Plant Pal

MQTT Publish / Subscribe

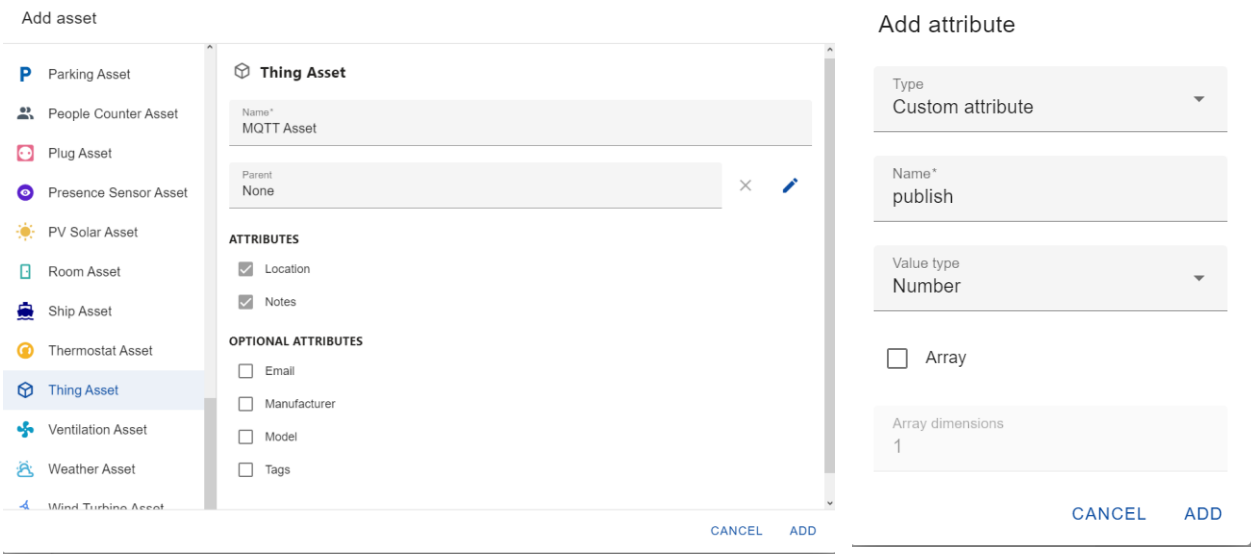
Introduction

This document contains a walkthrough of a python script, connecting to the integrated MQTT broken of OpenRemote using the MQTT protocol. It contains information about prerequisites, publishing, and subscribing to OpenRemote assets.

Setup & Prerequisites

To be able to authenticate you'll need to create a service user using the Manager UI (must be logged in as super user to access this functionality), please refer to the Manager UI user guide. After creating a user through the Manager UI, copy the **username** and auto **generated secret**. The username we will use in this demo is 'usermqtt'.

You will also need to create an asset using the Manager UI. For this demo, we will create a Thing Asset, call it 'MQTT Asset' and press ADD to add the asset to the asset tree.



Next, we will modify the asset and add two attributes, one will be named **publish** and the other will be named **subscribe**. Both of Type 'Custom attribute' and Value type 'Number'. Don't forget to save the new modifications.

Navigate to the top of the page and copy the asset ID from the URL. This is the last part of the URL:

We have now collected the necessary values from the OpenRemote Manager UI:

- Host: Host of the manager (e.g. `natlab.openremote.app`)
- Port: 8883 (if running with SSL i.e. our AWS hosted deployment)

  1883 (if running without SSL proxy i.e. a locally hosted deployment)

- Encryption/TLS: true (port 8883) false (port 1883)
- Username: `master:usermqtt`
- Password: `{secret}`
- ClientId: MQTTpub & MQTTsub

Publish:

{realm}/{clientId}/writeattributevalue/{attributeName}/{assetId}

'master/MQTTpub/writeattributevalue/publish/6oIoXiueHOisQXDXDp3NrJ'

Subscribe:

{realm}/{clientId}/attribute/{attributeName}/{assetId}

'master/MQTTsub/attribute/subscribe/6oIoXiueHOisQXDXDp3NrJ'

For the script you will also need to install the following:

- Python (>= 3.10)
- Certifi (>= 2022.5.18.1)
- Paho-MQTT (>= 1.6.1)
- Time (Python Standard Library)
- JSON (Python Standard Library)

Publishing

The full example code can be found in the 'publish.py' file.

To start publishing data to OpenRemote, we will first import all the necessary libraries:

```
import paho.mqtt.client as mqttClient
import time
import certifi
```

Next, we will create variables for each of the values mentioned in the setup, we will also add a variable called 'Connected' which will be used later in the code.

```
Connected = False  #global variable for the state of the connection
username = 'master:usermqtt'
secret = 'secret'
host = 'natlab.openremote.app'
port = 8883
clientID = 'MQTTpub'
assetID = '6oIoXiueHOisQXDXDp3NrJ'
attribute = 'publish'
attribute_value = 15
```

With these variables we can now create out MQTT client using the Paho-MQTT library. One very important thing we are doing is 'clientMQTT.tls_set(certify.where()). This line of code locates the appropriate CA certificate for the server, making a secure connection possible.

