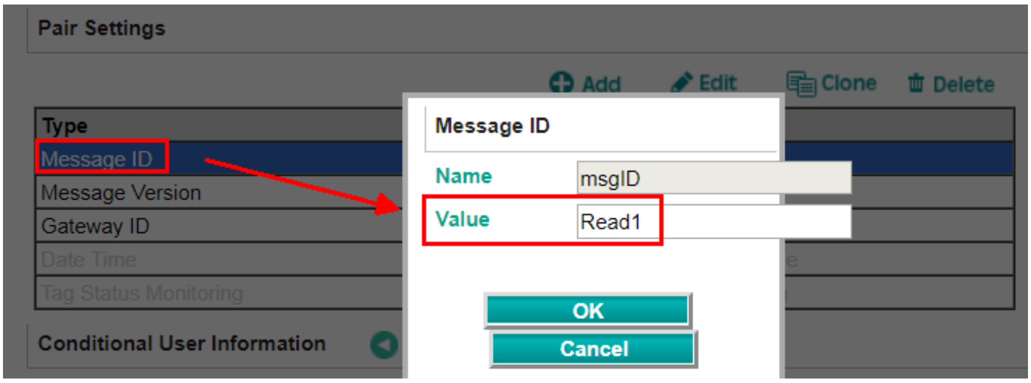
3. Publish Messages:
Click the **Add** button to create a **Publish Message** and click it to edit the message settings.



Publish Messages

Message ID

In **Pair Settings**, click **Message ID** to edit **Name**, and set **Value** as **Read1**.

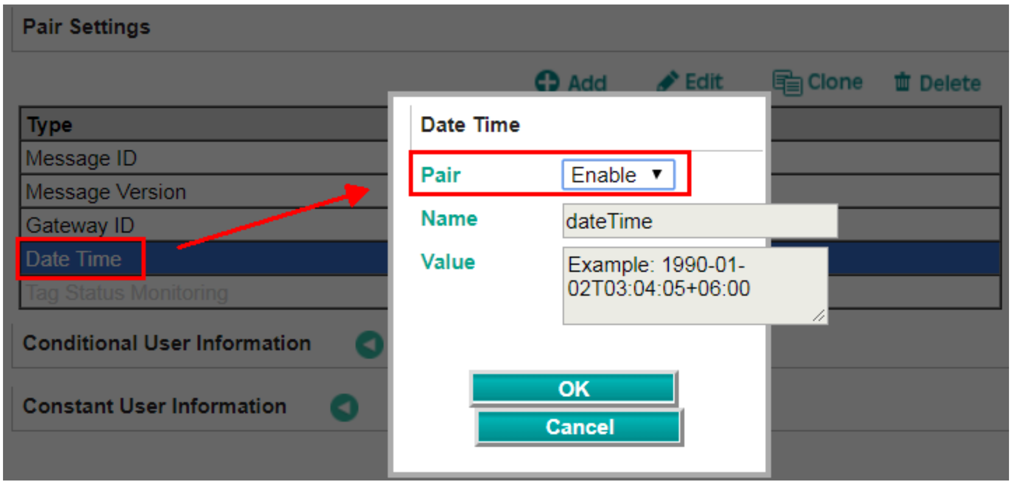


Message ID

Name: msgID

Value: Read1

Click **Date Time** to enable **dateTime** padding in the message.



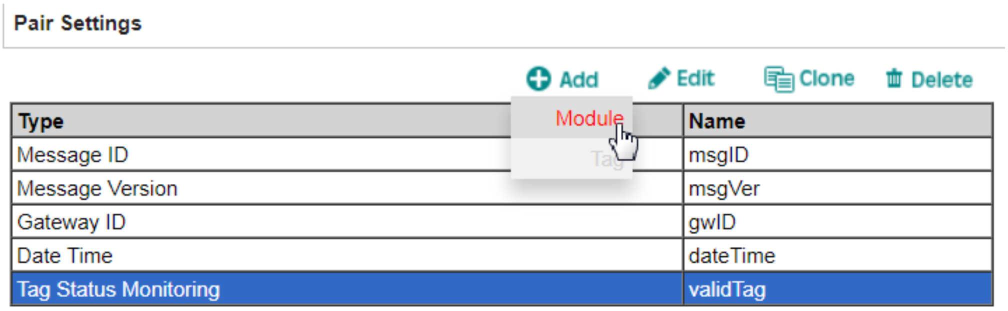
Date Time

Pair: Enable

Name: dateTime

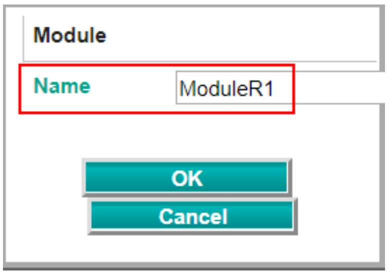
Value: Example: 1990-01-02T03:04:05+06:00

Click **Add** → **Module** to create a new module.



