SystemCORP ENERGY

JSON

eNode Configuration Manual

This PDF Document contains internal hyperlinks for ease of navigation.
For example, click on any item listed in the Table of Contents to go to that page.

Documentation Control

Author:	Nicholas Rixson
Version:	1.01
Version History:	1.00 – Initial release 1.01 – Add subscription models, communication transport security, logs and CSV.
Creation Date:	21 August 2018
Last Version Date:	23 January 2020
Product Reference:	**197-1400**
Document Status:	Released

Table of Contents

Table of Figures

List of Tables

1 Introduction

This document describes how to use the JSON eNode Designer Module to configure SystemCORP's JSON ADH Application within the eNode Designer.

This allows the eNode Designer user to configure communication of JavaScript Object Notation (JSON) formatted messages containing information from the ADH, as well as device information and static data. This format describes a very simple, structured way to communicate and is widely used in internet-based technologies.

This is combined with a transport mechanism to deliver the message to a destination. Currently only MQTT is supported – this is widely used in Internet of Things (IoT) field to transfer data and is very often combined with JSON. Combining these makes the module a great for interfacing with remote database servers, cloud applications and dashboards.

1.1 Document Reference

[1] Document Title: eNode Designer User Manual: 197-0100
Revision: Version 1.00 or higher

1.2 List of Abbreviations

ADH	= Application Data Hub
JSON	= JavaScript Object Notation
IP	= Internet Protocol
MQTT	= Message Queueing Telemetry Transport
TCP	= Transmission Control Protocol

2 Overview

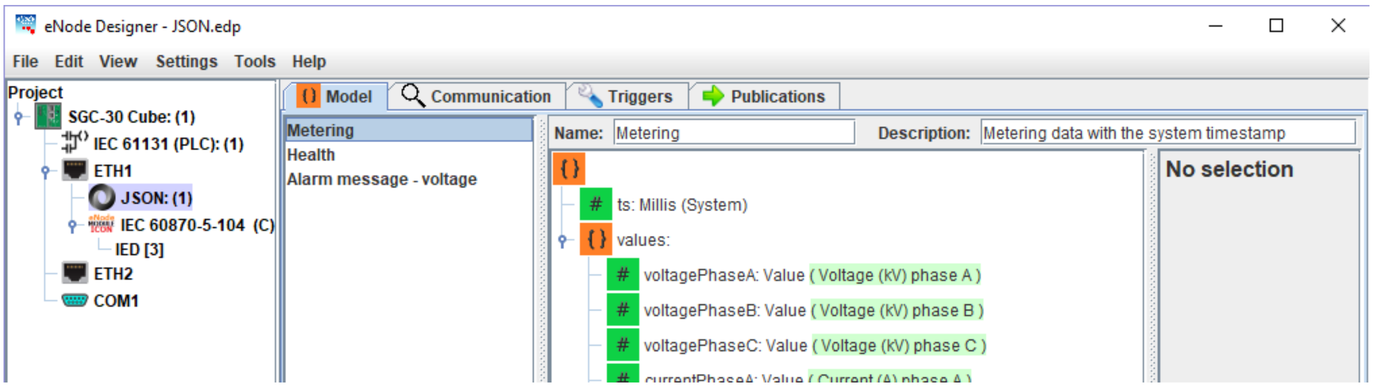
The JSON eNode Designer module can be used to configure publishing JSON messages over MQTT on any supported device. There is support for **sending** information from ADH and **receiving** information or commands using JSON and CSV messages.

Many JSON models, communication destinations, trigger options and publications can be defined in the single JSON instance. It is highly flexible to send any message to any destination, in a many-to-many relationship. For example: one model can go to multiple destinations, and many models can be sent to the same destination without duplicating any configuration.

2.1 General Screen Description

The main module window is split in to five categories:

- Model – to configure the JSON models used in the messages
- Communication – to configure destinations to send messages, and any associated parameters.
- Triggers – to define when to generate a message
- Logs – to define when to store messages in non-volatile storage
- Publications – A selection of a combination of model, communication and trigger defining an actual message publication.



Name: Metering
Description: Metering data with the system timestamp

```
{ }
  # ts: Millis (System)
  { } values:
    # voltagePhaseA: Value ( Voltage (kV) phase A )
    # voltagePhaseB: Value ( Voltage (kV) phase B )
    # voltagePhaseC: Value ( Voltage (kV) phase C )
    # currentPhaseA: Value ( Current (A) phase A )
```

Figure 2-1 - General screen description

3 Configuration Guide

3.1 Adding the Module in eNode Designer

The JSON application can be added to *Ethernet* ports only. **Right click** the desired port, use “Add ADH Application” and select JSON.

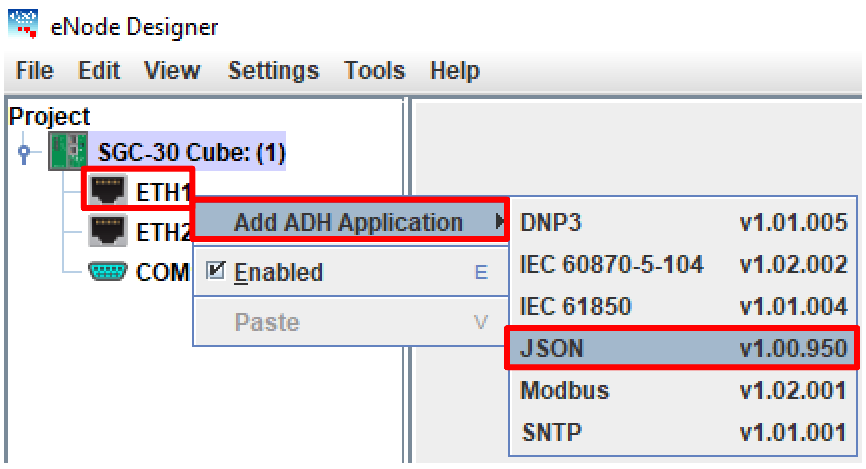


Figure 3-1 - Add the module in eNode Designer

This will add an empty default module:

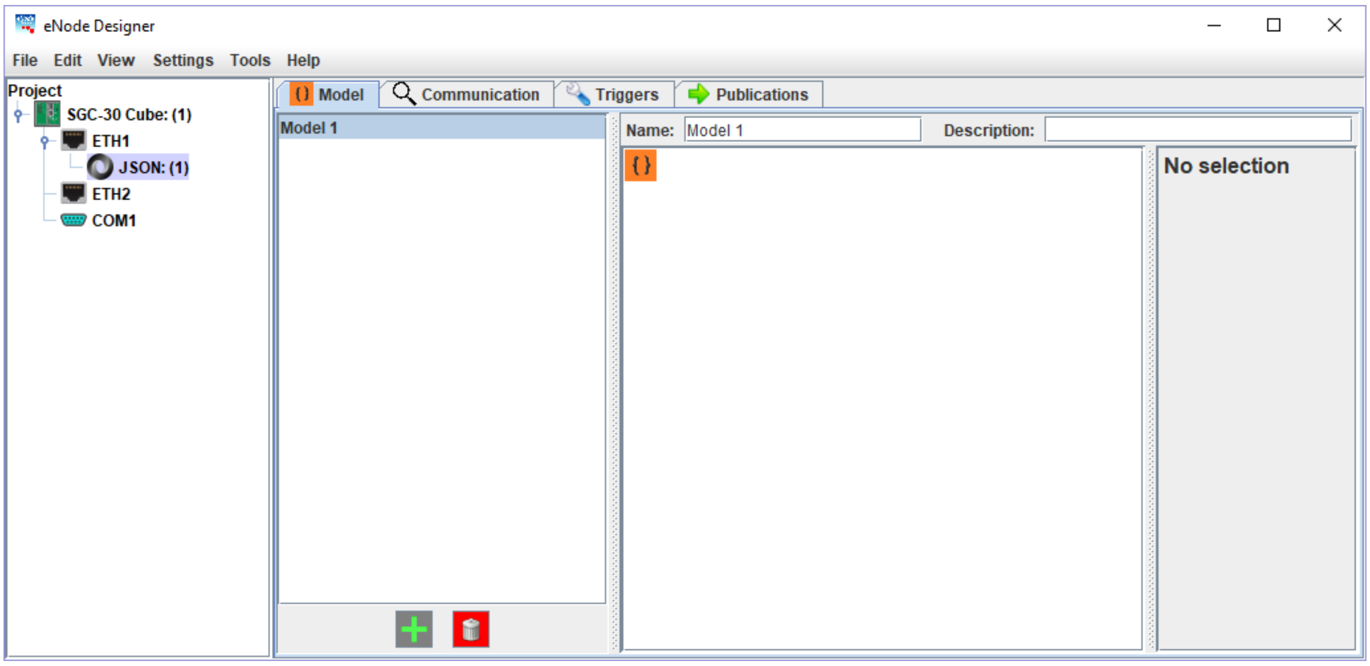


Figure 3-2 - Default module appearance when added

Each configuration section can support many sub-items. For example, the model section can configure many JSON models. This is handled in the same way for each section, as described below.

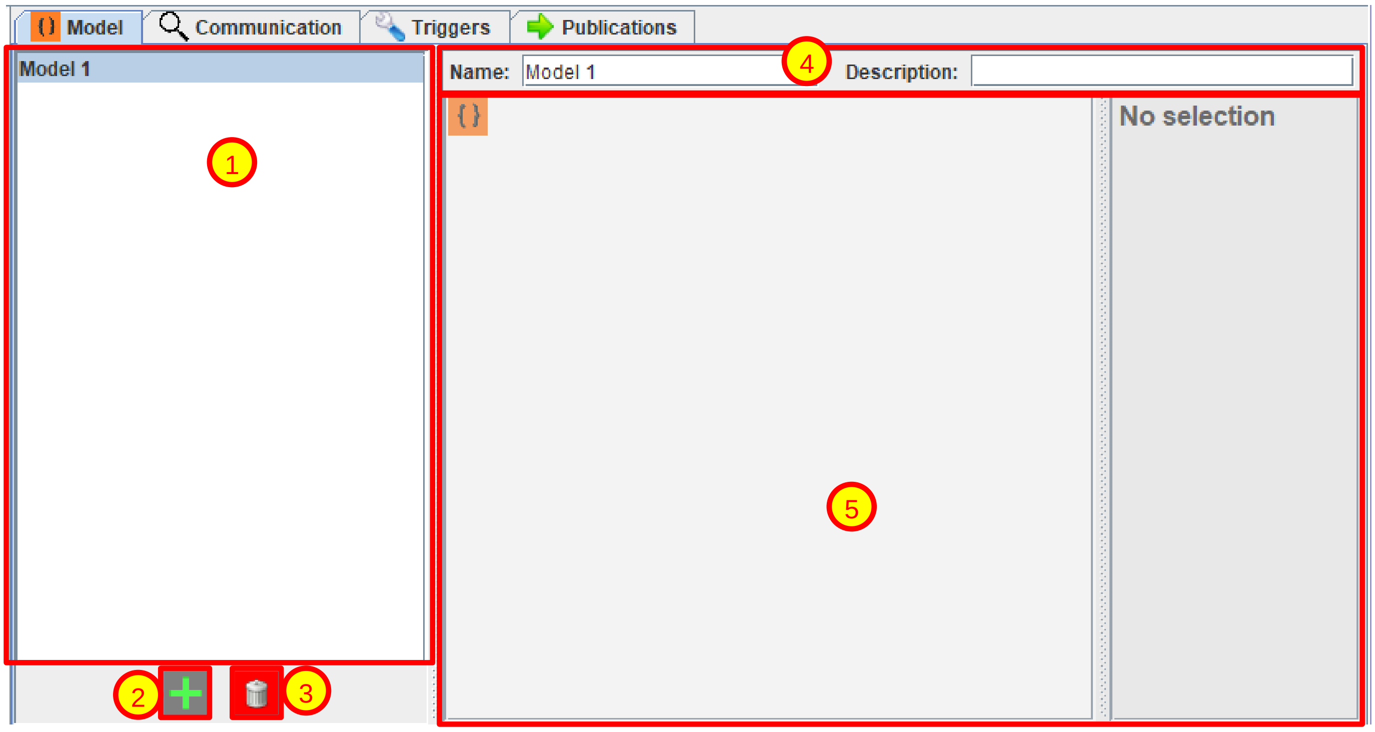


Figure 3-3 - Detailed Screen Description

1 **Selection area** – click on the item you want to select. This will redraw the right-hand side of the screen: area (4) and (5).

2 **Add new** – Adds a new default item to the list in (1) and selects it.

3 **Delete selected item** – Removes the selected item from (1) and removes all references to it. If the item is currently in use, a message is shown to the user to confirm the removal.

4 **Item name and description** – Shows the currently selected item (1)'s name and description. These can be modified and the name in (1) will automatically update to reflect any changes in the name.

5 **Item configuration area** – The selected item can be configured here. This will appear different depending on the section – model, communication, triggers or publication.

3.2 Data Model

The data model defines the format of the data message which will be generated. JSON itself describes a way of structuring data, and this section allows you to configure the structure according to the rules required in JSON. The data model has also been extended to support a comma-separated values format, as required by some customers.

Usually most JSON messages start with an object and contain data. It is however possible to generate JSON messages which are not contained in an object.

JSON defines only a few types:

- Object – contains information in key-value pairs. The key is a string, the value may be any JSON type.
- Array – contains any number of JSON values in an ordered list
- String – Sequence of characters
- Number – May represent floating point or integers
- true – A Boolean truth value
- false – A Boolean false value
- null – Represents the absence of a value

To be more compatible with the rest of eNode Designer and the ADH, the “true” and “false” values are combined into a “Boolean” type in the JSON model builder. This allows simpler data point mapping.

This guide will show you how to build a model step-by-step.

