Neotys

Testing Applications using MQTT over WebSocket

Introduction

The goal of this document is to explain how to design scenarios with NeoLoad to test MQTT over WebSocket.

Supported versions

- MQTT 3 and MQTT 3.1
- The MQTT websocket module has been tested with NeoLoad 5.4.0.

Requirements

- **NeoLoad** - First of all you need to download NeoLoad and install it on your machine.http://www.neotys.com/support/download-neoload.html

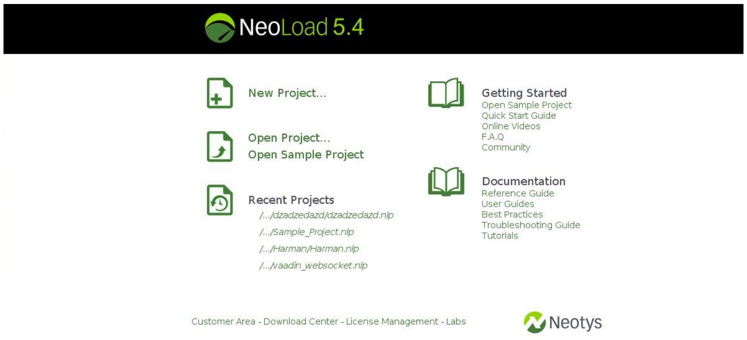
- **Neoload MQTT WebSocket's Data Format Extension** - Download the latest release. *This extension needs to be added on each new NeoLoad project using MQTT over WebSocket.*

- **MQTT Framework** - To design your scenario, you need to configure MQTT technical parameters as well as parameters related to your functional use case. In order to respect the messaging workflow related to MQTT, it is recommended to add the MQTT framework into Neoload. NeoLoad will manage all "synchronous messages with the help of the Neoload Mqtt framework. Download the latest release.