Type	Name
Message ID	msgID
Message Version	msgVer
Gateway ID	gwID
Date Time	dateTime
Tag Status Monitoring	validTag

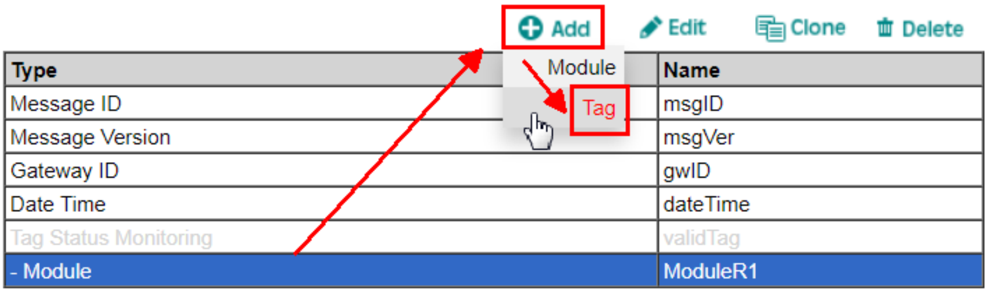
Set **Name** as **ModuleR1**.



Module

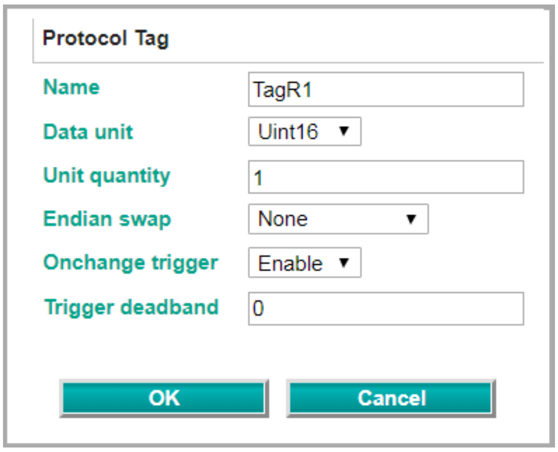
Name: ModuleR1

Choose ModuleR1 and then click **Add** → **Tag**.



Type	Name
Message ID	msgID
Message Version	msgVer
Gateway ID	gwID
Date Time	dateTime
Tag Status Monitoring	validTag
- Module	ModuleR1

Create a Protocol Tag as below:



Protocol Tag

Name: TagR1

Data unit: Uint16

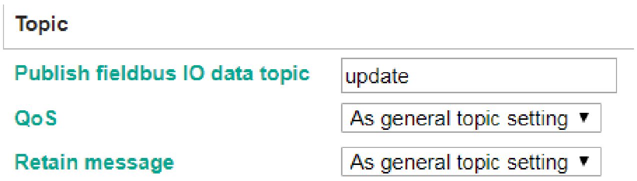
Unit quantity: 1

Endian swap: None

Onchange trigger: Enable

Trigger deadband: 0

We set the topic name of this message as **update**.



Topic

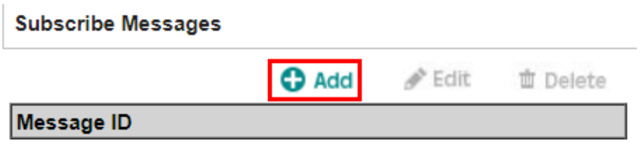
Publish fieldbus IO data topic: update

QoS: As general topic setting

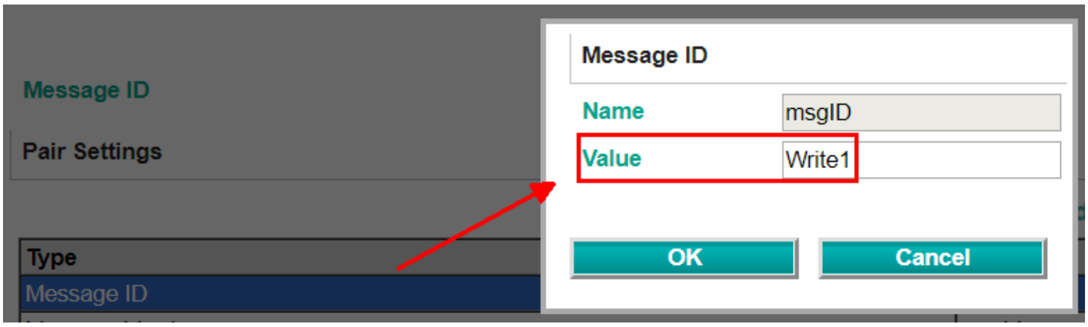
Retain message: As general topic setting

4. Subscribe Messages:

   Click the **Add** button to create a subscribe message and click it to edit message settings.