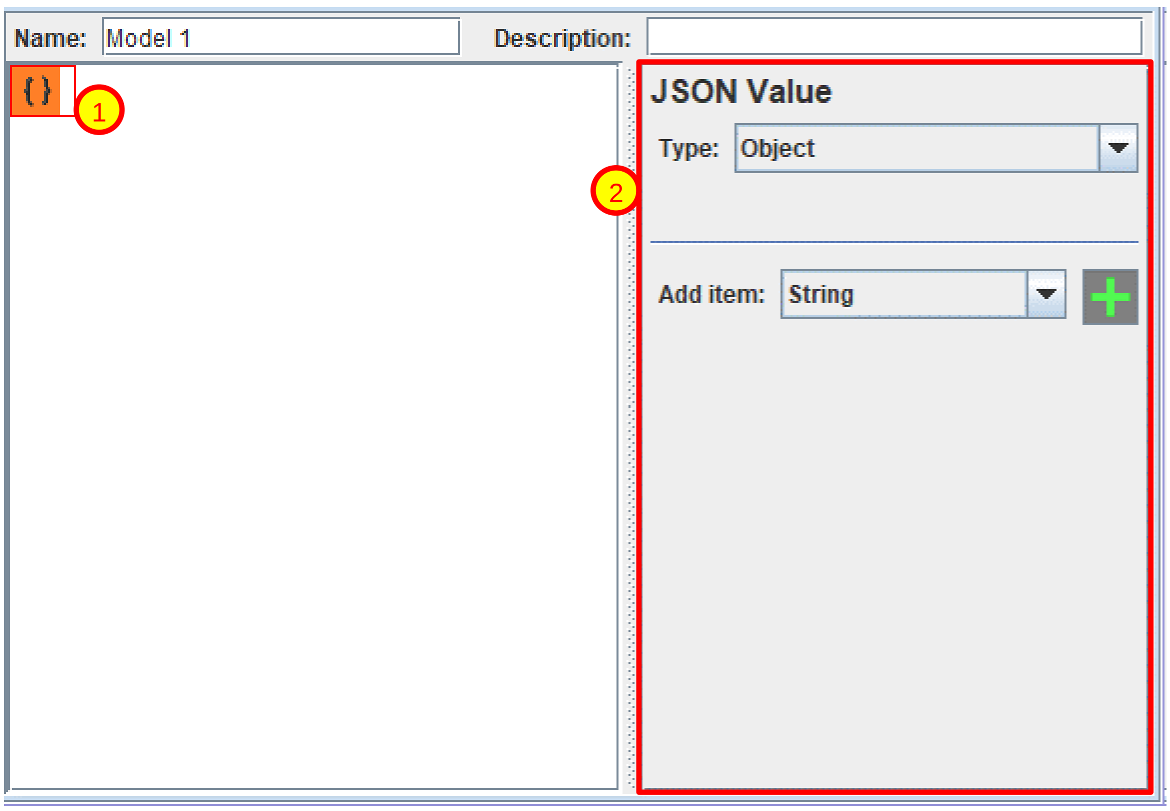
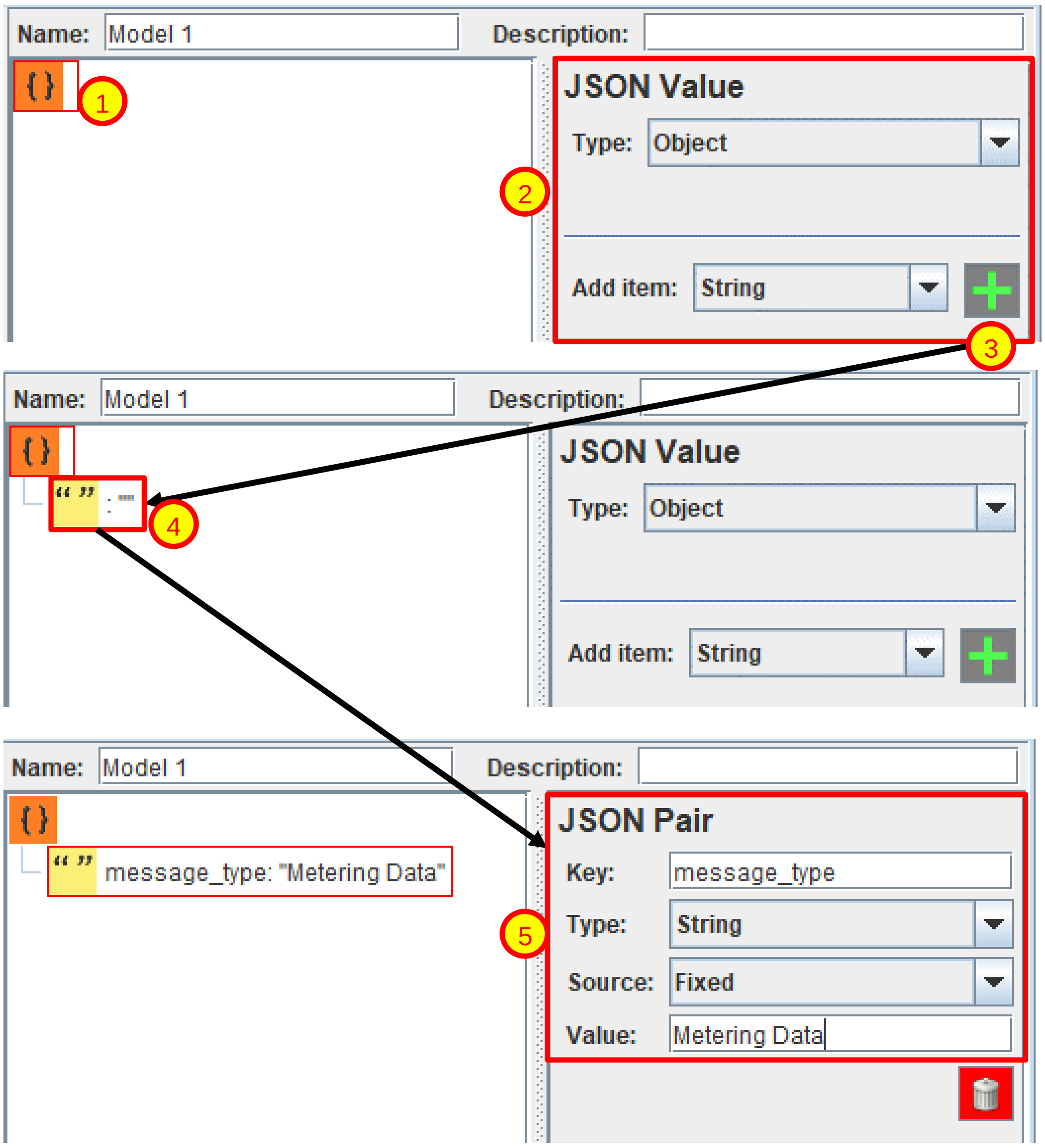


Figure 3-4 - Select an object

1 **Select a model item** – This will show a red box around the selected item and allows its properties to be configured in area (2).
2 **Properties of selected item** – Shows the current values of the item. Here you can select its JSON data type, configure the “key” (if present), delete the item (if legal), and add sub-items (for Objects and Arrays).

Adding “items” or “children” to a JSON type means it adds an element inside the structure. Only Objects and Arrays may contain items. Object children contain a “key” describing what kind of value it is. Arrays are just a sequence of data – it does not have a “key” to describe the meaning of each item. Usually arrays are used for items of the same type, and for fixed pre-defined structures.

An example of adding children to an object is shown below.



Name: Model 1
Description:
JSON Value
Type: Object
Add item: String

Name: Model 1
Description:
JSON Value
Type: Object
Add item: String

Name: Model 1
Description:
message_type: "Metering Data"
JSON Pair
Key: message_type
Type: String
Source: Fixed
Value: Metering Data

Figure 3-5 - Adding JSON model children

1 **Select the model item by mouse click** – This will show a red box around the selected item and allows its properties to be configured in area (2).

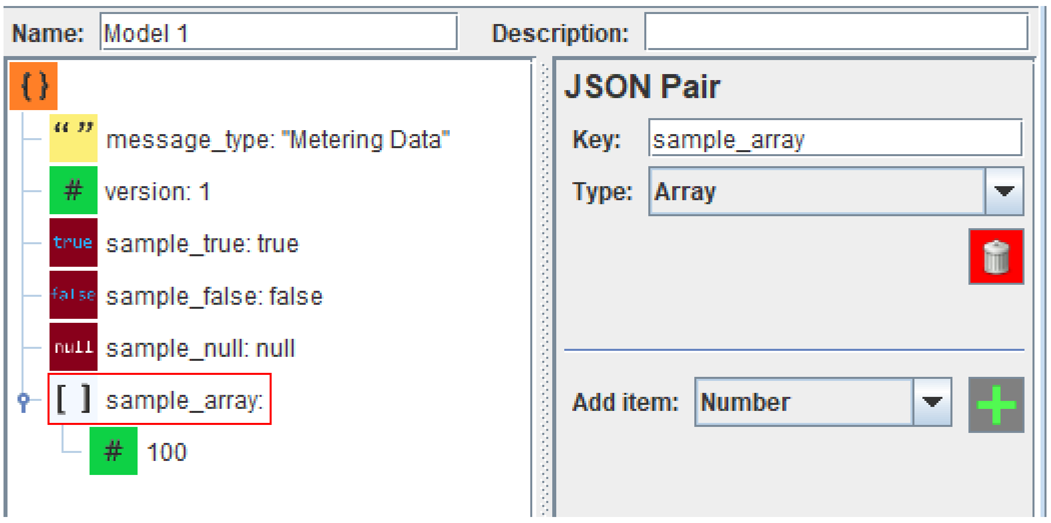
2 **Define child type** – Define the type of the new item with the combo box.

3 **Add child** – Click the “+” button to add a new child.

4 **(Optional) Select the child by mouse click** – Shows its properties, just like (1).

5 **(Optional) Modify properties** – Modifying the data will live-update the tree model.

This process can be continued to build the JSON model in a tree-like manner. An example of the result is shown below.


Name: Model 1
Description:
JSON Pair
Key: sample_array
Type: Array
Add item: Number

```
{ }
  message_type: "Metering Data"
  version: 1
  sample_true: true
  sample_false: false
  sample_null: null
  sample_array:
    100
```

Figure 3-6 - Continuing to build to JSON model

The JSON message payload then looks like:

```
{
  "message_type": "Metering Data",
  "version": 1.0,
  "sample_true": true,
  "sample_false": false,
  "sample_null": null,
  "sample_array": [
    100.0
  ]
}
```

A CSV model is created by changing the root node to type "CSV". Then it will create the comma-separated values format instead of creating a JSON format model. CSV is a simplified model, which simply has consecutive values separated by commas. For example, the model below is used to produce a CSV payload:

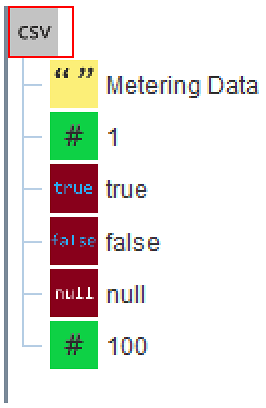

```
CSV
  Metering Data
  1
  true
  false
  null
  100
```

Figure 3-7 - CSV basic example

The CSV message payload then looks like:

```
Metering Data,1,true,false,null,100
```

Arrays and Objects can be used to introduce a 'line' of the CSV. The object keys can be useful for the **eNode Designer user**, to indicate the meaning of each part of the model, however the object keys are not included in the payload; only the value parts are placed into the output CSV string. An example is shown below.

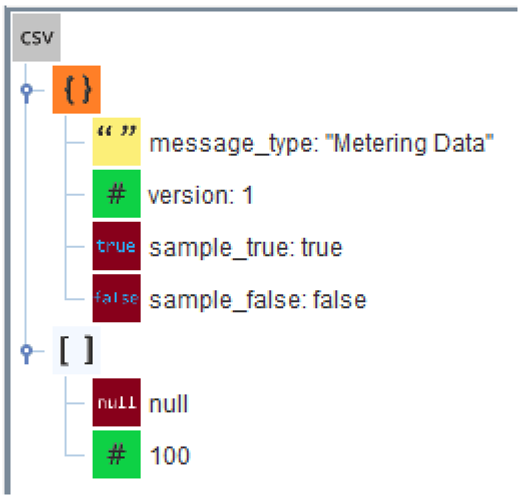


Figure 3-8 - Multi-line CSV model

```
Metering Data,1,true,false
null,100
```

Here we see how objects and arrays introduce lines, and that object keys are not included in the output CSV.

3.2.1 *Changing the Model Structure*

One way the model structure can be modified by adding and removing children as described in the previous section. This allows new nodes to enter the system and to be removed from it. However, changing an existing model is very important, as the initial design is not always final.

All objects in the tree can use the *Drag and Drop* method to move JSON model objects to different locations. To do this, click and hold on an object in the tree, move the mouse to the desired drop location, and release the mouse. This will move the model nodes to the new location.

Note: only valid “drop” locations are allowed. This means that nodes may only be dropped into Objects and Arrays, and the item may not be dropped on a location which is “inside” itself.

To move the item to be the last value in a container-type, you can drop it directly on to the container itself.

3.2.2 *Source of a JSON Value*

Every JSON object can be statically defined to contain a fixed “literal” value. This is very useful for structuring the data and defining meta-data about the contents of the message.

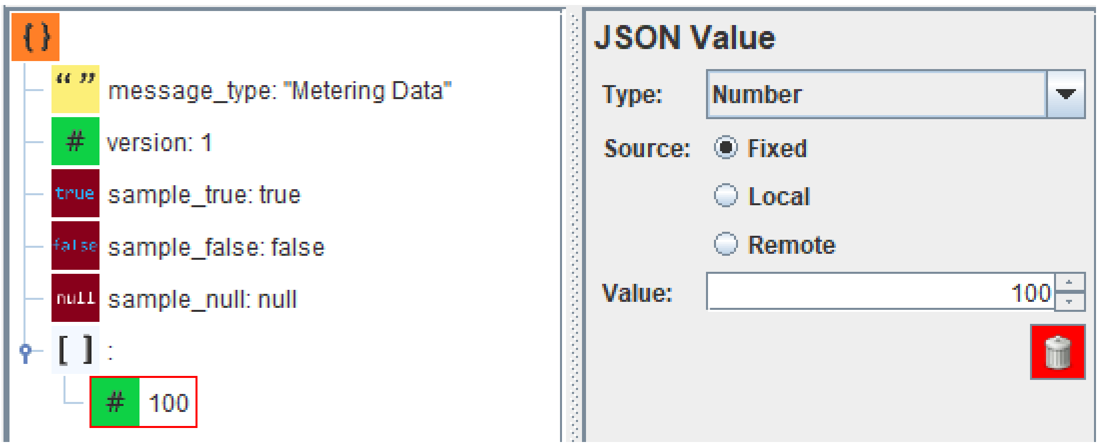
The contents of the value-containing types can also be taken from the system and ADH data points at run-time. The dynamic value types supported are:

- String contents
- Number contents
- Boolean (true and false) value

The values which do not support dynamic binding are:

- Objects
- Arrays
- null
- Object’s children “key” values – the key value is always fixed.
- Value type – For example, a configured string is always a string, and will never be a “Number” or “null”.

Types which support dynamic binding have the option displayed in its property configuration area, under the “Source” of the value – as shown below.



