




Hyper-**N**etwork for **e**lectro**M**obility

## NeMo Data Translators

### - Mapping creation guide

Work package	WP3: NeMo Data Management
Task	Task 3.2: Data Translators
Authors	Singular Logic
Dissemination level	Confidential (CO)
Status	Draft
Document date	24/05/2018
Version number	0.5
	<i>This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement no 713794.</i>





## Table of contents

1. Introduction .....	1
2. Data translation mapping by example .....	2
<i>Listing 1</i> .....	3
<i>Listing 2</i> .....	5
<i>Listing 3</i> .....	7
<i>Listing 4</i> .....	8
<i>Listing 5</i> .....	10
<i>Listing 6</i> .....	12

# 1. Introduction

The main role of Data Translators in NeMo is to transform messages between platform specific data models (PSMs) used as input and output data of registered services and the NeMo Common Information Model (CIM), so that services within the Hypernetwork are represented in a homogenous manner. In order to achieve the above, the Data Translator instance deployed at the NeMo node of the Service Provider (SP) owning the service must be configured accordingly, so that this service can be made seamlessly accessible. During this procedure the Data Translator is provided with the PSM-to-CIM mapping, which, expressed through the appropriate XML structure, reflects the exact relationships between the platform-specific data model based on which the service is implemented and the CIM, in both semantic and syntactic terms. Further, this mapping allows the SP to specify visibility levels for all data involved at a fine-grained level (expressed in CIM). Based on this information, the Data Translator internally generates the logic required to transform incoming requests and outgoing responses at invocation time, expressed in the Transformation Workflow Language.

The mapping XML file is generated during service creation time. This configuration is foreseen to be specified with the help of an XML Editor within the Eclipse Environment. The output of file will then be pushed to an artefacts server, which will be a Gitlab Server, where it is available for further processing during registration time.

The following section presents the main features of the XML language used for the mapping, along with examples illustrating said features and demonstrating how a complete mapping file for a service can be created.

## 2. Data translation mapping by example

The root of the translation XML file is a `<MAPPINGS>` element. This consists of a number of `<MAPPING>` elements defining the correspondences between any PSM and the CIM in a *bidirectional* way. More specifically, for each `<MAPPING>` element the user mandatorily specifies the following two sub-elements:

- the `<SOURCE_TYPE>` (multiple instances are allowed): this is the PSM data type which will be the source to translate to a corresponding CIM data object. It will be defined as a path, so as to indicate the hierarchical location of the transformation source. The optional attribute `data_type`, acquiring the corresponding string values, may be used in order to denote if the data in question will be represented as `string`, `double`, `int`, etc., for cases where this cannot be directly derived.
- the `<TARGET_TYPE>` (multiple instances are allowed): this is the CIM data type that will be derived as a result of the current mapping. Here, the attribute `visibility` can additionally be used so that the user can specify the visibility levels for all data involved at a fine-grained level (expressed in CIM). The `visibility` attribute can acquire one of two string values, `SP_ONLY` and `SP_CONTRACTS`. The former is used to indicate that the corresponding data object is accessible only by the current SP, by this or potentially any other of their services; the latter means that disclosure of the data object must be averted against any entity that has not signed a contract with the current SP. The translator, based on this indication, will automatically employ suitable security transformations through dedicated built-in functions each time data leave an SP's domain. The default value according to the Translator specification is `SP_CONTRACTS`, leaving it to the user to prescribe a stricter policy when needed.

Note that the reverse transformations, i.e., from CIM to PSM, will be derived automatically by the same mapping file by reversing `<SOURCE_TYPE>` and `<TARGET_TYPE>` and through additional adaptations explained in what follows.

From there on there are two possibilities; either the `<OPERATION>` to be performed on the source(s) in order to produce the target(s) is defined, or another `<MAPPINGS>` element is defined, with the latter encompassing a series of nested `<MAPPING>` elements, denoting hierarchical/recursive mapping definitions.

The listing below shows a very simple case where a PSM `DataAccessClearanceRequest` is mapped to a CIM object of the same name. After mapping the top-level objects, another `<MAPPINGS>` element is used to map source and target nested fields, in this case `clearanceId` and `clearanceId2`, respectively.