In **Pair Settings**, click **Message ID** to edit **Name** and set **Value** as **Write1**.

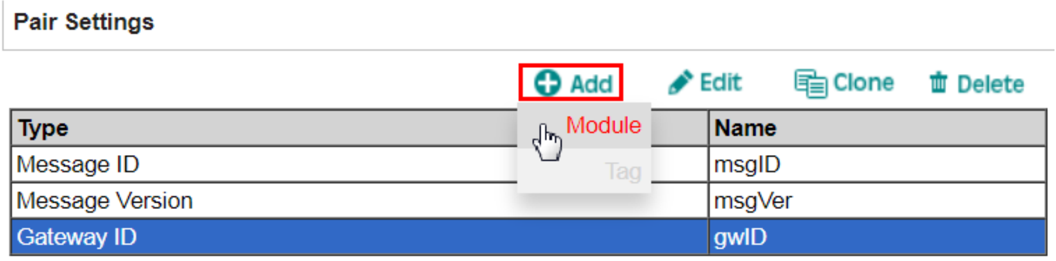


Message ID

Name: msgID

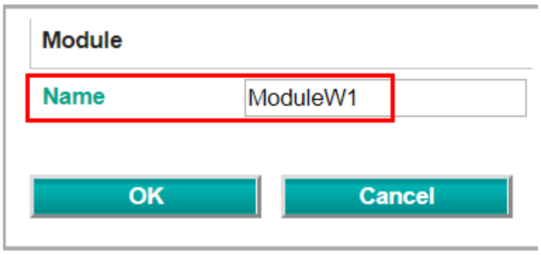
Value: Write1

Click **Add** → **Module** to create a new module.



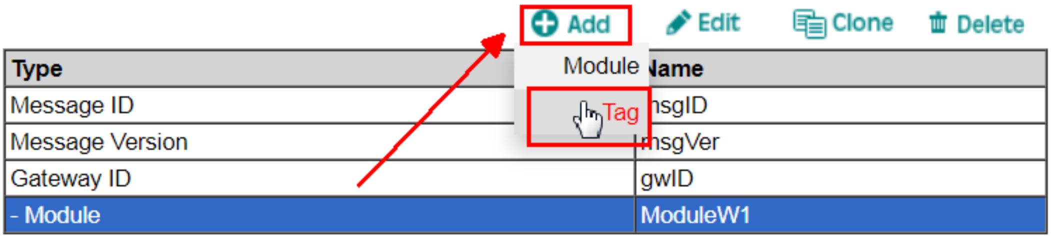
Type	Name
Message ID	msgID
Message Version	msgVer
Gateway ID	gwID

Set **Name** as **ModuleW1**.



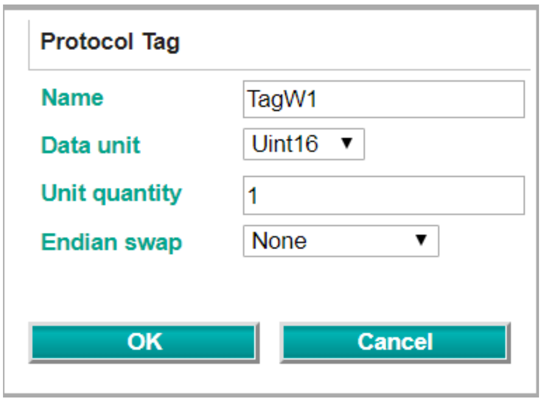
Name: ModuleW1

Choose **ModuleW1**, then click **Add** → **Tag**.



Type	Name
Message ID	msgID
Message Version	msgVer
Gateway ID	gwID
- Module	ModuleW1

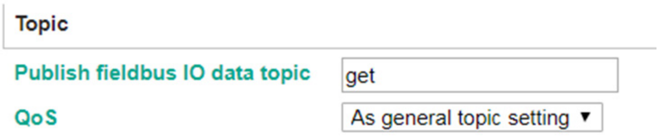
Create a Protocol Tag as below:



Protocol Tag

Name	TagW1
Data unit	Uint16
Unit quantity	1
Endian swap	None

We set the topic name of this message as **get**.

