SESSION
03

Edge Analytics Fundamentals

STREAMING DATA INPUTS

Manage data properties in Crosser Flow Studio

Session 3

Agenda

- Re-format messages
  - Property Mapper module
- External streaming inputs (push)
  - MQTT Client Sub module
- Multi-path flows
  - Split module
- Math operations
  - Math module
- Exercise 2: Process streaming data

crosser

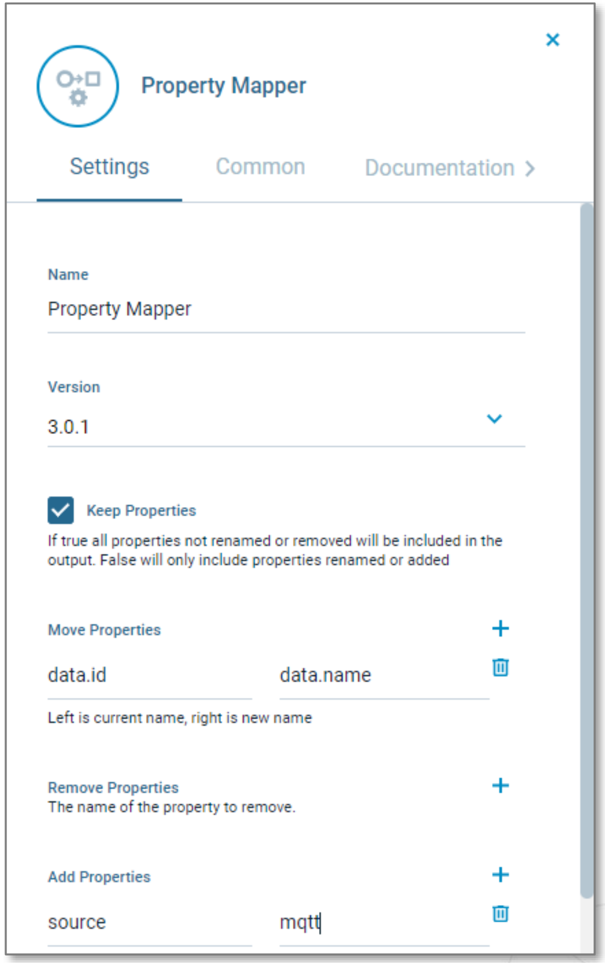
THE PROPERTY MAPPER MODULE

The Swiss army knife for re-formatting messages

Module

Property Mapper

- The Property Mapper module is used to change the format of messages (not the values!).
- Use it to:
  - Rename properties
  - Move or copy properties between any hierarchy levels
  - Remove properties
  - Add new properties
- Two modes of operation:
  - *Keep properties* = **True**
    - Start with a copy of the input message
    - Use Move/Remove/Add to modify the message
  - *Keep properties* = **False**
    - Start with an empty output message
    - Use Move and Add to specify the content of the output message



Name
Property Mapper

Version
3.0.1

✓ Keep Properties
If true all properties not renamed or removed will be included in the output. False will only include properties renamed or added

Move Properties
data.id data.name
Left is current name, right is new name

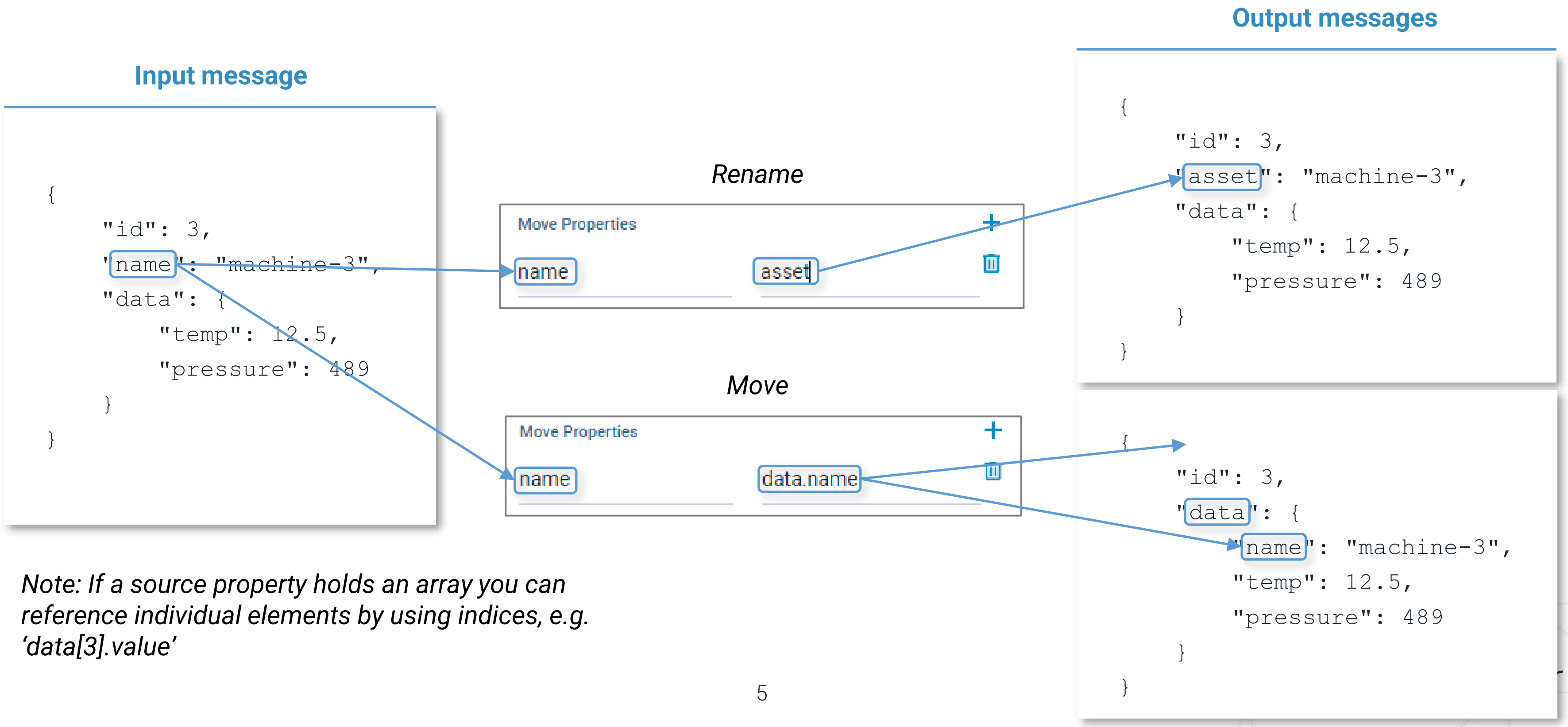
Remove Properties
The name of the property to remove.

