# Simio API Note: MQTT Steps

July 2020 (Dhouck)

Contents

[Simio API Note: MQTT Steps 1](#_Toc54425838)

[Overview 2](#_Toc54425839)

[Some Background Information on MQTT 3](#_Toc54425840)

[Doing MQTT with the Simio MQTT Steps API 4](#_Toc54425841)

[MQTT Steps Code Overview 6](#_Toc54425842)

[Running the Model 7](#_Toc54425843)

[Notes on Use 8](#_Toc54425844)

[The Experiment (multi-thread) Problem. 8](#_Toc54425845)

[Adding Logic 8](#_Toc54425846)

[TroubleShooting 9](#_Toc54425847)

[Make sure the Mosquitto Server/Broker is running. 9](#_Toc54425848)

[If the Server is on a remote computer, check your firewall. 9](#_Toc54425849)

[Appendix – Using Client Certificates 10](#_Toc54425850)

# Overview

This API Note describes a User Extension that permits Simio to communicate using the popular IoT MQTT communications protocol.

The techniques discussed here provide the following:

1. Creating a Simio Process Step that Publishes.
2. A Simio Subscribe Element that Fires an Event when a MQTT topic arrives.

For testing, an Mqtt Test utility is provided, as well as a Simio example model. An explanation of how to acquire and run a 3rd party MQTT Broker/Server.

## Some Background Information on MQTT

MQTT (Message Queueing Telemetry Transport) is a lightweight Publish and Subscribe protocol that is used for IoT (Internet of Things) communications. Communications conversations each have a unique Topic that is hierarchical and formatted using slashes. A packet information conveyed about a Topic is called a Payload.

For example, if we are communicating information about information in a factory, a Topic might be of the form Location/Machine/Information So, for example roughing-pulpit/rollstand-7/speed. If our client subscribes to this Topic, it might get a Payload of “Setpoint=23,Average=22.4”. But you can see that it entirely up to the designers of the communications to specify the formats for the Topics and Payloads.

Note that the topics are case-sensitive, so it has become a convention to make the topics lower-case… but it is of course up to you.

The process responsible for brokering these conversations is called a MQTT Server. When a client wants to listen to a conversion, it Subscribes to a Topic. If it also wants to contribute to the conversation, it can also Publish a Payload with that (or any other) Topic.

The missing piece of information is how to communicate with the Server. This is done by referencing its address, which is a URL and Port. As you may infer from this, the underlying protocol is IP.

# Doing MQTT with the Simio MQTT Steps API

The implementation of MQTT for Simio involves:

1. A MqttSubscribeElement
2. A MqttPublishStep
3. A MqttRpcStep (future)

The MqttSubscribeElement references an MqttSubscribeConnector, which contains the MQTT broker/server information. This element subscribes to an MQTT topic and fires Simio Events when a message is received. There are three pre-defined Simio events: OnMqttReceivedEvent2, OnMqttReceivedEvent2 OnMqttReceivedEvent3. The MQTT payload determines which event to fire, and also includes a payload that the Element will place in the Simio Payload State variable. An example payload might look like this: SimioEvent=2,Payload=this is just an example payload.

An MqttPublishStep references a MqttPublishConnector, which contains the MQTT broker/server information. When a Simio Entity enters the Step, the Payload associated with the Step is published under the given Topic. This could be used to signal events to external programs. For example, that the Entity has entered a particular server. The format of the payload is up to you.

Both the MqttSubscribeElement and the MqttPublishStep use a MqttConnector element that specifies the address of the MQTT Broker/Server. The difference between them is that the MqttPublishServerElement also holds a reference to an MQTT Client so that the internal code does not have to keep creating and destroying it.

The MqttRpcStep is a future enhancement that will include a combination of Subscribe and Publish and implements an RPC (Remote Procedure Call). Within the step a Publish with a unique topic is made and then subscribes to the unique response to that topic. The simulation is therefore paused until the response is received (or a timeout occurs). This step would be used, for example, if the simulation required information from an external program. For example, a request for an optimization from Matlab or some other optimization routine.