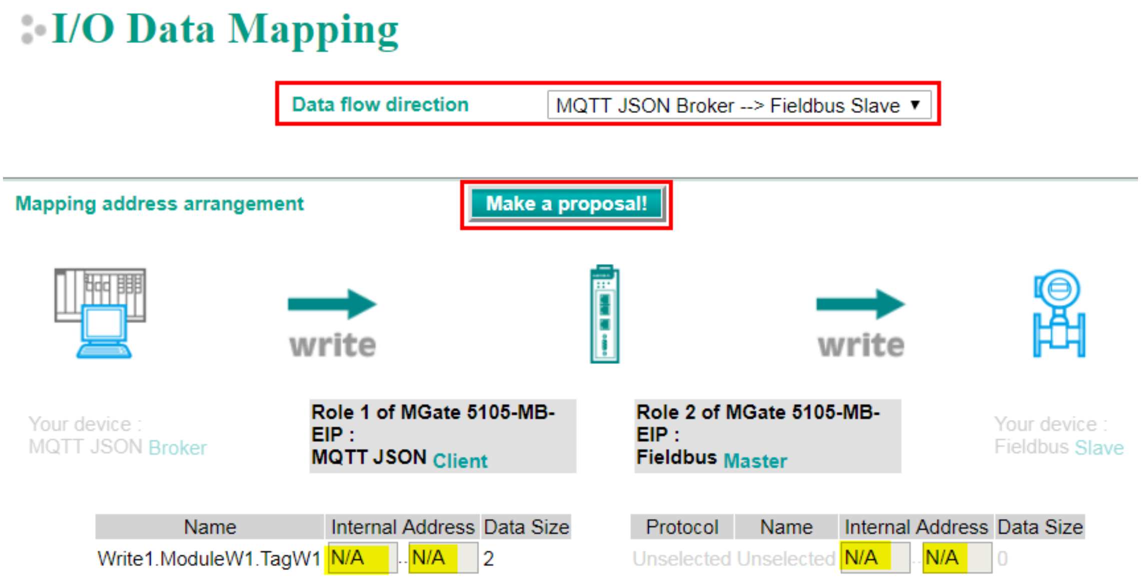


Topic

Publish fieldbus IO data topic	get
QoS	As general topic setting

4.4 I/O Data Mapping

When the protocol settings are done, only one more step of I/O Data mapping for protocol configuration is required. Click the **Make a proposal** button for auto mapping in both **MQTT JSON Broker → Fieldbus Slave** direction and **Fieldbus Slave → MQTT JSON Broke**r direction.



I/O Data Mapping

Data flow direction: MQTT JSON Broker --> Fieldbus Slave ▼

Mapping address arrangement [Make a proposal!]

Your device : MQTT JSON Broker

Role 1 of MGate 5105-MB-EIP : MQTT JSON Client

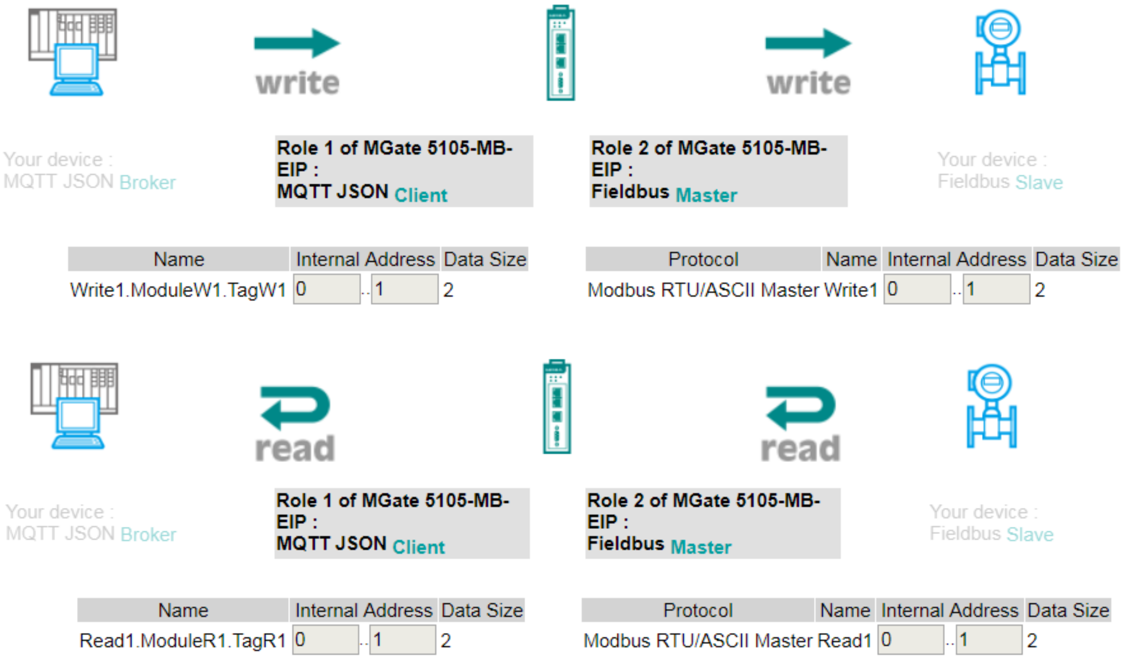
Role 2 of MGate 5105-MB-EIP : Fieldbus Master