{ }
message_type: "Metering Data"
version: 1
sample_true: true
sample_false: false
sample_null: null
[ ] :
100

JSON Value
Type: Number
Source: [✓ Fixed] Local Remote
Value: 100

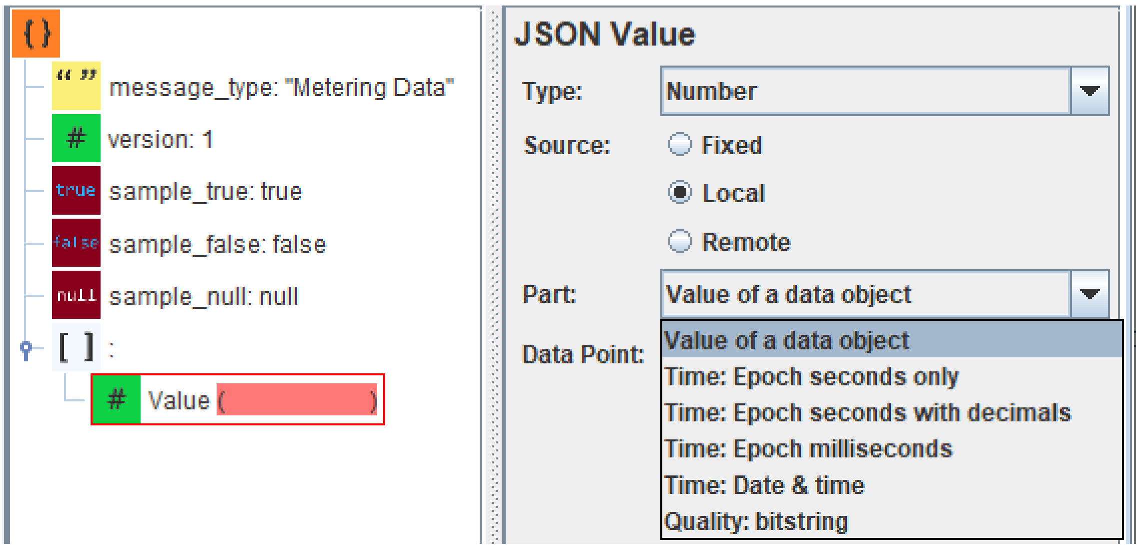
Figure 3-9 - Choosing a data source

Source	Usage
Fixed	A fixed, “literal”, unchangeable value which is always present in the model.
Local	The value comes from a known source on the local device. This means it comes either from the ADH data points that already exist, or from the system (e.g. system time). These must be used with “Publisher” communication options.
Remote	The value comes from the other end of the connection – it is incoming data to be processed. These must be used with “Receiver” or “Subscriber” communication options (not publishing).

Table 3-1 - Value source options

In Fixed mode, a constant value in the model can be defined. Every JSON object can be statically defined to contain a fixed “literal” value. This is very useful for structuring the data and defining meta-data about the contents of the message.

In Local source mode, the following options are available.



{ }
message_type: "Metering Data"
version: 1
sample_true: true
sample_false: false
sample_null: null
[ ] :
Value ( )

JSON Value
Type: Number
Source: Fixed [✓ Local] Remote
Part: Value of a data object
Data Point:
Value of a data object
Time: Epoch seconds only
Time: Epoch seconds with decimals
Time: Epoch milliseconds
Time: Date & time
Quality: bitstring

Figure 3-10 - Local source options

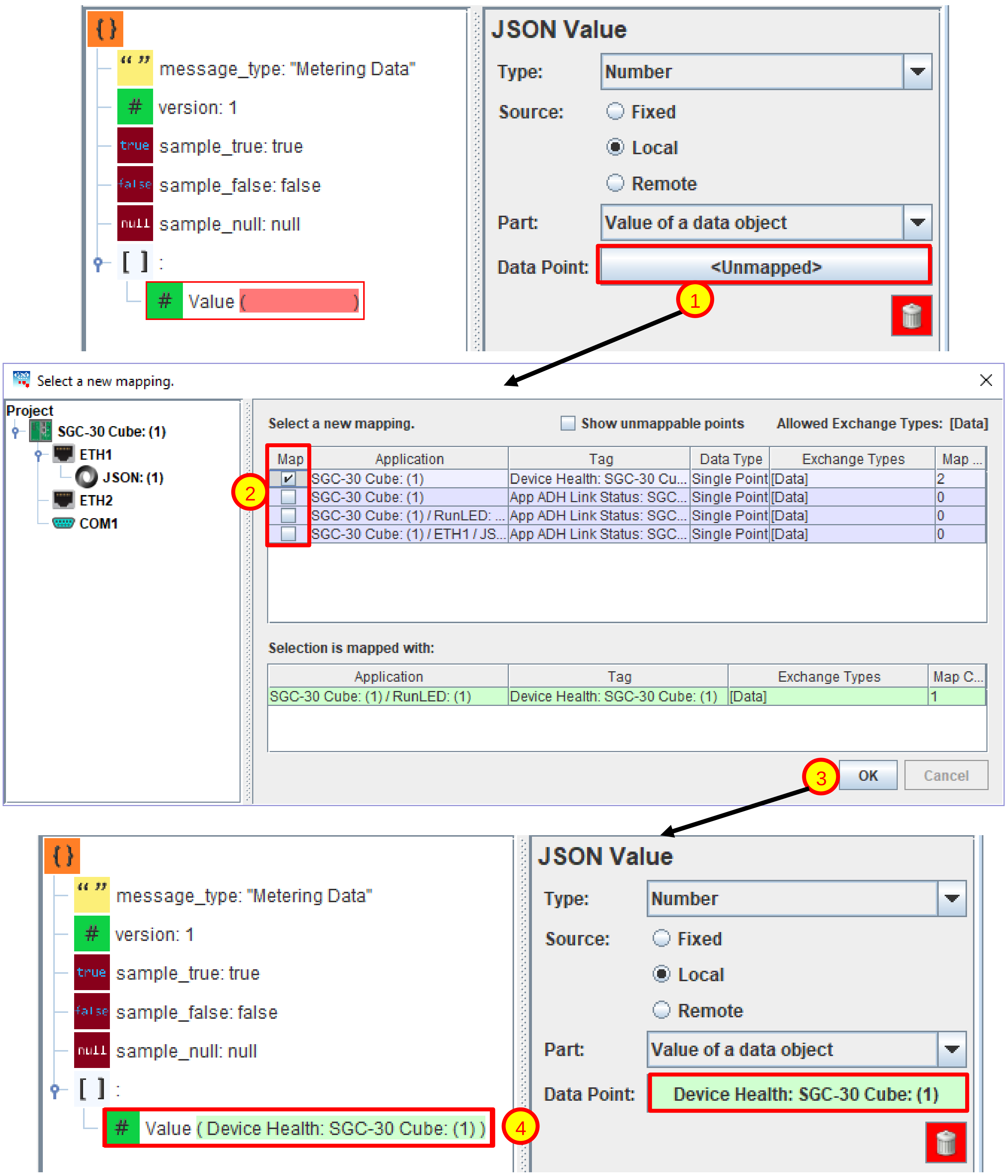
Local Source	Usage
Value of a data object	The value comes directly from the value part of a data object, converted into JSON format. Example: 1000 (from a data object whose value is 1000 at runtime)
Time: Epoch seconds only	The number of seconds since the "Epoch" – 1 Jan 1970 00:00:00 UTC, rounded down to the nearest integer. This is also known as the "Unix Timestamp" – and is very common in computer systems. Example: 1534911044
Time: Epoch seconds with decimals	The number of seconds since Epoch with trailing decimals. Example: 1534911044.183005
Time: Epoch milliseconds	The number of milliseconds since Epoch, as an integer Example: 1534911044183.
Time: Date & time	The date and time expressed in ISO-8601 long form. Example string (yyyy-MM-dd**T**hh:mm:ss.nnnnnn**Z**) : "2018-08-22T04:10:44.183005Z" Example number: 20180822041044183
Quality: bitstring	The quality flags of a data point as a bitwise OR-mask of ADH quality flags. Example string (0x plus 6 hexadecimal digits): "0x000001" Example number: 1

Table 3-2 - Local source options

The meaning of the quality flags and their bitmask value is defined in section *4.1*.

3.2.2.1 *Local Source: Value of a Data Object*

Selecting the "Value of a Data Object" as the source of a value shows the following screen. A data point must then be chosen from which to take the value. This is achieved by pressing the data point button: initially saying it is "<Unmapped>". This will raise the standard eNode Designer mapping selection window. Select a point to be mapped.



message_type: "Metering Data"
version: 1
sample_true: true
sample_false: false
sample_null: null
[ ] :
Value ( )

JSON Value

Type: Number

Source: Fixed [✓ Local] Remote

Part: Value of a data object

Data Point: <Unmapped>

Select a new mapping.

Project
- SGC-30 Cube: (1)
  - ETH1
    - JSON: (1)
  - ETH2
  - COM1

Select a new mapping. Show unmappable points Allowed Exchange Types: [Data]

Map	Application	Tag	Data Type	Exchange Types	Map ...
✓	SGC-30 Cube: (1)	Device Health: SGC-30 Cu...	Single Point	[Data]	2
	SGC-30 Cube: (1)	App ADH Link Status: SGC...	Single Point	[Data]	0
	SGC-30 Cube: (1) / RunLED: ...	App ADH Link Status: SGC...	Single Point	[Data]	0
	SGC-30 Cube: (1) / ETH1 / JS...	App ADH Link Status: SGC...	Single Point	[Data]	0

Selection is mapped with:

Application	Tag	Exchange Types	Map C...
SGC-30 Cube: (1) / RunLED: (1)	Device Health: SGC-30 Cube: (1)	[Data]	1

message_type: "Metering Data"
version: 1
sample_true: true
sample_false: false
sample_null: null
[ ] :
Value (Device Health: SGC-30 Cube: (1))

JSON Value

Type: Number

Source: Fixed [✓ Local] Remote

Part: Value of a data object

Data Point: Device Health: SGC-30 Cube: (1)

1 **Click the remap data point button** – This will bring up a mapping window to select the data point source (2).

2 **Select a data point** – At run-time, the JSON model will be populated with the “value” part of the data object.

3 **Click OK** – to accept the mapping.

4 **Model is updated** – you can see the mapping has come into effect in the JSON tree and mapping button.

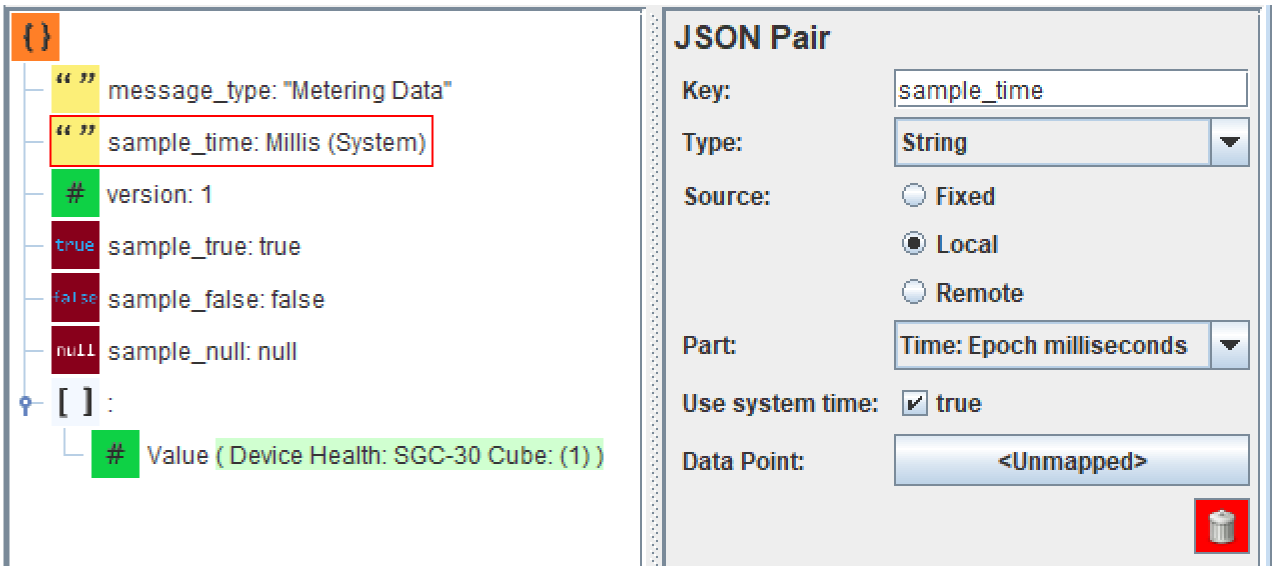
3.2.2.2 *Local Source: Timestamp*

Timestamp types can be mapped in two ways:

- To take the timestamp of the last update of a data point. Mapping works the same as mapping a data point value, explained in section *3.2.2.1*.
- System time. Each time a message is generated, the device's current system time will be placed in the JSON model.

Only one of these choices may be used at a time. If you map to a data point, it will automatically remove the "Use system time" flag; selecting "Use system time" will un-map the current data point.

The formats are shown in Table 3-2 - Local source options.



{ }
"" message_type: "Metering Data"
"" sample_time: Millis (System)
# version: 1
true sample_true: true
false sample_false: false
null sample_null: null
[ ] :
Value (Device Health: SGC-30 Cube: (1))

JSON Pair

Key: sample_time
Type: String
Source: ( ) Fixed [✓ Local] ( ) Remote
Part: Time: Epoch milliseconds
Use system time: [✓] true
Data Point: <Unmapped>

Figure 3-11 - Timestamp source selection screen

3.2.2.3 *Local Source: Quality*

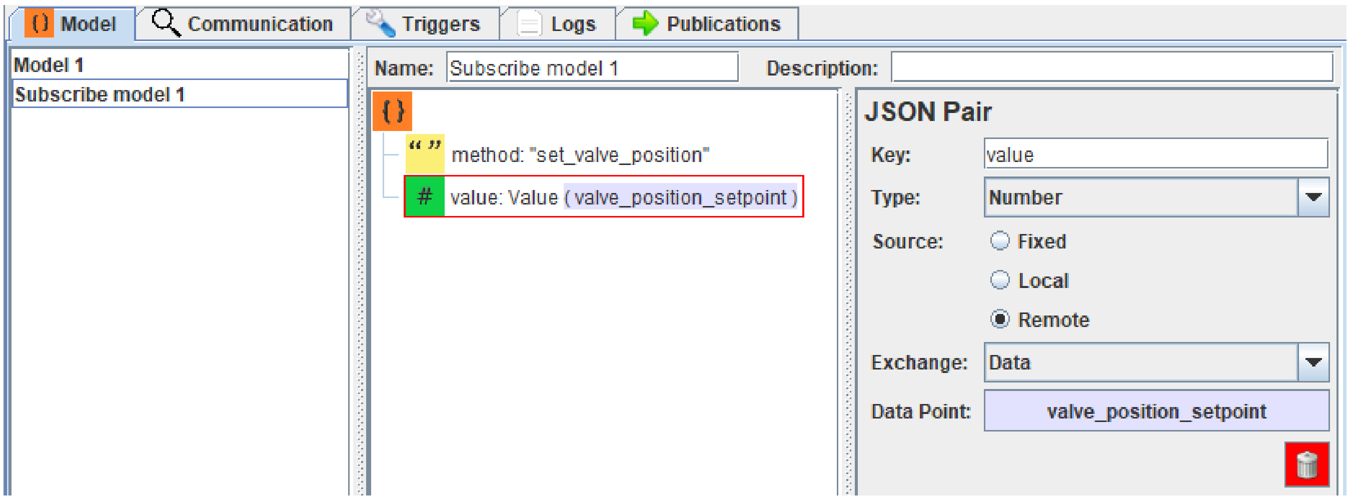
Mapping the quality flags of a data point works the same as mapping a data point value, explained in section *3.2.2.1*.

