**UML-YANG Mapping Tool User Guide**

# Overview

The UML-YANG mapping tool translates a UML (Unified Modeling Language) model to a YANG model defined in RFC6020. The output YANG files can be conveniently used in REST style API definition.

# Mapping Rules

The mapping rules of this UML-YANG mapping tool is based on ONF onf2015.261\_Mapping\_Gdls\_UML-YANG.04, IETF RFC 6020 and OMG Unified Modeling Language TM (OMG UML) Version 2.5.

# Programing Language

Programming language：JavaScript

Running environment: node.js

# History

|  |  |
| --- | --- |
| **Date** | **Features Added/Issues Fixed** |
| May 17 | Support units substatements.  Remove config if it is true.  Remove order-by if it is ordered-by system. |
| May 13 | Fix multiple “range” bug.  Support default value.  Fix upperValue issue, change “\*” to “max”. |
| May 6 | Fix “range” and “enum” bugs. |
| Apr 19 | The tool now translates "context" to "Container Context". |
| Mar 29 | Remove prefix “G\_” for grouping. |
| Mar 16 | Add package dependency and support single literal enumeration. |
| Mar 17 | Resolve rpc multiplixity issue. |
|  | In version 1.3, we add the following features:   * Add package dependency. The use don’t need to install xmlreader again! * Support single literal in enumeration * Support new OpenModelProfile StereoType “isPartOfObjectKey” * Support multiple keys for an object class. * Support “self-composite” and “cross-reference”. * Compatible with both lowercase and uppercase stereotype names in OpenModelProfile. * Add “revision” statement. * Remove “key.cfg” file. * Resolve “revision” issue and “namespace” issue. |
|  | In version 1.2, we add the following functions:   * Supports both UML and XML format as input. * Realize the notification mapping. * Realize the “support” and “condition” mappings. * Realize the “isOrdered” “valueRange” “passedByReference” mappings of attribute. * Realize the “lifecycle stereotypes” mapping. * Flexible “key” adding function: the tool supports self-defined “key” value by writing some “key” information in key.cfg file. * Error alert functionality: cross-reference among “yang module” files. * Fixed some undefined type error by adding “string” type to .xml files. * Fixed the bug that some xml file couldn’t create yang files caused by different platforms. * Fixed the UML element “datatype” mapping issue,adding some complex datatype mapping. * Add the function that Unify the YANG identifiers naming conventions when appear the illegal characters by underline. |

# Architecture and Components

Figure 1 illustrates the project structure of the UML-to-YANG mapping tool. The main functions and project entry is in the main.js, which uses UML files as input and generates yang files as output. The left side modules of the main doc.js relate to UML object classes extraction and analysis. The right side modules of the main.js relate to yang objects generation.



Figure 1 Project Files of UML-to-YANG Mapping Tool

Below is the description of the files and directories in the project.

* /main.js This is the project entry. This file contains main processing functions of UML to yang.
* /model This directory includes multiple files related to UML and YANG model.
* /model /Association.js This file extracts the “association” in UML file.
* /model /ObjectClass.js This file extracts “objectClass” in UML file.
* /model /ownedAttribute.js This file extracts the “ownedAttribute” in UML file.
* /model /OpenModelObject.js This file extracts the “OpenModelProfile” content in

UML file.

* /model /yang This directory includes the files related to yang elements.
  + /leaf.js This file translates the yang “leaf” node.
  + /leaf\_list.js This file translates the yang “leaf-list” node.
  + /module.js This file translates the yang “module”.
  + /node.js This file translates the yang “node”.
  + /rpc.js This file translates yang “rpc”.
  + /type.js This file translates yang “type”.
  + /uses.js This file translates yang “uses”.
  + /feature.js This file translates yang “feature”.
* /project This directory stores UML files and the generated yang files.
* package.json This file adds node module dependency.