Your device : Fieldbus Slave

Name	Internal Address	Data Size
Write1.ModuleW1.TagW1	N/A .. N/A	2

Protocol	Name	Internal Address	Data Size
Unselected	Unselected	N/A .. N/A	0

The mapping result is as below:



Your device : MQTT JSON Broker

Role 1 of MGate 5105-MB-EIP : MQTT JSON Client

Role 2 of MGate 5105-MB-EIP : Fieldbus Master

Your device : Fieldbus Slave

Name	Internal Address	Data Size
Write1.ModuleW1.TagW1	0 .. 1	2

Protocol	Name	Internal Address	Data Size
Modbus RTU/ASCII Master	Write1	0 .. 1	2

Your device : MQTT JSON Broker

Role 1 of MGate 5105-MB-EIP : MQTT JSON Client

Role 2 of MGate 5105-MB-EIP : Fieldbus Master

Your device : Fieldbus Slave

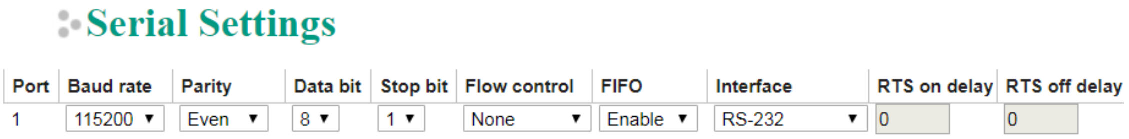
Name	Internal Address	Data Size
Read1.ModuleR1.TagR1	0 .. 1	2

Protocol	Name	Internal Address	Data Size
Modbus RTU/ASCII Master	Read1	0 .. 1	2

4.5 Serial Settings

Serial Port1 connects to Modbus RTU device, so you must set the serial parameters of Port1.

Set as below:

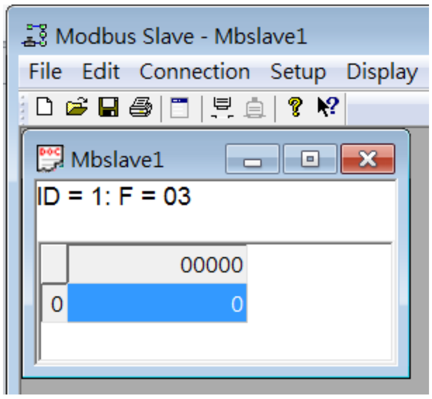


Serial Settings

Port	Baud rate	Parity	Data bit	Stop bit	Flow control	FIFO	Interface	RTS on delay	RTS off delay
1	115200	Even	8	1	None	Enable	RS-232	0	0

5. Modbus Slave Tool Settings

PC1 runs **Modbus Slave tool** and connects to MGate 5105's Serial Port. Add the Modbus definition below:



ID = 1: F = 03

	00000
0	0

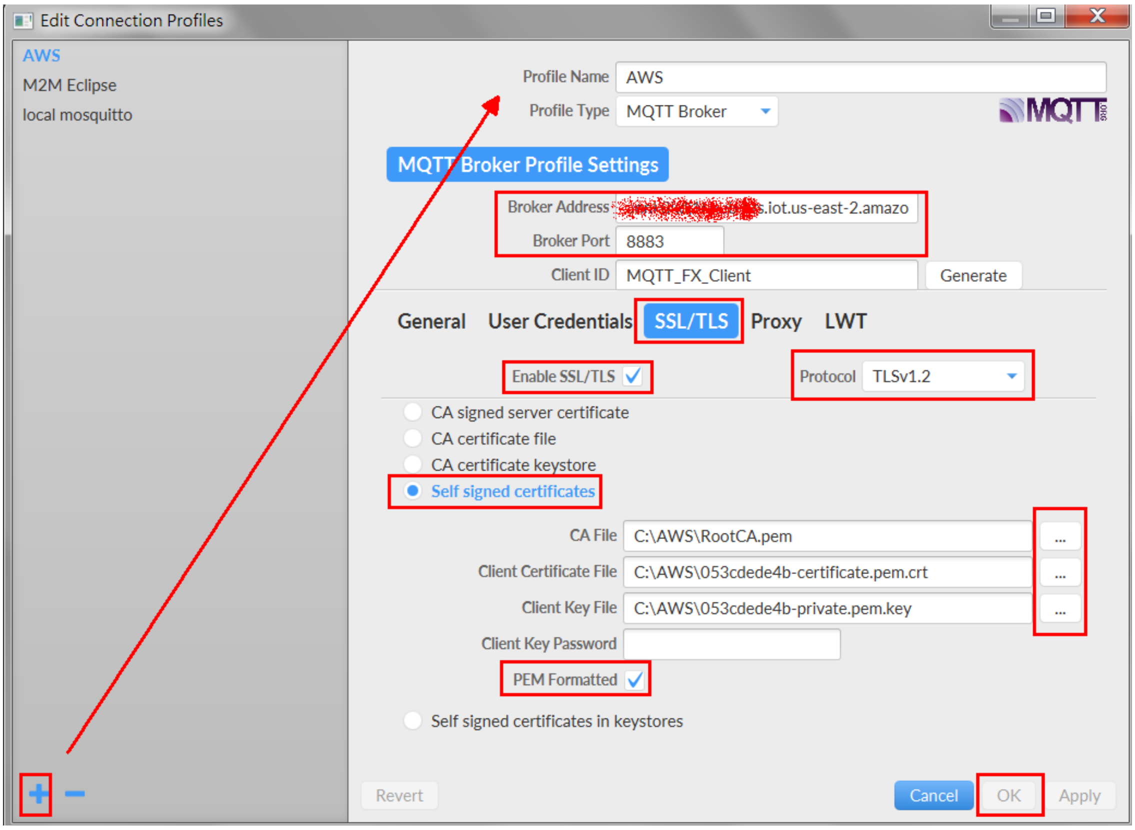
6. MQTT.fx Settings

6.1 Connection Settings

PC2 runs the MQTT.fx client. Create a new connection profile by going to the following menu option : **Extras** > **Edit Connection Profiles** and then click the **+** button (bottom left).

Input the correct Broker Address and Broker Port as 8883. In the **SSL/TLS** tab, enable **SSL/TLS** and choose **TLSv1.2** for **Protocol**.

Select **Self signed certificates in keystores**, then upload **Root CA certificate**, **Client Certificate**, and **Client Key**. Enable **PEM Formatted**. Then press **OK** or **Apply** to finish the connection settings.



Profile Name AWS

Profile Type MQTT Broker

Broker Address [redacted].iot.us-east-2.amazo

Broker Port 8883

Client ID MQTT_FX_Client Generate

Enable SSL/TLS ✓

Protocol TLSv1.2

○ CA signed server certificate

○ CA certificate file

○ CA certificate keystore

● Self signed certificates

CA File C:\AWS\RootCA.pem ...

Client Certificate File C:\AWS\053cdede4b-certificate.pem.crt ...

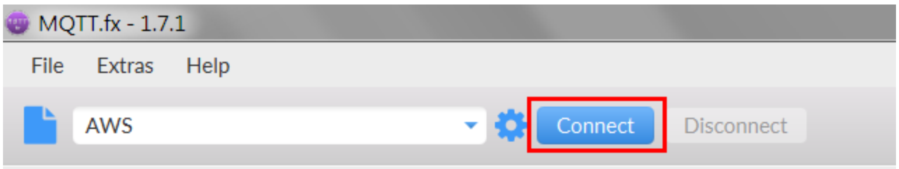
Client Key File C:\AWS\053cdede4b-private.pem.key ...

Client Key Password

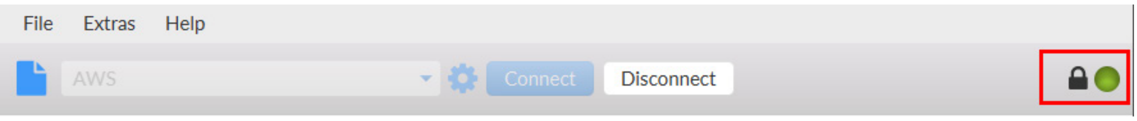
PEM Formatted ✓

○ Self signed certificates in keystores

Select your AWS connection profile then click **Connect**:

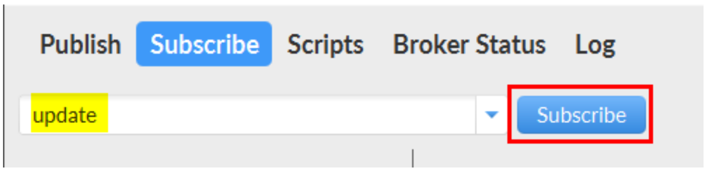


When connected, the connection status signal turns green.



6.2 Subscribe Topic

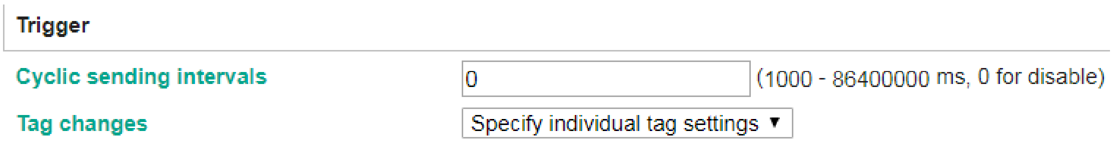
In the Subscribe tab, input **update**, which is the MGate 5105's publish topic name. Then click **Subscribe**.