The formats are shown in Table 3-2 - Local source options.

3.2.2.4 *Remote Source: Data*

Information from the remote side can use used as basic data in eNode Designer and the ADH, meaning that the new value is taken from the value in the message.

The JSON application can be configured to process messages of any pre-defined model, with an example below. To match, all fixed values must match the incoming packet, and the JSON data types must match the update point.



Model 1
Subscribe model 1

Name: Subscribe model 1
Description:

{ }
method: "set_valve_position"
value: Value (valve_position_setpoint)

JSON Pair
Key: value
Type: Number
Source: Fixed Local [✓ Remote]
Exchange: Data
Data Point: valve_position_setpoint

Figure 3-12 - Remote source data model

If the JSON application receives the message matching the format prescribed:

```
{
  "method": "set_valve_position",
  "value": <number value here>
}
```

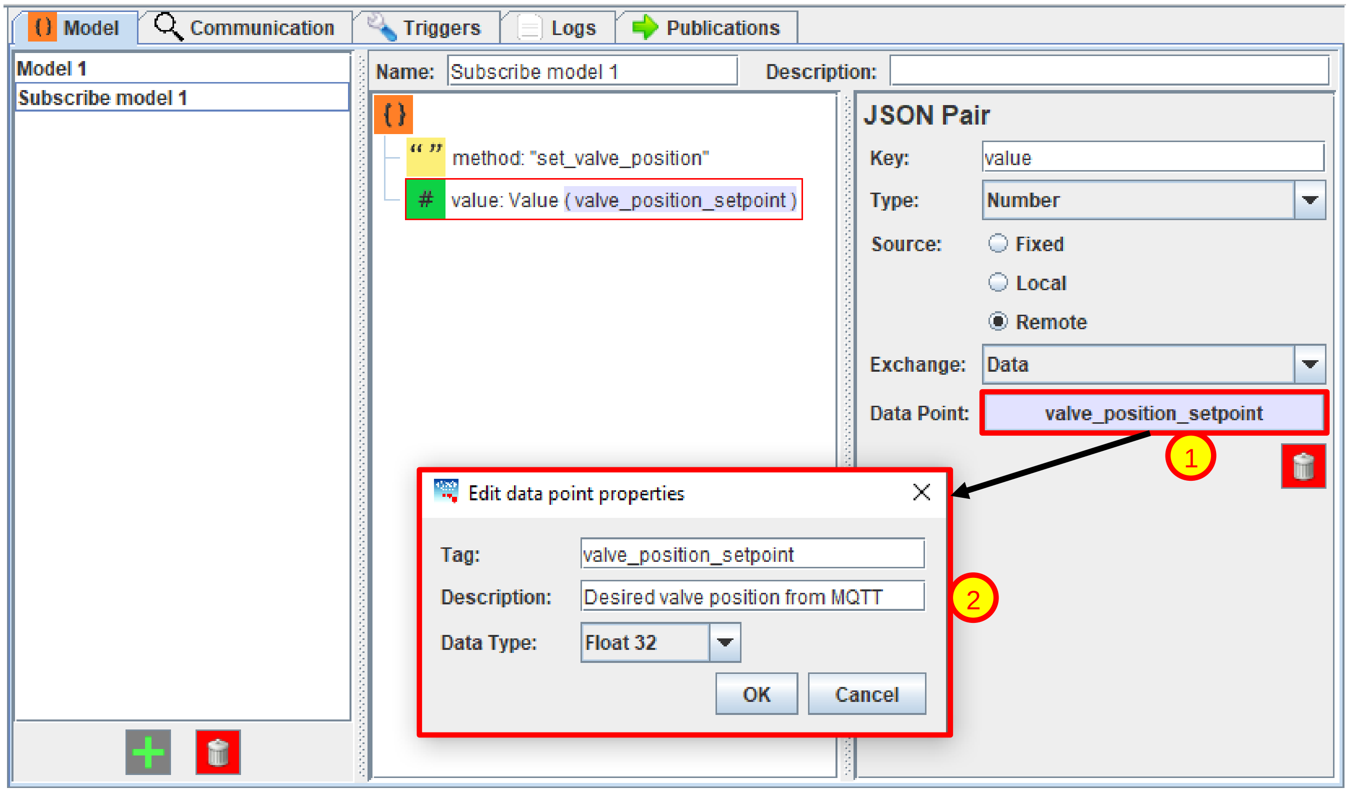
Then the new value will be placed into the ADH data point "`valve_position_setpoint`" to be used elsewhere in the ADH system via mapping.

For example, if the incoming message was:

```
{
  "method": "set_valve_position",
  "value": 85.2
}
```

Then the value 85.2 would be send to the ADH under this data point.

To configure the data point properties, simply click the data point button.


Model
Communication
Triggers
Logs
Publications
Model 1
Subscribe model 1
Name: Subscribe model 1
Description:
method: "set_valve_position"
value: Value (valve_position_setpoint)
JSON Pair
Key: value
Type: Number
Source: Fixed Local Remote
Exchange: Data
Data Point: valve_position_setpoint
1
Edit data point properties
Tag: valve_position_setpoint
Description: Desired valve position from MQTT
Data Type: Float 32
OK
Cancel
2

1 **Click the data point button** – This will bring up the data point properties window.

2 **Set properties** – Set the new tag, description and data type of the data point, and press OK.

Models *can* be made to contain multiple data and command points in the one model. All values are updated or controlled as per the mapping.

3.2.2.5 *Remote source: Command*

Incoming messages can also be used to trigger commands. Select the exchange type of Command (single stage).

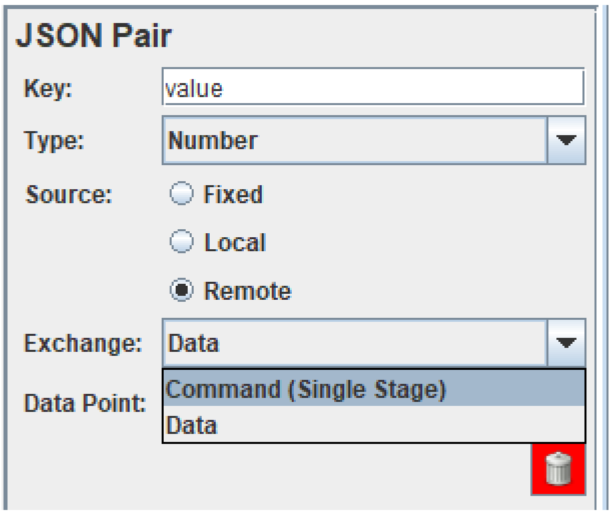


Figure 3-13 - Select command type for remote source

Since the command needs to *go* somewhere in the ADH, it needs to be mapped to a command.

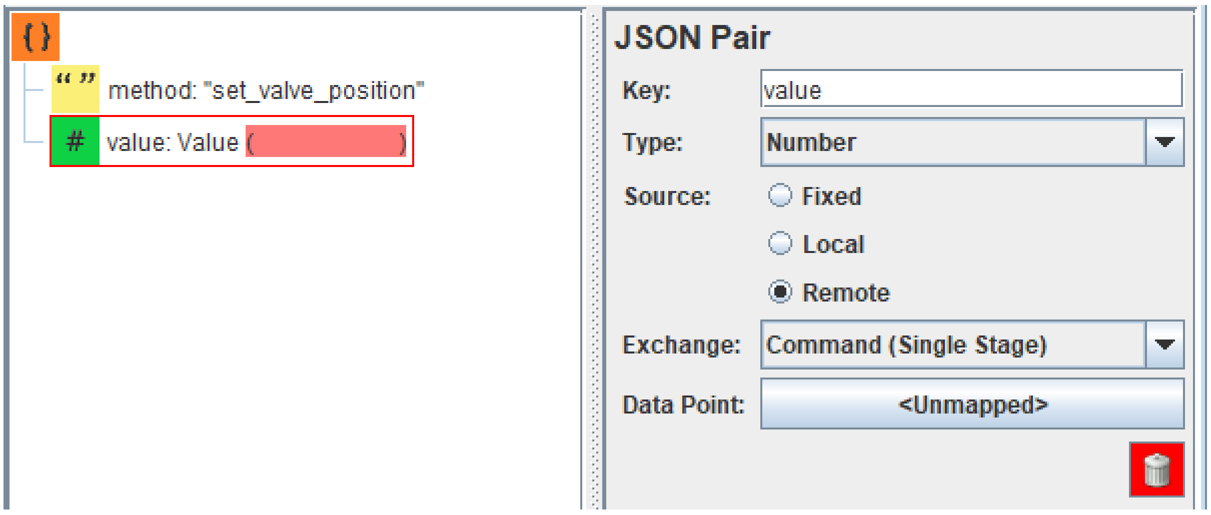
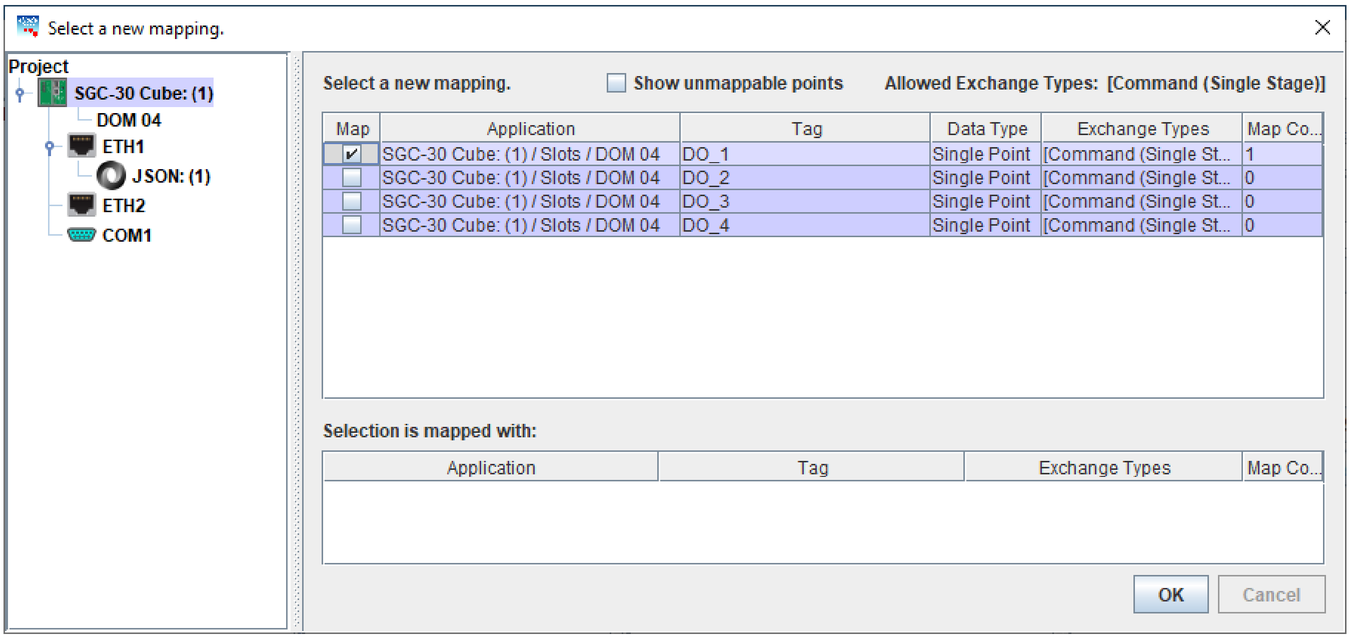

{ }
method: "set_valve_position"
value: Value ( )
JSON Pair
Key: value
Type: Number
Source: Fixed / Local / [✓ Remote]
Exchange: Command (Single Stage)
Data Point: <Unmapped>

Figure 3-14 - Remote command initially unmapped

Click the data point button and map it to an existing controllable data point that supports single-stage commands. Multi-stage commands are not currently supported, so cannot be mapped.


Select a new mapping.

Project
- SGC-30 Cube: (1)
  - DOM 04
  - ETH1
    - JSON: (1)
  - ETH2
  - COM1

Select a new mapping. ☐ Show unmappable points Allowed Exchange Types: [Command (Single Stage)]

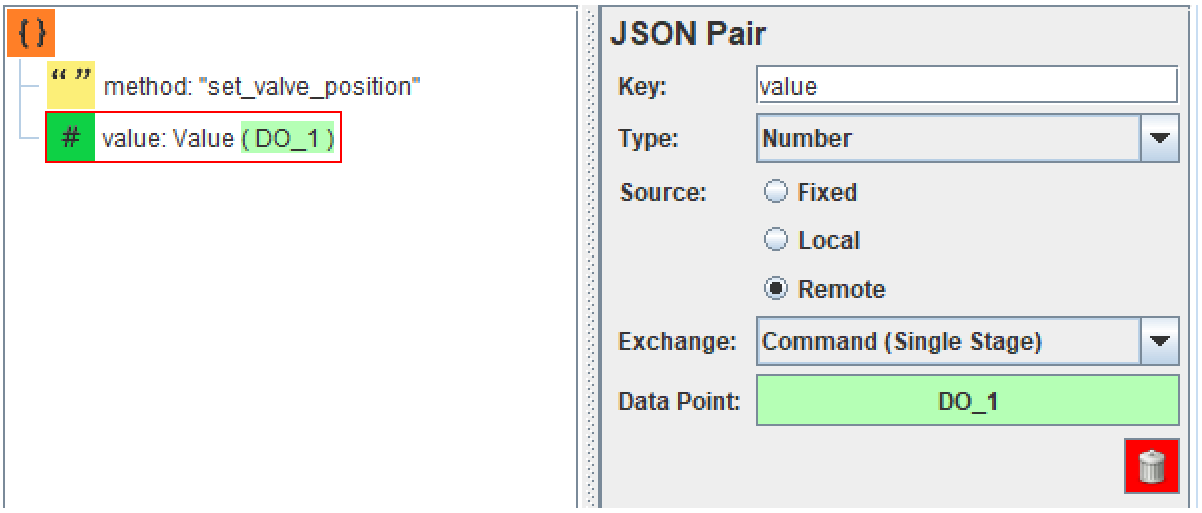
Map	Application	Tag	Data Type	Exchange Types	Map Co...
✓	SGC-30 Cube: (1) / Slots / DOM 04	DO_1	Single Point	[Command (Single St...	1
☐	SGC-30 Cube: (1) / Slots / DOM 04	DO_2	Single Point	[Command (Single St...	0
☐	SGC-30 Cube: (1) / Slots / DOM 04	DO_3	Single Point	[Command (Single St...	0
☐	SGC-30 Cube: (1) / Slots / DOM 04	DO_4	Single Point	[Command (Single St...	0

Selection is mapped with:

Application	Tag	Exchange Types	Map Co...