Underneath that we are setting the behavior we want when we are connected to the broker and when we successfully published information to OpenRemote. These will be explained further in the demo.

Next, we connect our client to the specified host over the right port, start the loop, and wait for a connection to be established.

```
clientMQTT = mqttClient.Client(clientID)
clientMQTT.username_pw_set(username, password=secret)
clientMQTT.tls_set(certifi.where())
clientMQTT.on_connect = on_connect
clientMQTT.on_publish = on_publish
clientMQTT.connect(host, port=port)
clientMQTT.loop_start()

while Connected != True:
    time.sleep(0.1)
```

Once a connection is established, we can publish our data, using the MQTT API and the format mentioned above, directly to the integrated MQTT broker.

After the data is successfully published, we disconnect the client from the broker and stop the loop.

```
clientMQTT.publish(f"master/{clientID}/writeattributevalue/{attribute}/{assetID}", attribute_value)

clientMQTT.disconnect()
clientMQTT.loop_stop()
```

For the final part, we will add the methods that will be called when the client connects to the broker, and when the data is successfully published to OpenRemote.

```python
def on_connect(client, userdata, flags, rc):
    if rc == 0:

        print("Connected to broker")

        global Connected  #Use global variable
        Connected = True  #Signal connection

    else:

        print("Connection failed")


def on_publish(client, userdata, result):
    print("Data published \n")
    pass
```

Now that all the code we need is in place, we can run the script. If everything went correct, the output would look something like this in the terminal:

```
Connected to broker
Data published
```

The updated asset attribute in the Manager UI:

Congratulations! You have now published data to an asset attribute through the OpenRemote MQTT broker using the MQTT Manager API.

Subscribing

The full example code can be found in the 'subscribe.py' file.

The process of subscribing is in many aspects very similar to that of publishing, however it is important to note a few critical differences.

To start subscribing to attribute events of OpenRemote assets, we will first import all the necessary libraries:

```
import paho.mqtt.client as mqttClient
import time
import certifi
import json
```

As you might have noticed, these are mostly the same libraries as we used in the publishing example, with the addition of the python standard library JSON. This library will be used to load the message as a dictionary to access the individual variables.

Next, we will create variables for each of the values mentioned in the setup, we will also add a variable called 'Connected' which will be used later in the code.

```
Connected = False  #global variable for the state of the connection
username = 'master:usermqtt'
secret = 'secret'
host = 'natlab.openremote.app'
port = 8883
clientID = 'MQTTsub'
assetID = '6oIoXiueHOisQXDXDp3NrJ'
attribute = 'subscribe'
```

With these variables we can now create out MQTT client using the Paho-MQTT library. One very important thing we are doing is 'clientMQTT.tls_set(certify.where()). This line of code locates the appropriate CA certificate for the server, making a secure connection possible.

Underneath that we are setting the behavior we want when we are connected to the broker and when we successfully received information from OpenRemote. These will be explained further in the demo.

Next, we connect our client to the specified host over the right port, start the loop, and wait for a connection to be established.

```
client = mqttClient.Client(clientID)
client.username_pw_set(username, password=secret)
client.tls_set(certifi.where())
client.on_connect= on_connect
client.on_message= on_message
client.connect(host, port=port)
client.loop_start()

while Connected != True:
    time.sleep(0.1)
```

Once a connection is established, we can subscribe to the attribute of our asset, using the MQTT API and the format mentioned above, directly to the integrated MQTT broker.

After the client is successfully subscribed, we start a loop to listen for attributeEvents. When the loop is broken, we close the client connection and stop the loop.

```
client.subscribe(f"master/{clientID}/attribute/{attribute}/{assetID}")

try:
    while True:
        time.sleep(1)

except KeyboardInterrupt:
    print("Exiting")
    client.disconnect()
    client.loop_stop()
```

For the final part, we will add the methods that will be called when the client connects to the broker, and when the data is received from OpenRemote.

```
def on_connect(client, userdata, flags, rc):

    if rc == 0:

        print("Connected to broker")

        global Connected                #Use global variable
        Connected = True                #Signal connection

    else:

        print("Connection failed")


def on_message(client, userdata, message):
    msg = json.loads(message.payload.decode("utf-8"))
    id = msg["attributeState"]["ref"]["id"]
    att_name = msg["attributeState"]["ref"]["name"]
    att_value = msg["attributeState"]["value"]
    print(f'Message received. Asset ID: {id}. Attribute name: {att_name}. Attribute value: {att_value}.')
```

Now that all the code we need is in place, we can run the script. Then we will go over to our subscribe attribute in the Manager UI and change the value to 10. If everything went correct, the output would look something like this in the terminal:

```
Connected to broker
Message received. Asset ID: 6oIoXiueHOisQXDXDp3NrJ. Attribute name: subscribe. Attribute value: 10.0.
```

The updated asset attribute in the Manager UI:

Congratulations! You have now subscribed to data from an asset attribute through the OpenRemote MQTT broker using the MQTT Manager API.