

CIMSYNTAXGEN MANUAL

v10.4.0 | 10 September 2026

For CimSyntaxGen software version 4.0.17

ICTC approved on 10 September 2026

ENTSO-E Mission Statement

Who we are

ENTSO-E, the European Network of Transmission System Operators for Electricity, is the association for the cooperation of the European transmission system operators (TSOs). The 40 member TSOs, representing 36 countries, are responsible for the secure and coordinated operation of Europe's electricity system, the largest interconnected electrical grid in the world. In addition to its core, historical role in technical cooperation, ENTSO-E is also the common voice of TSOs.

ENTSO-E brings together the unique expertise of TSOs for the benefit of European citizens by keeping the lights on, enabling the energy transition, and promoting the completion and optimal functioning of the internal electricity market, including via the fulfilment of the mandates given to ENTSO-E based on EU legislation.

Our mission

ENTSO-E and its members, as the European TSO community, fulfil a common mission: Ensuring the security of the inter-connected power system in all time frames at pan-European level and the optimal functioning and development of the European interconnected electricity markets, while enabling the integration of electricity generated from renewable energy sources and of emerging technologies.

Our vision

ENTSO-E plays a central role in enabling Europe to become the first climate-neutral continent by 2050 by creating a system that is secure, sustainable and affordable, and that integrates the expected amount of renewable energy, thereby offering an essential contribution to the European Green Deal. This endeavour requires sector integration and close cooperation among all actors.

Europe is moving towards a sustainable, digitalised, integrated and electrified energy system with a combination of centralised and distributed resources. ENTSO-E acts to ensure that this energy system keeps consumers at its centre and is operated and developed with climate objectives and social welfare in mind.

ENTSO-E is committed to use its unique expertise and system-wide view – supported by a responsibility to maintain the system's security – to deliver a comprehensive roadmap of how a climate-neutral Europe looks.

Our values

ENTSO-E acts in solidarity as a community of TSOs united by a shared responsibility.

As the professional association of independent and neutral regulated entities acting under a clear legal mandate, ENTSO-E serves the interests of society by optimising social welfare in its dimensions of safety, economy, environment, and performance.

ENTSO-E is committed to working with the highest technical rigour as well as developing sustainable and innovative responses to prepare for the future and overcoming the challenges of keeping the power system secure in a climate-neutral Europe. In all its activities, ENTSO-E acts with transparency and in a trustworthy dialogue with legislative and regulatory decision makers and stakeholders.

Our contributions

ENTSO-E supports the cooperation among its members at European and regional levels. Over the past decades, TSOs have undertaken initiatives to increase their cooperation in network planning, operation and market integration, thereby successfully contributing to meeting EU climate and energy targets.

To carry out its legally mandated tasks, ENTSO-E's key responsibilities include the following:

- › Development and implementation of standards, network codes, platforms and tools to ensure secure system and market operation as well as integration of renewable energy;
- › Assessment of the adequacy of the system in different timeframes;
- › Coordination of the planning and development of infrastructures at the European level (Ten-Year Network Development Plans, TYNDPs);
- › Coordination of research, development and innovation activities of TSOs;
- › Development of platforms to enable the transparent sharing of data with market participants.

ENTSO-E supports its members in the implementation and monitoring of the agreed common rules.

TABLE OF CONTENTS

CIMSyntaxGen manual.....	1
<i>Table of Contents</i>	<i>3</i>
1. EXECUTIVE SUMMARY	7
<i>Licensing</i>	<i>8</i>
Notice.....	8
1. Modelling methodology reminder.....	9
1.1. Introduction.....	9
1.2. CIM Profiles	9
1.3. Modelling Methodology	10
1.4. CimSyntaxGen Add-In and Modelling Methodology	12
2. Important things to remember when doing syntactic generation	13
General	13
2.1. WARNING.....	13
3. Class Packages Template and Dependencies	14
4. Datatypes package template and dependencies	16
5. CimSyntaxGen Add-In Installation and Overview	17
5.1. Platform Architecture	17
5.2. Downloading “CimSyntaxGen”	17
5.3. Before installing “CimSyntaxGen”	18
5.4. Installation.....	18
5.5. License.....	18
5.6. Configuration	19
5.7. Overview.....	21
5.8. "RDF" Menu.....	22
5.9. “XSD” Menu.....	24
5.10. “Html documentation” Menu	24
5.11. "Manage CodeLists" Menu	25
5.12. "Json" Menu	26
5.13. "Options" Menu	27
5.14. "CodeComponent" Menu	28
5.15. “About” Menu	29

6.	<i>How to use CimSyntaxGen Add-In for syntactic model generation</i>	29
7.	<i>RDF Schema generation</i>	30
7.1.	Overview	30
7.2.	Recommended pre-requisite	31
7.3.	Select profile package	31
7.4.	DCAT-3 compliance	31
7.5.	Launch RDFS generation	32
8.	<i>Generation of RDFS according to 501:2006 augmented style</i>	32
8.1.	Overview	32
8.2.	New file structure	33
8.3.	Copyright	33
8.4.	Special characters handling	34
8.5.	Launch RDFS augmented generation	34
9.	<i>Generation of RDFS according to 501-Ed2</i>	38
9.1.	Overview	38
9.2.	New file structure	39
9.3.	Copyright	39
9.4.	Special characters handling	39
9.5.	Flattening datatypes	39
9.6.	Launch 501-Ed2 generation	39
9.7.	XSD Generation	43
10.	<i>Generation of WG19 style XSD</i>	43
10.1.	Overview	43
10.2.	Root Class checking	43
10.3.	Error Message: no root class	44
10.4.	Select profile package and launch XSD WG 19	45
11.	<i>Generation of WG16 Style XSD</i>	47
11.1.	Overview	47
11.2.	Root Class checking	48
11.3.	Select Assembly Model package and launch XSD WG 16	48
11.4.	ENTSO-E parameters	49
11.5.	Generation process	50
12.	<i>XSD by Ref</i>	51

13.	<i>XSD to Prof Menu</i>	52
14.	<i>How to use CimSyntaxGen for HTML documentation generation.....</i>	54
14.1.	Overview.....	54
14.2.	Select profile package	54
14.3.	Select HTML generation	55
15.	<i>Generic HtmlDocumentation generation</i>	55
15.1.	Overview.....	55
15.2.	HTML file.....	55
15.3.	HTML generation example using the CIM to build a Work Profile.....	56
16.	<i>How to use CimSyntaxGen for HTML ENTSOE Documentation generation</i>	58
17.	<i>How to use CimSyntaxGen for ESMP HTML documentation generation.....</i>	59
	Overview.....	59
17.1.	ENTSO-E Documentation.....	59
17.2.	Part 351 IEC standard generation	60
17.3.	Conceptual and assembly models IEC standard generation.....	60
17.4.	Set of conceptual and assembly models IEC standard generation	60
18.	<i>How to use CimSyntaxGen for AsciiDoc documentation generation</i>	61
18.1.	Overview.....	61
18.2.	Document generation	61
19.	<i>How to use CimSyntaxGen to Manage CodeList.....</i>	61
19.1.	Description.....	61
	Import	62
19.2.	Generation of the codelists and documentation.....	62
20.	<i>JSON Schema export.....</i>	63
20.1.	Overview.....	63
20.2.	JSON schema versions	63
20.3.	UML profile styles.....	64
20.4.	Connectivity between Schema, Codelist and External Codelist files.....	64
20.5.	External CodeList Management	64
20.6.	Launching Json schema export	66
21.	<i>Generation of JSON Schema WG19 Style.....</i>	66
22.	<i>Generation of JSON Schema WG16 Style.....</i>	68
22.1.	Launching Json schema export	68

22.2.	CodeList management	69
23.	<i>Avro Schema export</i>	<i>69</i>
23.1.	Overview.....	69
23.2.	Codelist.....	69
24.	<i>Code Component export.....</i>	<i>70</i>
24.1.	Overview.....	70
24.2.	IEC Copyright files.....	70
24.3.	Delivery package name	70
24.4.	Launching CodeComponent	71
25.	<i>CimSyntaxGen configuration file.....</i>	<i>72</i>
25.1.	Managing configuration file.....	72
25.2.	Overview.....	72
25.3.	AppSettings parameters.....	72
25.4.	DataProfile parameters.....	74
25.5.	Profdata parameters	75
25.6.	Profstereo parameters.....	77
25.7.	SHACL parameters	77
25.8.	JSON parameters	78
26.	<i>UML/XSD generation principles</i>	<i>78</i>
26.1.	XSD generation basic principles	78
26.2.	CIMDatatype generation principles.....	80
27.	<i>Subclass inherited association and XSD generation principles.....</i>	<i>81</i>

1. EXECUTIVE SUMMARY

CimSyntaxGen is an Enterprise Architect Add-In and a companion tool for CimConteXtor. It has three functionalities:

- Generate syntactic models from EA packages. The first syntactic generations are [W3C](#) XML (Extensible Markup Language) [Schema](#), [RDF](#) (Resource Description Framework) Schema, JSON (Java Script Object Notation) Schema and Avro Schema.
- Generate HTML and AsciiDoc documentation from EA packages,
- Manage CodeLists.

CimSyntaxGen's syntactic generations are based on [IEC](#) (International Electrotechnical Commission) TC 57 standards: IEC 61970-501 (CIM RDF Schema), IEC 62361-100 (XML Naming and Design Rules), and IEC 62361-104 (CIM profiles to JSON schema mapping).

NOTE: To use *CimSyntaxGen* Add-In, one should be familiar with:

- UML (Unified Modelling Language) class diagram modelling,
- Enterprise Architect UML tool,
- Rules for creating a profile based on a UML Information Model (see Annexes of the CimConteXtor user's manual and [UN/Cefact](#)¹ CCTS)
- W3C² XML [Schema Recommendation](#),
- W3C RDF [Schema Recommendation](#),
- IEC TC 57 61970 - 501 "CIM Resource Description framework",
- IEC TC 57 62361-100 "XML Naming and Design Rules,"
- IEC TC 57 62361-104 draft "CIM profiles mapping to JSON schema".

"Enterprise Architect" is a tool developed and distributed by [Sparx System](#).

CimSyntaxGen was originally developed at [Zamiren](#) by Sébastien Maligue-Clausse, André Maizener and Jean-Luc Sanson and distributed under CeCILL-B open-source license.

ENTSO-E contributed expanding CimSyntaxGen and publishes it under the Apache 2.0 license on the [ENTSO-E website](#). The open-source tool development is under governance of ENTSO-E.

¹ UN/Cefact : [United Nations Center for Trade Facilitation and e-Business](#)

² W3C : [World Wide Web Consortium](#)

Licensing

Licensed to the Apache Software Foundation (ASF) under one or more contributor license agreements. See the NOTICE file distributed with this work for additional information regarding copyright ownership. The ASF licenses this file to you under the Apache License, Version 2.0 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at

<http://www.apache.org/licenses/LICENSE-2.0>

Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

Notice

Apache CimSyntaxGen

Copyright 2025 The Apache Software Foundation.

Portions of this software were developed at ENTSO-E (<https://www.entsoe.eu/>) and Zamiren (<https://www.zamiren.fr/index.php/en/>).

1. Modelling methodology reminder

1.1.Introduction

The development of CIM – originally as an internal database model (relational Schema) for an EMS API (Energy Management System Interface) – was created by Electric Power Research Institute (EPRI) started in the mid-90s. Early in the stage of development CIM switched to the object-oriented approach. Therefore, the CIM extensively uses Unified Modelling Language (UML) class diagrams to describe each class of objects, their attributes, and their relationships with other classes.

1.2.CIM Profiles

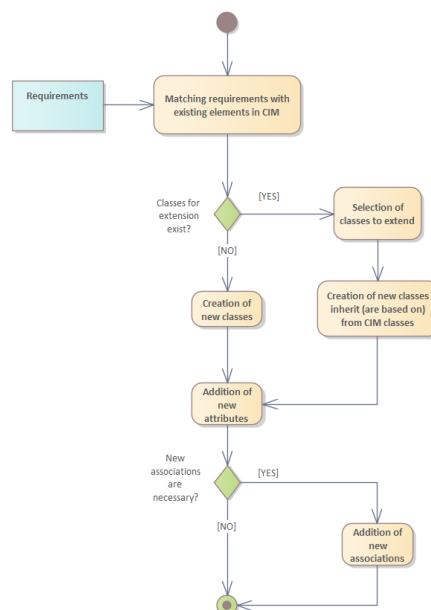
CIM profiles are

- excerpts from the entire CIM data model of a fixed version,
- which may contain additional „own“ classes, attributes and associations,
- and describe the syntax and semantics of data exchanged by interfaces

Why do we not use the whole data model?