Figure 3-15 - Map the remote command

Once mapped, it will appear in the model tree and on the data point button. The data point button can be clicked again to change the mapping.


{ }
“ ” method: "set_valve_position"
# value: Value (DO_1)

JSON Pair
Key: value
Type: Number
Source: Fixed / Local / [✓ Remote]
Exchange: Command (Single Stage)
Data Point: DO_1

Figure 3-16 - Mapped remote command

This example uses the SystemCORP Cube controller with a 4 relay output module, so it is mapped to control one of those outputs. When the matching JSON message is received, the output will be controlled to that value. To match, all fixed values must match the incoming packet, and the JSON data types must match the controlled point.

Turn off the relay:
```
      {
        "method": "set_valve_position",
        "value": 0
      }
```

Turn on the relay:
```
      {
        "method": "set_valve_position",
        "value": 1
      }
```

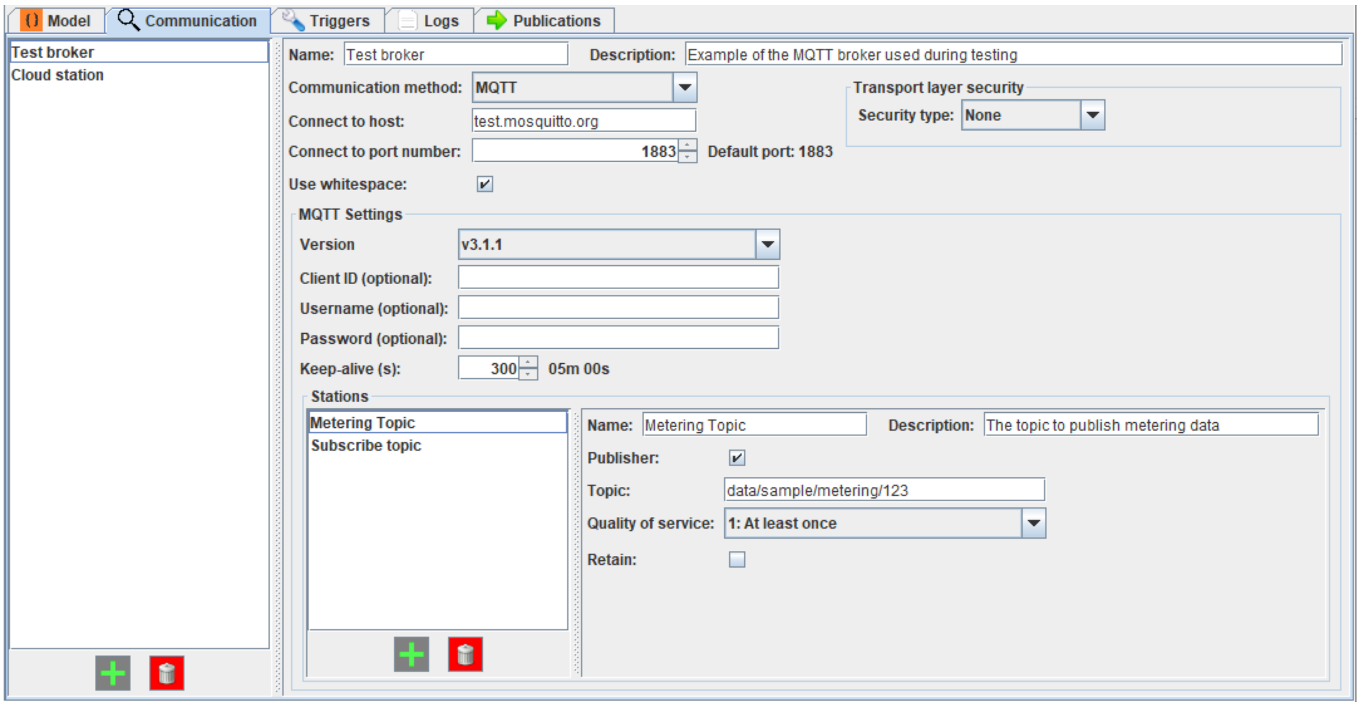
Models *can* be made to contain multiple data and command points in the one model. All values are updated or controlled as per the mapping.

3.3 Communication

The communication section defines the locations where the JSON data will be sent, and all associated parameters. The destinations are defined by IP address and port pairs, with a chosen communication method.

MQTT is widely used with JSON for data transfer, so this was chosen as the first transport method to implement. Currently, it is the only transport method supported, but more may be added in the future as required.

The configuration screen is shown below.



Test broker
Cloud station

Name: Test broker Description: Example of the MQTT broker used during testing
Communication method: MQTT
Transport layer security
Security type: None
Connect to host: test.mosquitto.org
Connect to port number: 1883 Default port: 1883
Use whitespace: ☑

MQTT Settings
Version v3.1.1
Client ID (optional):
Username (optional):
Password (optional):
Keep-alive (s): 300 05m 00s

Stations
Metering Topic
Subscribe topic

Name: Metering Topic Description: The topic to publish metering data
Publisher: ☑
Topic: data/sample/metering/123
Quality of service: 1: At least once
Retain: ☐

Figure 3-17 - Communication screen

3.3.1 *Parameter Definitions*

3.3.1.1 *Communication Method*

Description	The transport protocol to use to communicate with the device.
Data Entry	Combo box
Values	*MQTT*
Input Option	Mandatory

3.3.1.2 *Connect to host*

Description	The hostname, URL or IP address of the device to communicate with. For example, 192.168.1.110, or test.mosquitto.org.
Data Entry	Text
Input Option	Mandatory

3.3.1.3 *Connect to port number*

Description	Define the remote port number to connect to.
Data Entry	Number spinner
Minimum	0
Maximum	65535
Input Option	Mandatory

3.3.1.4 *Use whitespace*

Description	Flag that the message generated from the JSON model should include unnecessary whitespace. Including it makes it much easier for humans to read, but usually makes no difference to computers. Including whitespace increases the amount of data required to produce the message.
Data Entry	Checkbox
Values	*Checked* (add extra whitespace), *unchecked* (do not add extra whitespace)
Max Length	N/A
Input Option	Mandatory

3.3.1.5 *Transport layer security: Security type*

Description	The kind of transport layer security to use. See section 3.3.2 for details.
Data Entry	Combo box
Values	*None, Certificates, Pre-shared key*
Input Option	Mandatory

3.3.1.6 *MQTT: Client ID*

Description	The client ID to use in the MQTT connection. If not defined, a random one will be generated
Data Entry	String
Min Length	0
Max Length	N/A
Input Option	Optional

3.3.1.7 *MQTT: Username*

Description	The MQTT username to use for verification when initialising the connection. If blank, no username will be used.
Data Entry	String
Min Length	0
Max Length	N/A
Input Option	Optional

3.3.1.8 *MQTT: Password*

Description	The MQTT password to use in combination with the username for verification. If blank, none is used. If the username is blank, the password is not used.
Data Entry	String
Min Length	0
Max Length	N/A
Input Option	Optional

3.3.1.9 *Keep-alive (s)*

Description	The amount of time between MQTT “ping” messages to determine if the MQTT connection is OK, when no other messages are exchanged.

Data Entry	Number spinner
Minimum	0 (use default of 60 seconds)
Maximum	3600 (1 hour)
Input Option	Mandatory

3.3.1.10 *MQTT: Topic*

Description	The MQTT topic to publish on. Topics act like directory structures, with each "/" analogous to a directory. Example "data/sample/metering/123"
Data Entry	String
Min Length	0
Max Length	N/A
Input Option	Mandatory

3.3.1.11 *MQTT: Quality of service*

Description	The MQTT quality of service to use. This defines whether how to retry messages. The higher the quality, the more messages need to be exchanged.
Data Entry	Combo box
Values	*0: At most once, 1: At least once, 2: Exactly once*.
Input Option	Mandatory

3.3.1.12 *MQTT: Retain*

Description	Flag that the broker must maintain the last value of the message. This means that new connecting clients will be immediately sent the last message stored in the broker.
Data Entry	Check box
Values	*Checked* (Broker must retain the message), *Unchecked* (broker will not retain the message)
Input Option	Mandatory

3.3.2 Transport Layer Security

Transport layer security can be used to provide authentication of both ends and encryption of data. The options are:

Scheme	Meaning
None	No transport layer security added; no authentication or encryption.
Certificates	Use standard certificate-based SSL/TLS support. Use PEM-encoded certificate authority, and optionally a client certificate and client private keys to perform SSL/TLS authentication and encryption of data.
Pre-shared key	Use pre-shared-key based TLS support.

With None selected, there are no transport layer security options required.

3.3.2.1 *Certificates*

When Certificates is selected, the following options are available:

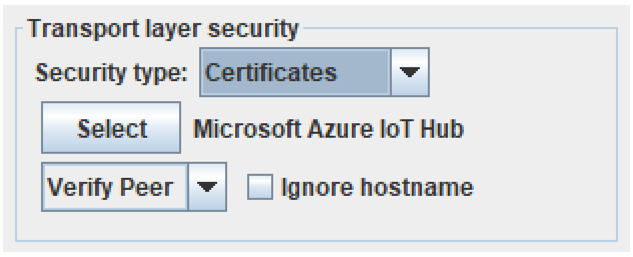
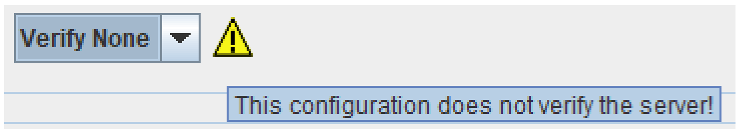
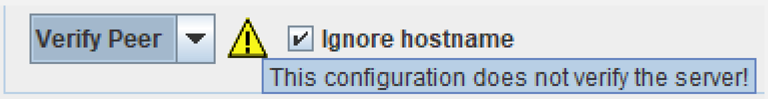


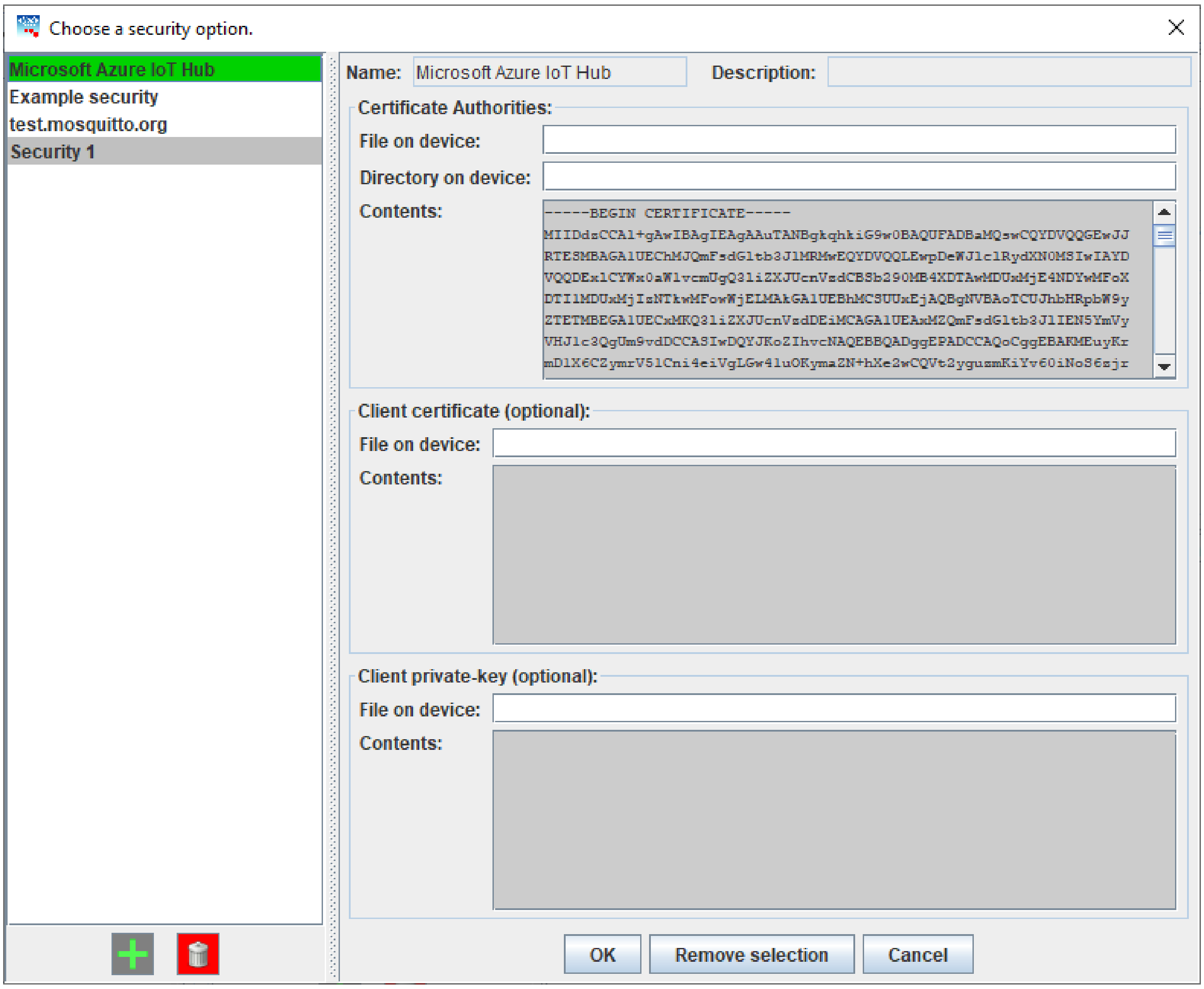
Figure 3-18 - Certificates selection window

Here you can select:

Option	**Meaning**
Certificate set	The set of certificate information to use for the connection
Verification	Verify Peer; or Verify None. Whether to verify the identity of the server. If not verified, the server cannot be determined to be correct. A warning will be shown.  This configuration does not verify the server!
Ignore hostname	Whether to ignore the server's hostname in negotiation. If ignored, the server cannot be determined to be correct. A warning will be shown.  This configuration does not verify the server!

Clicking "Select" allows you to change the active certificate set. By default, the eNode module comes with the Microsoft Azure IoT hub certificate authority's certificate, with not client certificate or private key enabled – this is the default settings for communications with Azure's IoT Hub.

Like the other configuration screens, new certificate-sets can be created using the + and delete icons.