## Listing 1

```
<MAPPINGS xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="NeMo_mapping_v0.9.1.xsd">

  <MAPPING>

    <SOURCE_TYPE>DataAccessClearanceRequest</SOURCE_TYPE>

    <TARGET_TYPE>DataAccessClearanceRequest</TARGET_TYPE>

    <MAPPINGS>

      <MAPPING>

        <SOURCE_TYPE>DataAccessClearanceRequest/clearanceId</SOURCE_TYPE>

        <TARGET_TYPE>DataAccessClearanceRequest/clearanceId2</TARGET_TYPE>

        ...

      </MAPPING>

    </MAPPINGS>

  </MAPPING>

</MAPPINGS>
```

Upon reaching the innermost mapping level, the user specifies the `<OPERATION>`, an element defining the function that will eventually be used for the transformation. It will further specify the exact parameters to be used by each function and its exact output, among the ones potentially implied as part of the source(s) and target(s), through its sub-elements `<INPUT>` and `<OUTPUT>`. The optional string attributes `input_container` and `output_container` are used to cover cases where the input is part of the source (respectively, the output is part of the target) specified in the mapping, hence the sources and/or targets concerning this specific operation must be further distinguished. The optional attribute `data_type` may be used in order to denote if the data in question will be represented as `string`, `double`, `int`, etc., for cases where this cannot be directly derived. We currently distinguish between three types of operations.

- *Predefined operations* represent basic built-in functions, defined through the `<FUNCTION>` element together with its reverse if applicable (`<REVERSE_FUNCTION>` element), with the latter intended to be used for the reverse transformation, i.e., from CIM to PSM. For both regular and reverse functions, the user may define a `<FUNCTION_NAME>` and, optionally, one or more parameters through `<FUNCTION_PARAMETER>` elements. (any value type is allowed). The `<FUNCTION_NAME>` can acquire one of the following contents, corresponding to the built-in functions currently made available by the Translator: `VALUE_COPY`, `UPPER_CASE`, `LOWER_CASE`, `CAMEL_CASE`, `SPLIT` (requires separator value as parameter), `CONCAT` (requires separator value as parameter), `REPLACE` (requires regular expression and replacement parameters), `DEFAULT_VALUE` (requires the default value as parameter), `SUBSTRING` (requires the start and/or end index as parameter), `UUID` (creates a new UUID

value), *VALUE\_MULTIPLIER* (requires multiplier value as parameter, in case of not just multiplication between input values), *VALUE\_DIVIDER* (requires divider value as parameter, in case of not just division between input values). When the reverse function is used, inputs, outputs, input containers and output containers will be reversed as well.

- *Value mappers* define correspondence pairs between different enumerations (<MAP\_PAIRS> element). The <MAP\_PAIRS> element consists of a series of <MAP\_PAIR> elements, corresponding to single PIM to CIM value mapping. Each of these pairs defines a set of <SOURCE\_VALUE> and a set of <TARGET\_VALUE> (any value type is allowed). The former specifies the PSM value(s) while the latter the corresponding CIM value(s). Further, <MAP\_PAIR> selection priority can be specified by the optional attribute *priority* (integer), in order to cover cases where one PSM value set can possibly correspond to more than one CIM value set and vice versa.
- *Custom operations* allow the user to “load” the translator with custom transformation functions in order to cover more complex cases, indicating for each transformation case one class for both the regular and reverse conversions. For this type of operations, the string element <OPERATION\_CLASS> points to the implementation, so that the Translator can locate and invoke it when needed. <CONVERT\_PARAMETER> and <REVERSE\_CONVERT\_PARAMETER> (any value type is allowed) allow specifying necessary execution parameters distinctively for each conversion type.

Progressing the previous example, it is defined below that for the mapping of clearance IDs, the value of the source is simply copied to the target field.

## Listing 2

```
<MAPPINGS xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="NeMo_mapping_v0.9.1.xsd">

  <MAPPING>

    <SOURCE_TYPE>DataAccessClearanceRequest</SOURCE_TYPE>

    <TARGET_TYPE>DataAccessClearanceRequest</TARGET_TYPE>

    <MAPPINGS>

      <MAPPING>

        <SOURCE_TYPE>DataAccessClearanceRequest/clearanceId</SOURCE_TYPE>

        <TARGET_TYPE>DataAccessClearanceRequest/clearanceId2</TARGET_TYPE>

        <OPERATION xsi:type="PredefinedOperationType">

          <INPUT>DataAccessClearanceRequest/clearanceId</INPUT>

          <OUTPUT>DataAccessClearanceRequest/clearanceId2</OUTPUT>

          <FUNCTION>

            <FUNCTION_NAME>VALUE_COPY</FUNCTION_NAME>

          </FUNCTION>

          <REVERSE_FUNCTION>

            <FUNCTION_NAME>VALUE_COPY</FUNCTION_NAME>

          </REVERSE_FUNCTION>

        </OPERATION>

      </MAPPING>

    </MAPPINGS>

  </MAPPING>

</MAPPINGS>
```