- Individual parts still need to be added – the CIM standard is constantly improved and extended.
- CIM is large – the CIM 17 standard contains
 - 8011 attributes in
 - 1616 classes in
 - 146 packages.
- So that unnecessary amounts of data do not always have to be transferred.

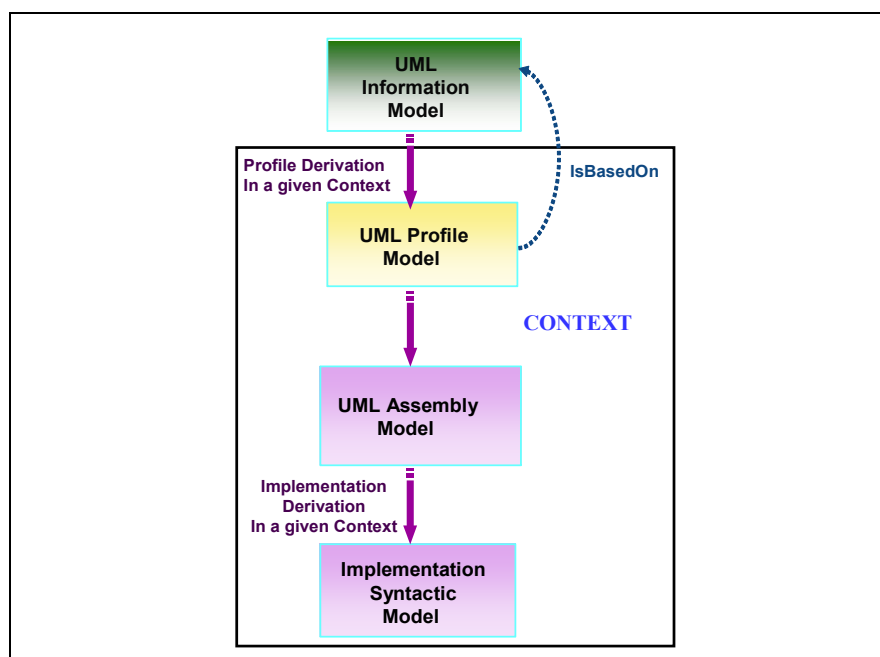
The following activity diagram depicts the schematically the procedure to create a profile:

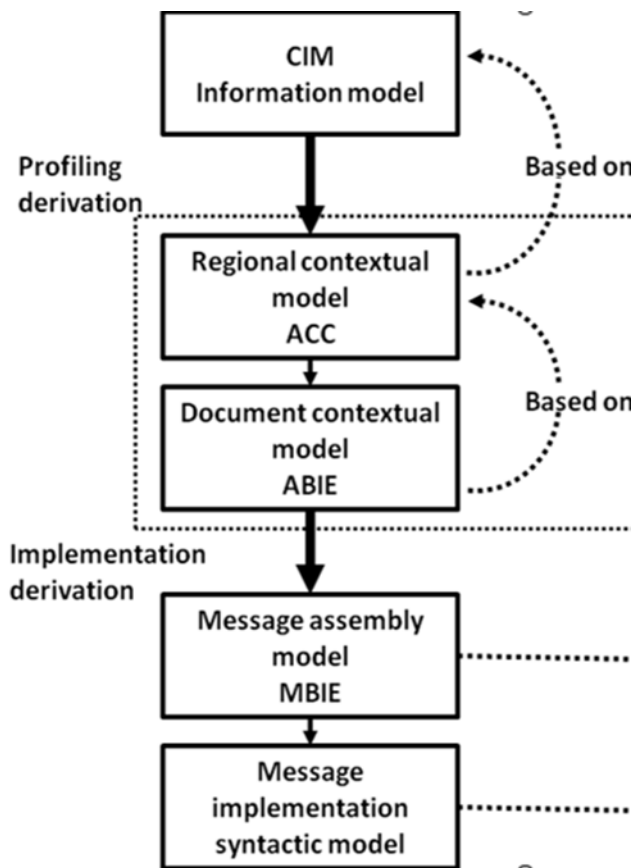


1.3. Modelling Methodology

The methodology that is behind *CimConteXtor* and *CimSyntaxGen* Add-Ins is derived from UN/Cefact Core Component Technical Specification (CCTS), has been adapted for IEC needs, and defines four modelling levels:

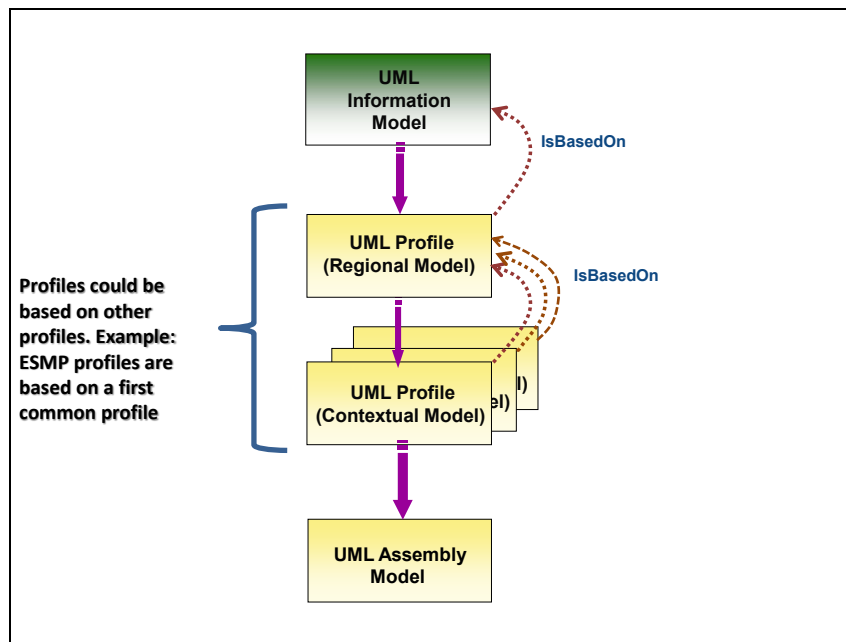
1. the Information Model level that defines a domain model (for example, the CIM for Electrotechnical Domain),
2. the Profile Model Level, which defines how this information model is used in a context, or which subset of this information model is used in some given context (example CPSM—Common Power System Modelling—for power network exchange): This is the concept of “*IsBasedOn*”,
3. the Assembly level, that defines how the profile artefacts are assembled for exchange,
4. the syntax level, that defines in which syntax the exchange will take place and how the Assembly level will be mapped to the chosen syntax.





The first three levels are modelled in UML.

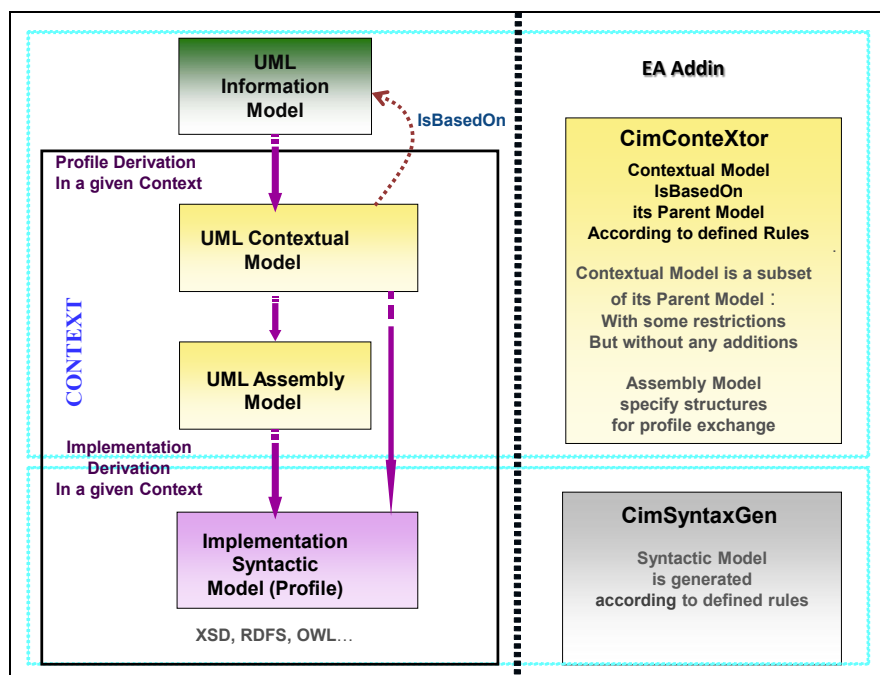
NOTE: a profile could be based on another profile (recursion): this is used for example for ESMP profiles, where there is an additional level (called Regional Profile) to mutualize some specific restrictions on an appropriate subset of the information model. So, the UML levels could be like:



1.4.CimSyntaxGen Add-In and Modelling Methodology

CimConteXtor is a tool that builds a UML Profile (and optionally a UML Assembly Model) based on a UML Information Model (See *CimConteXtor* User Guide for profile and Assembly model generation).

CimSyntaxGen Add-In is a companion tool for *CimConteXtor* that generates syntactic schemas (XSD or RDFS) according to given Syntax Naming and Design Rules.



CimSyntaxGen Add-In implements Syntactic model generation, for both RDFS, XSD and Json Schema that conform to IEC standards.

Next *CimSyntaxGen* Add-In versions will implement other syntactic models, when specifications will be available.

Apart from syntactic model generation, *CimSyntaxGen* provides also other features like model documentation generation.

2. Important things to remember when doing syntactic generation

General

The profile, from which you want to generate a syntactical model, SHALL

- be clearly identified as a specific package (and sub-packages),
- have explicit “*IsBasedOn*” dependencies with Information Model packages see “*Packages Template and dependencies*” section below,
- use datatypes (and compounds) that are clearly identified:
 - they belong to specific packages both at Information Model (for example “Domain” package in CIM) and profile levels
 - their names are unambiguous on the project or profile name space,
- use datatypes that have all their “*IsBasedOn*” dependencies defined,
- follow the naming rules defined in the modelling methodology for Class, Attribute, Association role end, Datatype names.

More, when the target is an XSD, *CimSyntaxGen* Add-In is performing some functionalities that, normally, are performed by the UML Assembly level like adding a header (named “*Envelop*” in IEC 62361-100) and its relationships with profile root classes.

2.1.WARNING

Important things to remember when creating an XSD or a Json schema:

- Verify that the selected package is the UML profile or assembly model package for which you want to generate the schema,
- Verify that all attribute types used in this profile are conforming to the datatype options you choose (See *CimConteXtor* Integrity Check feature):

- could be any datatype (or primitive, enumeration, compound) on the project name space: this implies that all these elements have a unique name on that name space,
- could only be datatype belonging to the profile: this means that uniqueness is just on the profile name space.
- Verify that there is a root class.

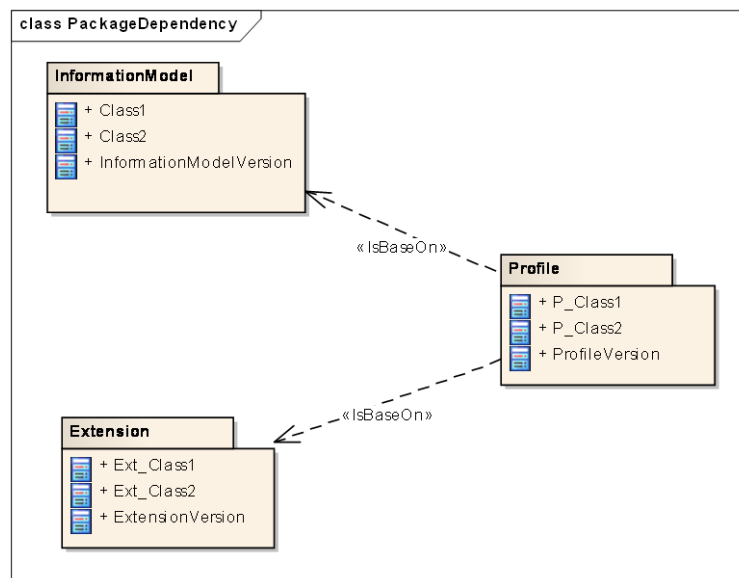
3. Class Packages Template and Dependencies

“*IsBasedOn*” Package Hierarchy must be defined now when you work out Package Template definition.

