# Simio API Note: MQTT Steps

July 2020 (Dhouck)

Contents

[Simio API Note: MQTT Steps 1](#_Toc54192443)

[Overview 2](#_Toc54192444)

[Some Background Information on MQTT 3](#_Toc54192445)

[Doing MQTT with the Simio MQTT Steps API 4](#_Toc54192446)

[MQTT Steps Code Overview 6](#_Toc54192447)

[Running the Model 7](#_Toc54192448)

[Notes on Use 8](#_Toc54192449)

[The Experiment (multi-thread) Problem. 8](#_Toc54192450)

[Adding Logic 8](#_Toc54192451)

[TroubleShooting 9](#_Toc54192452)

[Make sure the Mosquitto Server/Broker is running. 9](#_Toc54192453)

[If the Server is on a remote computer, check your firewall. 9](#_Toc54192454)

[Appendix – Using Client Certificates 10](#_Toc54192455)

# 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 MQTT message.
2. A Simio Subscribe Element that Fires an Event when a MQTT message arrives.

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

## Some Background Information on MQTT

MQTT (Message Queueing Telemetry Transport) is a lightweight Publish and Subscribe (“Pub-Sub”) 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 MqttServerElement
2. A MqttSubscribeElement
3. A MqttPublishStep
4. A MqttRpcStep

The MqttServerElement holds a reference to an MQTT broker. It has properties that includes the address (URL and Port) of the broker, as well as a Simio Event that is fired when a connection is made or broken.

The MqttSubscribeElement that has properties for the MQTT Server address (URL and Port), and a MQTT Topic that is to be subscribed to, and a Simio Event. Each time the Subscribed Topic arrives, the Simio Event fires.

An MqttPublishStep references a MQTT Server Element, which contains the MQTT 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 data needed for display by R is now available.

The MqttRpcStep is 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.

In the example models, the first model has a Steps is placed at the exit of the sink. When the Entity leaves the server, it sends a message to the second model, which then creates an Entity. In this way the two simulations are connected in series.

Start the Broker/Server

A screenshot of a cell phone

Description automatically generated

Start the Explorer App

A screenshot of a cell phone

Description automatically generated

## Using MQTT in the Model

Both the MqttPublishStep and the MqttSubscriberElement require a connection to the MQTT Broker. So that changes to the broker can be more easily done, these connectors are implemented as Elements called MqttPublishConnector and MqttSubscriberConnecter, and then it is these connectors that are reference by the MqttPublishStep and MqttSubscriberElement.

## MQTT Steps Code Overview

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

The code for the Process Step is in the

# 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 automatically run 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

# 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>