**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

# 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.

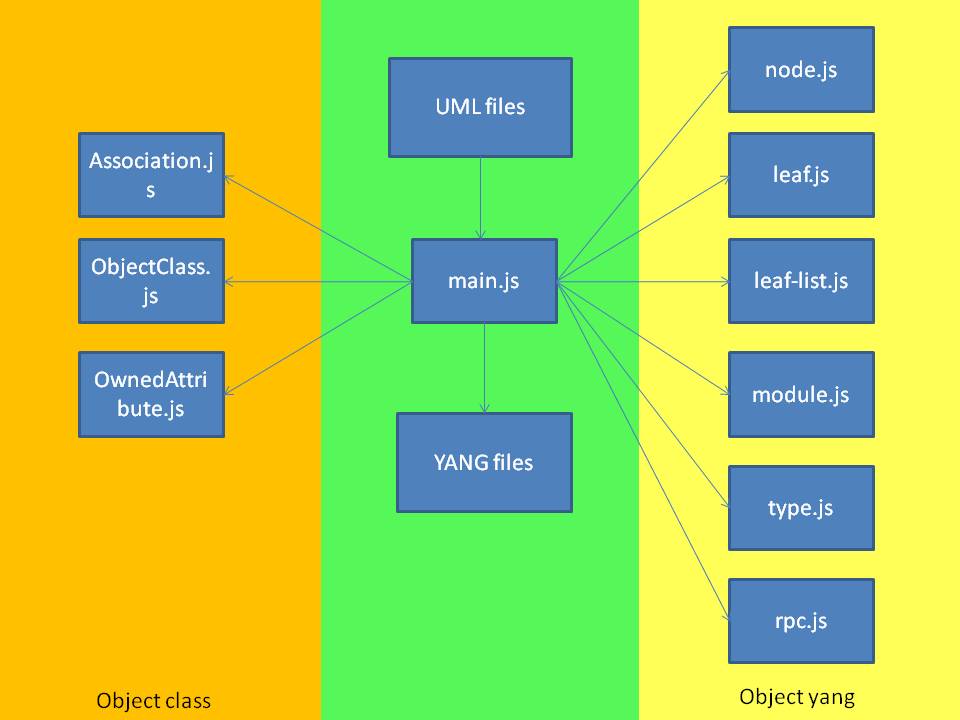


Figure Project Files of UML-to-YANG Mapping Tool

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 /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”.
* /project This directory includes UML files and the generated yang files.

# Implemented Mapping Rules

**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; |

**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 |  |

**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 |  |

**Interface Mapping**

| UML Artifact | YANG Artifact | remark |
| --- | --- | --- |
| comment | “description” substatement |  |

**Operation Mapping**

|  |  |  |
| --- | --- | --- |
| UML Artifact | YANG Artifact | remark |
| comment | “description” substatement |  |
| input parameter | “input” substatement |  |
| output parameter | “output” substatement |  |

**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 |  |

# Mapping Rules Not Implemented Yet

**Object Class Mapping**

|  |  |  |
| --- | --- | --- |
| UML Artifact | YANG Artifact | Remark |
| objectCreationNotification [YES/NO/NA] | “notification” statement | In progress. |
| objectDeletionNotification [YES/NO/NA] | “notification” statement | In progress. |
| support | “if-feature” substatement | In progress. |
| condition | In progress. |
| operation | “action” substatement | CIM UML doesn’t have this artifact. |

**Attribute Mapping**

|  |  |  |
| --- | --- | --- |
| UML Artifact | YANG Artifact | Remark |
| isOrdered | “ordered-by” substatement ("system" or "user”) | Just use “system” by default? |
| isInvariant | “extension” substatement 🡪 ompExt:isInvariant | In progress. |
| valueRange | “range” or “length” substatement of “type” substatement | In progress. |
| 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) | In progress. |
| support | “if-feature” substatement | In progress. |
| condition | In progress. |

**Operation Mapping**

|  |  |  |
| --- | --- | --- |
| UML Artifact | YANG Artifact | Remark |
| pre-condition | “extension” substatement🡪 ompExt: preCondition | Current CIM and TAPI model did not use this artifact. |
| post-condition | “extension” substatement🡪 ompExt: postCondition | Current CIM and TAPI model did not use this artifact. |
| operation exceptions | “extension” substatement🡪 ompExt:operationExceptions | Current CIM and TAPI model did not use this artifact. |
| isOperationIdempotent | “extension” substatement🡪 ompExt:isOperationIdempotent | In progress. |
| isAtomic | “extension” substatement🡪 ompExt:isAtomic | Current CIM and TAPI model did not use this artifact. |
| support | “if-feature” substatement | In progress. |
| condition | In progress. |
| hyperlink | “reference” substatement | Current CIM and TAPI model did not use this artifact. |
| lifecycle stereotypes | “status” substatement | In progress. |

# How to Run This Tool

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

* Step 1: The user should download Node.js source code or a pre-build installer for your platform before you run this tool. The users should also install the xmlreader (a Node Packaged Module) with the following command.

*npm install xmlreader*

* Step 2: The given uml files should be first renamed to .xml files and then copied to the "project" folder, e.g. a.uml should be renamed to a.xml. Note that if “a.xml” depends on other files, they need to be copied to the “project” folder as well.
* Step 3: The user should specify the file needed to be transferred in the following texts in main.js. The texts below refers the “CoreModel.xml” file.

*var xml = fs.readFileSync("./project/CoreModel.xml", {encoding: 'utf8'});*

* Step 4: The user can use the following command under this directory to run this tool.

*node main.js*

* Step 5: After running the tool, the .yang files will be generated by the mapping tool.