The “*IsBasedOn*” hierarchy could be of different kinds:

1° Profile Package is “*IsBasedOn*” an Information Model Package:

==> Profile classes are « *IsBasedOn* » on Information Model Package classes. Information Model Package can have sub-packages or not. But in the profile package there are no sub-packages.

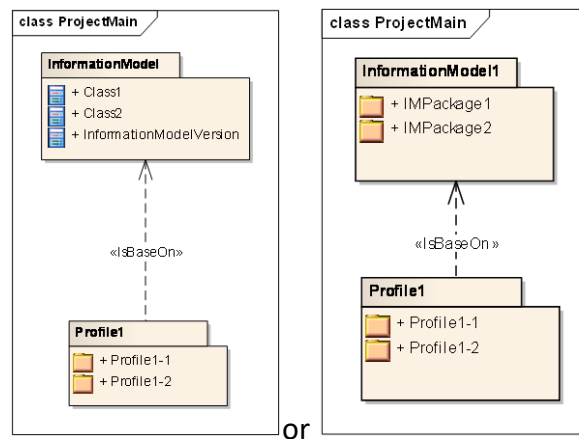


2° A Profile Package has Sub-Packages and is based on an Information Model Package Dependency

Profiles rules:

- All profile classes are in sub-packages,

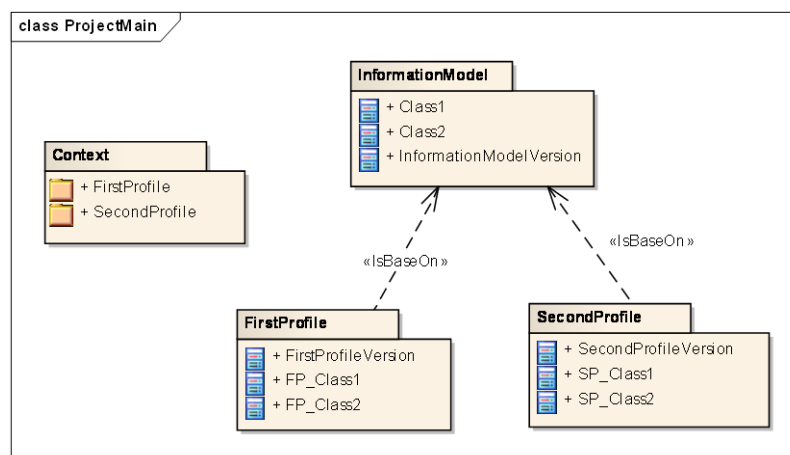
- all profile class names are unique in the context of the profile package (a profile sub-package class could not have the same name as another class of the sub-package, but also as another class in another different profile sub-package),
- associations between sub-package classes are allowed.



3° Profile packages are « *IsBasedOn* » an Information Model Package and are grouped in a common Package that is not itself « *IsBasedOn* » the Information Model

This is the case when different profiles are made in each Context:

- All profile class names are unique in the context of a sub-package (classes in different profile sub-packages could have the same names).
- Associations between classes of different sub-packages are not allowed.



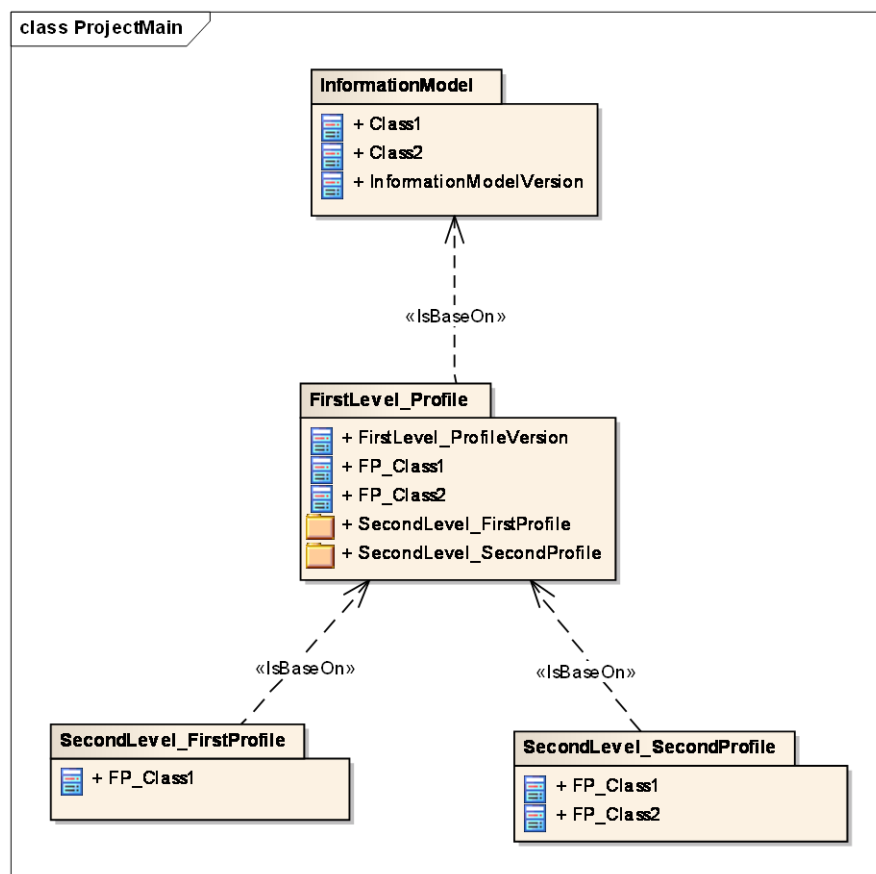
4° Sub-packages classes are based on the package classes

This case will happen when doing sub-profiles which are based on a given profile.

Profile Package is “*IsBasedOn*” an Information Model Package. Profile classes are contained in the Profile Package and are not organized in sub-packages.

Sub-Profile packages are contained by the profile package.

This case is like the following one:



4. Datatypes package template and dependencies

There are different ways to manage and use profile primitives, enumerations, datatypes and compounds. This must deal primarily with profile package standalone capacity and with datatypes reusability.

1° If you want profile standalone package, then all the primitives, enumerations, datatypes and compounds that are used in the profile package must be defined in this profile package or in a profile sub-package best named “*Profile Domain*”.

In this case:

- primitives, enumerations, datatypes and compound names should be unique in the profile name space,
- but it means also that, if you use primitives, enumerations, datatypes or compounds already defined at the Information Model level, you should recreate them (with “*IsBasedOn*” function) at the profile level. When doing that, profile elements could have the same names as the Information model level ones.

2° If you do not want profile standalone package, then primitives, enumerations, datatypes and compounds that are used in the profile package could be defined at any modelling level. This means:

- that primitive, enumeration, datatype and compound names must be unique on the project name space (use of qualifier is required when doing an “*IsBasedOn*”).

3° If you want datatype reusability in multiple profiles, the best is to put all primitives, enumerations, datatypes and compounds in a shared “*Domain*” package. This will imply that all names are unique on the project name space (use of qualifier is required when doing an “*IsBasedOn*”).

5. CimSyntaxGen Add-In Installation and Overview

5.1. Platform Architecture

The CimSyntaxGen Add-in is now available for two target platforms beginning with EA version 16+.

- For 32-bit platforms

This applies to all EA versions 15 and earlier.

- For 64-bit platforms

This applies to all EA versions 16 and later.

In general, the 64-bit version of the add-in is the preferred version for distribution and maintenance. The 32-bit version is available upon request but is considered deprecated.

NOTE: When using the EA 64-bit version the project files are locally stored as *.qea (SQLite) files.

5.2. Downloading “CimSyntaxGen”

On the [ENTSO-E website](#), you can find a .zip file containing:

- *CimSyntaxGenSetup_installer*: An ENTSO-E signed Windows installer (.exe and .msi files).

- LICENSE.txt: The Apache 2.0 license.
- NOTICE.txt: File used to include third application party notice.
- *key_20270101.txt*: A general license key with an expiry date 01/01/2027. ENTSO-E will update this key accordingly.
- *CimSyntaxGen_Manual_v10-0-3.pdf*: A manual with the necessary documentation. This is the current document the user is reading.
- *cimsyntaxgen_SourceCode.zip*: A package with the source code.

5.3. Before installing “CimSyntaxGen”

Check on ENTSO-E site (<https://www.entsoe.eu/data/cim/#cim-profiling-tools>) the version you are using and the requirement for it in terms of Operating Systems and Enterprise Architect version compatibility.

5.4.Installation

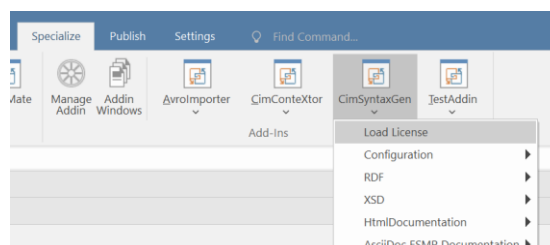
To install CimConteXtor, double-click on the delivered “Setup*CimSyntaxGen*.msi” file and follow the wizard instructions (Do remember to uninstall a previous installed version through the Windows “Add or suppress programs”).

NOTE: CimSyntaxGen versions before 2024 required installing Python for JSON and AVRO schema generation. Newer versions since end of 2023 include these schema generations in the Add-in. An installation of Python is not necessary anymore.

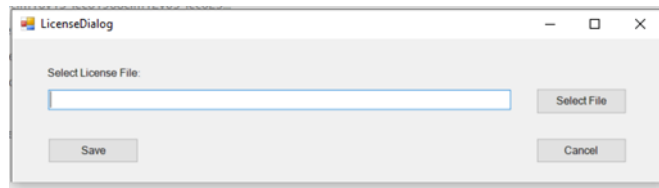
5.5.License

To be able to use *CimSyntaxGen* it is necessary to load a valid license. The license is a text file containing the license key and is provided by ENTSO-E.

After installing *CimSyntaxGen* you open the menu of the Add-in in the EA’s Ribbon “Specialize”:



This will open a dialogue enabling to load the license file:



When the file is selected and loaded by pressing the button 'Save', the license is stored encrypted in a file in the *CimSyntaxGen* resources directory. After that the license file is not needed anymore. If the encrypted file is removed and the Add-in is used, a warning message of an invalid license will come up and the Add-in will not work anymore. To continue to work requires a re-loading of a valid license file.

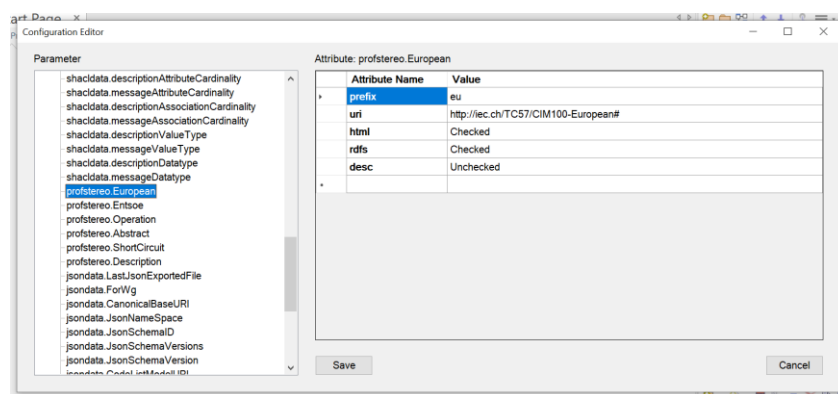
A warning mechanism starts to inform 14 days before expiry of the license how much days are still left till a renewal of the license is necessary.

5.6.Configuration

The *CimSyntaxGen* has a built-in default configuration. Menu entries enable to import and export the configuration. If a custom configuration is imported as XML file, the default configuration is replaced and the custom configuration is stored in the EA project as Notes of an element "*CimSyntaxGen*" in a package "Configuration".