Add Properties
source mqtt

crosser

Example 1

Rename & Move



Input message

```
{
    "id": 3,
    "name": "machine-3",
    "data": {
        "temp": 12.5,
        "pressure": 489
    }
}
```

Rename

Move Properties

name	asset

Move

Move Properties

name	data.name

Output messages

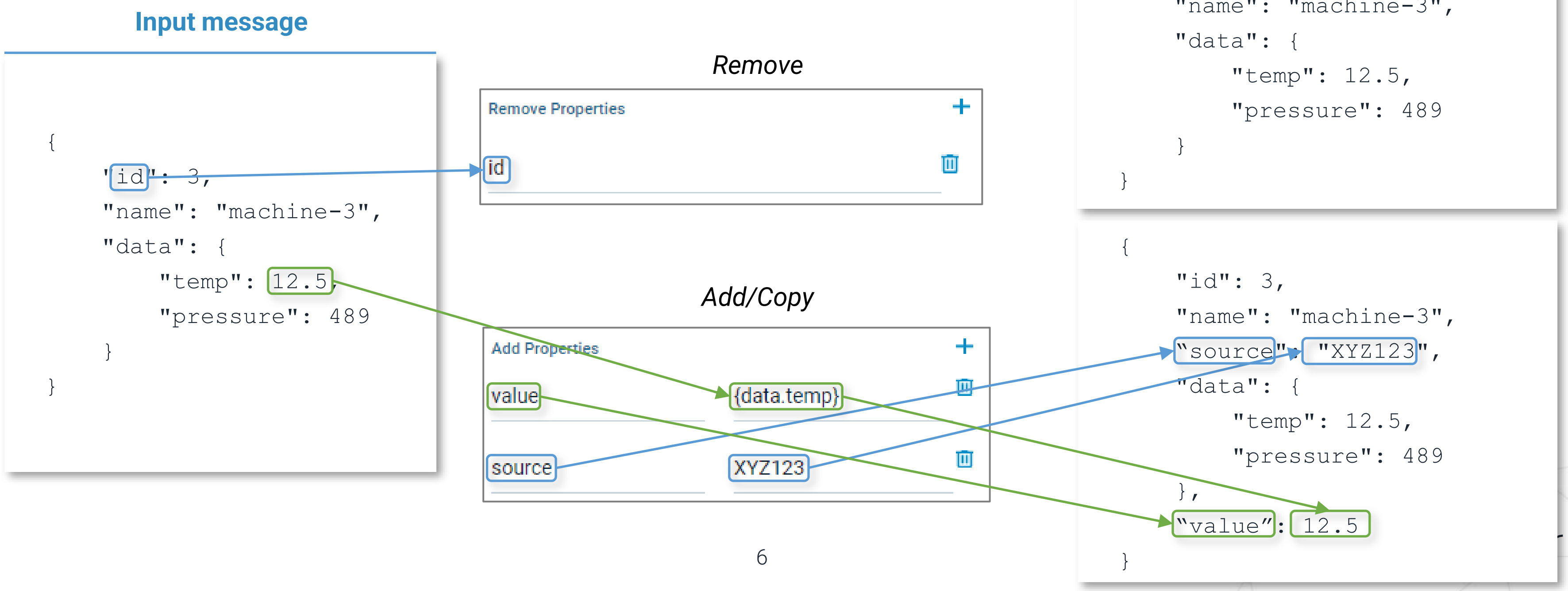
```
{
    "id": 3,
    "asset": "machine-3",
    "data": {
        "temp": 12.5,
        "pressure": 489
    }
}
```

```
{
    "id": 3,
    "data": {
        "name": "machine-3",
        "temp": 12.5,
        "pressure": 489
    }
}
```

Note: If a source property holds an array you can reference individual elements by using indices, e.g. 'data[3].value'

Example 2

Remove & Add



Input message

```
{
    "id": 3,
    "name": "machine-3",
    "data": {
        "temp": 12.5,
        "pressure": 489
    }
}
```