Create a new Neoload project with the MQTT module

Create a new project

1. Create a new NeoLoad project from the NeoLoad main menu.

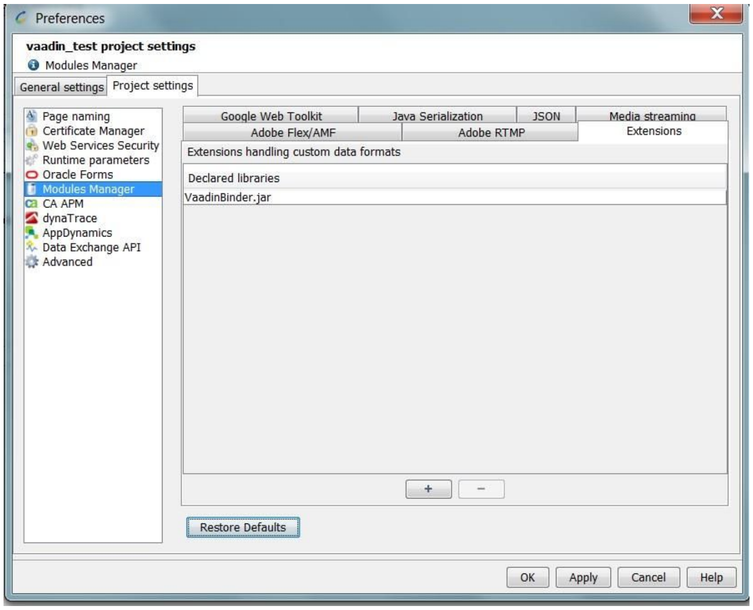


Change the settings of the new project

1. Open the Preferences screen (Edit/Preferences)

2. Select the **Project settings** tab.

3. On the left panel, select the **Modules Manager** section.

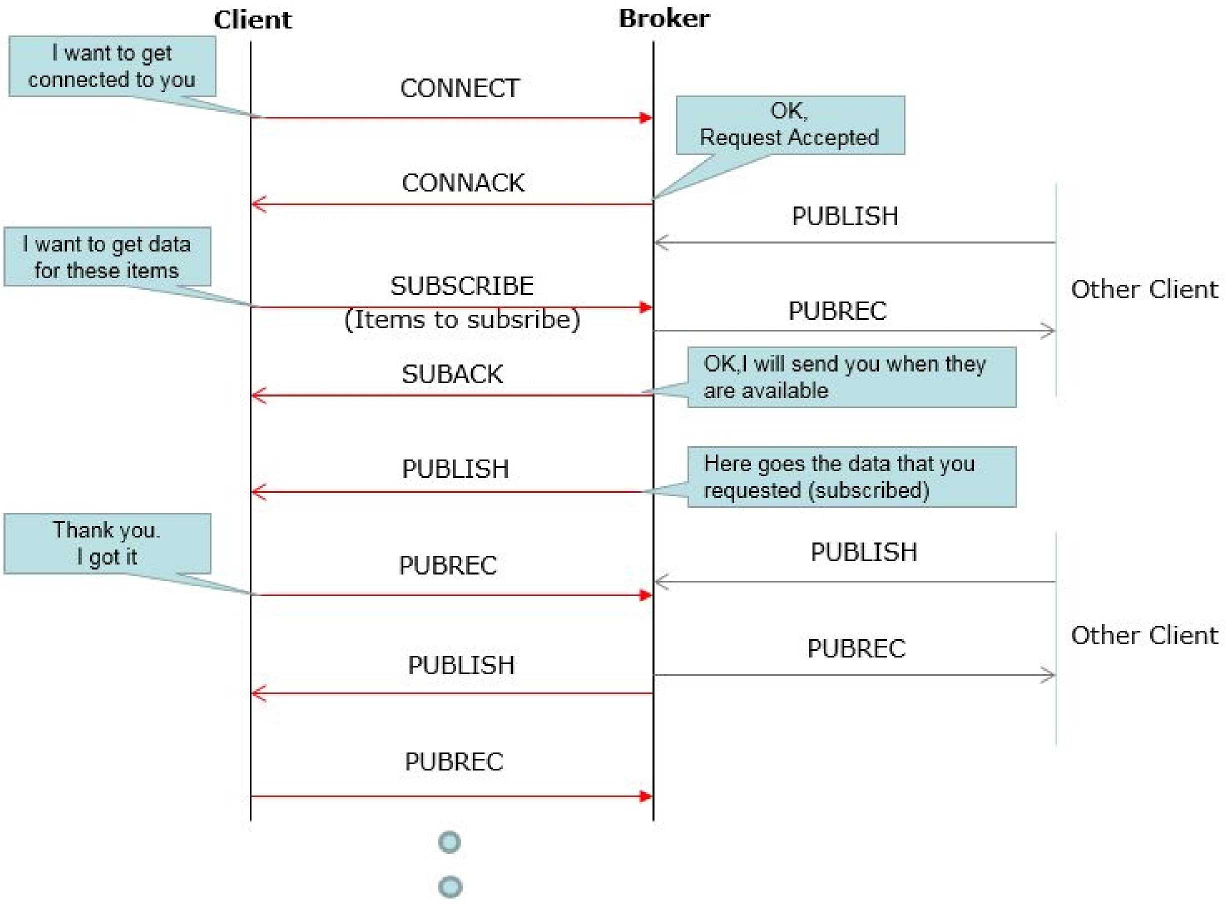


4. Select the **Extensions** tab.

5. Add a new extension by opening the mqttwebsocket.jar.

MQTT messaging workflow

The Protocol specification of MQTT describes the following workflow between the different messages:



This scheme highlights that the client won’t send any message before receiving the confirmation of the server that the client is properly connected.
Therefore in NeoLoad the WebSocket messages below need to wait for the acknowledgement of the server:

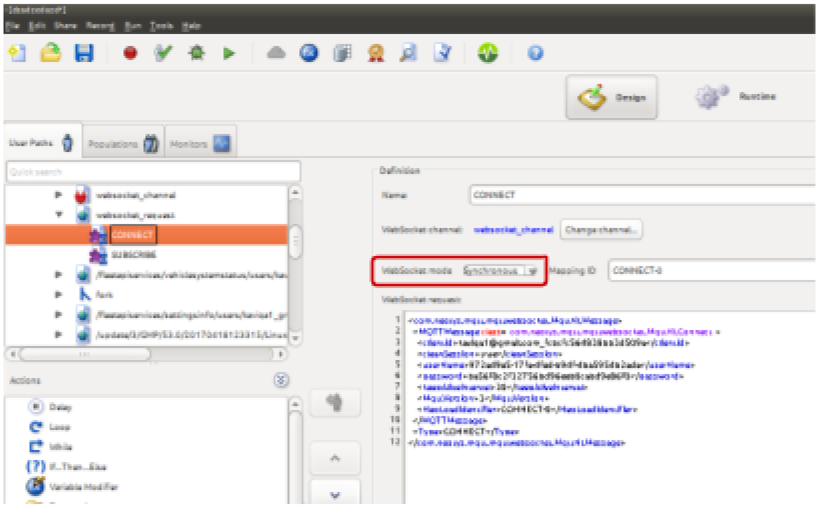
Type of client message	Response required by the server
CONNECT	CONNACK
SUBSCRIBE	SUBACK
UNSUBSCRIBE	UNSUBACK

Using the MQTT framework

The main purpose of the MQTT NeoLoad framework is to position a mapping between each of those WebSocket requests.

The Framework is creating the “message mapping” rule, but you will still need to change the WebSocket message into Synchronous mode.

After recording your scenario (and having the framework parameters applied), it is required to specify to NeoLoad that each CONNECT,SUBSCRIBE and UNSUBSCRIBE requests are synchronous:



Definition

Name: CONNECT

WebSocket channel: websocket_channel Change channel...

WebSocket mode: Synchronous Mapping ID: CONNECT-0

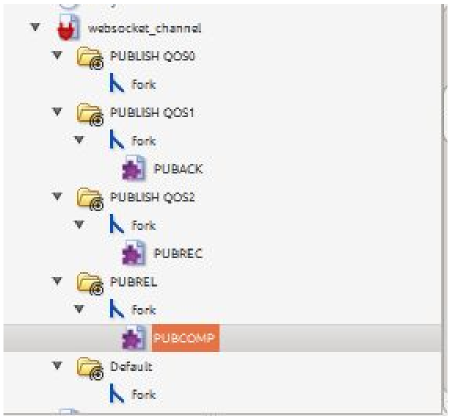
WebSocket request:

```
<com.neotys.mqtt.mqttwebsocket[illegible].MqttMessage>
  <MQTTMessage class="com.neotys.mqtt.mqttwebsocket[illegible].MqttConnect">
    <clientId>[illegible]</clientId>
    <cleanSession>true</cleanSession>
    <userName>[illegible]</userName>
    <password>[illegible]</password>
    <keepAliveInterval>30</keepAliveInterval>
    <MqttVersion>3</MqttVersion>
    <NeoLoadIdentifier>CONNECT-0</NeoLoadIdentifier>
  </MQTTMessage>
  <Type>CONNECT</Type>
</com.neotys.mqtt.mqttwebsocket[illegible].MqttMessage>
```

Specific behavior of the PUBLISH message

According to the QOS, the mqtt protocol specification specifies which behavior is expected depending on the PUBLISH message.

In order to respect the RFC you will need to create this logic within the WebSocket channel.