This mechanism enables to install the *CimSyntaxGen* without having to care about a meaningful configuration. After installation the Add-In is ready for use without external configuration XML file.

In addition, the configuration parameters can be edited via a dialogue (menu entry 'Configuration / Configuration Editor'):



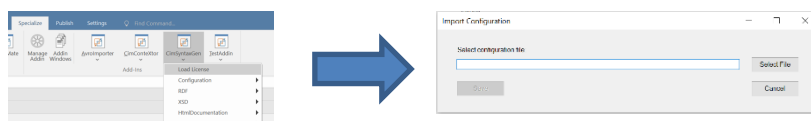
The following actions are possible with a configuration file:

1. Installing and using the *CimSyntaxGen* Addin without external configuration file:
The Addin is installed and can be used right away. The default configuration is internally in the EA package 'Configuration' in the tagged values of the element 'CimSyntaxGen'.

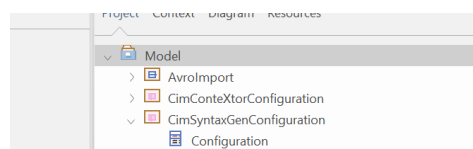
NOTE: It is not intended to edit the configuration directly there. CimSyntaxGen contains a configuration editor (see below) to enable amending the configuration.

2. Import of a custom configuration:

An existing custom configuration XML file can be imported by the Menu “Import Configuration”:



The custom configuration replaces the default configuration and is stored in the element “Configuration” in a newly created package “CimSyntaxGenConfiguration”:



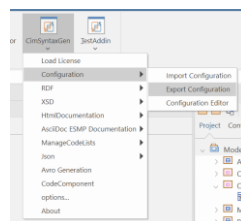
When loading the XML configuration, a quick XML validation routine checks the configuration file. If there is an error in the XML a message is displayed telling the possible reason (e.g. an XML element name is misspelled and which element).

3. Changing the configuration

Each change of the configuration (Menu item “Options”) causes either the creation of a custom configuration stored in the model (as described below in the element ‘Configuration’) or an update of the configuration, if there exists already a custom configuration.

4. Export of a configuration

There is a menu entry for the export of the configuration:



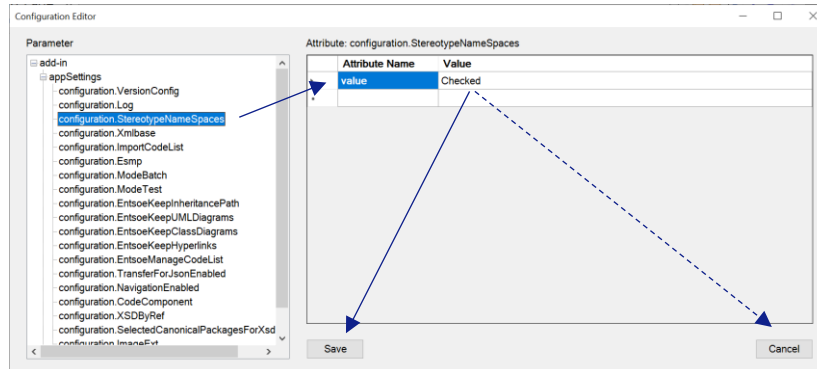
This opens a dialogue for storing the configuration XML file.

If there is no custom configuration, the default configuration is exported. This enables to check the default configuration.

5. Editing the configuration via the ‘Configuration Editor’ dialogue.

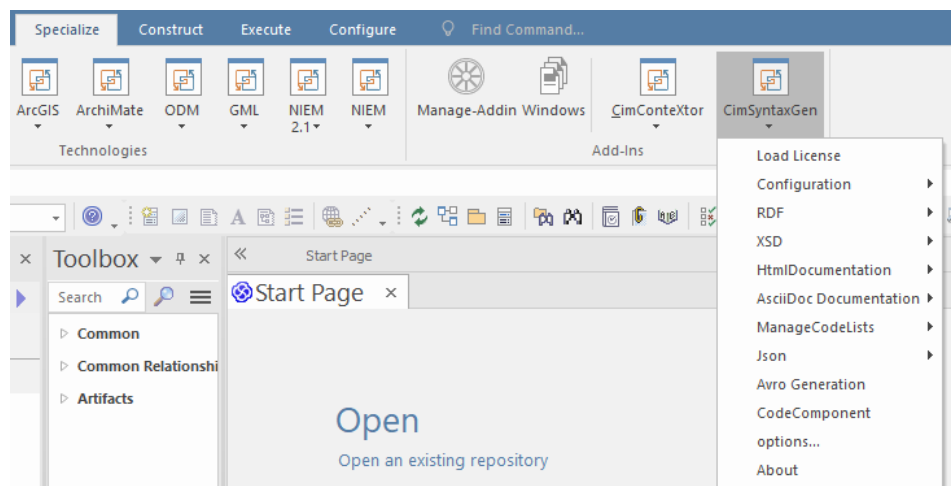
The dialogue lists on the left-hand side the parameters divided in categories. Clicking on a parameter updates the right-hand side showing the attributes of the parameter with the

current value. The value can be changed. When saving the changes the configuration is automatically updated.



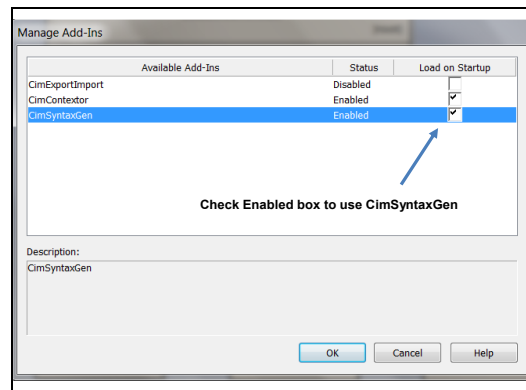
5.7.Overview

CimSyntaxGen Add-In is managed as an EA Add-In. It is referenced in the EA tool bar in "Extension":

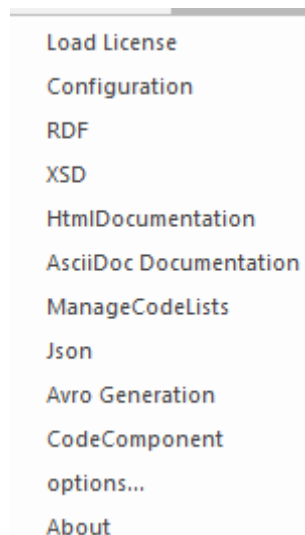


The *CimSyntaxGen* Add-In is enabled after installation. The Add-In can be switched off by disabling it. To do that, click on the “*Manage Add-Ins*”, this will display the “*Manage Add-Ins*” window where you tick off *CimSyntaxGen*’s checkbox:

:

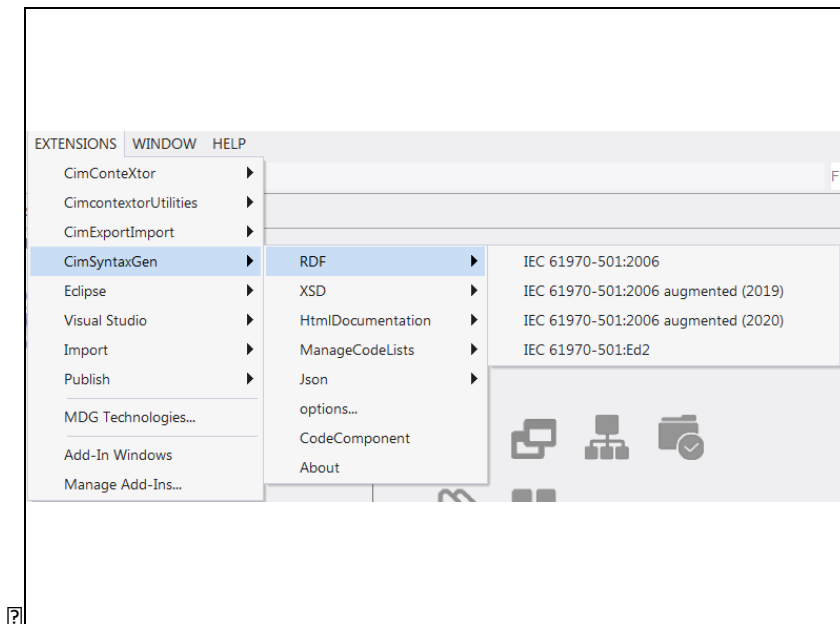
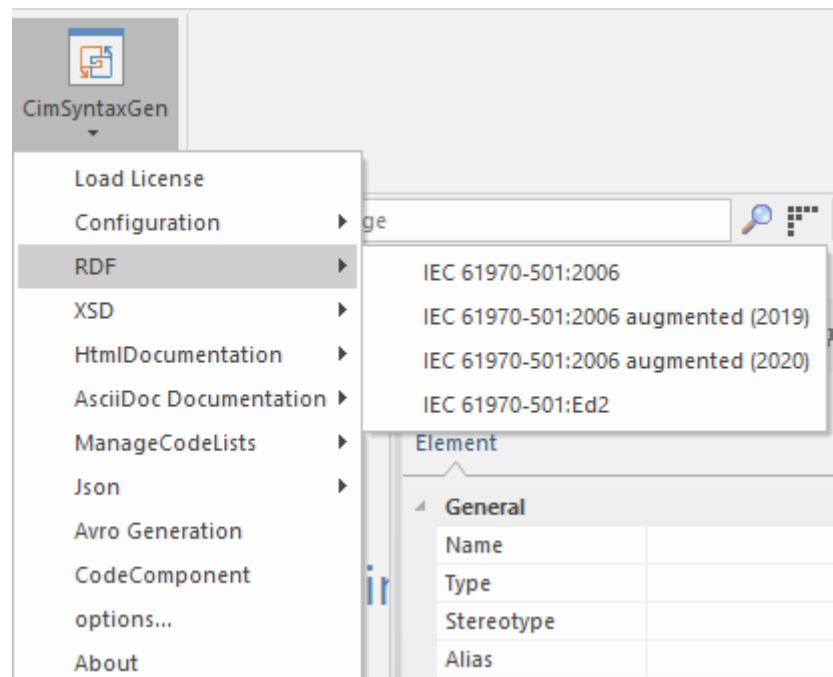


CimSyntaxGen Add-In has the following functionalities that are given by the *CimSyntaxGen* Add-In Menu items:



5.8. "RDF" Menu

The “*RDF Menu*” is used to generate RDF Schemas.

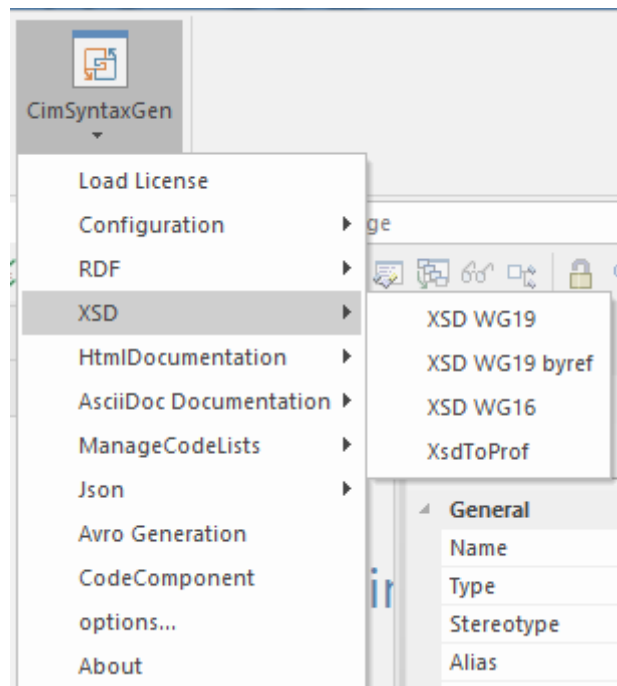


Four kinds of RDF Schemas can be generated:

- IEC 61970-501:2006: RDF Schema conforming to IEC TC 57 61970 – 501 Ed1,
- IEC 61970-501:2006 augmented (2019): RDF Schema with some syntactical differences with RDFS 501 Ed1.
- IEC 61970-501:2006 augmented (2020): RDF Schema with Profile version class and header.
- IEC 61970-501:Ed2: RDF Schema conforming to IEC TC 57 61970 – 501 Ed2 draft.