7. Communication Test

7.1 Publish message

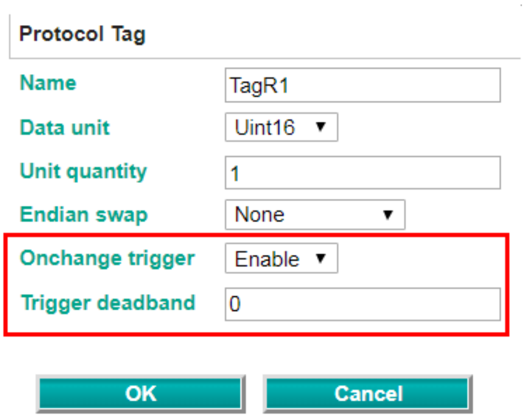
We set **Trigger** as follows: For Cyclic sending interval, choose **0**; for tag changes, choose **Specify individual tag settings**:



Trigger

Cyclic sending intervals	0 (1000 - 86400000 ms, 0 for disable)
Tag changes	Specify individual tag settings ▼

We set TagR1 **Onchange trigger** as enable with **Trigger deadband** as 0.

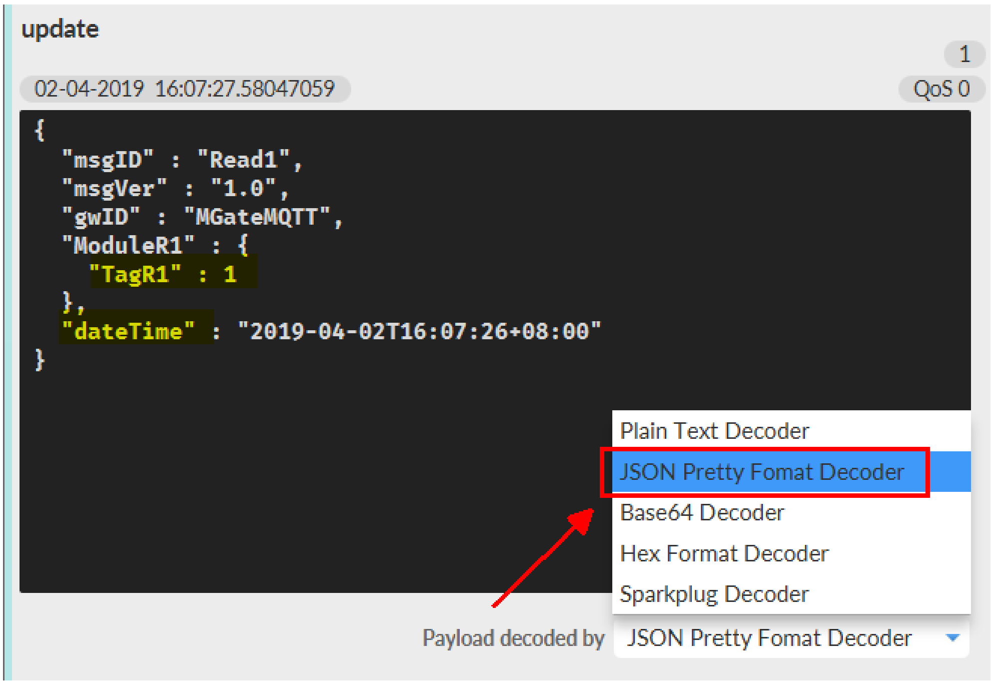


Protocol Tag

Name	TagR1
Data unit	Uint16
Unit quantity	1
Endian swap	None
Onchange trigger	Enable
Trigger deadband	0

So when the MGate 5105 gets Modbus RTU device Register0’s value changed, it triggers to publish message to AWS IoT Thing.

Now, update Modbus Register0’s value as 1. In MQTT.fx tool, TagR1’s value is shown as 1 and with dateTime padding.



update

1

02-04-2019 16:07:27.58047059

QoS 0

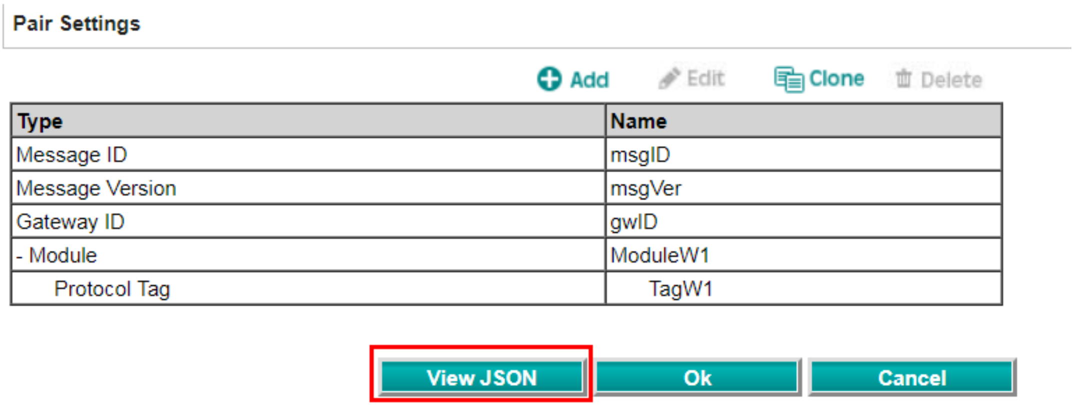
```
{
  "msgID" : "Read1",
  "msgVer" : "1.0",
  "gwID" : "MGateMQTT",
  "ModuleR1" : {
    "TagR1" : 1
  },
  "dateTime" : "2019-04-02T16:07:26+08:00"
}
```