Remove

Remove Properties

id

Add/Copy

Add Properties

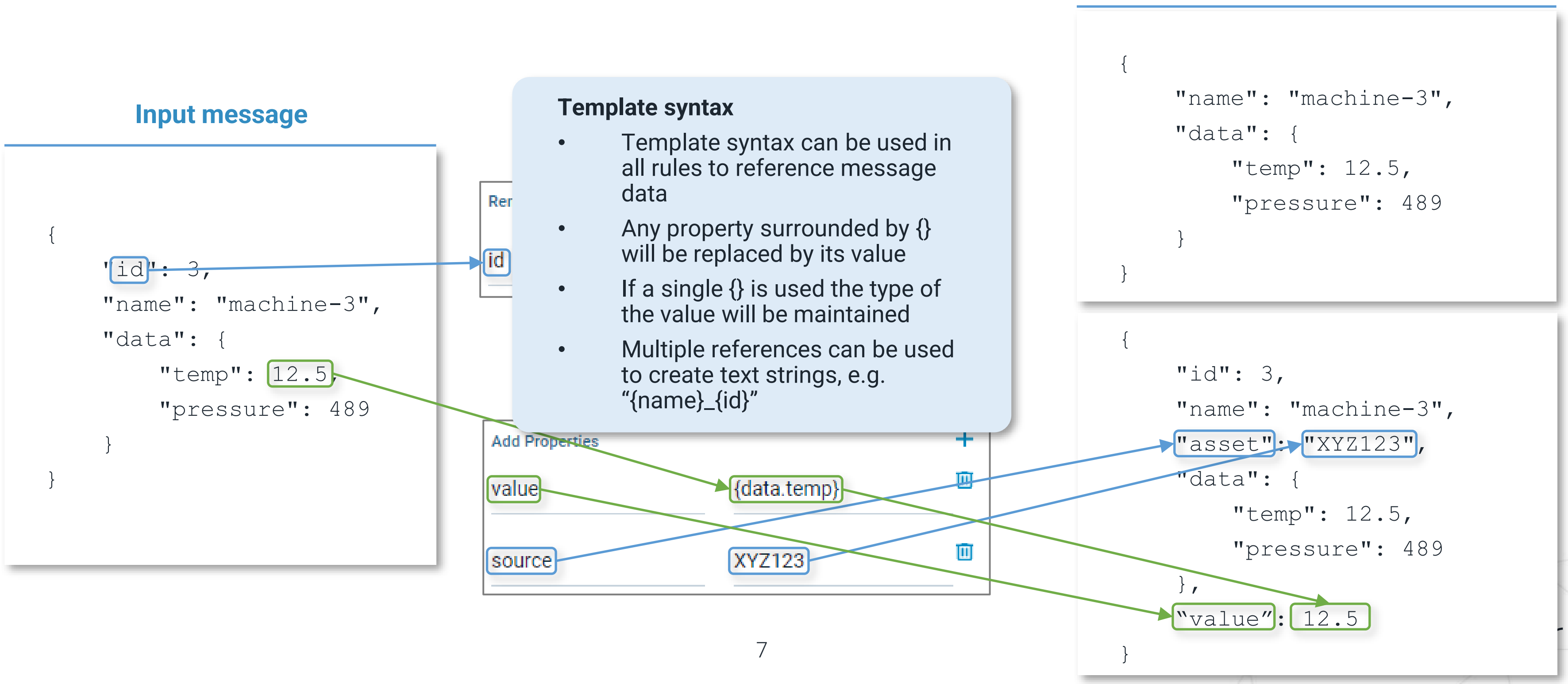
value	{data.temp}
source	XYZ123

Output messages

```
{
    "name": "machine-3",
    "data": {
        "temp": 12.5,
        "pressure": 489
    }
}
```

```
{
    "id": 3,
    "name": "machine-3",
    “source": "XYZ123",
    "data": {
        "temp": 12.5,
        "pressure": 489
    },
    “value”: 12.5
}
```

Example 2
Template syntax



Input message

```
{
    "id": 3,
    "name": "machine-3",
    "data": {
        "temp": 12.5,
        "pressure": 489
    }
}
```

Ren

id

Template syntax

- Template syntax can be used in all rules to reference message data
- Any property surrounded by {} will be replaced by its value
- If a single {} is used the type of the value will be maintained
- Multiple references can be used to create text strings, e.g. “{name}_{id}”