Name: Microsoft Azure IoT Hub Description:

Certificate Authorities:

File on device:

Directory on device:

Contents:

```
-----BEGIN CERTIFICATE-----
MIIDdzCCAl+gAwIBAgIEAgAAuTANBgkqhkiG9w0BAQUFADBaMQswCQYDVQQGEwJJ
RTESMBAGA1UEChMJQmFsdGltb3JlMRMwEQYDVQQLEwpDeWJlclRydXN0MSIwIAYD
VQQDExlCYWx0aW1vcmUgQ3liZXJUcnVzdCBSb290MB4XDTAwMDUxMjE4NDYwMFoX
DTI1MDUxMjIzNTkwMFowWjELMAkGA1UEBhMCSUUxEjAQBgNVBAoTCUJhbHRpbW9y
ZTETMBEGA1UECxMKQ3liZXJUcnVzdDEiMCAGA1UEAxMZQmFsdGltb3JlIEN5YmVy
VHJ1c3QgUm9vdDCCASIwDQYJKoZIhvcNAQEBBQADggEPADCCAQoCggEBAKMEuyKr
mD1X6CZymrV51Cni4eiVgLGw41uOKymaZN+hXe2wCQVt2yguzmKiYv60iNoS6zjr
```

Client certificate (optional):

File on device:

Contents:

Client private-key (optional):

File on device:

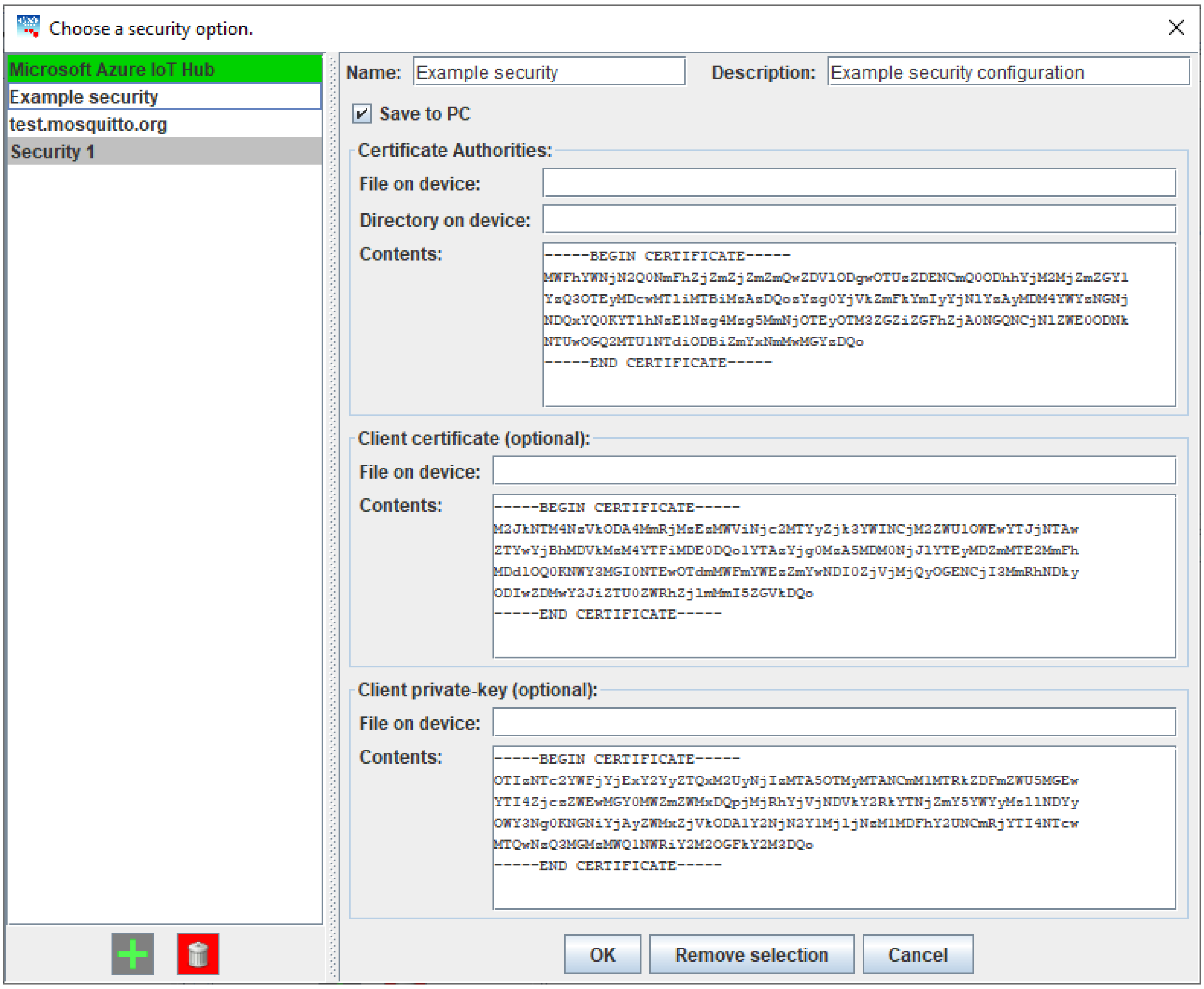
Contents:

Figure 3-19 - Certificate selection window

This allows the user to create certificate options that can be saved to the PC or just the eNode project. Certificate authorities define the root certificate, which is used to authenticate the server, and is mandatory. The client certificates are optional and allow the server to verify the client that is connecting. Some servers may require the client to use the client certificates.

Configuration	Usage
File on device	Use the certificate or private key from the *device's* file system. The certificate or key must be at the specified location for this to succeed. The JSON application currently provides no mechanism to transfer the file, so it is outside the scope of the eNode module.
Directory on device	Search the directory from the *device's* file system. Similarly to the file on the device, it is outside the scope of this module.
Contents	The contents of the certificate or private key file can be copied and pasted directly into this text area, which will be transferred to the device within the configuration files sent to the device. Open the PEM-encoded file in a text editor such as Notepad or Notepad++ to see its contents, to copy.

Any default available certificate sets cannot be modified. User-created sets can be modified and saved to the PC, as shown below.



Name: Example security

Description: Example security configuration

[✓] Save to PC

Certificate Authorities:

File on device:

Directory on device:

Contents:

```
-----BEGIN CERTIFICATE-----
MWFhYWNjN2Q0NmFhZjZmZjZmZmQwZDV1ODgwOTUzZDENCmQ0ODhhYjM2MjZmZGY1
YzQ3OTEyMDcwMTliMTBiMzAzDQozYzg0YjVkZmFkYmIyYjN1YzAyMDM4YWYzNGNj
NDQxYQ0KYTlhNzE1Nzg4Mzg5MmNjOTEyOTM3ZGZiZGFhZjA0NGQNCjN1ZWE0ODNk
NTUwOGQ2MTU1NTdiODBiZmYxNmMwMGYzDQo
-----END CERTIFICATE-----
```

Client certificate (optional):

File on device:

Contents:

```
-----BEGIN CERTIFICATE-----
M2JkNTM4NzVkODA4MmRjMzEzMWViNjc2MTYyZjk3YWINCjM2ZWU1OWEwYTJjNTAw
ZTYwYjBhMDVkMzM4YTFiMDE0DQo1YTAzYjg0MzA5MDM0NjJ1YTEyMDZmMTE2MmFh
MDd1OQ0KNWY3MGI0NTEwOTdmMWFmYWEzZmYwNDI0ZjVjMjQyOGENCjI3MmRhNDky
ODIwZDMwY2JiZTU0ZWRhZjlmMmI5ZGVkDQo
-----END CERTIFICATE-----
```

Client private-key (optional):

File on device:

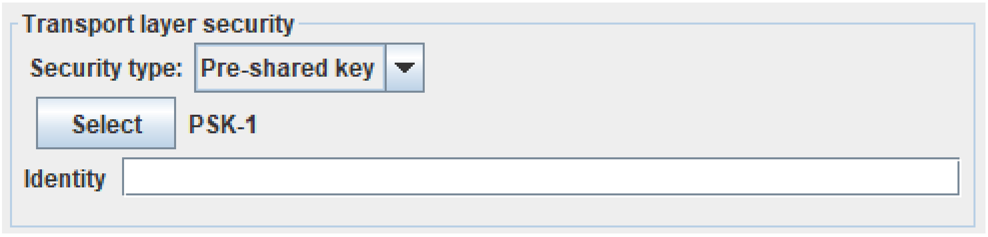
Contents:

```
-----BEGIN CERTIFICATE-----
OTIzNTc2YWFjYjExY2YyZTQxM2UyNjIzMTA5OTMyMTANCmM1MTRkZDFmZWU5MGEw
YTI4ZjczZWEwMGY0MWZmZWMxDQpjMjRhYjVjNDVkY2RkYTNjZmY5YWYyMzl1NDYy
OWY3Ng0KNGNiYjAyZWMxZjVkODA1Y2NjN2Y1Mj1jNzM1MDFhY2UNCmRjYTI4NTcw
MTQwNzQ3MGMzMWQ1NWRiY2M2OGFkY2M3DQo
-----END CERTIFICATE-----
```

Figure 3-20 - Configuring user-defined certificate information

3.3.2.2 *Pre-shared Key*

Pre-shared keys can be configured similarly to the certificates, but with different configuration options:



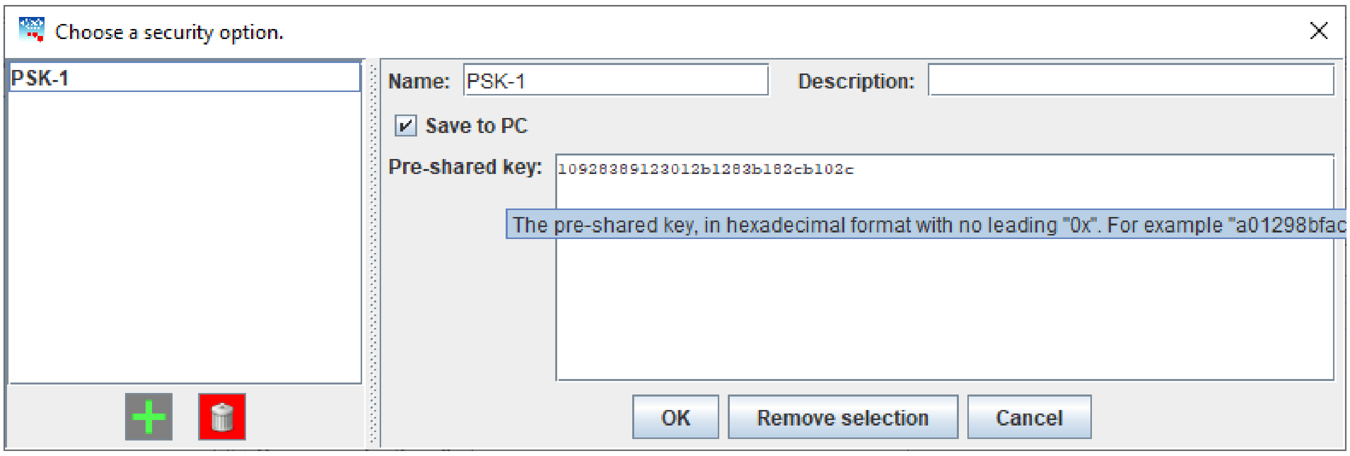
Security type: Pre-shared key

PSK-1

Identity

Configurable	Meaning
Pre-shared key	The pre-shared key is given in hexadecimal format, with no leading ‘0x’ characters.
Identity	The identity of the client connecting. May be used as a username depending on the server settings.

Clicking the “Select” button allows the user to select which pre-shared key they wish to use, which can also be saved to the PC. Mousing-over the labels can be used to give more details, so you are less likely to need to refer to the user manual each time.


Choose a security option.

PSK-1

Name: PSK-1

Description:

[✓] Save to PC

Pre-shared key: 10928389123012b1283b182cb102c

The pre-shared key, in hexadecimal format with no leading "0x". For example "a01298bfac

Figure 3-21 - Pre-shared key selection window

3.4 Trigger Trees

Triggers are used to define *when* to send messages. The currently supported triggers are:

- Periodic (a maximum period between messages)
- Data point trigger (on data point change beyond thresholds or reaching a specific value)

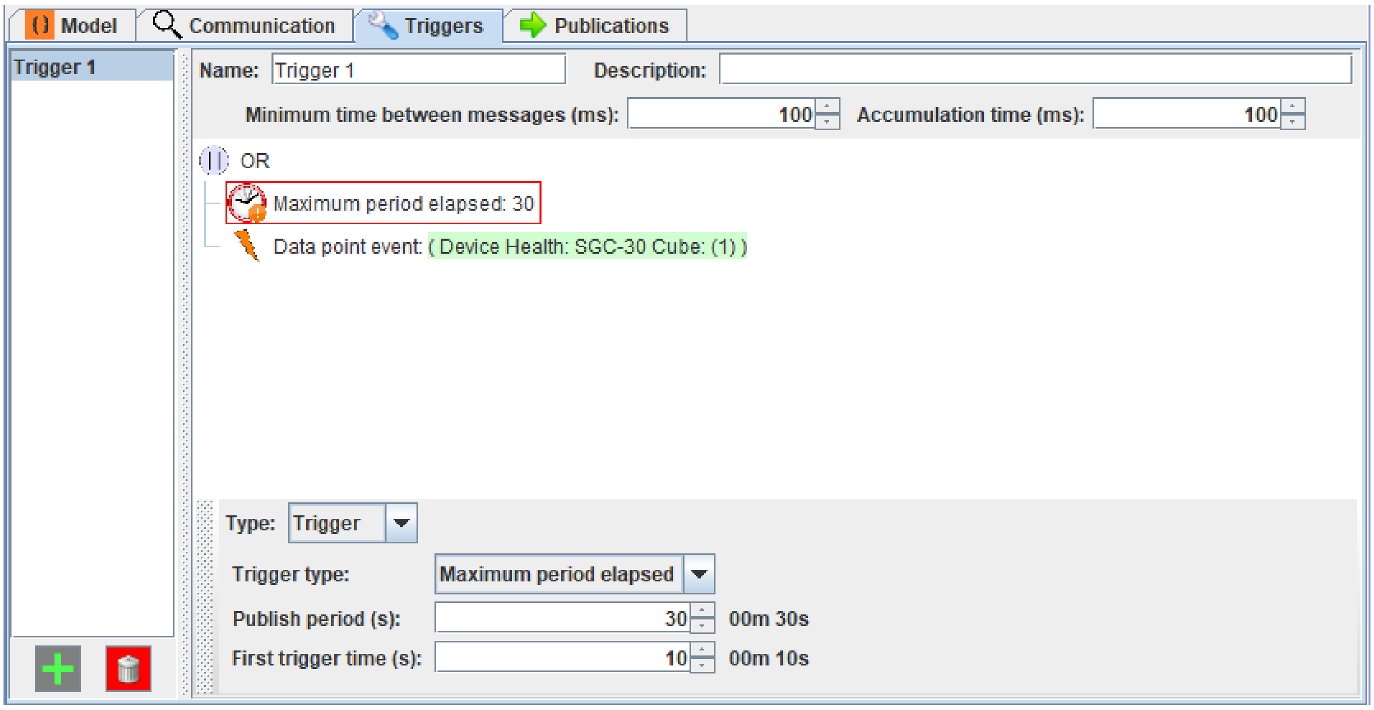
To increase the flexibility of triggers, there are support nodes:

- Logical AND – considered triggered if and only if all its direct children are triggered.
- Logical OR – considered triggered if one or more if its direct children are triggered.
- Logical NOT – triggered if its direct child is not triggered; not triggered if its direct child is triggered.