# The Example Models

In the example models, one of the Steps is place at the entrance of the Simio Server. When the Entity enters the server, it MQTT Publishes at Topic its name as the Payload with the topic of server1/enter, and publishes an Entity Create message for Model2 as it enters its Sink.

The second model MQTT Subscribes to an Entity/Create message. Upon this topic being published, a new entity is created which then flows through the model Publishing information as it enters the Server, and publishes an Entity create for Model1 when it enters the Sink.

To run the Models you will need an MQTT Broker/Server. We will use the Mosquitto free server. Information about getting and using Mosquitto is contained in an Appendix.

To initiate messages we will also use the free MQTT Explorer app that is available in the Microsoft Store. It was written by Thomas Nordquist and is a very useful debugging and learning tool for MQTT.

A screenshot of a cell phone

Description automatically generated

Start the Explorer App

A screenshot of a cell phone

Description automatically generated

Graphical user interface, application

Description automatically generated

## MQTT User-Defined Objects Overview

Using the MQTT users-defined objects is rather straightforward.

The Publish and Subscribe connectors have identical properties. You specify the address and port of the MQTT broker/server. The connectors are then references by the MQTT Publish Step or the MQTT Subscribe Element.

The MQTT Step has properties where you can reference the MQTT Connector, and then specify the Topic and Payload as strings.

Graphical user interface, application

Description automatically generated

## MQTT Steps Code Overview

The MQTT client-side code for the steps was written using the NuGet package MQTTNet (by Christian Cratky)

The code includes a user-defined Simio Step for publishing, MQTT connector Elements for both Publishing and Subscribing, and a Simio Element for Subscribing to a MQTT topic and then raising a Simio Event when a message with that topic appears.

# Running the Model

There are some moving parts involved in this setup, but it is rather straightforward, and once it is setup with the MQTT service automatically starting it is easy to use.

This was tested with the popular MQTT Server (aka broker) called Mosquitto Server ( <https://mosquitto.org> ).

The steps to get this running are:

Download and install the Mosquitto server as a service and start it. You can set it up to run automatically each time your computer starts.

Also, download the free MQTT Explorer using the Microsoft Store. This will be used to test the Simio MQTT components and provides a way to watch what is happening.

Place the MqttSteps.DLL in your {username}/Documents/SimioUserExtensions folder.

Launch the MQTT Explorer.

Open the Simio project.

Now do the following:

1. MqttTest utility: Subscribe to server1/enter and server1/exit
2. MqttTest utility: Set the Publish topic to entity/launch
3. Simio: Start the Simio simulation
4. MqttTest utility: Press Publish
5. Simio: Observe that an Entity is emitted from the source, and when it hits the Server1…
6. MqttTest utility: … Observer that payloads for server1/enter and server1/exit are logged.

# Notes on Use

## The Experiment (multi-thread) Problem.

## Adding Logic

# TroubleShooting

## Make sure the Mosquitto Server/Broker is running.

## If the Server is on a remote computer, check your firewall.

Check the Mosquitto Server using utilities such as MQTT Explorer (Windows)

# Appendix – Using Client Certificates

Much of this information is derived from this great article here: : <http://www.steves-internet-guide.com/creating-and-using-client-certificates-with-mqtt-and-mosquitto/>

This references other articles:

<http://www.steves-internet-guide.com/mosquitto-tls/>

<http://www.steves-internet-guide.com/ssl-certificates-explained/>

The main points of these articles are repeated here, so for more details go to Steves-Internet-Guide.

Our examples use OpenSSL, which is available here:

<https://www.openssl.org/>

For the examples here, we used the OpenSSL binary provided by FireDaemon:

<https://kb.firedaemon.com/support/solutions/articles/4000121705>

For ease-of-use, the files found deep under the x64\bin folder where copied to c:\tools\openssl and the path to this was added to the local PATH environment variable for easier access to OpenSSL.

