MQTT for Unity V1.1.0

Andreas Vogler

Introduction

"MQTT for Unity" is a powerful Unity Package designed to seamlessly integrate MQTT (Message Queuing Telemetry Transport) functionality into Unity projects, offering a user-friendly and efficient solution for enabling real-time communication and data exchange within Unity applications. This package simplifies the process of implementing MQTT protocol, allowing Unity developers to effortlessly connect their projects to MQTT broker servers and create interactive and responsive applications.

Connection

Please also take a look at the SampleScene in the scene folder.

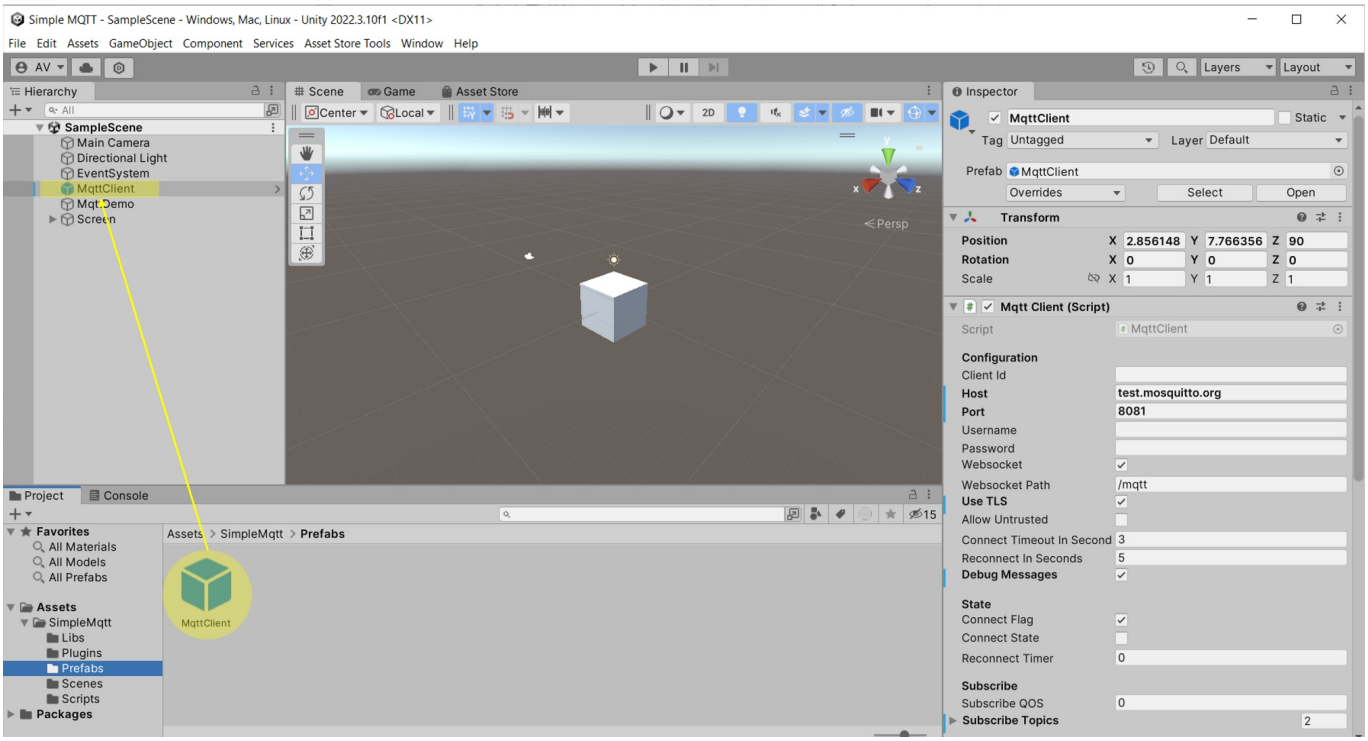
Prefab

The MqttClient Prefab consists of three components:

- Mqtt Client: Here we have the configuration settings.
- Mqtt Client Web: This one will be active if you build for WebGL.
- Mqtt Client Net: This one will be active in all other cases (Not WebGL).