The "trigger tree" is said to be triggered when it's root node is triggered. Once triggered, a message will be generated after a configurable small accumulation time.

Combining these features allows, for example, to trigger every 15 minutes – unless an important event occurs when a message needs to be sent immediately. This would be achieved by using OR (period-15min, data-point-event).

An example configuration screen is shown below:



Trigger 1

Name: Trigger 1
Description:
Minimum time between messages (ms): 100
Accumulation time (ms): 100

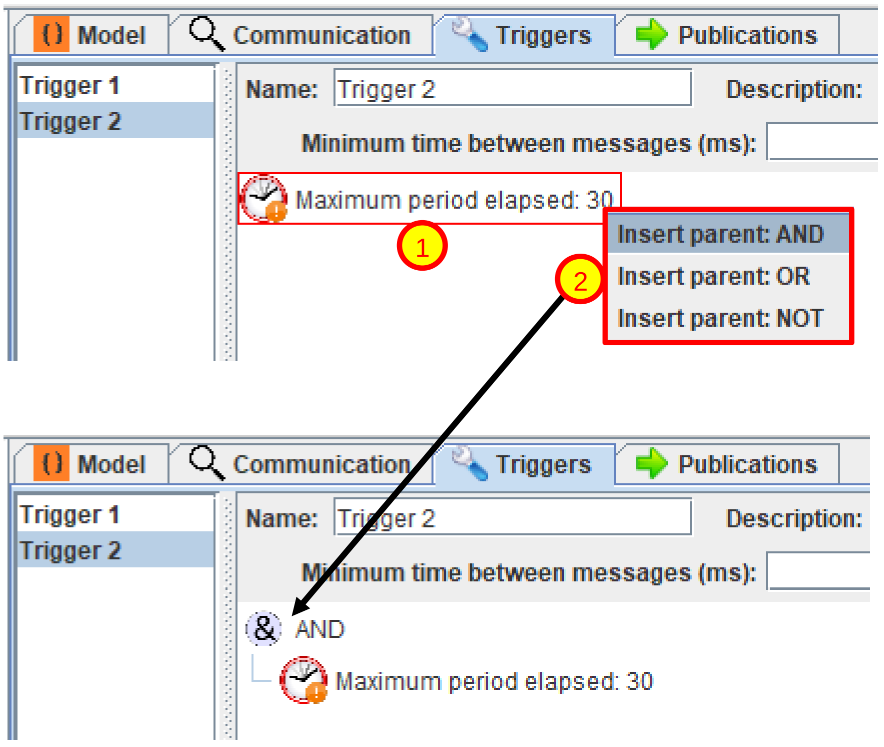
OR
Maximum period elapsed: 30
Data point event: (Device Health: SGC-30 Cube: (1))

Type: Trigger
Trigger type: Maximum period elapsed
Publish period (s): 30 00m 30s
First trigger time (s): 10 00m 10s

Figure 3-22 - Trigger screen

3.4.1 Defining a Trigger

When a trigger is added, there is a single item: the periodic trigger, containing some default values. The parameters are explained in the following sections. The trigger structure is modified by using a right-click context menu and drag-and-drop functionality as in the JSON model.



Name: Trigger 2

Maximum period elapsed: 30

AND

Maximum period elapsed: 30

Figure 3-23 - Insert trigger parent

1 **Right click the tree item** – This brings up a context menu where you can choose an action. The choices shown are dependent on the item and its current location – only relevant operations are shown.

2 **Select action** – For example, selecting “Insert parent: AND” will insert a new parent node of the “Logical AND” type. The result is shown in the lower screen capture.

Other context-menu options are available, as shown in the figures below. This menu is the method to add and remove items from the tree. Nodes can be moved by drag-and-drop, the same as the JSON model.

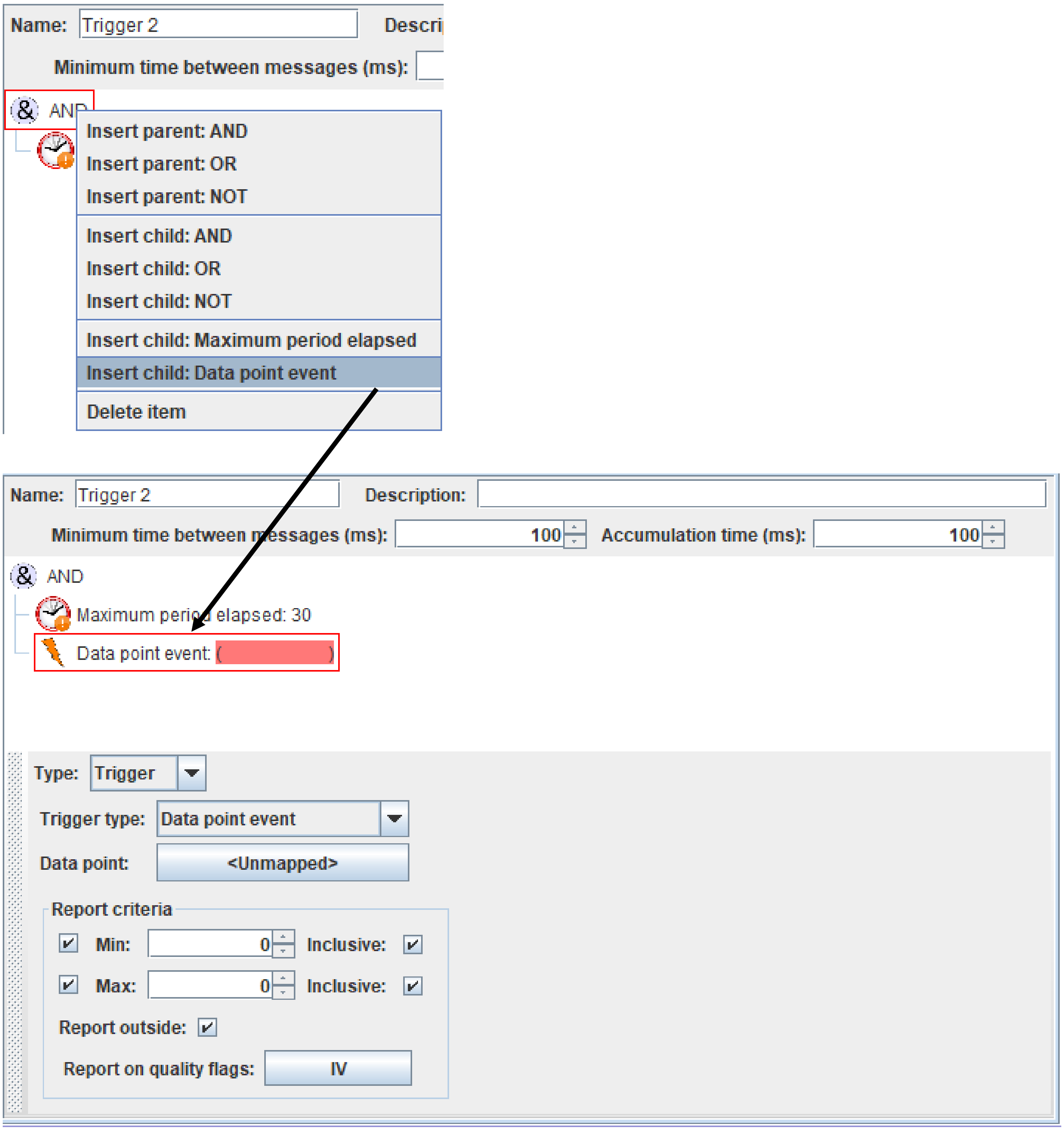


Figure 3-24 - Insert trigger child

3.4.1.1 *Minimum time between messages (ms)*

Description	The minimum time between generating messages due to this trigger, in milliseconds. If another event occurs within this time, it will be triggered at the end of this time.
Data Entry	Number spinner
Minimum	0 (None)
Maximum	2147483647 (~24 days)

Input Option	Mandatory

3.4.1.2 *Accumulation time (ms)*

Description	The amount of time after a trigger before sending the message, in milliseconds. This can be used to accumulate multiple events into the one JSON message, if there are likely to be many changes close together.
Data Entry	Number spinner
Minimum	0 (Immediately)
Maximum	2147483647 (~24 days)
Input Option	Mandatory

3.4.2 Trigger: Maximum period elapsed

This is a kind of periodic trigger, which fires after there have been no generated messages for a given period. Therefore, it is not exactly a "periodic" timer which just fires every set period. The timer is re-started each time the message generated, even if the overall trigger was caused by another "branch".

The configuration options are described below.

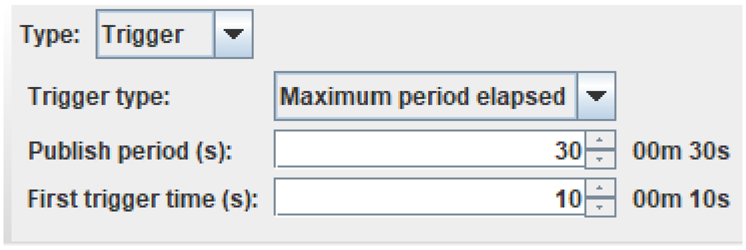

Type: Trigger
Trigger type: Maximum period elapsed
Publish period (s): 30 00m 30s
First trigger time (s): 10 00m 10s

Figure 3-25 - Maximum period elapsed configuration options

3.4.2.1 *Publish period (s)*

Description	Consider this node to be "triggered" if the current "trigger tree" has not been triggered for at least this amount of time, in seconds. Does not include the first trigger.
Data Entry	Number spinner
Minimum	0 (Never)
Maximum	2147483 (~24 days)
Input Option	Mandatory

3.4.2.2 *First trigger time (s)*

Description	The time after start-up before generating the first message, in seconds.
Data Entry	Number spinner
Minimum	0 (Immediately)
Maximum	2147483 (~24 days)
Input Option	Mandatory

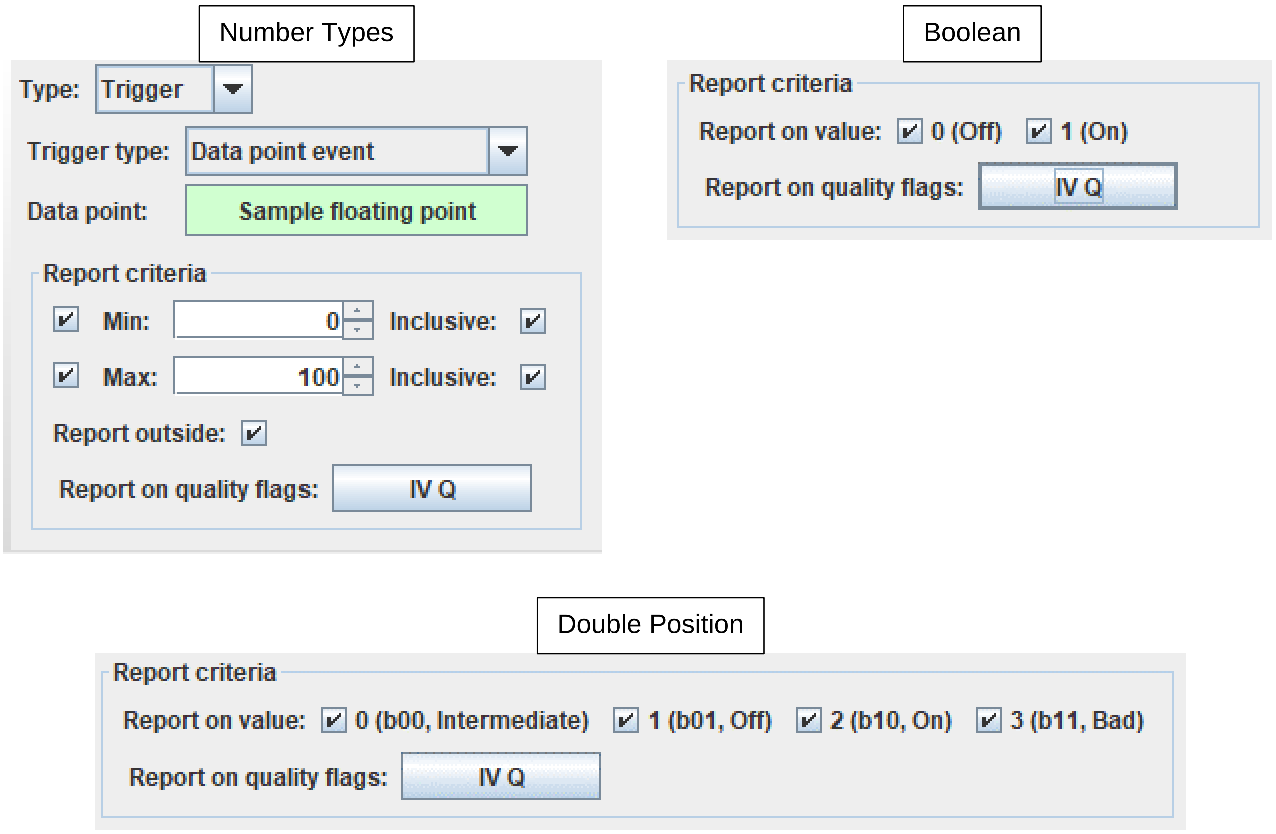
3.4.3 *Trigger: Data point event*

This trigger is fired when an ADH data point enters or exits a specified range or reaches a set of specified values. It can also fire when the quality flags of the point change – as a choice of the user.

Boolean and double position types have options for when the value becomes any of their states. The other number types define ranges that the value must enter or exit to be considered a trigger.

Quality flag triggers are generated when the quality flag gains or loses any of the specified quality flags defined.

The configuration options are described below.