The end goal here is to make an encrypted connection between the MQTT broker and the MQTT client.

For some diagnostic information, read this:

<https://stackoverflow.com/questions/3463723/determining-web-http-authentication-methods>

# Appendix – Using Mosquitto as the MQTT Broker/Server

Perhaps the most popular free MQTT Broker/Server is Mosquitto (and yes, it is a misspelling of Mosquito, I suppose so they could get all of M-Q-T-T).

Mosquitto is free from Eclipse and downloadable from here: <https://mosquitto.org/download/>

Download the binary (the Windows 64 is used here).

The install is straightforward and the result is a service called

Graphical user interface, application, Word

Description automatically generated

Graphical user interface, text, application

Description automatically generated

Graphical user interface, text, application

Description automatically generated

Graphical user interface, text, application

Description automatically generated

# Appendix – Video Narration Script

This video describes a way to extend Simio to interface with external processes and devices using MQTT

MQTT is perhaps the most popular communication mechanism for IoT.

It is an open source and ISO standard for a publish-subscribe (sometimes refer pub-sub) messaging protocol.

This example demonstrates using a user-defined Simio process step to publish MQTT topics, and a user-defined Simio Element to subscribe to and fire Simio Events when the message arrives.

There are many applications for this, from custom simulator monitoring, data collection, and dashboards. It can also be used for emulation, and for allowing several distributed simulations to work in concert.

\*\*\* Show the running Sample1 and Sample2 \*\*\*

The included example models demonstrate two Simio projects interacting, with entities from Project 1 “transferring” to Project 2 upon completing. At the same time, we will be using a small Windows app to interface and debug MQTT messages.

\*\*\* Diagram showing MQTT \*\*\*

Diagram

Description automatically generated

First, a quick review of MQTT. It is called a pub-sub because the clients can both “publish” a message with a given topic, and also subscribe to topics and therefore receive a message anytime a topic changes. The data within a topic is called a “payload”.

In MQTT, topics are case-sensitive, hierarchical text with a slash used as the hierarchy delimiter, plus wildcard delimiters “#” and “+” that can be used for levels of the hierarchy. This is best shown with an example for a distributed security example for a company that has facilities in multiple cities.

The topic hierarchy is {city}/{zone}/{device}

And the payload is comma delimited string of sensor-name = value

So, the topic OilCity/Warehouse/Temperature might return as its payload as something like: “Current=64/High=67/Low=60”

\*\*\* Table shows Topic examples \*\*\*

Some example subscriber topic requests and results are:

|  |  |
| --- | --- |
| OilCIty/# | Any OilCity zones and sensors |
| OIlCity/+/Temperature | Temperatures for all zones |
| OilCity/Warehouse | All sensors for zone “Warehouse” |

Using MQTT with Simio

\*\*\*Diagram show how Simio uses MQTT\*\*\*

Diagram

Description automatically generated

The user-defined objects in this example show one way to do an MQTT implementation.

Let us look at our example models, which are two simple Source-Server-Sink models.

First Model 1:

Model1 has an MQTT Subscriber Element MqttSubscriberElement1 that subscribes to the topic MqttSample1/Entity/Create and fires the OnMqttReceivedEvent1 when such a message arrives.

Its Source has Entity Arrival logic that creates an entity when OnMqttReceivedEvent1 fires.

As the Entity passes through the model, it publishes the topic MqttSample1/Input@Server1 when the entity enters the server.

Finally, when the Sink is entered it publishes a message with the topic MqttSample2/Entity/Create and payload information necessary for creating an Entity.

Next Model 1:

Model2 is nearly identical to Model1, but when an Entity enters its Sink, it MQTT publishes to the MqttSample1/Entity/Create, which cause Model1 to create an Entity, and thus continues the loop.

Cleary, the uses for MQTT communications are endless, and the Simio MQTT add-in allows for easy integration with any MQTT eco-system.

Happy Simulation.