Further, the user is enabled to specify conditions that must be applicable to source types in order for the transformation to occur, through the `<CONDITION>` element. Simple conditions consist of a `<SUBJECT>`, an `<OPERATOR>` and a `<VALUE>`. The `<SUBJECT>` element defines the source of the data that the condition applies to. The optional string attribute `subject_container` is used to denote the specific source that the condition applies to, in case the condition can not be applied directly to any source of the mapping. The `<VALUE>` can be of any type, while the `<OPERATOR>` can take one of the following string values: *lessThan*, *greaterThan*, *lessEqual*, *greaterEqual*, *noEqual*, *equalTo*, *in*, *exists*. The *in* operator allows the definition of complex conditions where the condition's value should belong to the specified set(s) of values. For the reverse transformation, the `<CONDITION>` element in fact implies that the target type to be created must fulfil the specified constraints.

Presenting a more collective example, the listing below shows part of the mapping definition in order to obtain an `EVSE` and associated business objects, as defined in the CIM, from a list of `EVSERichDescrip` elements described according to the eMIP protocol<sup>1</sup>. The EVSE (Electronic Vehicle Supply Equipment) or Charging Point is a system than can charge one EV at a time, and is managed by the corresponding CPO (Charge Point Operator).

First, it is defined that each `EVSERichDescrip` within `EVSERichDescripList` corresponds to a CIM `EVSE`. Proceeding with filling in each `EVSE` object, a specific `EVSEAttribute` from the `EVSEAttributeList` comprising `EVSERichDescrip` is mapped to the CIM `EVSEID`; the attribute is chosen among all others in the list based on its `attributeId` (3001) through the corresponding condition. The operation used for the mapping is a predefined one, specifically `VALUE_COPY`, meaning that the `attributeValue` (specified as `INPUT`, where the `input_container` is the same as the `SOURCE_TYPE`) will be used as is to form the `EVSEID`; naturally the reverse function is the same in this case.

---

<sup>1</sup> This, as well as all other examples that follow, are taken from the mapping xml file used for the configuration of the translator regarding the provision of the eMIP web service named "eMIP\_ToIOP\_GetEVSEData\_FullList", used by an eMSP to get data of all EVSEs he can access.



### Listing 3

```
<MAPPINGS xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="NeMo_mapping_v0.9.1.xsd">

  <MAPPING>

    <SOURCE_TYPE>EVSERichDescripList/EVSERichDescrip</SOURCE_TYPE>
    <TARGET_TYPE>EVSE</TARGET_TYPE>
    <MAPPINGS>
      <MAPPING>
        <SOURCE_TYPE>EVSERichDescripList/EVSERichDescrip/EVSEAttributeList/EVSEAttribute</SOURCE_TYPE>
        <TARGET_TYPE>EVSE/EVSEID</TARGET_TYPE>
        <OPERATION xsi:type="PredefinedOperationType">
          <INPUT
input_container="EVSERichDescripList/EVSERichDescrip/EVSEAttributeList/EVSEAttribute"/>attributeValue</INPUT>
          <OUTPUT>EVSE/EVSEID</OUTPUT>
          <FUNCTION>
            <FUNCTION_NAME>VALUE_COPY</FUNCTION_NAME>
          </FUNCTION>
          <REVERSE_FUNCTION>
            <FUNCTION_NAME>VALUE_COPY</FUNCTION_NAME>
          </REVERSE_FUNCTION>
        </OPERATION>
        <CONDITION xsi:type="SimpleConditionType">
          <SUBJECT
subject_container="EVSERichDescripList/EVSERichDescrip/EVSEAttributeList/EVSEAttribute"/>attributeId</SUBJECT>
          <OPERATOR>equalTo</OPERATOR>
          <VALUE>3001</VALUE>
        </CONDITION>
      </MAPPING>
    </MAPPINGS>
  </MAPPING>
</MAPPINGS>
```

Below it is shown how the mapping between enumerated values is performed. The value of EVSEAttribute with id 3041 corresponds to field OperState attribute of the OperationalState BO in CIM. But since the permitted values are different in the two data models and the user knows the correspondence, they specify a number of MAP\_PAIR elements so that the Translator can infer at invocation time the value of the OperState (TARGET\_VALUE) based on the attributeValue of said EVSEAttribute (SOURCE-VALUE), and vice versa.