Add Properties

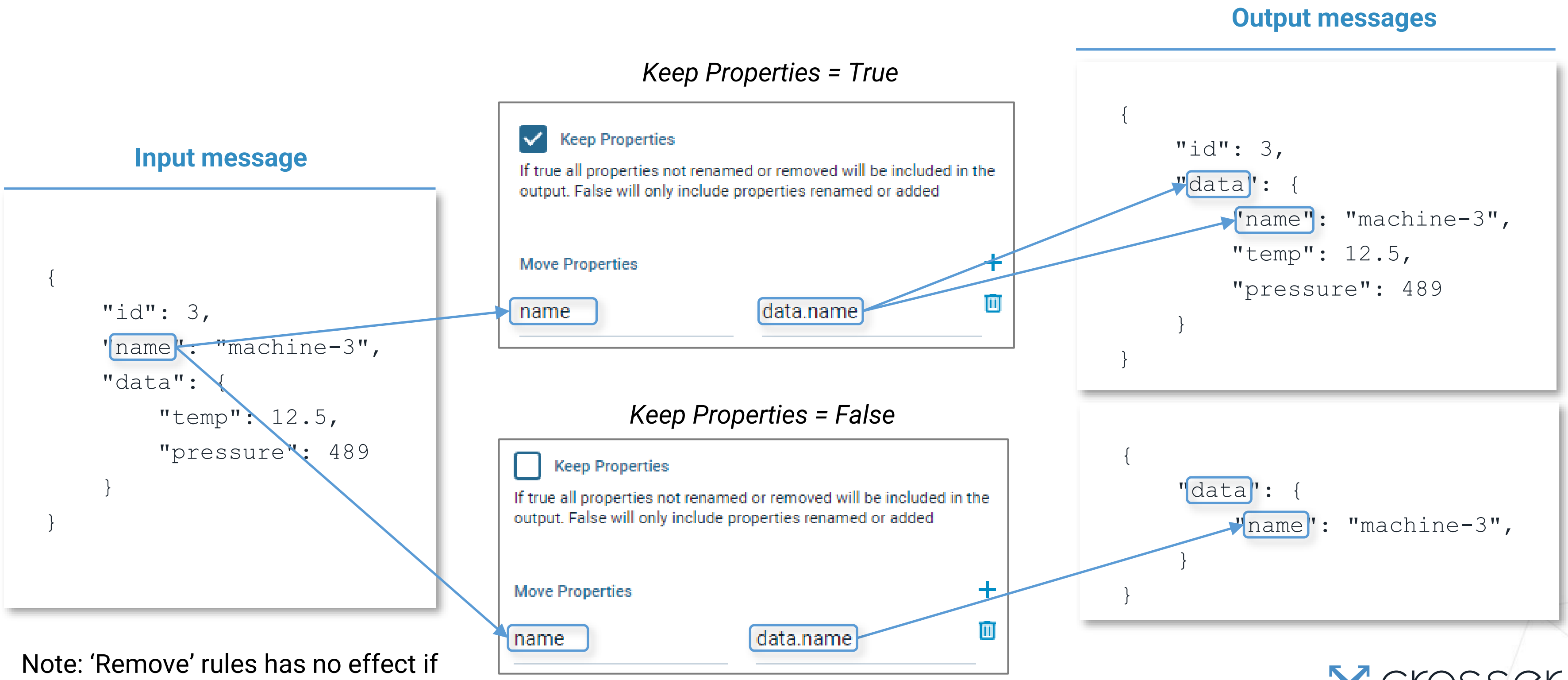
value	{data.temp}
source	XYZ123

Output messages

```
{
    "name": "machine-3",
    "data": {
        "temp": 12.5,
        "pressure": 489
    }
}
```

```
{
    "id": 3,
    "name": "machine-3",
    "asset": "XYZ123",
    "data": {
        "temp": 12.5,
        "pressure": 489
    },
    “value”: 12.5
}
```

Example 3
‘Keep Properties’



Input message

```
{
    "id": 3,
    "name": "machine-3",
    "data": {
        "temp": 12.5,
        "pressure": 489
    }
}
```

Keep Properties = True

✓ Keep Properties

If true all properties not renamed or removed will be included in the output. False will only include properties renamed or added

Move Properties

name	data.name

Keep Properties = False

☐ Keep Properties

If true all properties not renamed or removed will be included in the output. False will only include properties renamed or added

Move Properties

name	data.name

Output messages

```
{
    "id": 3,
    "data": {
        "name": "machine-3",
        "temp": 12.5,
        "pressure": 489
    }
}
```

```
{
    "data": {
        "name": "machine-3",
    }
}
```

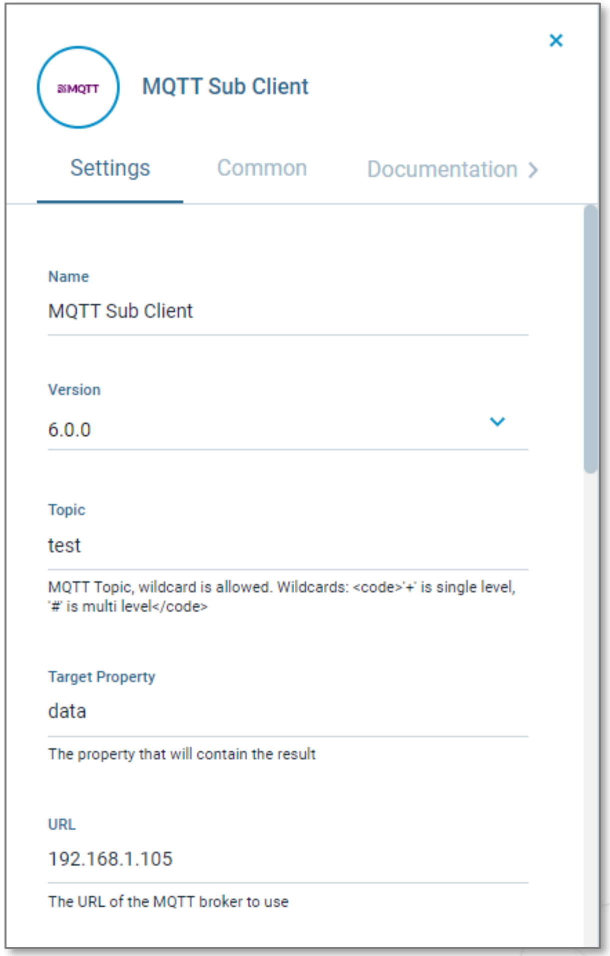
Note: ‘Remove’ rules has no effect if ‘Keep Properties’ is disabled