5.9. "XSD" Menu

The "XSD Menu" is used to generate XML Schemas.



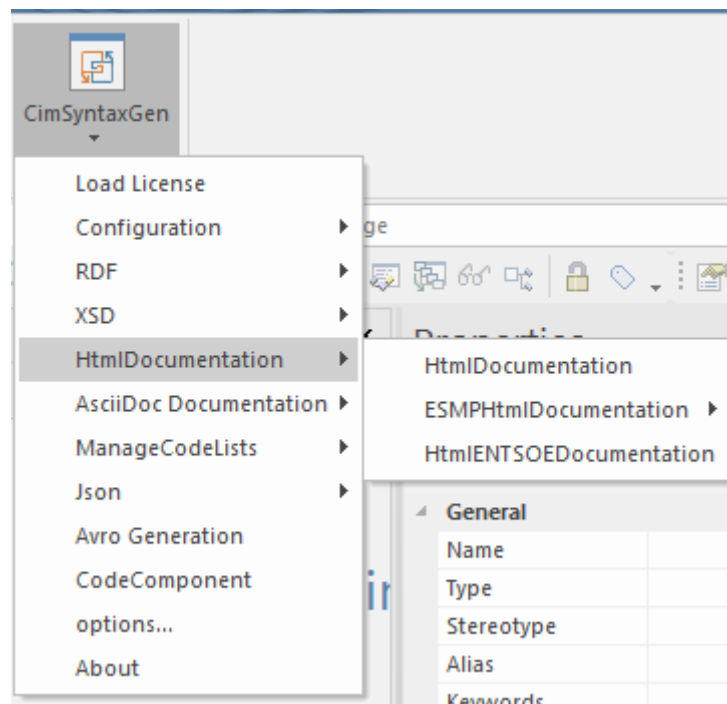
Three kinds of XML Schemas can be generated:

- XSD WG19: XML Schema conforming to IEC 62361-100 "XML Naming and Design Rules" standard,
- XSD WG19: by Ref: use with Graph profiles to generate XML Schema conforming to IEC 62361-100, but using the "by Ref" feature,
- XSD WG16: XML Schema conforming to IEC 62361-100 "XML Naming and Design Rules", using different options than the ones used for XSD WG19.

"XsdToProf" Menu is used for reverse engineering: map an XML Schema to UML, based on a CIM model.

5.10. "Html documentation" Menu

The "Html Documentation" menu lets you generate the html documentation of a UML package, according to different contexts:

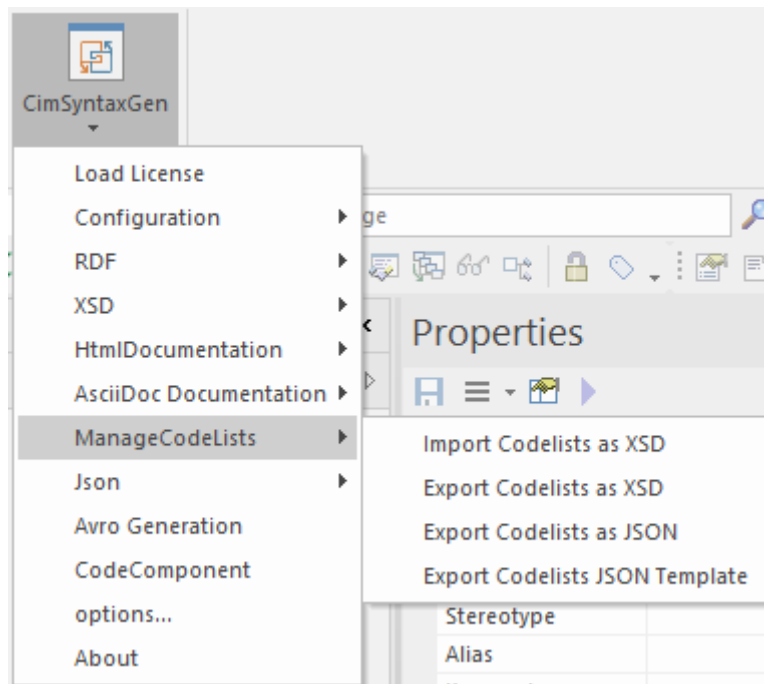


There are three different contexts:

- HtmlDocumentation: standard output for TC/57/WG13 profiles (basically CPSM),
- ESMPHtmlDocumentation: outputs for TC57/WG16 European Style Market profile. This output could be further tailored for specific purposes:
 - Output of both Contextual and Assembly Models,
 - Output of a regional profile (basically IEC 62325-351),
 - Output of a full package with multiple contextual and assembly models,
 - Output used for ENTSO-E Market Documentation
- HtmlENTSOEDocumentation: output for ENTSGE CGMES.

5.11. "Manage CodeLists" Menu

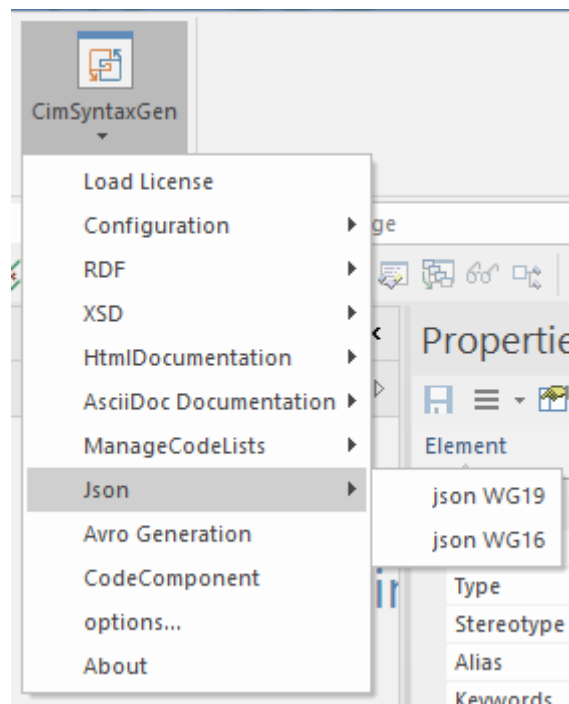
The "ManageCodeLists" is used in ESMP Profile to import ENTSO-E CodeLists in a UML package or to generate the ENTSO-E XSD CodeLists with the associated documentation and to export Codelists in JSON format. In addition, it is possible to export a JSON codelist template for local codelists (see JSON schema generation):



NOTE: to generate a CodeList XSD from "351" enumeration, do not forget to create for new enumeration a "Uid" Tagged Value with the proper value.

5.12."Json" Menu

The "JSON Menu" is used to generate JSON and AVRO Schemas.



Three kinds of Schemas can be generated:

- Json WG19: Json Schema conforming to IEC 62361-104 "CIM Profiles mapping to Json schema" standard,
- Json WG16: Json Schema conforming to IEC 62361-104, using different options than the ones used for XSD WG19, especially the use of external codelists.

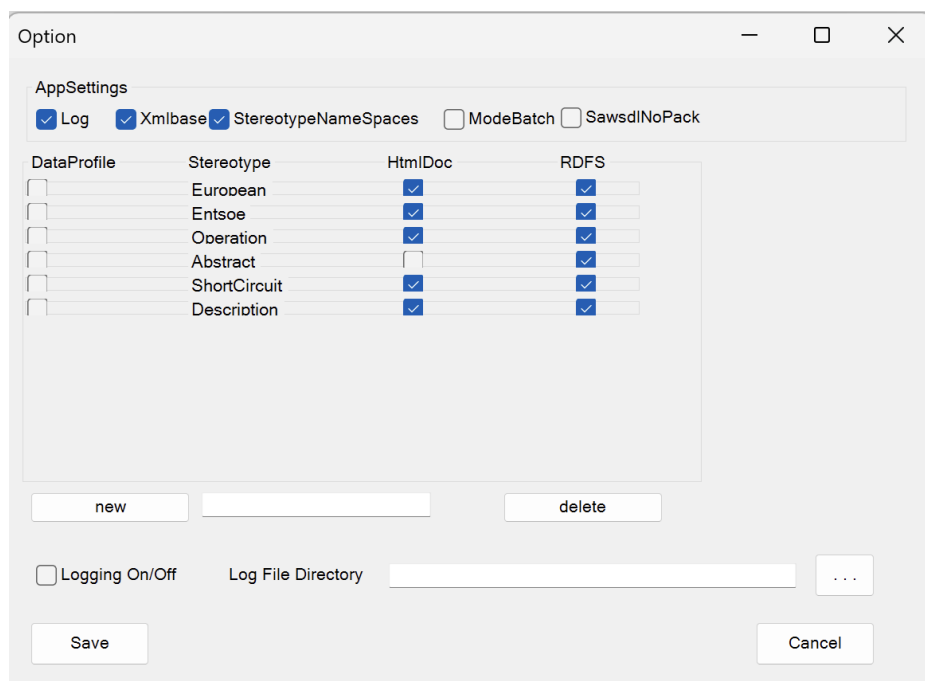
5.13."Options" Menu

The "Option" Menu is used in parallel with CimSyntaxGen configuration file. It lets you fix some parameters used by CimSyntaxGen to tailor some behaviors:

- Log or not results of command,
- Use of XmlBase in schema generations,
- For RDFS generation, use namespace for stereotyped element output,
- For RDFS generation, do things in Batch Mode,
- For XSD generation, output or not "sawdl" elements
- For Graph profiles, differentiate outputs with according to stereotypes:
 - For HtmlDoc output, order or not order output by stereotypes,
 - For RDFS output, use or not a specific name space for each stereotypes.

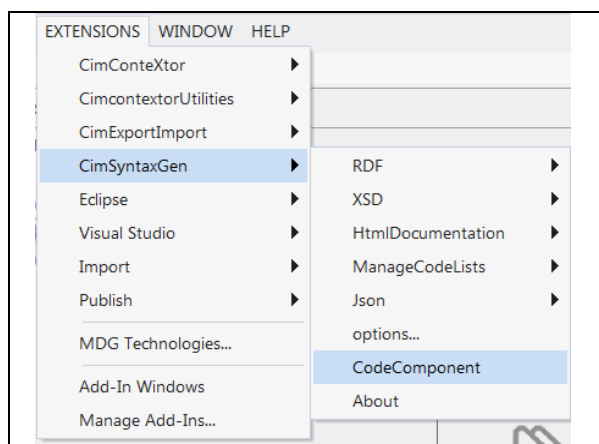
The list of stereotypes is given by the configuration file, but the option menu lets you extend this list.

The "Save" button updates the configuration file (see configuration file section).



5.14."CodeComponent" Menu

The "CodeComponent Menu" manage the inclusion of code component(s) into an IEC deliverable, produce and publish such code component(s) as a package, according to the proposal: "Handling of Code Components in IEC Standards, including Copyright Licensing v8.0 - 2019-08-23". For this version, it is tailored for WG16 ESMP code components.



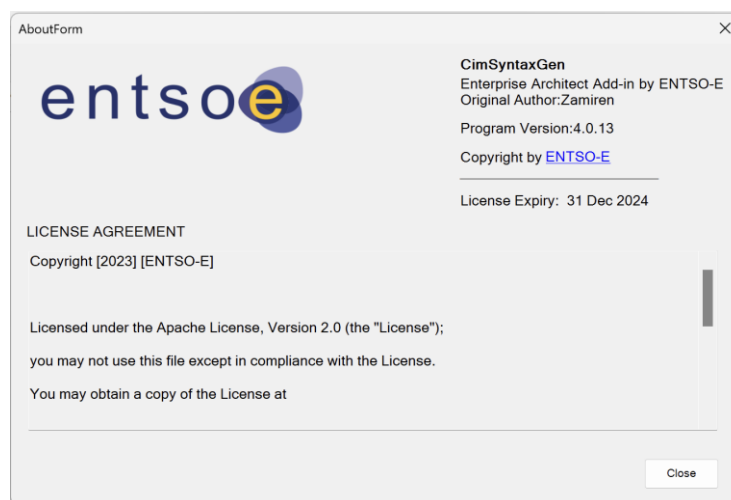
The menu allows gathering copyright information and generation of a Core Components delivery package.