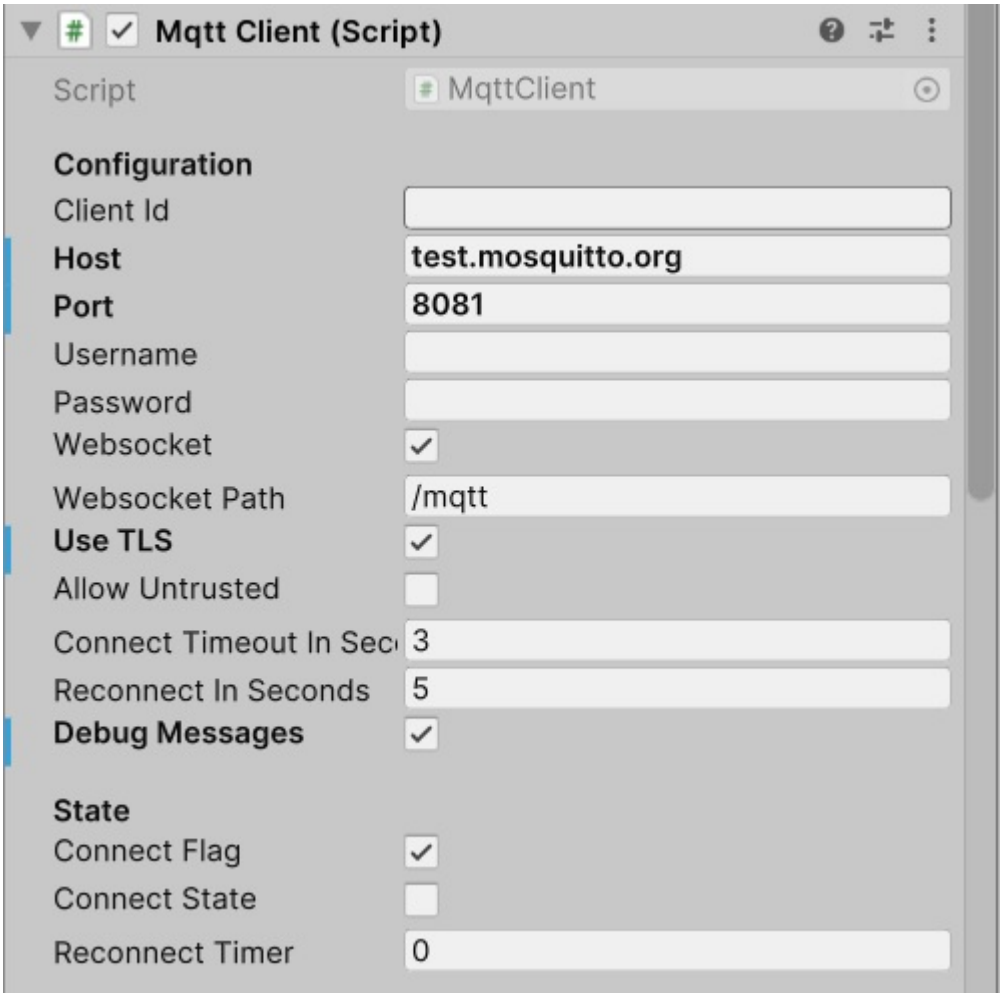
If you only build your application for Non-WebGL, then you can also create your own game object and only add the "MqttClientNet" and "MqttClient" component to it.

Drag and drop the Prefab "MqttClient" to your scene.