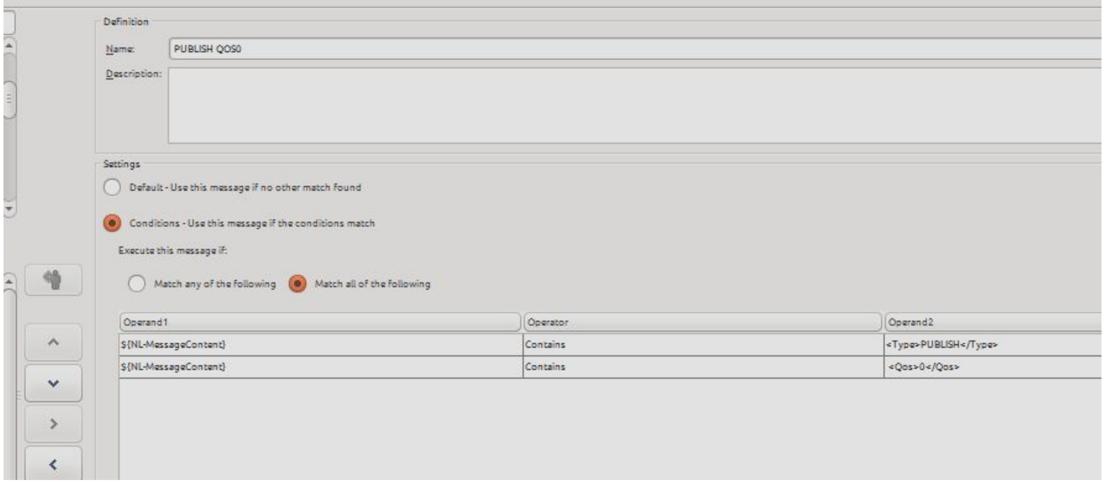
- websocket_channel
  - PUBLISH QOS0
    - fork
  - PUBLISH QOS1
    - fork
      - PUBACK
  - PUBLISH QOS2
    - fork
      - PUBREC
  - PUBREL
    - fork
      - PUBCOMP
  - Default
    - fork

it is required to:

Message received	Message to send
PUBLISH with Qos 0	No messages
PUBLISH with Qos1	PUBACK with the same MessageID
PUBLISH with Qos2	PUBREC with the same MessageID
PUBREL	PUBCOM with the same Message ID

Those push messages need to be defined this way:

1. Right-click on **websocket_channel** and select **Add as a Child a Push Messages**.
2. Create a push message named “PUBLISH QOS0”.
   The rule of this message is: ${NL-MessageContent} contains “<Type>PUBLISH</Type>” and ${NL-MessageContent} contains
   « <Qos>0</Qos>.



Definition

Name: PUBLISH QOS0

Description:

Settings

○ Default - Use this message if no other match found

● Conditions - Use this message if the conditions match

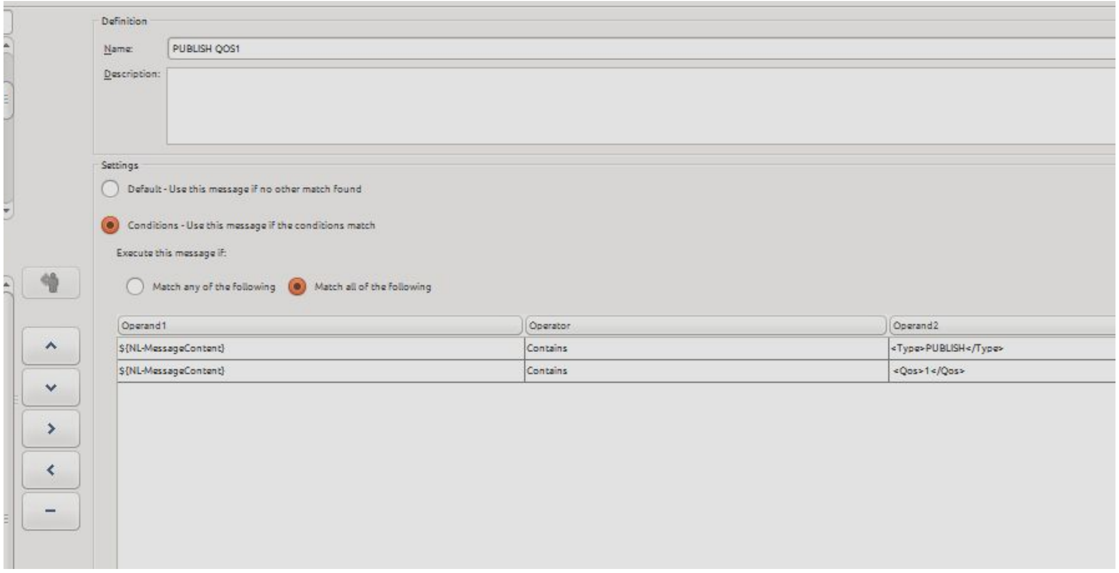
Execute this message if:

○ Match any of the following ● Match all of the following

Operand1	Operator	Operand2
${NL-MessageContent}	Contains	<Type>PUBLISH</Type>
${NL-MessageContent}	Contains	<Qos>0</Qos>

3. Insert as a child in this push message a Fork logical action.

4. Create a push message named “PUBLISH QOS1”.
   The rule of this message is: ${NL-MessageContent} contains “<Type>PUBLISH</Type>” and ${NL-MessageContent} contains
   « <Qos>1</Qos>.



Definition

Name: PUBLISH QOS1

Description:

Settings

○ Default - Use this message if no other match found

● Conditions - Use this message if the conditions match

Execute this message if:

○ Match any of the following ● Match all of the following

Operand1	Operator	Operand2
${NL-MessageContent}	Contains	<Type>PUBLISH</Type>
${NL-MessageContent}	Contains	<Qos>1</Qos>

5. Insert as a child in this push message a Fork logical action.

6. Drag and drop one of the PUBACK requests captured by NeoLoad during the recording (with the following logo).

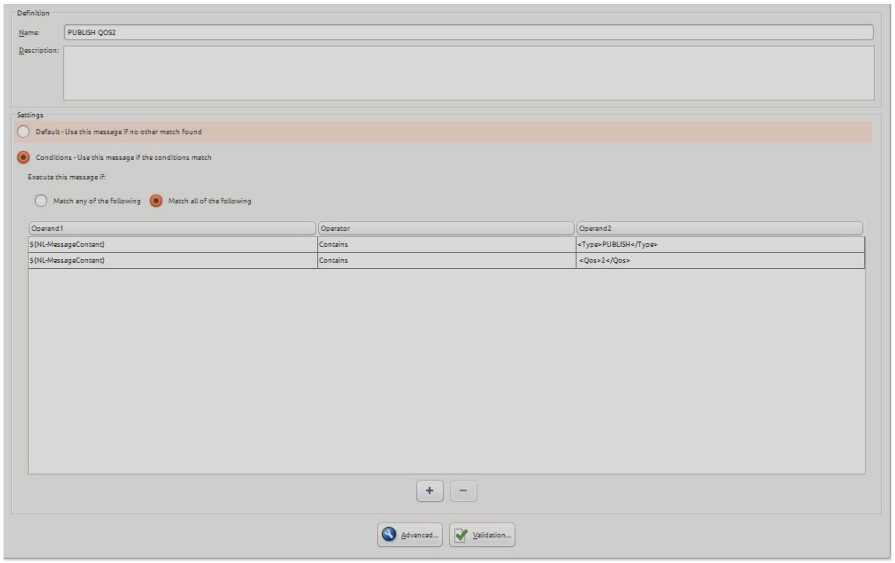
7. Replace the content of the message with:

```
<com.neotys.mqtt.mqttwebsocket.MqttNLMessage>
  <MQTTMessage
class="com.neotys.mqtt.mqttwebsocket.MqttNLPuback">
    <MessageID>${MessageID}</MessageID>
  </MQTTMessage>
  <Type>PUBACK</Type>
</com.neotys.mqtt.mqttwebsocket.MqttNLMessage>
```

8. Remove all PUBACK request from the UserPath

9. Create a push message named “PUBLISH QOS2”.

   The rule of this message is: ${NL-MessageContent} contains “<Type>PUBLISH</Type>” and ${NL-MessageContent} contains « <Qos>2</Qos>



Definition

Name: PUBLISH QOS2

Description:

Settings

○ Default - Use this message if no other match found

● Conditions - Use this message if the conditions match

Execute this message if:

○ Match any of the following ● Match all of the following

Operand1	Operator	Operand2
${NL-MessageContent}	Contains	<Type>PUBLISH</Type>
${NL-MessageContent}	Contains	<Qos>2</Qos>

10. Insert as a child in this push message a Fork logical action.

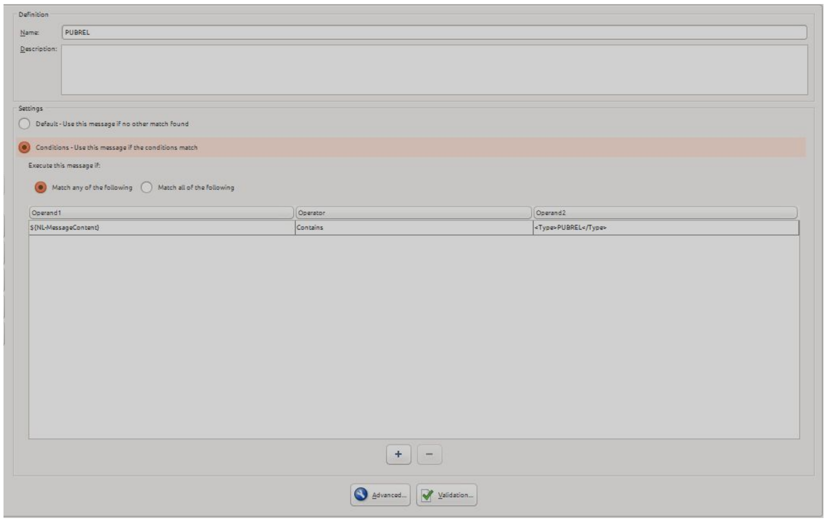
11. Drag and Drop one of the PUBREC requests captured by NeoLoad during the recording

```
<com.neotys.mqtt.mqttwebsocket.MqttNLMessage>
<MQTTMessage
class="com.neotys.mqtt.mqttwebsocket.MqttNLPubRec">
```

```
    <MessageID>${MessageID}</MessageID>
  </MQTTMessage>
  <Type>PUBREC</Type>
</com.neotys.mqtt.mqttwebsocket.MqttNLMessage
```

12. Remove all PUBREC request from the UserPath

13. Create a push message named “PUBREL”.
    The rule of this message is: ${NL-MessageContent} contains “<Type>PUBREL</Type>”.



Definition

Name: PUBREL

Description:

Settings

○ Default - Use this message if no other match found

● Conditions - Use this message if the conditions match

Execute this message if:

● Match any of the following ○ Match all of the following

Operand1	Operator	Operand2
${NL-MessageContent}	Contains	<Type>PUBREL</Type>

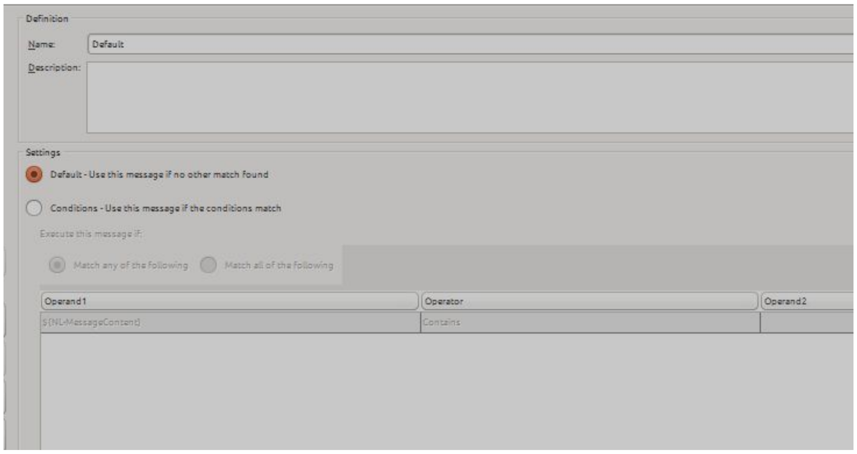
14. Insert as a child in this push message a Fork logical action.