The Core Component delivery package is a zip file, which includes:

- a manifest xml file (conforming to the IEC Manifest XSD) that describes:
 - the publication(s) from where the CodeComponents are defined,
 - the CodeComponent file(s) description,
 - the History file(s) describing the changes which have been considered in the associated package, since the last IEC publication (at least).
- An IEC copyright XML file (conforming to the IEC Copyright XSD).
- the code component file(s) extracted from the IEC publication(s) (typically XSD file, XML file, SNMP MIB file ...).

5.15. “About” Menu

The “About” menu is used to give information on the current *CimSyntaxGen* Add-In, like version, release date, expiration date and license agreement.



6. How to use CimSyntaxGen Add-In for syntactic model generation

There are several steps to use *CimSyntaxGen* Add-In:

1. Define the package that serves as the starting point for the generation,
2. Verify that there is at least one root class if you want to generate an XSD,
3. Define all parameters (profile name space for example) for syntactic model,
4. Define syntactic model name, version and location,

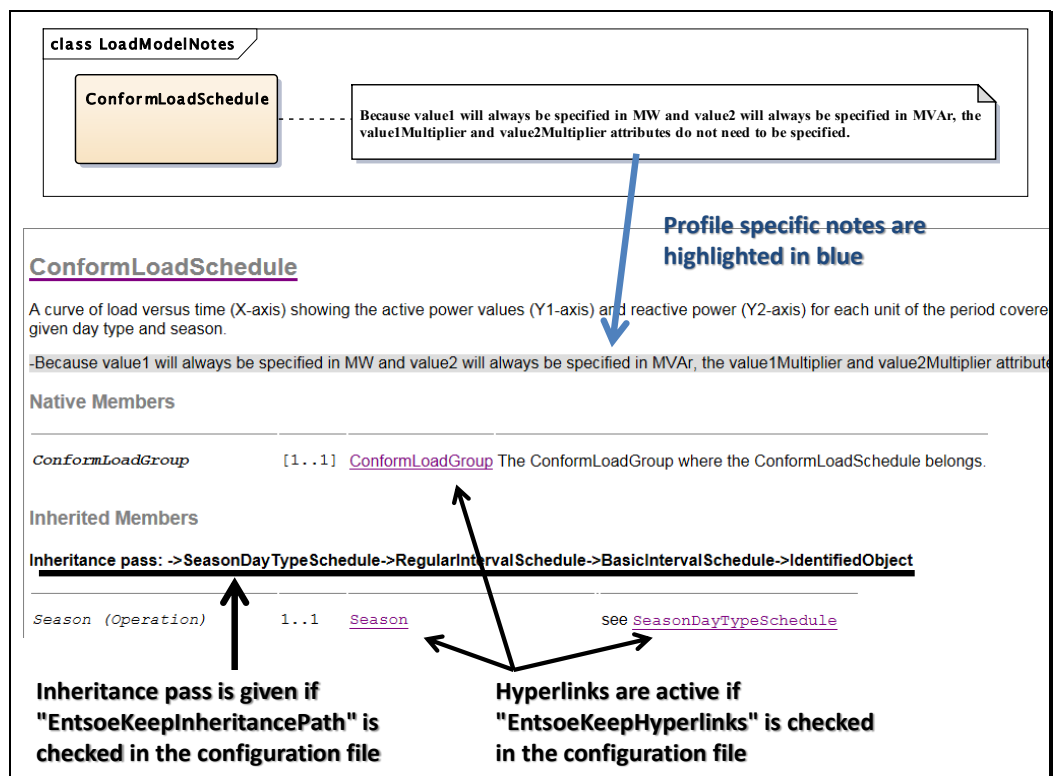
5. Launch appropriate *CimSyntaxGen* Add-In function.

7. RDF Schema generation

7.1.Overview

RDF Schema generation is done according to:

- IEC 61970 part 501 Ed1 standard, with some extensions:



- "cims:IsDefault" and "cims:IsFixed" for specifying attribute default and fixed values,
- "cims:datatype" to state which datatype is used for an attribute value,
- "cims:AssociationUsed" to define which association will be used in the RDF instances.
- IEC 61970-501 augmented (stamped as 2019 version):
 - use of 501 Ed1 with modifications adopted for IEC interop tests (mainly everything is described by a "rdf:description" instead of "rdfs:class" or "rdf:property",
 - presence in UML of a <profilename>Version class,
 - possible use of a copyright comment.

- IEC 61970-501 augmented (stamped as 2020 version):
 - use of 501 Ed1 with modifications adopted for IEC interop tests (mainly everything is described by a "*rdf:description*" instead of "*rdfs:class*" or "*rdf:property*",
 - possible use of a copyright comment,
 - Use of a header part that takes into account a new "Ontology Profile class" included in the UML profile.
- IEC 61970 part 501 Ed2 draft standard:
 - Generation of two files: one for the vocabulary and one for the constraints expressed as SHACL shapes,
 - Possible use of a copyright comment,
 - Use of a header part.

7.2.Recommended pre-requisite

Before launching the RDFS export, it is wise to check if the UML profile is valid, by making a CimContextor Utilities "*CGMES Integrity check*" before. The result CVS file is named "*CheckError<ProfileName>.cvs*" and is stored in:

"C:\Users\<UserName>\AppData\Roaming\ENTSO-E\CimContextor"

7.3.Select profile package

The first step is to select the package on which you want to apply RDFS generation.

NOTE: *CimSyntaxGen* Add-in will use this profile package to generate all selected package (and/or its sub-packages) classes and their properties. But it will grab all the datatypes that have been used in the profile packages where it will find them (see datatypes template and dependencies section).

7.4.DCAT-3 compliance

In compliance to the Data Catalog Vocabulary (DCAT) Version 3, the RDF generation includes now the following attributes of the Header (see ENTSO-E's METADATA AND DOCUMENT HEADER DATA EXCHANGE SPECIFICATION V2.2 ³)

- dcat:version
Specifies the version.

³ https://eepublicdownloads.entsoe.eu/clean-documents/CIM_documents/Grid_Model_CIM/MetadataAndHeaderDataExchangeSpecification_v2.2.pdf

- `dcterms:title`
Specifies the title.
- `adms:versionNotes`
A description of changes between this version and the previous version of the resource.
- `dcterms:issued`
Specifies the release date.

7.5. Launch RDFS generation

Go to the EA “Add-Ins” Menu, select “RDF” and select “IEC 61970-501:2006”, “IEC 61970-501:2006 augmented (2019)”, “IEC 61970-501:2006 augmented (2020)” or “IEC 61970-501:Ed2”:

CimConteXtor	▶		
CimcontextorUtilities	▶		
CimExportImport	▶		
CimSyntaxGen	▶	RDF	▶ IEC 61970-501:2006
Eclipse	▶	XSD	▶ IEC 61970-501:2006 augmented (2019)
Visual Studio	▶	HtmlDocumentation	▶ IEC 61970-501:2006 augmented (2020)
Import	▶	ManageCodeLists	▶ IEC 61970-501:Ed2

If “IEC 61970-501:2006” is selected, “Windows save file” window will be displayed to select where you want to put the RDFS file and how you will name it.

If another choice is selected (501:2006 augmented or 501-Ed2), then appropriate dialog will be started.

8. Generation of RDFS according to 501:2006 augmented style

8.1. Overview

Two kinds of RDFS augmented could be generated:

- “IEC 61970-501:2006 augmented (2019)” nearly the same as the old RDFS augmented, with two new options:
 - Add Copyright as a comment,

- Map special characters of the comment section.
- "IEC 61970-501:2006 augmented (2020)", with following changes and options:
 - Only applies to new UML Profiles with an Ontology Class,
 - Add Copyright as a comment,
 - Map special characters of the comment section.
 - Copyright in Ontology Class <dcterms:rights>
 - Use of header
 - New file structure

8.2.New file structure

The new file structure is the following one:

- Beginning: <rdf:RDF xmlns....
- Copyright as a comment (option)
- Header part, only for RDFS 2020 augmented style with:
 - rdf:description of the Profile_Ontology Class,
 - Copyright in OntologyClass.rights
- Profile part in case of RDFS 2019 augmented including all profile classes and:
 - ProfileVersion Class
 - Ontology Class if present
- Profile part of RDFS 2020 augmented including all profile classes except Profile_Ontology Class.

8.3.Copyright

A copyright notice could be added in the RDFS file as a comment for RDFS 2019 and 2020. This notice could be also inserted as a "dcterms:rights" tag in the "Profile_Ontology" class.

The notice is in the form of:

< Copyright >

< Notice >

COPYRIGHT (c) IEC 2020

This version of this RDFS is part of IEC 61970-DiagramLayoutProfile see the IEC 61970 for full legal notices.

In case of any differences between the here - below code and the IEC published content, the here - below definition supersedes the IEC publication.

it may contain updates. See history files (if it exists).

The whole document shall be taken into account in order to have a full description of this code component.
See www.iec.ch/CCv1 for copyright details

</Notice>

<Notice >

RDFS for 61970-DiagramLayoutProfile version=3.0.0 released 2020-07-14 as Draft

</ Notice >

<License uri =" www.iec.ch/CCv1 " kind ="Draft" > IEC License </ License >

</Copyright>

The appropriate fields are coming from a dialog box when you launch the RDFS generation (see below). Those fields are also updated and kept in the configuration file.

8.4.Special characters handling

In the comment part of an element there could be characters that are XML mark-up characters (like <, >, &, " and '). These characters must not be interpreted as mark-up of the RDF file. There are two ways to handle this:

- Use the `rdf:parseType="Literal"` feature,
- Or map those special characters.

The option is open to use one or the other. If mapping is selected, the mapping will be:

& - &

< - <

> - >

" - "

' - '

A comment like "OPC Unified Architecture" will be generated as:

If no mapping is selected":

```
<rdf:comment parseType="Literal">
```

```
    <i>OPC Unified Architecture</i>
```

If mapping is selected:

```
<rdf:comment rdf:datatype=http://www.w3.org/2001/XMLSchema#string>
```

```
&lt;i&gt;OPC Unified Architecture&lt;i&gt;
```

8.5.Launch RDFS augmented generation

RDFS (Augmented) has been selected, a first dialog window will appear:

There are two parts in the dialog box:

- The left part is about copyright information,
- The right part is the choice for RDFS generation.

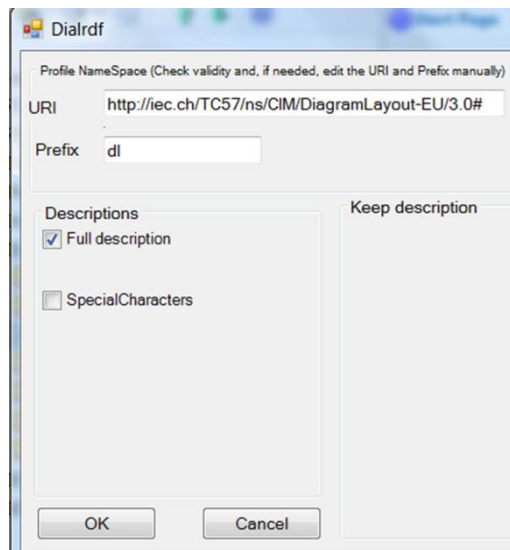
Fulfill Copyright information part:

Type Of Document	IS	Standard type: IS/TS/TR
StandardNumber	61970-600 ed1	
Profile Name	DiagramLayoutProfile	Profile package name
CodeComponentName	RDFS_Schema	Kind of code component
VersionStateInfo	3.0.0	Version of the profile
Light or Full	light	Without or with comments
DiffusionState	Draft	Draft/IEC/EntsoE/Other
Publication (Modification) Date	2020-10-12	

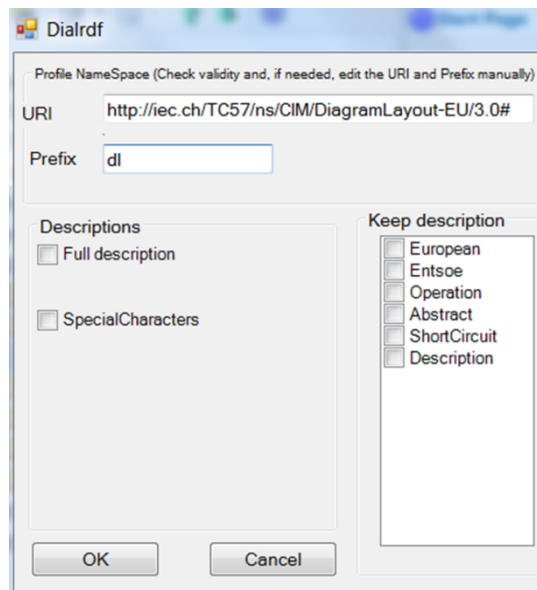
Then fulfill the left part:

- Check or enter the namespace of the profile
- Check if mapping of special characters

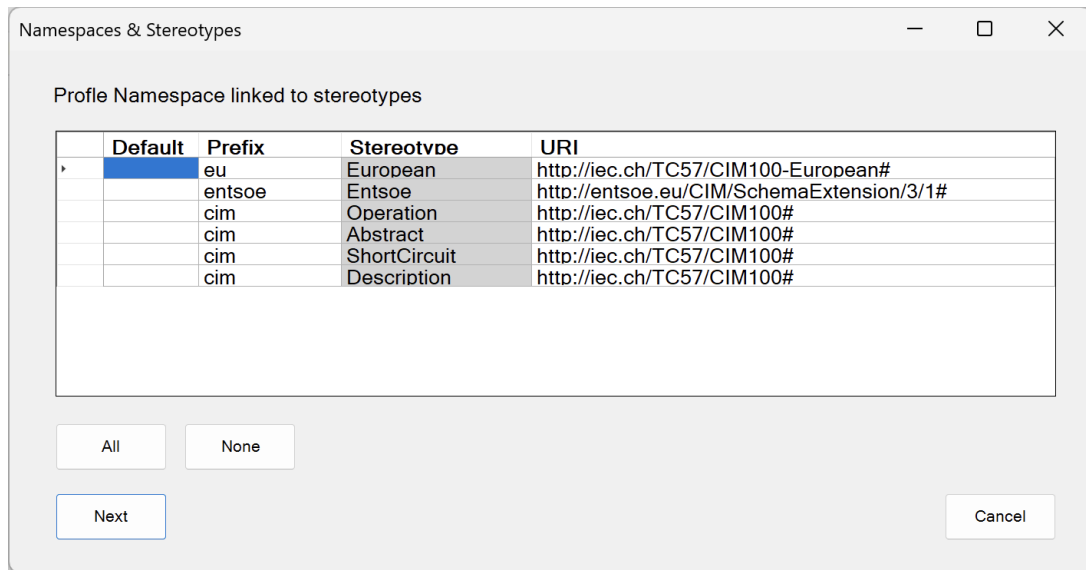
- Check if you want to have element with its description



- Or if you do not want the full description, except for element with a given stereotype:

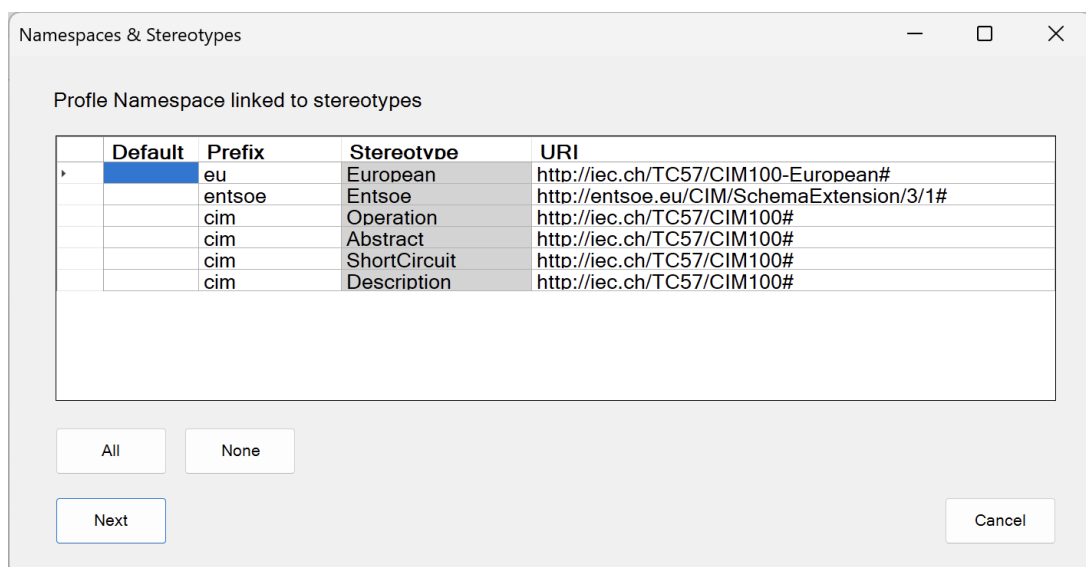


Once the required information has been entered, click "OK", a new window will be displayed:

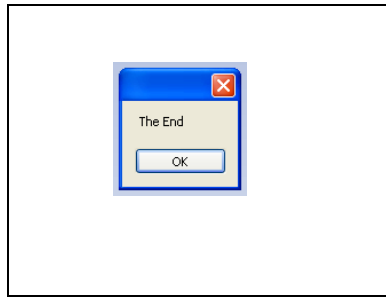


This window is used to say if you want to use or not special namespace for elements that are stereotyped. The list of stereotypes (defined in the configuration file or in the Option Menu) is shown:

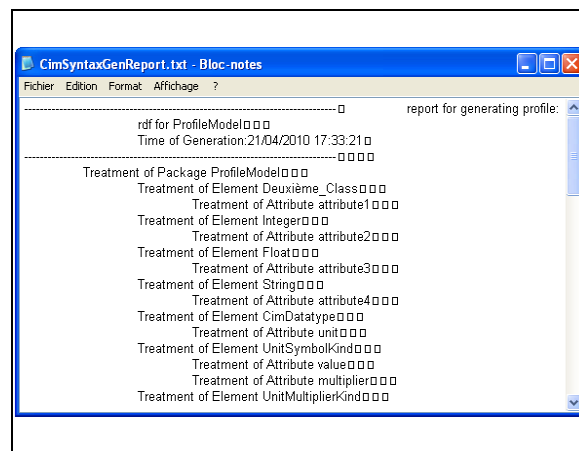
- If stereotype boxes are selected, the default name space (the profile one) will be used,
- If stereotype box is unchecked, a namespace will be used for these stereotype elements. The default namespaces are given by the configuration file. But this can be changed if stereotype box is unchecked: a new namespace could be edited and save in the configuration when clicking on the OK button.



By saying OK, the RDFS generation process is launched, a window is popping up to give the name of the file and its location, then the generation is finished, the “End” window is displayed:



The RDF file has been created alongside with a log file named “*CimSyntaxGenReport.txt*”:



NOTE: RDFS augmented parameters are given by the configuration file, the Option Menu and what is entered during the RDFS process.

9. Generation of RDFS according to 501-Ed2

9.1.Overview

The generation "IEC 61970-501-Ed2" is done as follow:

- Only applies to new UML Profiles with an Ontology Class,
- A Copyright is added as a comment,
- Map special characters for the comment section is always done.
- A Copyright notice is put in Ontology Class <dc:rights>
- A Header part is outputted
- An option is used for flattening datatype
- Two files are generated:
 - A vocabulary file for the new rdfs/owl profile mapping,
 - A constraint file for the UML constraints expressed as SCHACL shapes.

9.2.New file structure

The new file's structure is the following one:

- Beginning: <rdf:RDF xmlns....
- Copyright as a comment
- Header part with:
 - rdf:description of the Profile_Ontology Class,
 - Copyright in OntologyClass.rights
 - "Constraints" in OntologyClass.theme
- Profile part in the vocabulary file including all profile classes
- Profile part in the constraint file including SHACL shapes.

9.3.Copyright

A copyright notice is added in the two files as described in clause 8.3.

9.4.Special characters handling

Special characters mapping is done according to clause 8.4.

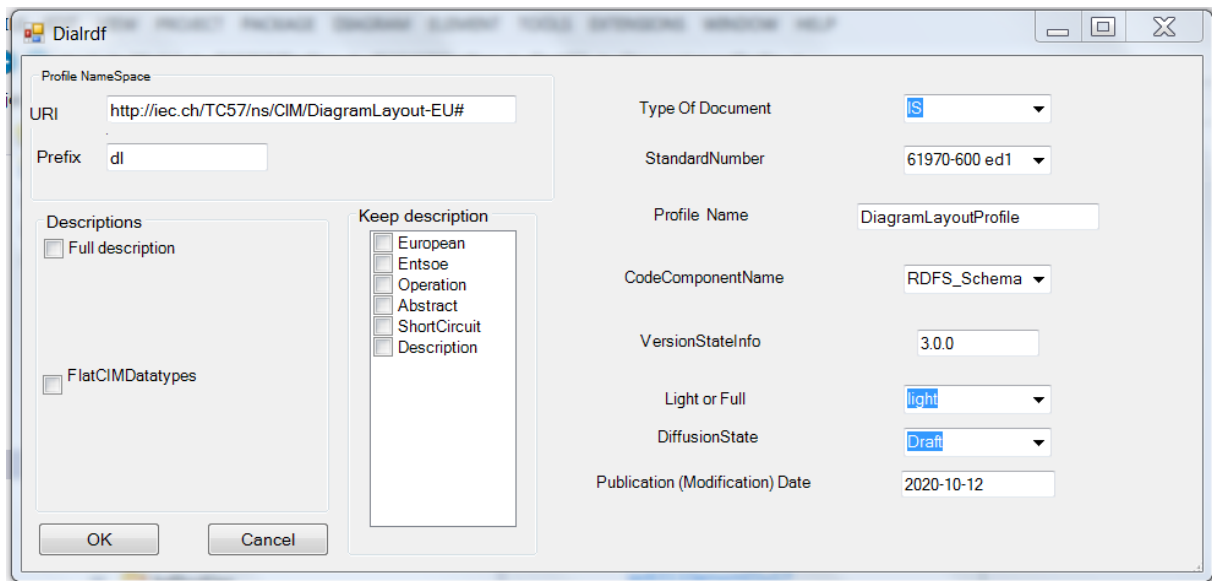
9.5.Flattening datatypes

In some cases, there is a need to make a CIMDataType equivalent with the CIMDataType.value. So an option is proposed to add the following statement will be added to a CIMDatatype class (e.g. AngleDegrees):

```
<owl:EquivalentProperty rdf:resource="#AngleDegrees.value"/>
```

9.6.Launch 501-Ed2 generation

When 501-Ed2 is selected, a first dialog window will appear:



There are two parts in the dialog box:

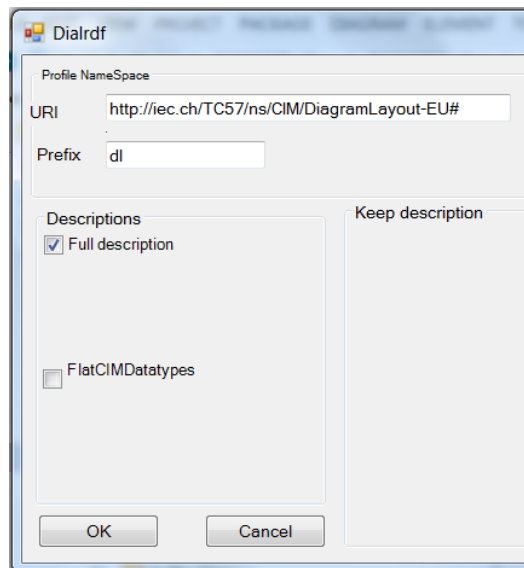
- The left part is about copyright information,
- The right part is the choice for RDFS/OWL generation.

Fulfill Copyright information part as in clause 8.3.