THE MQTT, SPLIT AND MATH MODULES

Module

MQTT Sub Client

- The MQTT Sub Client module is used to subscribe to MQTT messages from an external MQTT Broker.
  - Data will be pushed from the broker, i.e. can arrive at any time
- MQTT Topic
  - Select which data to receive
  - Wildcards
    - '+' is single level wild card
      Example of usage: *myhome/+/temperature* matches *myhome/abc/temperature* but <u>not</u> *myhome/a/b/c/temperature*
    - '#' is multi-level wild card
      Example of usage: *myhome/#/temperature* matches *myhome/abc/temperature* <u>and</u> *myhome/a/b/c/temperature*
- Output Format (message parsing)
  - Raw / JSON / XML (JSON default)
- The *MQTT Sub Broker* module has the same functionality but gets data from the internal MQTT broker



Name
MQTT Sub Client

Version
6.0.0

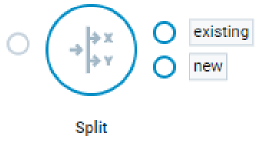
Topic
test
MQTT Topic, wildcard is allowed. Wildcards: <code>'+' is single level, '#' is multi level</code>

Target Property
data
The property that will contain the result

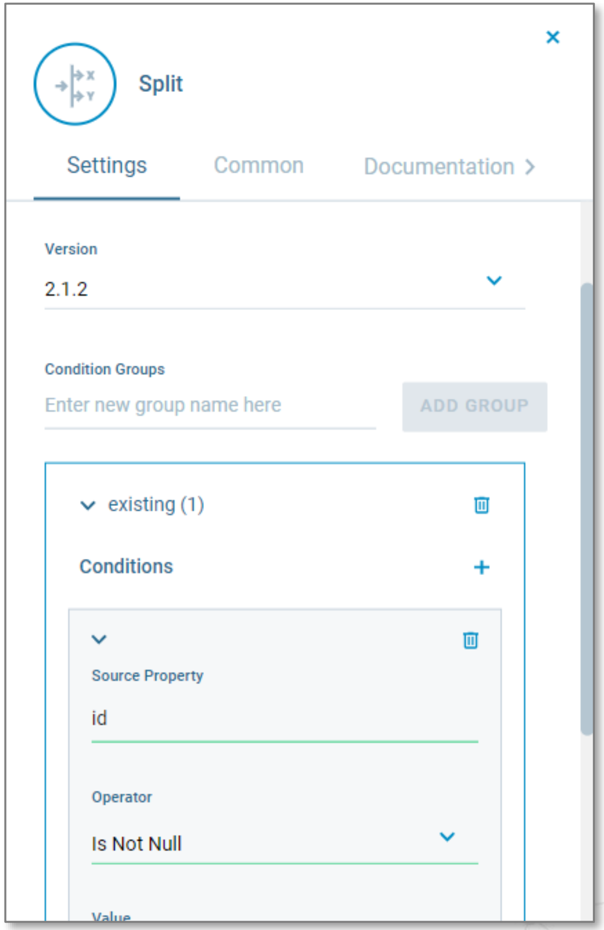
URL
192.168.1.105
The URL of the MQTT broker to use

Module

Split



- The Split module is used to split/route messages to different outputs using condition groups, one output per condition group
- The only module with multiple outputs
- Conditions Groups
  - One or several conditions on message content
  - How to handle multiple conditions:
    All conditions in group fulfilled (true (AND) or false (OR))
  - What to do when expected data is missing from a message: MissingOK
    - If 'MissingOK' is enabled conditions with missing properties will be treated as 'true'



Version
2.1.2

Condition Groups
Enter new group name here

existing (1)

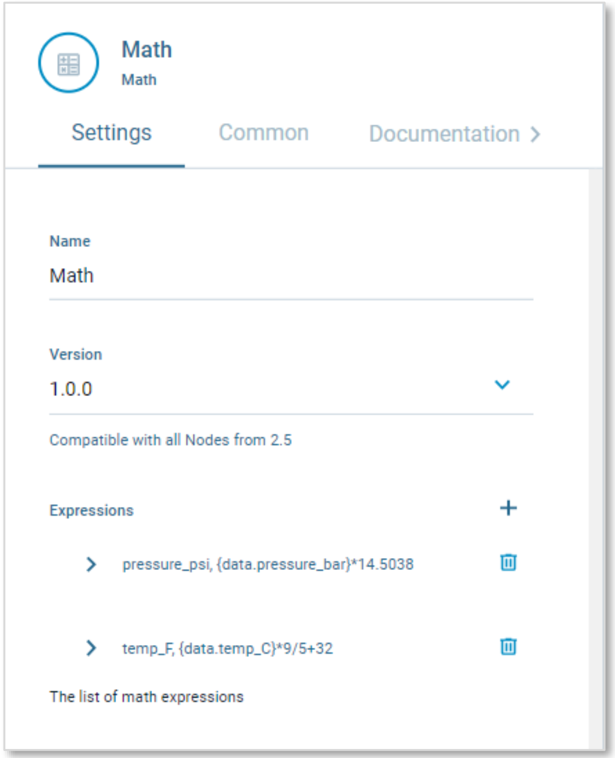
Source Property
id

Operator
Is Not Null

Value

MODULE
Math

- Perform arbitrary mathematical calculations on flow data
- Write expressions with flow data using template syntax ({})
- The result of a previous calculation can be used in the following expressions (all results will be present on the output message)
- Large number of mathematical functions supported (see docs):
  - +, -, *, /
  - Trigonometry
  - Log/Power
- Conditional outputs using *if* statements (cf Excel)