# Implemented Mapping Rule

**Object Class mapping**

|  |  |  |
| --- | --- | --- |
| UML Artifact | YANG Artifact | remark |
| comment | “description” substatement |  |
| isReadOnly | “config”substatement | isReadOnly=true  config=false;  isReadOnly=false  config=true |
| abstract | "grouping" statement | abstract =true  can’t be Instantiated;  abstract =false  should be Instantiated; |
| support | “if-feature” substatement |  |
| condition |  |

**Attribute Mapping**

|  |  |  |
| --- | --- | --- |
| UML Artifact | YANG Artifact | remark |
| comment | “description” substatement |  |
| type | “type” substatement | built-in or derived type |
| isReadOnly | “config” substatement | isReadOnly=false  config=true;  isReadOnly=true  config=false |
| Multiplicity  (upperValue  lowerValue) | “mandatory” or “min-elements” and “max-elements” substatements [0..1] => no mapping needed; is default [1] => mandatory substatement = true [0..x] => no mapping needed; is default [1..x] => min-elements substatement = 1 [0..3] => max-elements substatement = 3 |  |
| defaultValue | "default" substatement |  |
| isOrdered | “ordered-by” substatement ("system" or "user”) |  |
| valueRange | “range” or “length” substatement of “type” substatement |  |
| passedByReference | if passedByReference = true 🡪 type leafref { path “/<object>/<object identifier>"  if passedByReference = false 🡪 either “list” statement (key property, multiple instances) or “container” statement (single instance) |  |
| support | “if-feature” substatement |  |
| condition |  |

**Type Mapping**

|  |  |  |
| --- | --- | --- |
| UML Artifact | YANG Artifact | remark |
| Primitive Type | Built-In Type if defined; |  |
| Enumeration | “enum” statement |  |
| Data Type | “typeDef” statement or grouping statement |  |

**Enumeration Type Mapping**

|  |  |  |
| --- | --- | --- |
| UML Artifact | YANG Artifact | remark |
| comment | “description” substatement |  |
| literal name | enum name |  |
| literal integer | “value” substatement |  |
| defaultValue | "default" substatement |  |

**Data Type Mapping**

|  |  |  |
| --- | --- | --- |
| UML Artifact | YANG Artifact | remark |
| comment | “description” substatement |  |
| type | “type” substatement(built-in type) |  |
| defaultValue | "default" substatement |  |
| ownedAttribute | The same as Attribute Mapping |  |

**Association Mapping**

|  |  |  |
| --- | --- | --- |
| UML Artifact | YANG Artifact | remark |
| generalization | “uses” substatement |  |
| aggregation =composition | “uses” substatement |  |
| aggregation=shared | “leafref” statement |  |

**Operation Mapping**

|  |  |  |
| --- | --- | --- |
| UML Artifact | YANG Artifact | remark |
| comment | “description” substatement |  |
| input parameter | “input” substatement |  |
| output parameter | “output” substatement |  |
| lifecycle stereotypes | “status” substatement | This artifact appears in OpenModelProfile and this work is in progress. |
| support | “if-feature” substatement | This artifact appears in OpenModelProfile and this work is in progress. |
| condition | This artifact appears in OpenModelProfile and this work is in progress. |

**Parameter Mapping**

|  |  |  |
| --- | --- | --- |
| UML Artifact | YANG Artifact | remark |
| comment | “description” substatement |  |
| Direction(in/out) | “input”/“output”substatement |  |
| type | see mapping of attribute types (grouping, leaf, leaf-list, container, list, typedef, uses) |  |
| multiplicity |  |
| defaultValue |  |
| complex parameter | “uses” substatement |  |

**Notification Mapping**

|  |  |  |
| --- | --- | --- |
| UML Artifact | YANG Artifact | Comments |
| common | “description” substatement |  |
| support | “if-feature” substatement | Support and condition belong together. If the “support” is conditional, then the “condition” explains the conditions under which the class has to be supported. |
| condition |
| lifecycle stereotypes | “status” substatement | "current", "deprecated", "obsolete", default = current |
| attributes | see mapping of attribute types (grouping, leaf, leaf-list, container, list, typedef, uses) |  |
| complex attribute | “uses” substatement |  |

**Other Mappings**

|  |  |  |
| --- | --- | --- |
| UML Lifecycle | | |
| UML Artifact | YANG Artifact | Comments |
| Lifecycle Stereotypes | “status“ substatement | YANG: "current", "deprecated", "obsolete", default = current  UML: «Example», «Experimental», «Faulty», «LikelyToChange», «Deprecated», «Obsolete», «Preliminary» |

|  |  |  |
| --- | --- | --- |
| Interface | | |
| UML Artifact | YANG Artifact | Comments |
| Comment | “description” substatement |  |
| interface | Module |  |