Type: Trigger
Trigger type: Data point event
Data point: Sample floating point
Report criteria
☑ Min: 0 Inclusive: ☑
☑ Max: 100 Inclusive: ☑
Report outside: ☑
Report on quality flags: IV Q

Report criteria
Report on value: ☑ 0 (Off) ☑ 1 (On)
Report on quality flags: IV Q

Report criteria
Report on value: ☑ 0 (b00, Intermediate) ☑ 1 (b01, Off) ☑ 2 (b10, On) ☑ 3 (b11, Bad)
Report on quality flags: IV Q

Figure 3-26 - Data point trigger configuration options

3.4.3.1 *Data point*

Description	The ADH data point to test.
Data Entry	Mapping window
Values	*Any number-type ADH data point*
Maximum	2147483 (~24 days)
Input Option	Mandatory

3.4.3.2 *Min Checkbox*

Description	Apply a “minimum value” threshold.
Data Entry	Checkbox

Values	*Checked* (Minimum value is used), *Unchecked* (Minimum value is not used)
Input Option	Mandatory

3.4.3.3 *Min Value*

Description	The minimum value threshold.
Data Entry	Number spinner
Minimum	*-infinity*
Maximum	*+infinity*
Input Option	Mandatory

3.4.3.4 *Min Inclusive*

Description	Whether the threshold “includes” the exact “min value” specified.
Data Entry	Checkbox
Minimum	*Checked* (Minimum value included), *Unchecked* (Minimum value excluded)
Input Option	Mandatory

3.4.3.5 *Max Checkbox*

Description	Apply a “maximum value” threshold.
Data Entry	Checkbox
Values	*Checked* (Maximum value is used), *Unchecked* (Maximum value is not used)
Input Option	Mandatory

3.4.3.6 *Max Value*

Description	The maximum value threshold.
Data Entry	Number spinner
Minimum	*-infinity*
Maximum	*+infinity*
Input Option	Mandatory

3.4.3.7 *Max Inclusive*

Description	Whether the threshold “includes” the exact “max value” specified.
Data Entry	Checkbox
Minimum	*Checked* (Minimum value included), *Unchecked* (Minimum value excluded)
Input Option	Mandatory

3.4.3.8 *Report outside*

Description	Whether to consider to be “triggered” when the value is outside or inside the specified range.
Data Entry	Checkbox
Values	*Checked* (Trigger when the data point value is outside the range) *Unchecked* (Trigger when the data point value is inside the range)
Input Option	Mandatory

3.4.3.9 *Report on value*

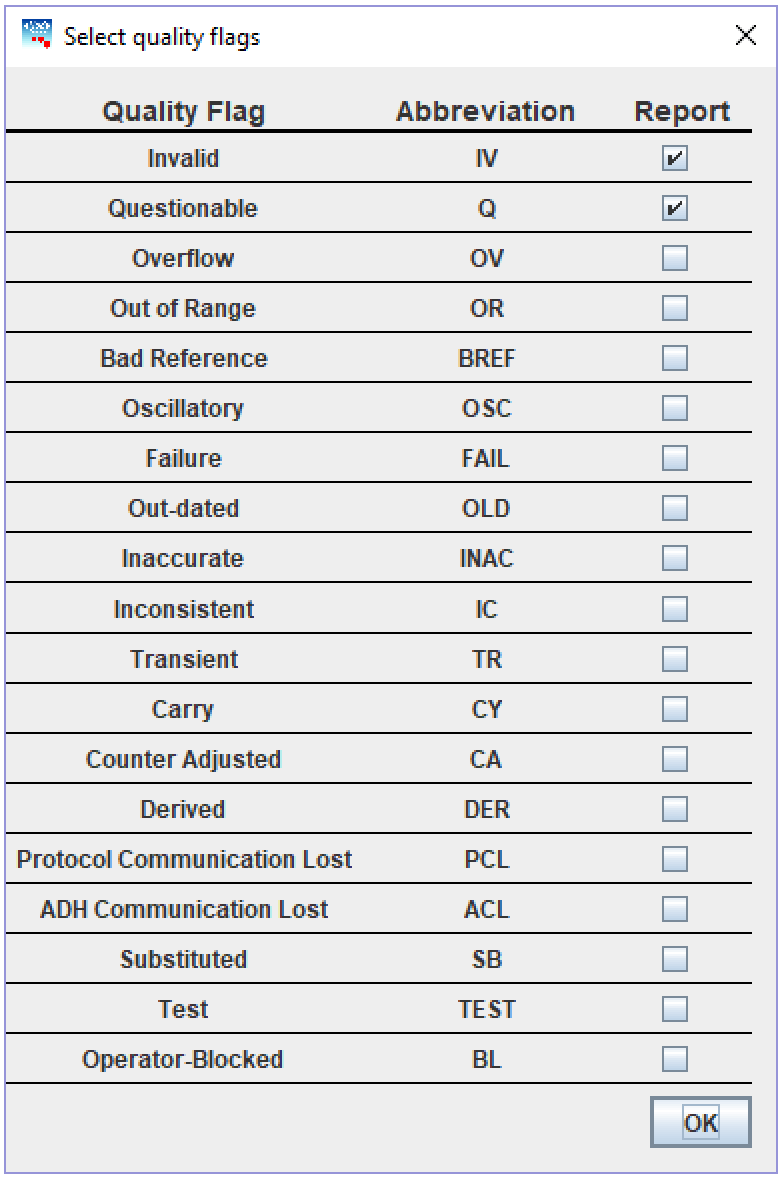
Description	Trigger when the data point is equal to one of the checked values.
Data Entry	Checkboxes
Options	According to data type. Boolean: *Off, On.* Double position: *Intermediate, Off, On, Bad*
Input Option	Optional (may specify no values)

3.4.3.10 *Report on quality flags*

Description	The quality flags whose change will create a trigger.
Data Entry	Popup window with checkboxes
Options	*Every ADH quality flag*. See next section.
Input Option	Optional (may specify no flags)

3.4.3.10.1 Quality flag options

Each quality flag option is shown below in the configuration window. Select the checkbox to report on changes in that quality flag. The precise meaning of each quality flag is given in the appendix: section *4.1*.


Select quality flags

Quality Flag	Abbreviation	Report
Invalid	IV	✓
Questionable	Q	✓
Overflow	OV	☐
Out of Range	OR	☐
Bad Reference	BREF	☐
Oscillatory	OSC	☐
Failure	FAIL	☐
Out-dated	OLD	☐
Inaccurate	INAC	☐
Inconsistent	IC	☐
Transient	TR	☐
Carry	CY	☐
Counter Adjusted	CA	☐
Derived	DER	☐
Protocol Communication Lost	PCL	☐
ADH Communication Lost	ACL	☐
Substituted	SB	☐
Test	TEST	☐
Operator-Blocked	BL	☐

Figure 3-27 - Quality flag trigger options window

3.5 Logs

Logs are used to store unacknowledged data in non-volatile storage. This means that if connection is temporarily lost to the remote side, the JSON application will continue to log the events to non-volatile storage. In case of the device's power loss, these events are stored. When power and an active is regained, the messages are then sent to the remote side of the connection. This ensures minimal loss of data, even in event of power loss.

The configuration parameters for logs are described below.

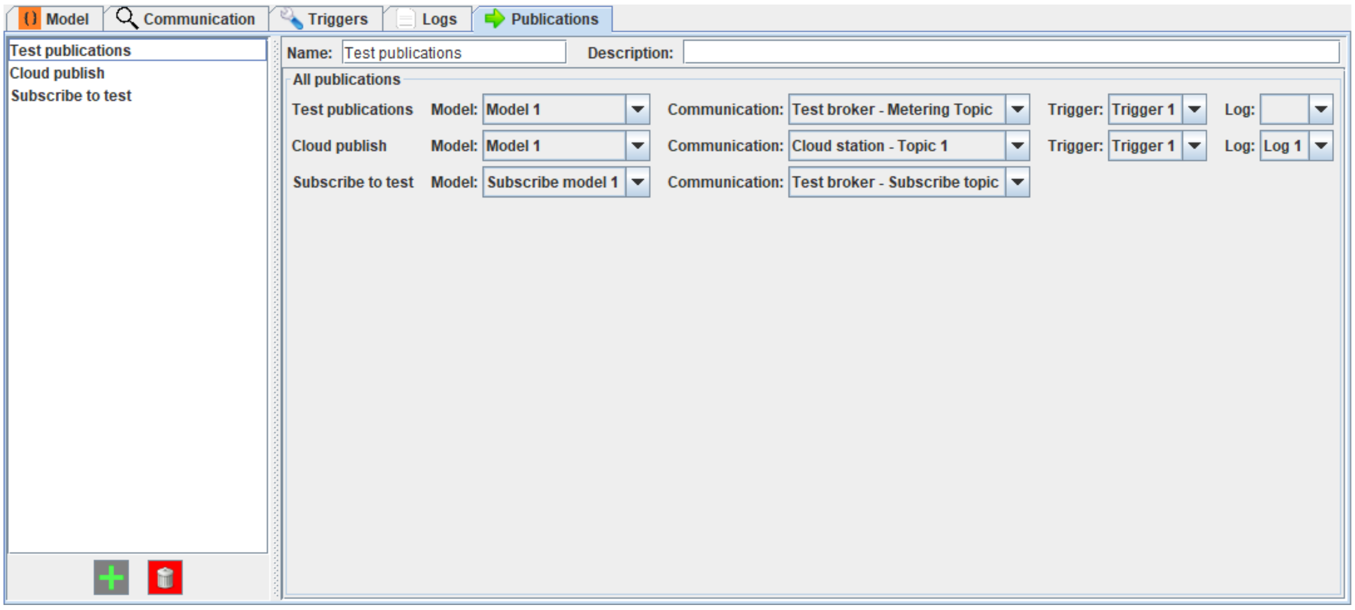
3.5.1.1 *Maximum acknowledgement wait time (ms)*

Description	The maximum amount of time to wait for an acknowledgement before writing the event to the log, in milliseconds. This is important to improve the lifetime expectation of the storage system (e.g. eMMC or SD card) of the device, as if the storage is written to constantly it can significantly reduce its lifetime. Set this to slightly higher than the highest expected delay to avoid writing to disk. However, note that if too high, the data will not be saved if the device loses power during this time.

Data Entry	Number spinner
Minimum	0 (Never)
Maximum	2147483647 (~24 days)
Input Option	Mandatory

3.6 *Publications*

Publications are a combination of model, communication and trigger tree which results in an actual message being published.



Test publications
Cloud publish
Subscribe to test

Name: Test publications Description:

All publications

Test publications	Model:	Model 1	Communication:	Test broker - Metering Topic	Trigger:	Trigger 1	Log:	
Cloud publish	Model:	Model 1	Communication:	Cloud station - Topic 1	Trigger:	Trigger 1	Log:	Log 1
Subscribe to test	Model:	Subscribe model 1	Communication:	Test broker - Subscribe topic				

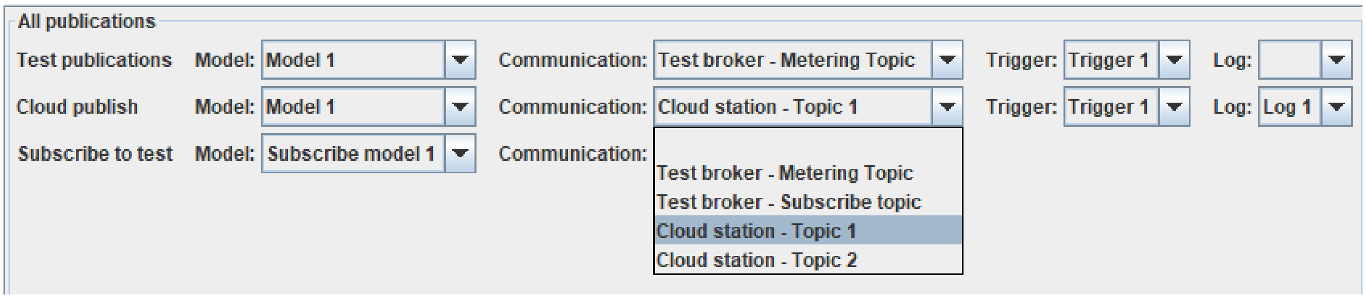
Figure 3-28 - Publication screen

The name and description can be changed in the text fields. They are only used as information for the user – they have no effect on the publication itself. Each publication has the name shown in the left pane matching the rest of the module. The difference here, is that selecting the module only changes the Name/description fields at the top of the screen. All publications are always shown in the centre right side, just to make it easier for the user to see all the combinations they have configured. All publication models, communication and triggers can be modified even if they are not selected, for convenience.

When the “trigger” is activated, messages will be generated for every publication it is paired with. These messages are based on the JSON model and are sent to the selected communication object.

The selected communication object is a set of the communication destination with a set of parameters for that message. For example, different publications can specify the same communication destination with different MQTT topics for different messages.

A closer look at the screen is shown below, with one of the combo boxes opened to show the configuration options.



All publications

Test publications	Model:	Model 1	Communication:	Test broker - Metering Topic	Trigger:	Trigger 1	Log:	
Cloud publish	Model:	Model 1	Communication:	Cloud station - Topic 1	Trigger:	Trigger 1	Log:	Log 1
Subscribe to test	Model:	Subscribe model 1	Communication:					

Test broker - Metering Topic
Test broker - Subscribe topic
Cloud station - Topic 1
Cloud station - Topic 2

Figure 3-29 - Publication screen centre area example

Each combo box shows all the available options. Model shows every model defined in the model configuration area, as do the trigger and log show their options. The communication option shows all available combinations of the communication destination and “Station” (or topic) configured for that destination.

4 Appendix

4.1 ADH Quality Flag Definitions

The ADH quality flags are as follows.

Flag	Short	Value	Definition
Good	OK	0x000000	Validity: no abnormal condition of the data acquisition is detected
Invalid	IV	0x000001	Validity: value is invalid or unknown
Questionable	Q	0x000002	Validity: value is in doubt and should not be relied on
Overflow	OV	0x000004	Detail: value cannot be represented in the given format (should have INVALID validity)
Out of Range	OR	0x000008	Detail: value is outside a predefined range
Bad Reference	BREF	0x000010	Detail: value may not be correct due to a reference being out of calibration (e.g. in an Analog to Digital Converter a known input gives wrong output)
Oscillatory	OSC	0x000020	Detail: value is currently oscillating (e.g. digital input continually changing 0/1). The data object value is not updated during this time
Failure	FAIL	0x000040	Detail: value is INVALID due to a failure
Out Dated	OLD	0x000080	Detail: value is out of date, it may have changed since the last notification (should have QUESTIONABLE validity)
Inaccurate	INAC	0x000100	Detail: value does not meet the stated accuracy of the source (should have QUESTIONABLE validity)
Inconsistent	INC	0x000200	Detail: an evaluation function has detected an inconsistency (should have QUESTIONABLE validity)
Transient	TR	0x000400	Detail: Equipment is in a transient state. Used in step-position information
Carry	CY	0x000800	Detail: Carry indicates (e.g. counter) overflow occurs when the value goes over-bound and resets to zero
Counter Adjusted	CA	0x001000	Detail: Counter was adjusted since the last reading (e.g. initialised to a new value)
Derived	DER	0x002000	Detail: The value is calculated, not a measurement
Protocol Communication Lost	PNC	0x004000	Detail: Protocol updating the ADH data point has lost communication to the source
ADH Communication Lost	ANC	0x008000	Detail: Not connected to the ADH application source of the data point
Substituted	SB	0x200000	Source: value, quality or time has been substituted
Test	TEST	0x400000	Test: test values should not be used for operational purposes
Operator Blocked	BL	0x800000	Further update of the value has been blocked by an operator. The value shall be the same as it was before blocking

Table 4-1 - ADH quality flags definitions

✹ End of Document ✹