Payload decoded by JSON Pretty Fomat Decoder

7.2 Subscribe message

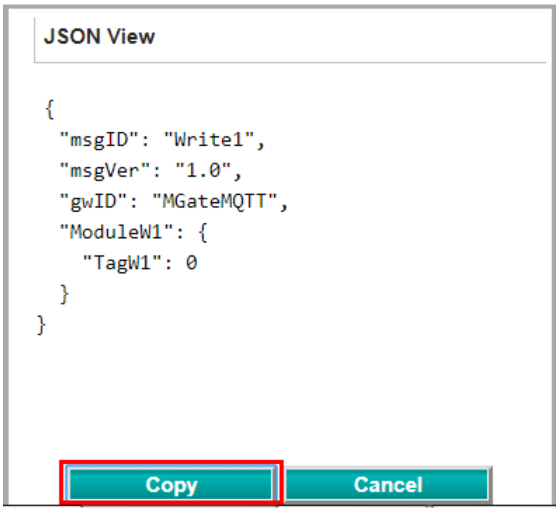
We use MQTT.fx to send messages to the device. You can follow the following steps:

1. Click **View JSON** button.



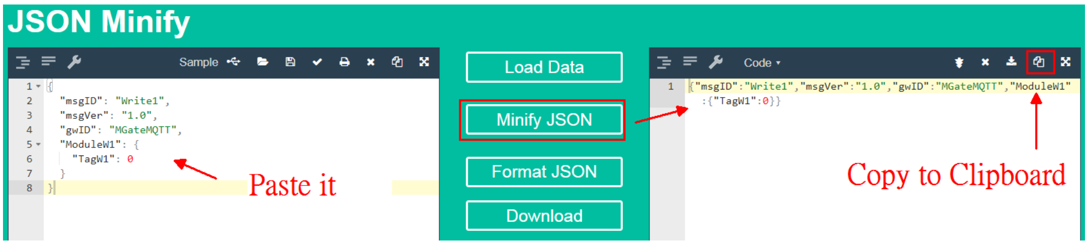
Type	Name
Message ID	msgID
Message Version	msgVer
Gateway ID	gwID
- Module	ModuleW1
Protocol Tag	TagW1

Copy Subscribe message JSON format:



```
{
  "msgID": "Write1",
  "msgVer": "1.0",
  "gwID": "MGateMQTT",
  "ModuleW1": {
    "TagW1": 0
  }
}
```

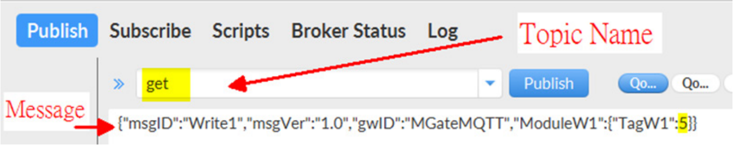
2. The copied message has a lot of space and line feed. Use tool to compact it. Download a free online tool: https://jsonformatter.org/json-minify
   Paste the message on the left side, then click **Minify JSON**. It will show a compact JSON format message on the right side. Click **Copy to Clipboard**.



```
{
  "msgID": "Write1",
  "msgVer": "1.0",
  "gwID": "MGateMQTT",
  "ModuleW1": {
    "TagW1": 0
  }
}
```

```
{"msgID":"Write1","msgVer":"1.0","gwID":"MGateMQTT","ModuleW1":{"TagW1":0}}
```

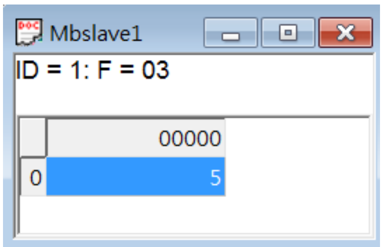
3. In the **MQTT.fx's Publish** tab, input **get** as Topic Name. Paste the message and change the **TagW1** value to 5 as below:



```
{"msgID":"Write1","msgVer":"1.0","gwID":"MGateMQTT","ModuleW1":{"TagW1":5}}
```

   Click **Publish** to send out message.

4. Check on Modbus Slave tool; Register0's value is updated as 5.



ID = 1: F = 03

	00000
0	5