#### Listing 4

```

<MAPPING>
  <SOURCE_TYPE>EVSERichDescripList/EVSERichDescrip/EVSEAttributeList/EVSEAttribute</SOURCE_TYPE>
  <TARGET_TYPE>EVSE/OperationalState/OperState</TARGET_TYPE>
  <OPERATION xsi:type="ValueMapperOperationType">
    <INPUT input_container="EVSERichDescripList/EVSERichDescrip/EVSEAttributeList/
EVSEAttribute"/>attributeValue</INPUT>
    <OUTPUT>EVSE/OperationalState/OperState</OUTPUT>
    <MAP_PAIRS>
      <MAP_PAIR>
        <SOURCE_VALUE>0</SOURCE_VALUE>
        <TARGET_VALUE>1</TARGET_VALUE>
      </MAP_PAIR>
      <MAP_PAIR>
        <SOURCE_VALUE>1</SOURCE_VALUE>
        <TARGET_VALUE>2</TARGET_VALUE>
      </MAP_PAIR>
      <MAP_PAIR>
        <SOURCE_VALUE>2</SOURCE_VALUE>
        <TARGET_VALUE>3</TARGET_VALUE>
      </MAP_PAIR>
      <MAP_PAIR>
        <SOURCE_VALUE>3</SOURCE_VALUE>
        <TARGET_VALUE>7</TARGET_VALUE>
      </MAP_PAIR>
    </MAP_PAIRS>
  </OPERATION>
  <CONDITION xsi:type="SimpleConditionType">
    <SUBJECT subject_container="EVSERichDescripList/EVSERichDescrip/EVSEAttributeList/
EVSEAttribute"/>attributeId</SUBJECT>
    <OPERATOR>equalTo</OPERATOR>
    <VALUE>3041</VALUE>
  </CONDITION>
</MAPPING>

```

In Listing 5, the use of two specific `data_type` attributes, `array` and `inner_array`, is showcased. The former, characterising the top `TARGET_TYPE`, indicates that from `EVSEAttribute` elements with `attributeIds` in the range 4001-4082 (as inferred by the condition in the end of the listing) an array of `ChargingConnector` BOs is created, rather than one single `ChargingConnector`. The nested mappings define how two fields of each `ChargingConnector` are created, `TypeOfConnector` and `MaxPower`. Here, data type `inner_array` in the `SOURCE_TYPE` denotes that each selected `EVSEAttribute` consists in itself in an array of elements, so that each `attributeValue` showing a `TypeOfConnector` is combined with the `attributeValue` showing to a `MaxPower` and holding the same position in its corresponding inner array. Schematically, the source and target structures are as follows:

- eMIP

```
<EVSEAttribute>
  <attributeId>4021</attributeId>
  <attributeValue>1</attributeValue>
  <attributeValue>2</attributeValue>
</EVSEAttribute>
<EVSEAttribute>
  <attributeId>4043</attributeId>
  <attributeValue>0.000</attributeValue>
  <attributeValue>5.000</attributeValue>
</EVSEAttribute>
```

- CIM

```
<ChargingConnector>
  <TypeOfConnector>1</attributeId>
  <MaxPower>0</attributeValue>
</ChargingConnector>
<ChargingConnector>
  <TypeOfConnector>2</attributeId>
  <MaxPower>5000</attributeValue>
</ChargingConnector>
```