Math
Math

Settings | Common | Documentation

Name
Math

Version
1.0.0

Compatible with all Nodes from 2.5

Expressions

pressure_psi, {data.pressure_bar}*14.5038

temp_F, {data.temp_C}*9/5+32

The list of math expressions

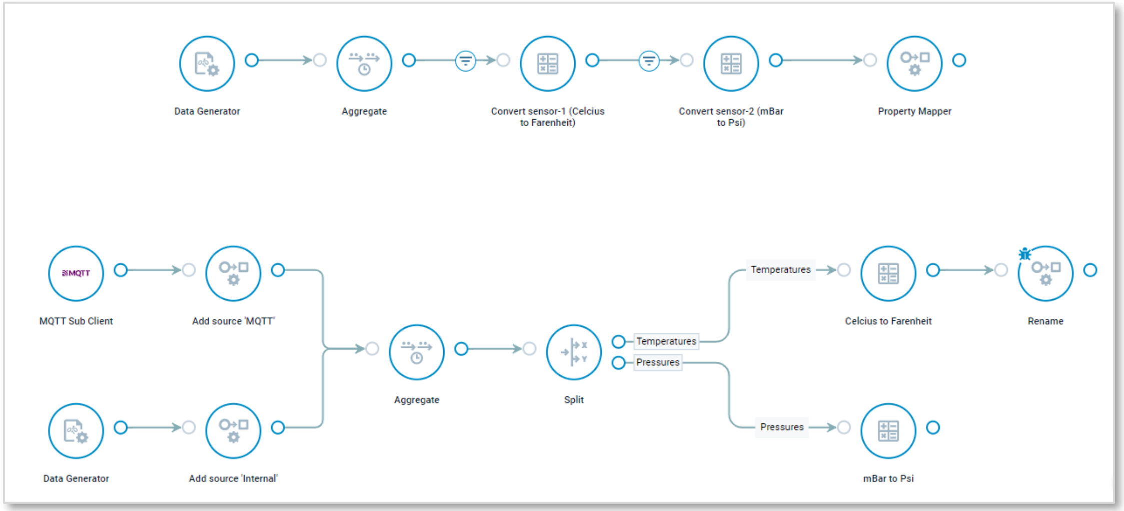
EXERCISE 2

Process streaming data

Exercise 2

Overview

- Process streaming data from multiple sources
- Use Message filters to do conditional processing
- Separate processing into multiple paths based on data
- Add metadata

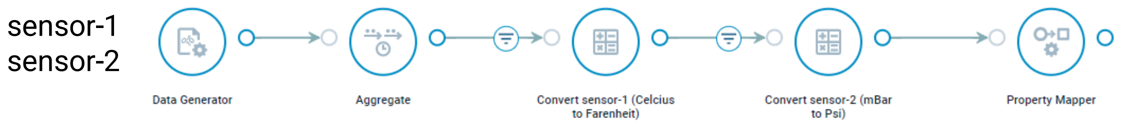


First flow

Single-path processing of multi-source data

Exercise 2.1

Overview



sensor-1
sensor-2
Data Generator
Aggregate
Convert sensor-1 (Celcius to Farenheit)
Convert sensor-2 (mBar to Psi)
Property Mapper

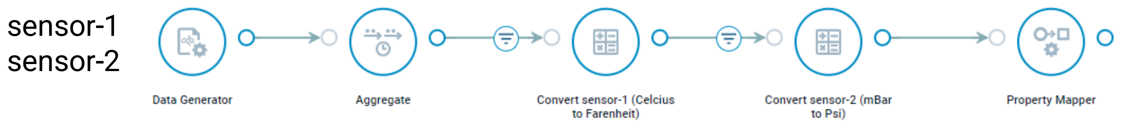
```
{
    "sensor-1": 77.48624
}
```

```
{
    "sensor-2": 17.17539
}
```

- Use internally generated data to simulate streaming data from two sensors (*sensor-1* and *sensor-2*)
- Average data over 10 seconds
- Convert the data (we will assume sensor-1 is temperature in Celsius and sensor-2 is pressure in Bar)
- Use message filters to separate data sources
- Reformat result

Exercise 2.1

Data Generator and Aggregate modules


sensor-1
sensor-2
Data Generator
Aggregate
Convert sensor-1 (Celcius to Farenheit)
Convert sensor-2 (mBar to Psi)
Property Mapper

```
{
    "sensor-1": 77.48624
}
```

```
{
    "sensor-2": 17.17539
}
```

After adding each module, run the flow in a sandbox and look at the output and make sure you understand how the settings affects the result