15. Drag and drop one of the PUBCOMP requests captured by NeoLoad during the recording

```
<com.neotys.mqtt.mqttwebsocket.MqttNLMessage>
<MQTTMessage class="com.neotys.mqtt.mqttwebsocket.MqttNLPubComp">
<MessageID>${MessageID}</MessageID>
</MQTTMessage>
<Type>PUBCOMP</Type>
</com.neotys.mqtt.mqttwebsocket.MqttNLMessage>
```

16. Remove all PUBCOMP request from the UserPath

17. Create a push message named ”Default”.
    The rule of this message: “Default message if no other rules match”.



Definition

Name: Default

Description:

Settings

● Default - Use this message if no other match found

○ Conditions - Use this message if the conditions match

Execute this message if:

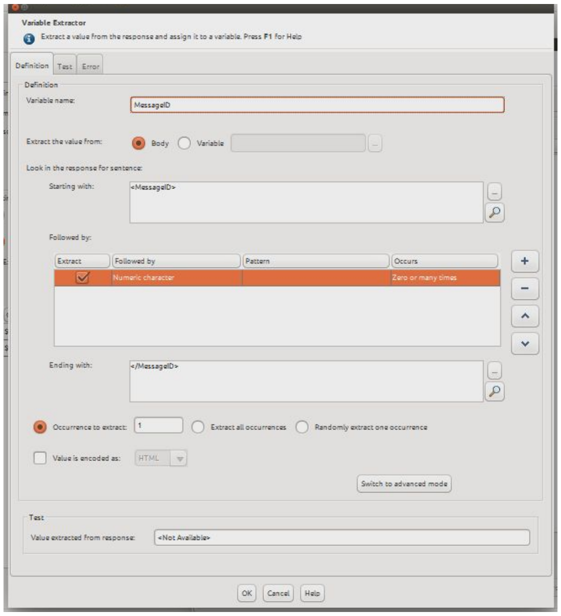
● Match any of the following ○ Match all of the following

Operand1	Operator	Operand2
${NL-MessageContent}	Contains	

18. Insert as a child in this push message a Fork logical action.

In order to send the message with the right reference of the message ID, you need to create a variable extractor on the specific push message (we are not going to define it directly on the websocket channel to avoid applying this extractor on each WebSocket messages).

The following extractor would have to be defined on the push channel:
- PUBLISH Qos1
- PUBLISH Qos2
- PUBREL



Variable Extractor

Extract a value from the response and assign it to a variable. Press F1 for Help

Definition | Test | Error

Definition

Variable name: MessageID

Extract the value from: ● Body ○ Variable

Look in the response for sentence:

Starting with: <MessageID>

Followed by:

Extract	Followed by	Pattern	Occurs
☑	Numeric character		Zero or many times

Ending with: </MessageID>

● Occurrence to extract: 1 ○ Extract all occurrences ○ Randomly extract one occurrence

□ Value is encoded as: HTML

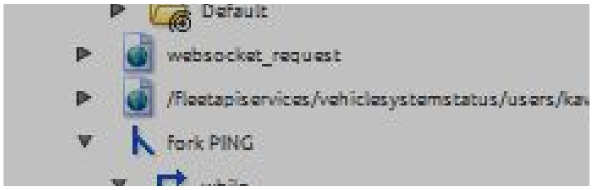
Switch to advanced mode

Test

Value extracted from response: <Not Available>

Ping mechanism required to maintain the WebSocket session

In order to maintain the session with the MQTT broker, it is required to send regular PINGREQ requests to the broker.
In order to make the User Path easier to maintain, you need to create the following polling mechanism within the User Path:



The value of the delay is the value of the keepalive parameter send in the CONNECT message.

(PINGREQ also needs the be a synchronous WebSocket message because it needs to wait from the acknowledgement of the server: PINGRESP).

Once the fork created in the User Path, you need to remove each PINGREQ request from the project.