Configuration

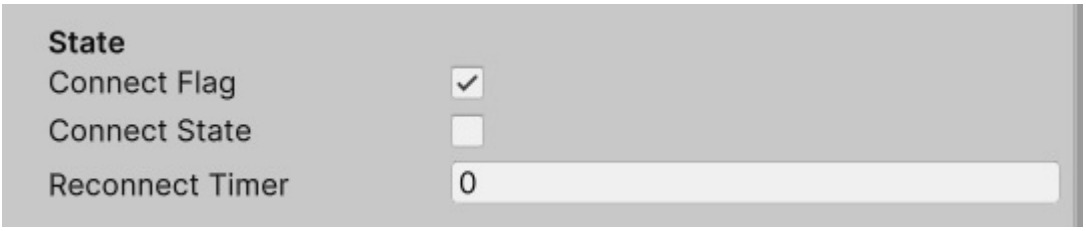
The "MqttClient" component holds the configuration for the MQTT connection. You can set it in the Inspector.



Configuration Options:

- ClientId: MQTT Client Id, if it is not set, then a UUID will be generated for this field.
- Host: Host of the MQTT Broker to which you want to connect
- Port: Port of the MQTT Broker to which you want to connect
- Username: Optional a Username if authentication is required
- Password: Optional a Password if authentication is required
- Websocket: If you want to connect via websocket protocol (Note: with WebGL build only Websocket is possible and this setting will be ignored. Be sure that you have set the right port for the websocket connection)
- Websocket Path: Most MQTT Broker are using "/mqtt" for Websocket connections, if your broker uses another path, then you can adapt it here. Be sure that your broker has Websocket protocol enabled if you use this option.
- Use TLS: Transport Layer Security (TLS) encrypts data. Note: if you use WebGL, then TLS should be used, because browser typically block any unsecured connection by default.
- Allow Untrusted: If you have a TLS connection and your MQTT Broker does not use a valid certificate, then this can be set to accept untrusted connections.
- Connect Timeout in Seconds: If connection cannot be opened within this time, then the open connection command is stopped.
- Reconnect in Seconds: If this is greater than 0, then a reconnect is done in x seconds when the connection got broken.
- Debug Messages: If set, then some debug log messages are printed.

State

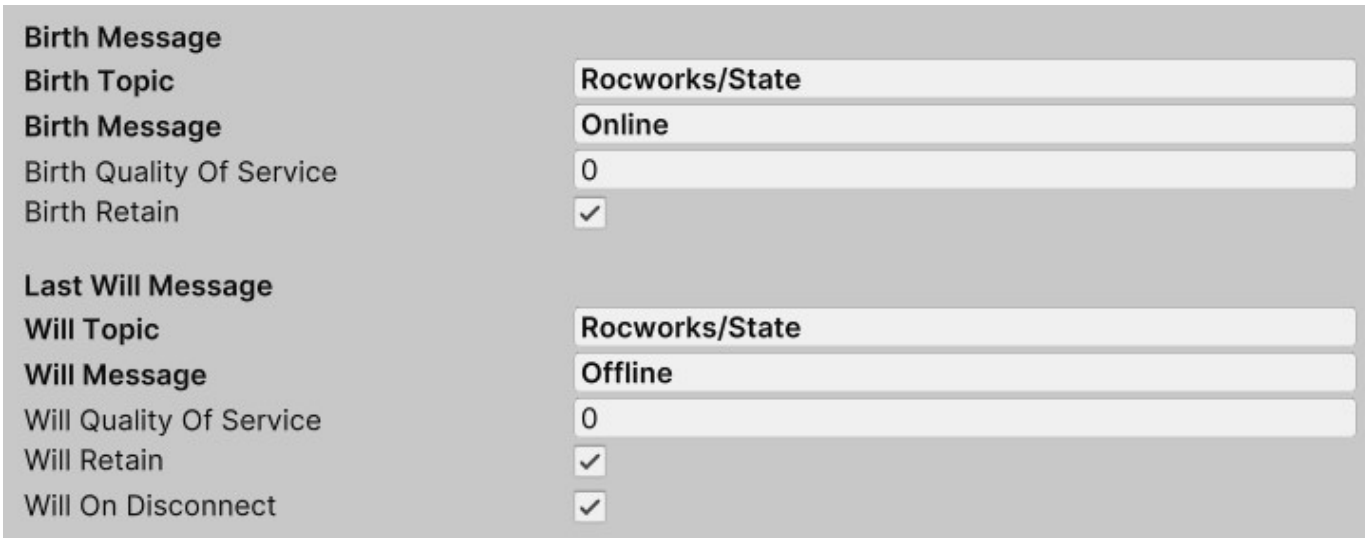


If you run the scene then you will see the current state in the Inspector.