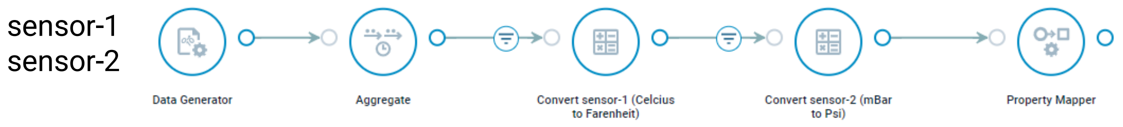
1. Create a new flow called Exercise 2.1
2. Add a Data Generator module with the following settings:
   - Number of samples: 2
   - Add JSON template from the box to the right and click on Update
   - Data Rules: data.name / Behavior: Identifier
3. Add an Aggregate module:
   - Source Property: data.name
   - Value Property: data.value
   - Target Property: stats
   - Interval: 10 seconds

Data Generator JSON template

```
{
  "data": {
    "name": "sensor",
    "value": 25.5
  }
}
```

Exercise 2.1

Math and Property Mapper modules


sensor-1
sensor-2
Data Generator
Aggregate
Convert sensor-1 (Celcius to Farenheit)
Convert sensor-2 (mBar to Psi)
Property Mapper

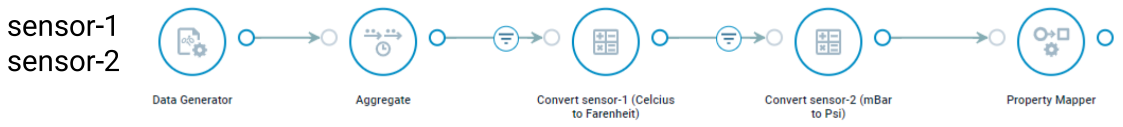
```
{
    "sensor-1": 77.48624
}
```

```
{
    "sensor-2": 17.17539
}
```

4. Add a Math module:
   - Add an Expression and set:
     - Target Property: convertedValue
     - Expression: {stats.average}*1.8+32
   - Add a Message Filter:
     - Source Property: stats.name, operator: Equal To value: sensor-1 Type: String
     - Bypass message to next module in flow if filters does not match: Enabled
5. Add a second Math module:
   - Add an expression and set:
     - Target Property: convertedValue
     - Expression: 14.503773773*{stats.average}/1000
   - Add a Message Filter:
     - Source Property: stats.name, operator: Equal To, value: sensor-2, Type: String
     - Bypass message to next module in flow if filters does not match: Enabled
6. Add a Property Mapper module:
   - Keep Properties: Disabled
   - Add Property: *New name:* {stats.name} *New value:* {convertedValue}

Exercise 2.1

Wrap-up


sensor-1
sensor-2
Data Generator
Aggregate
Convert sensor-1 (Celcius to Farenheit)
Convert sensor-2 (mBar to Psi)
Property Mapper

```
{
    "sensor-1": 77.48624
}
```

```
{
    "sensor-2": 17.17539
}
```

Things to test/consider:

- Why do we need filters on the Math modules?
- What happens if you turn off “Bypass message to next module in flow if filters does not match”?
- Why is the result in both Math modules assigned to the same property?
- Why is there {} around “stats.name” in the Property Mapper module?
- What happens if you enable “Keep Properties” on the Property Mapper module?

Second flow

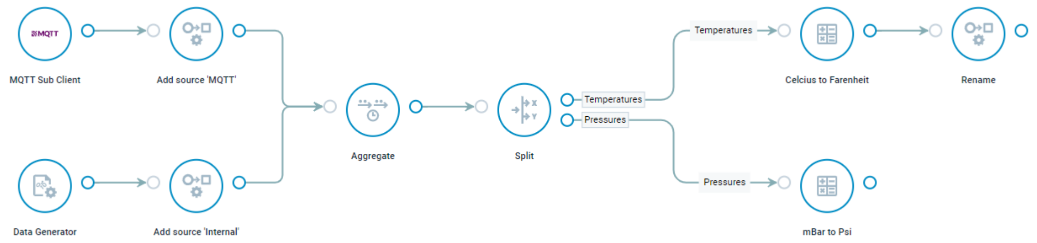
Multi-path processing of multi-source data

Exercise 2.2

Overview

temp3
pressure3

sensor-1
sensor-2



```
{
  "Temperature_MQTT": 77.48624
}
```

```
{
  "data": {
    "Pressure": 17.17539
  }
}
```

- Receive data from temperature and pressure sensors from two input sources every second
- Add metadata to separate inputs
- Split into separate paths based on naming
- Convert units: C → F and mBar → Psi
- Reformat result

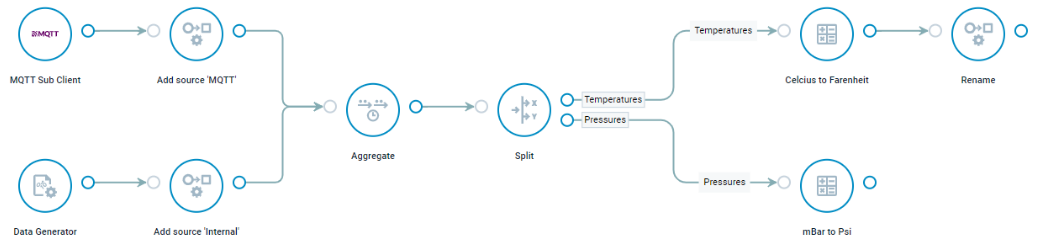
crosser

Exercise 2.2

MQTT and Property Mapper modules

temp3
pressure3

sensor-1
sensor-2



```
{
  "Temperature_MQTT": 77.48624
}
```

```
{
  "data": {
    "Pressure": 17.17539
  }
}
```