# Not Implemented Mapping Rule

**Object Class Mapping**

|  |  |  |
| --- | --- | --- |
| UML Artifact | YANG Artifact | Reason |
| objectCreationNotification [YES/NO/NA] | “notification” statement | This artifact appears in OpenModelProfile and this work is in progress. |
| objectDeletionNotification [YES/NO/NA] | “notification” statement | This artifact appears in OpenModelProfile and this work is in progress. |
| operation | “action” substatement | Could ObjectClass appear this form? |

**Attribute Mapping**

|  |  |  |
| --- | --- | --- |
| UML Artifact | YANG Artifact | Reason |
| isInvariant | “extension” substatement 🡪 ompExt:isInvariant | This artifact appears in OpenModelProfile and this work is in progress. |

**Operation Mapping**

|  |  |  |
| --- | --- | --- |
| UML Artifact | YANG Artifact | Reason |
| pre-condition | “extension” substatement🡪 ompExt: preCondition | Current UML model doesn’t have this artifact. |
| post-condition | “extension” substatement🡪 ompExt: postCondition | Current UML model doesn’t have this artifact. |
| operation exceptions | “extension” substatement🡪 ompExt:operationExceptions | Current UML model doesn’t have this artifact. |
| isOperationIdempotent | “extension” substatement🡪 ompExt:isOperationIdempotent | This artifact appears in OpenModelProfile and this work is in progress. |
| isAtomic | “extension” substatement🡪 ompExt:isAtomic | Current UML model doesn’t have this artifact. |

# How to Run This Tool

Running the UML-YANG mapping tool takes the following steps.

* Step 1: Download nodejs from <https://nodejs.org/en/> , choose the latest version for your system.
* Step 2: Install nodesjs by double click the "node-vxxx.msi" file. Use the default options are fine.
* Step 3: Open the terminal window in your system. If it’s Windows, run "cmd" to open the command line window.
* Step 4: In the command line, go to the directory you put "mapping tool". The format is “..\ AGLE-UML\_YANG-Mapping-Tooling\xmi2yang tool-vxx”. For example, I put this tool in D: work\Github\EAGLE-UML\_YANG-Mapping-Tooling\xmi2yang tool-vxx.
* Step 5: The uml files should be copied to the "project" folder. Please create the project folder if it does not exist under the “xmi2yang tool-vxx” folder. Note that if “a.uml” depends on other files, they need to be copied to the “project” folder as well.
* Step 6: The user can use the following command under this directory to run this tool.

*node main.js*

* Step 7: After running the tool, the .yang files will be generated by the mapping tool.

**About “key.cfg”(This file is removed in Version 1.3 because the tool now support “isPartOfKey” now.)**

a. The purpose of "key.cfg"

The implementation of "key.cfg" is a new feature in version 1.2, which is useful in the extension of CIM. "key.cfg" stores the keys that will appear in the transferred YANG files so that we can define keys flexibly. The "list" element in YANG requires a "key" value.

b. How to use "key.cfg"

Our objective is to add "key" values to "key.cfg". It takes the following procedures.

1) Find the multi-instance object class in your information model. For example, the “Topology” class is a multi-instance object class.

2) Identify the “key” value of this “List” element. For example, the “key” value for “Topology” is “uuid”.

3) Write the object class name “Topology” and the key attribute “uuid” as an entry in “key.cfg”. Multiple entries should be separated with a comma.

*[{"name":"Tapi::Topology","key":"uuid"},{……}]*

# Recommendation

We suggest the UML element identifiers are named according to YANG syntax (refer to chapter 6, RFC 6020). YANG syntax is similar to that of SMIng (chapter2.1, RFC3780). The detailed suggestion is as follows.

* Each identifier starts with an upper-case or lower-case character, dependent on the kind of SMIng item, followed by zero or more letters, digits, and hyphens.
* All identifiers defined in a namespace MUST be unique and SHOULD NOT only differ in case. Identifiers MUST NOT exceed 64 characters in length. Furthermore, the set of all identifiers defined in all modules of a single standardization body or organization SHOULD be unique and mnemonic. This promotes a common language for humans to use when discussing a module.
* When identifiers from external modules are referenced, there is the possibility of name collisions. As such, if different items with the same identifier are imported or if imported identifiers collide with identifiers of locally defined items, then this ambiguity is resolved by prefixing those identifiers with the names of their modules andthe namespace operator `::', i.e., `Module::item'. Of course, this notation can be used to refer to identifiers even when there is no name collision.