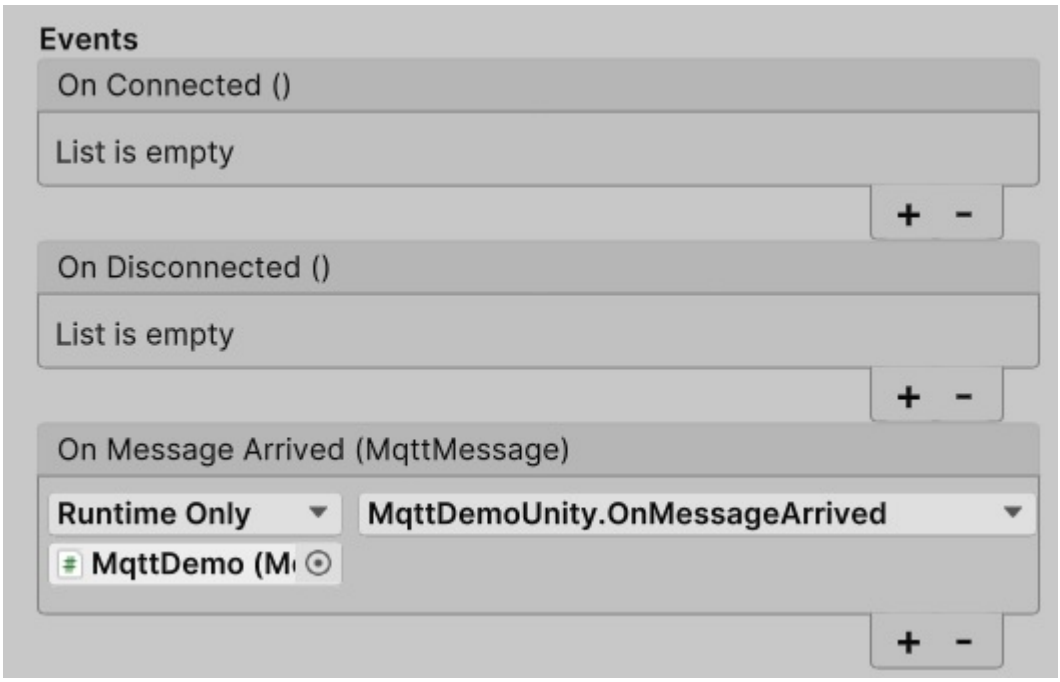
- Connect Flag: if set, it will try to connect to the broker. If unset then it will close the connection to the broker. Can be set in the inspector or via scripting. If you want to connect immediately after the scene is started, then set this flag to true.
- Connect State: if the connection opened then this is set to true, if the connection is not up or got broken, then it will be set to false. Don't set this in the inspector or via scripting!
- Reconnect Timer: if the connection got broken, then you will see here how long it waits until it will try to reconnect. The value is updated with every game-loop. Once this time is greater than the "Reconnect in Seconds" configuration, the system will try to reconnect to the broker.

Birth and Last Will

A birth message is sent everytime when the connection to the broker goes up. A last will message (testament message) is sent when the program goes down unintended (program kill, unintended lost connection). This message is sent by the broker. You can also define that the last will message is sent when the client disconnects the session normally - set the property "Will On Disconnect" to true. If Topic or Message is empty, then no birth or last will messages are sent.