## Listing 5

```
<MAPPING>
  <SOURCE_TYPE>EVSERichDescripList/EVSERichDescrip/EVSEAttributeList/EVSEAttribute</SOURCE_TYPE>

  <TARGET_TYPE data_type="array">EVSE/ChargingConnector</TARGET_TYPE>

  <MAPPINGS>
    <MAPPING>
      <SOURCE_TYPE
data_type="inner_array">EVSERichDescripList/EVSERichDescrip/EVSEAttributeList/EVSEAttribute</SOURCE_TYPE>

      <TARGET_TYPE>EVSE/ChargingConnector/TypeOfConnector</TARGET_TYPE>

      <OPERATION xsi:type="ValueMapperOperationType">
        <INPUT input_container="EVSERichDescripList/EVSERichDescrip/EVSEAttributeList/EVSEAttribute">/attributeValue</INPUT>

        <OUTPUT>EVSE/ChargingConnector/TypeOfConnector</OUTPUT>

        <MAP_PAIRS>
          <MAP_PAIR>
            <SOURCE_VALUE>2</SOURCE_VALUE>
            <TARGET_VALUE>21</TARGET_VALUE>
          </MAP_PAIR>
          <MAP_PAIR>
            <SOURCE_VALUE>5</SOURCE_VALUE>
            <TARGET_VALUE>19</TARGET_VALUE>
          </MAP_PAIR>
          <MAP_PAIR>
            <SOURCE_VALUE>6</SOURCE_VALUE>
            <TARGET_VALUE>20</TARGET_VALUE>
          </MAP_PAIR>
        </MAP_PAIRS>
      </OPERATION>

      <CONDITION xsi:type="SimpleConditionType">
        <SUBJECT subject_container="EVSERichDescripList/EVSERichDescrip/EVSEAttributeList/EVSEAttribute">/attributeId</SUBJECT>

        <OPERATOR>equalTo</OPERATOR>

        <VALUE>4021</VALUE>
      </CONDITION>
    </MAPPING>
  </MAPPINGS>
</MAPPING>
```

```

<MAPPING>
  <SOURCE_TYPE data_type="inner_array">EVSERichDescripList/
  EVSERichDescrip/EVSEAttributeList/EVSEAttribute</SOURCE_TYPE>
  <TARGET_TYPE>EVSE/ChargingConnector/MaxPower</TARGET_TYPE>
  <OPERATION xsi:type="PredefinedOperationType">
    <INPUT input_container="EVSERichDescripList/EVSERichDescrip/
  EVSEAttributeList/EVSEAttribute"/>attributeValue</INPUT>
    <OUTPUT>EVSE/ChargingConnector/MaxPower</OUTPUT>
    <FUNCTION>
      <FUNCTION_NAME>VALUE_MULTIPLIER</FUNCTION_NAME>
      <FUNCTION_PARAMETER>1000</FUNCTION_PARAMETER>
    </FUNCTION>
    <REVERSE_FUNCTION>
      <FUNCTION_NAME>VALUE_DIVIDER</FUNCTION_NAME>
      <FUNCTION_PARAMETER>1000</FUNCTION_PARAMETER>
    </REVERSE_FUNCTION>
  </OPERATION>
  <CONDITION xsi:type="SimpleConditionType">
    <SUBJECT subject_container="EVSERichDescripList/
  EVSERichDescrip/ EVSEAttributeList/EVSEAttribute"/>attributeId</SUBJECT>
    <OPERATOR>equalTo</OPERATOR>
    <VALUE>4043</VALUE>
  </CONDITION>
</MAPPING>
</MAPPINGS>
<CONDITION xsi:type="SimpleConditionType">
  <SUBJECT subject_container="EVSERichDescripList/EVSERichDescrip
  /EVSEAttributeList/EVSEAttribute"/>attributeId</SUBJECT>
  <OPERATOR>in</OPERATOR>
  <VALUE>(4001-4082)</VALUE>
</CONDITION>
</MAPPING>

```

The following excerpt shows how custom functions can be declared for transformations between source and target values. With custom operations, it is assumed that the logic for the reverse transformation is contained in the same class, differentiating what is being executed based on whether it receives the `INPUT` or the `OUTPUT` (in the reverse case), i.e., on the translation direction (PSM to CIM or CIM to PSM).

### Listing 6

```
...
<MAPPING>
  <SOURCE_TYPE
data_type="inner_array">EVSERichDescripList/EVSERichDescrip/EVSEAttributeList/EVSEAttribute</S
OURCE_TYPE>

  <TARGET_TYPE>EVSE/ChargingStation/ChargingPool/OpenHours</TARGET_TYPE>

  <OPERATION xsi:type="CustomOperationType">
    <INPUT
input_container="EVSERichDescripList/EVSERichDescrip/EVSEAttributeList/EVSEAttribute"></INPUT
>
    <OUTPUT>EVSE/ChargingStation/ChargingPool/OpenHours</OUTPUT>
    <OPERATION_CLASS>eu.nemo.translator.operation.gireve.OpenHoursConverter
</OPERATION_CLASS>
  </OPERATION>

  <CONDITION xsi:type="SimpleConditionType">
    <SUBJECT subject_container="EVSERichDescripList/EVSERichDescrip
/EVSEAttributeList/EVSEAttribute">/attributeId</SUBJECT>

    <OPERATOR>in</OPERATOR>

    <VALUE>(1101,1103\ -2,1103\ -3,1103\ -4) </VALUE>

  </CONDITION>
</MAPPING>
...
```