After adding each module, run the flow in a sandbox and look at the output and make sure you understand how the settings affects the result

1. Create a new flow from the Exercise 2.1 flow and change the name to Exercise 2.2
2. Add a MQTT Sub Client module:
   - Topic: sandbox
   - URL: 10.0.48.117
3. Add a Property Mapper module after each input module:
   - Add a Property called source and set the value to MQTT resp. Internal
4. Connect the outputs of the Property Mapper modules to the Aggregate module

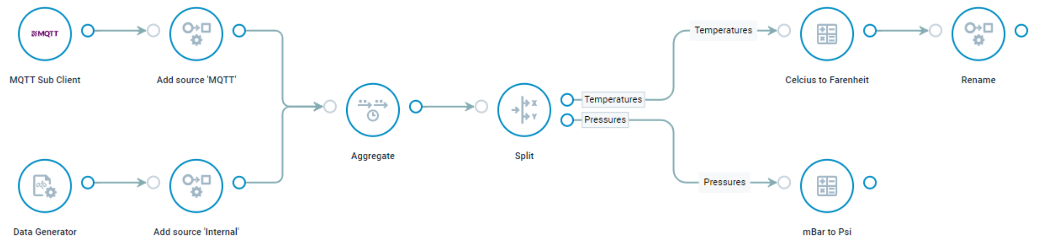
crosser

Exercise 2.2

Split module

temp3
pressure3

sensor-1
sensor-2



```
{
  "Temperature_MQTT": 77.48624
}
```

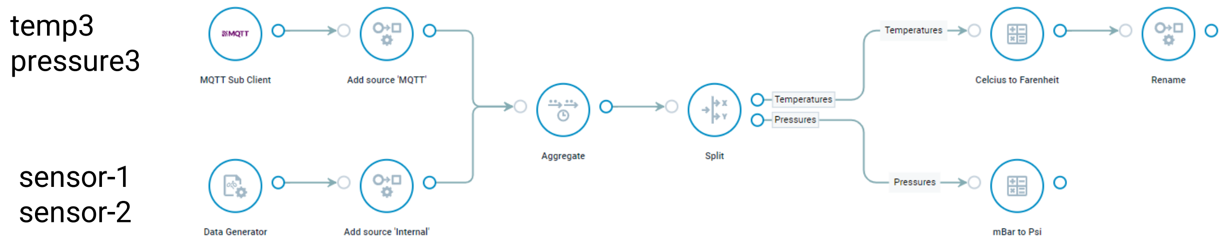
```
{
  "data": {
    "Pressure": 17.17539
  }
}
```

5. Add a Split module:
   - Add a first Condition Group:
     - Group Name: Temperatures
     - Rule: Or
     - First condition:
       - Name: stats.name
       - Operator: Contains
       - Value: temp
       - Type: String
     - Second condition:
       - Name: stats.name
       - Operator: Equal To
       - Value: sensor-1
       - Type: String
   - Add a second Condition Group:
     - Group Name: Pressures
     - Rule: Or
     - First condition:
       - Name: stats.name
       - Operator: Contains
       - Value: pressure
       - Type: String
     - Second condition:
       - Name: stats.name
       - Operator: Equal To
       - Value: sensor-2
       - Type: String

crosser

Exercise 2.2

Connect everything together



```
{
  "Temperature_MQTT": 77.48624
}
```

```
{
  "data": {
    "Pressure": 17.17539
  }
}
```

6. Remove the message filters from the Math modules and connect them to the respective outputs of the Split module. Also change the Target Property to data.Temperature and data.Pressure respectively

7. Add a Property Mapper module after the temperature conversion module:
   - Keep Properties: Disabled
   - Add Property: New name: Temperature_{source}, New value: {data.Temperature}

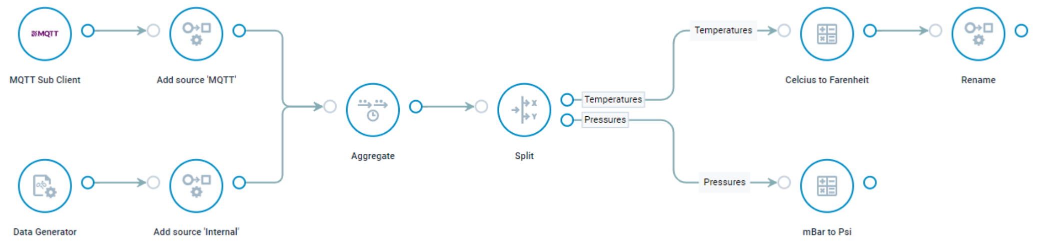
crosser

Exercise 2.2

Wrap-up

temp3
pressure3

sensor-1
sensor-2



```
{
  "Temperature_MQTT": 77.48624
}
```

```
{
  "data": {
    "Pressure": 17.17539
  }
}
```

Things to test/consider:

- Why do we <u>not</u> need filters on the Math modules now?
- How do you know which data that comes out through each output on the Split module?
- Why is {} used around both the *name* and the *value* settings in the last Property Mapper module?

SESSION – 03 END

How to work with external streaming inputs

Re-format messages

Split messages into different paths