Unity Events



Following events available:

- On Connected: Function without an argument. Will be fired every time when the connection goes up.
- On Disconnected: Function without an argument. Will be fired every time when the connection goes down.
- On Message Arrived: Function with one arguments of type "MqttMessage". The MqttMessage object contains the topic and the payload of the message. There is an additional property "On Message Arrived In Ui Thread", which defines if the event is executed in the MQTT background thread or in the Unity UI/Main thread (Note: in a WebGL build it is allways executed in the UI/Main thread).

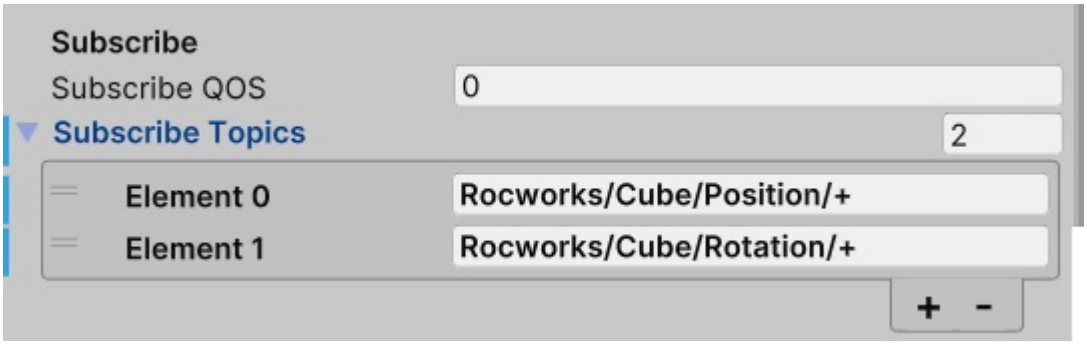
Add your own game object with a function to it, to execute your own business logic when events are fired.

Example OnMessageArrived function, which can be added to the On Message Arrived event:

```
public void OnMessageArrived(MqttMessage m)
{
```

```
        switch (m.GetTopic()) {
            case "Rocworks/Cube/Position/X":
                _positionX = float.Parse(m.GetString(), CultureInfo.InvariantCulture);
                break;
            case "Rocworks/Cube/Position/Y":
                _positionY = float.Parse(m.GetString(), CultureInfo.InvariantCulture);
                break;
            case "Rocworks/Cube/Position/Z":
                _positionZ = float.Parse(m.GetString(), CultureInfo.InvariantCulture);
                break;
            case "Rocworks/Cube/Rotation/X":
                _rotationX = float.Parse(m.GetString(), CultureInfo.InvariantCulture);
                break;
            case "Rocworks/Cube/Rotation/Y":
                _rotationY = float.Parse(m.GetString(), CultureInfo.InvariantCulture);
                break;
            case "Rocworks/Cube/Rotation/Z":
                _rotationZ = float.Parse(m.GetString(), CultureInfo.InvariantCulture);
                break;
        }
    }
```

Subscribe



In the Inspector you can add topics to which you want to subscribe. This topics will be automatically get subscribed when the MQTT connection gets up. You can add here as many topics you need. With the QOS property you can set the Quality of Service for the subscription.

- 0: at most once.
- 1: at least once.
- 2: exactly once.

You can also use the Subscribe function to subscribe to values. The function is available at the "Connection" item of the MqttClient component (MqttClient.Connection.Subscribe).

```
public void Subscribe(string topic, int qos);
```

Publish

There are two functions available to publish values to the MQTT broker. One uses a byte array as value and one uses a string as value. The functions are available at the "Connection" item of the MqttClient component (MqttClient.Connection.Publish).

```
public void Publish(string topic, string payload, int qos = 0, bool retain =
false);
public void Publish(string topic, byte[] payload, int qos = 0, bool retain =
false);
```

Example:

```
public class MqttDemoUnity : MonoBehaviour
{
    public MqttClient Client; // drag your MqttClient instance to here

    public void PublishMyValueToMqtt(float value)
    {
        Client.Connection.Publish("Rocworks/Cube/Position/X",
value.ToString(CultureInfo.InvariantCulture));
    }
    ...
```