Using MqttClient

Using MQTT client from library is very simple. First you have to create an instance of *MqttClient* class which provides only one mandatory parameter (the IP address or the host name of the broker you want to connect to) and some optional parameters with default values (MQTT broker port, secure connection and X.509 certificate). In the simpler case, you can use the default port (1883) and you don't have the support for secure connection based on SSL/TLS using default values for optional parameters and specifying only broker address (or host name).

```
MqttClient client = new
MqttClient(IPAddress.Parse("192.168.10.53"));
```

Before connect to the broker, you can register to the event *MqttMsgPublishReceived* raised when a message is published on a topic the client is subscribed to. In this case, you have to provide an event handler to handle the incoming message using the instance of *MqttMsgPublishEventArgs* that exposes the data bytes through the *Message* property.

```
client.MqttMsgPublishReceived += client_MqttMsgPublishReceived;
...
void client_MqttMsgPublishReceived(object sender,
MqttMsgPublishEventArgs e)
{
// access data bytes throug e.Message
}
```

You can also be interested to know if a subscription and/or unsubscription to a topic is completed and registered to the broker. In this case, you can define two event handlers for the events *MqttMsgSubscribed* and *MqttMsgUnsubscribed*.

```
client.MqttMsgSubscribed += client_MqttMsgSubscribed;
client.MqttMsgUnsubscribed += client_MqttMsgUnsubscribed;
...
```

```
void client_MqttMsgUnsubscribed(object sender,
MqttMsgUnsubscribedEventArgs e)
{
// write your code
}

void client_MqttMsgSubscribed(object sender,
MqttMsgSubscribedEventArgs e)
{
// write your code
}
```

顯示更多

If you are using QoS Level 1 or 2 to publish a message on a specified topic, you can also register to *MqttMsgPublished* event that will be raised when the message will be delivered (exactly once) to all subscribers on the topic.

```
client.MqttMsgPublished += client_MqttMsgPublished;
...
void client_MqttMsgPublished(object sender,
MqttMsgPublishedEventArgs e)
{
// write your code
}
```

After you have registered to all events you are interested in, you can use the *Connect()* method of *MqttClient* class to connect to the broker. The only mandatory parameter is the client ID that must be unique; the other parameters have some default values and they are related to :

- Username and password for client authentication (default values are *null*, no authentication);
- Will message feature (default values provides NO Will message);
- Clean session for removing subscriptions on disconnection (default value is *true*);
- Keep Alive period for keeping alive connection with ping message (default value is *60 seconds*);

Without using all the MQTT advanced features (authentication and “will” message), you can specify only the client ID (for example a generated GUID) and

leave the default keep alive period with clean session flag set.

```
client.Connect(Guid.NewGuid().ToString());
```

To subscribe and unsubscribe to a topic, the *MqttClient* class provides *Subscribe()* and *Unsubscribe()* methods. The former needs the list of topics and relative QoS levels to subscribe and the latter needs only the list of topic to unsubscribe.

```
string[] topic = { "sensor/temp", "sensor/humidity" };

byte[] qosLevels = { MqttMsgBase.QOS_LEVEL_AT_MOST_ONCE,
MqttMsgBase.QOS_LEVEL_AT_MOST_ONCE };
client.Subscribe(topic, qosLevels);
```

Subscribe() method returns the message id for subscription.

```
string[] topic = { "sensor/temp", "sensor/humidity" };

client.Unsubscribe(topics);
```

The last method is *Publish()* by which you can publish a message to a topic, specifying topic itself and the data bytes for the message. You can also set the QoS level parameter (that has a default value level 0) and the retain flag for delivery message also to the clients that aren't already connected when the message is published. In this case, the broker saves the message and send it to a new client as soon as it connects and subscribe to the topic.

```
client.Publish("sensor/temp", Encoding.UTF8.GetBytes(temp)
```

The M2Mqtt client library supports an internal "inflight queue" to execute all publish requests asynchronously (also subscribe and unsubscribe actions). The methods related to the above features (*Publish()*, *Subscribe()* and *Unsubscribe()*) return the message id assigned to the MQTT message sent to the broker. When the asynchronous operation is executed, the corresponding event is raised and the event args contains info on message id so that the user can match it.