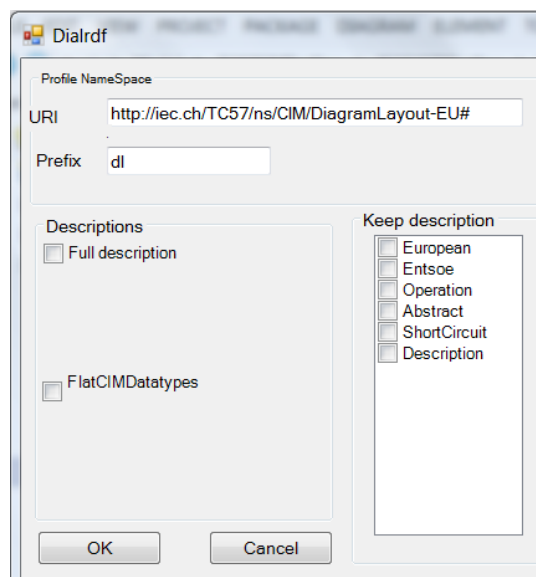
Type Of Document	IS	Standard type: IS/TS/TR
StandardNumber	61970-600 ed1	
Profile Name	DiagramLayoutProfile	Profile package name
CodeComponentName	RDFS_Schema	Kind of code component
VersionStateInfo	3.0.0	Version of the profile
Light or Full	light	Without or with comments
DiffusionState	Draft	Draft/IEC/EntsoE/Other
Publication (Modification) Date	2020-10-12	

Then fulfill the left part:

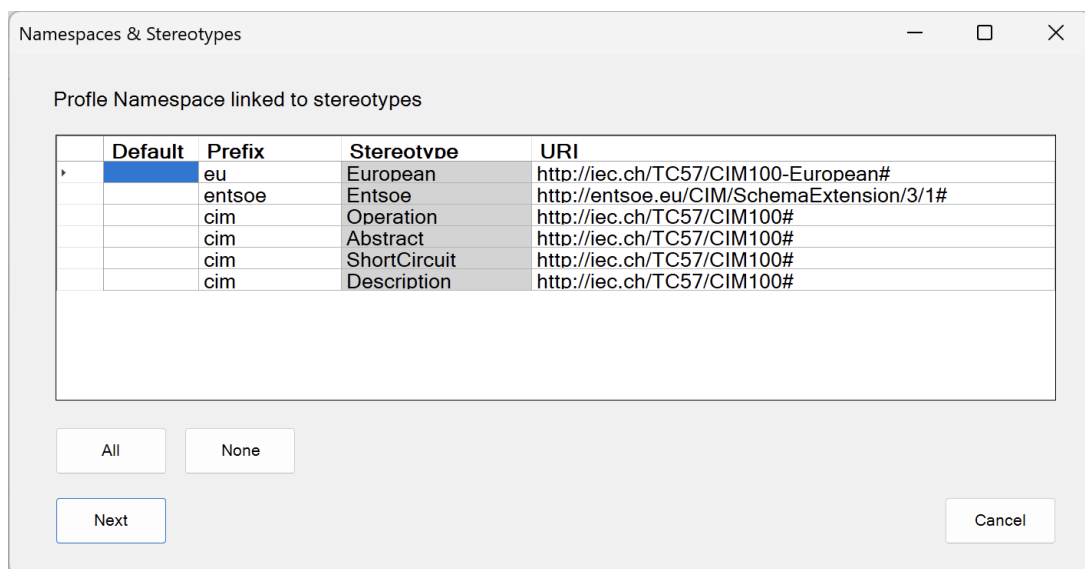
- Check or enter the namespace of the profile,
- Check if you want to have elements with their description,
- Check if you want to have datatype flattening.



- Or if you do not want the full description, except for element with a given stereotype:

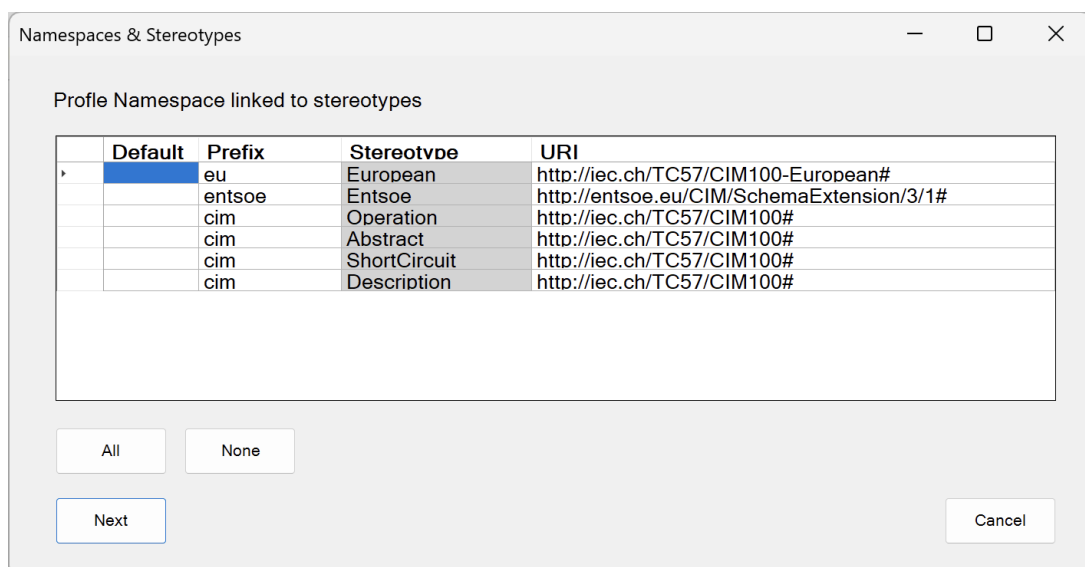


Once the required information has been entered, click "OK", a new window will be displayed:



This window is used to say if you want to use or not special namespace for elements that are stereotyped. The list of stereotypes (defined in the configuration file or in the Option Menu) is shown:

- If stereotype boxes are selected, the default name space (the profile one) will be used,
- If stereotype box is unchecked, a namespace will used for these stereotype elements. The default namespaces are given by the configuration file. But this can be changed if stereotype box is unchecked: a new namespace could be edited and save in the configuration when clicking on the OK button.



By saying OK, the 501-Ed2 generation process is launched, a window is popping up to give the name of the file and its location, then the generation is finished, the “End” window is displayed. Two files are generated:

- Vocabulary file (example: filename.rdf)
- Constraints file (example: filename-constraints.rdf)

The 501-Ed2 files are created alongside with a log file named "*CimSyntaxGenReport.txt*":

NOTE: 501-Ed2 parameters are given by the configuration file, the Option Menu and what is entered during 501-Ed2 process.

9.7.XSD Generation

XSD generation is done according to IEC 62361-100 "XML Naming and Design Rules".

XSD generation menu has 4 entries:

- Two are for generating XSDs corresponding to Hierarchical UML profiles:
 - WG19 style,
 - WG16 style,
- One for generating XSDs corresponding to Graph profiles (that usually are outputted as RDF Schemas):
 - XsdByRef style,
- Plus, a "XsdToProf" feature to map an XSD, conforming to IEC 62361-100, to UML (reverse engineering).

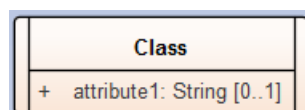
10.Generation of WG19 style XSD

10.1.Overview

WG19 style XSDs are conforming to the basic rules given by IEC 62361-100: use of an envelope name, order element by alphabetical order (See mapping rules of IEC 62361-100). In TC57, WG14 is following this style.

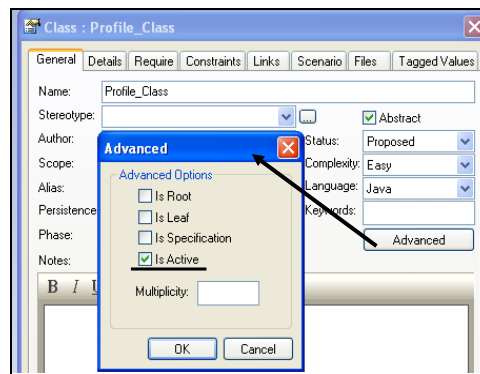
10.2.Root Class checking

Before launching the XSD generation, verify that you have at least one Root Class in the profile package or in its sub-packages. A Root Class has a special rendering in UML diagram:

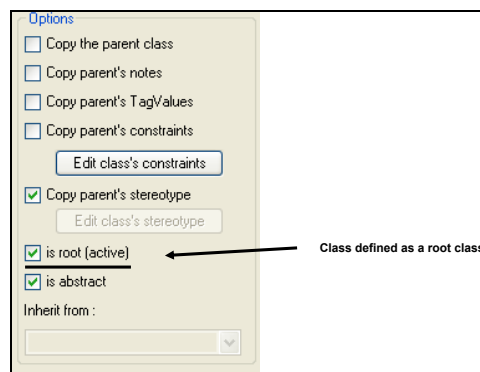


There could be several Root Classes in the profile, and in this case each Root Class drives its own hierarchy without any relationship between elements of different hierarchies.

In EA, a root class is marked with an “*IsActive*” property. This property could be found in the EA class property window, under the “*Advanced*” button:

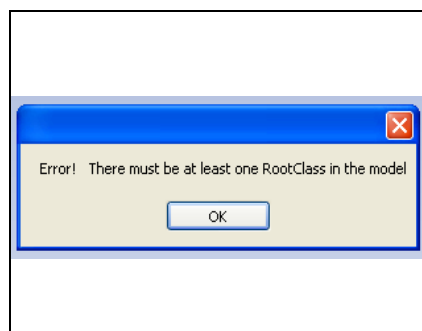


In CimConteXtor, the root class is defined by selecting the “*Edit IsBasedOn functionality*” window and checking the “*is root (active)*” box:

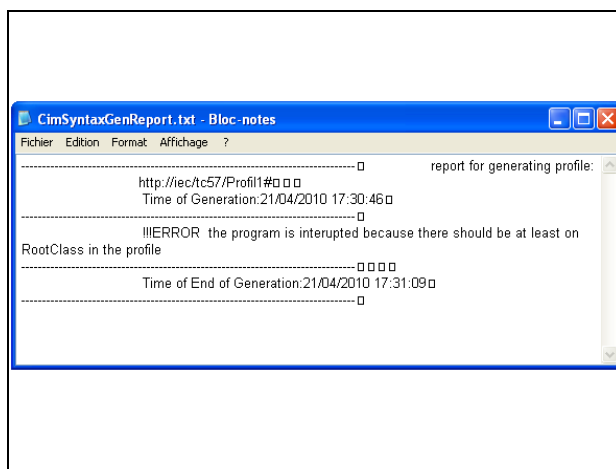


10.3.Error Message: no root class

If CimSyntaxGen Add-In does not find a root message an “*error*” window will be displayed:



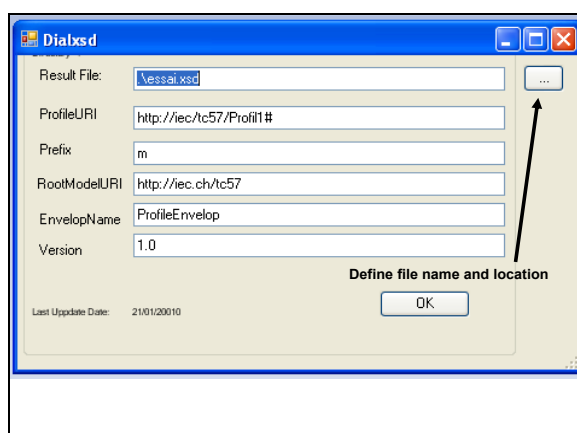
and a log file “*CimSyntaxGenReport.txt*” will be created:



10.4.Select profile package and launch XSD WG 19

Select the profile package from which you want to generate an XSD and select "XSD WG19". By doing this a first window is displayed. This window let you define your XML Schema characteristics:

- File name and location

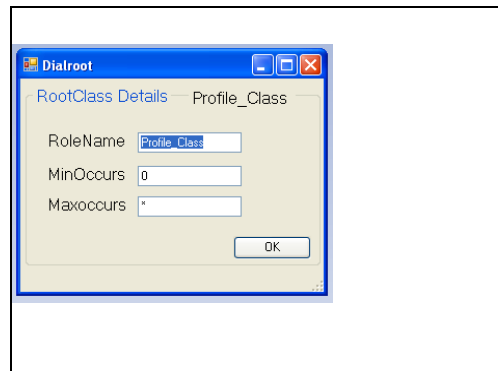


- Profile name space: this will enable to define both the profile name space and the target name.
 - Profile URI
 - Profile prefix
- Root Model URI: this is the URI of the Information Model. It is used to specify on which class a profile class is based on in the schema this will be used by the sawsdl attribute.
- EnvelopName: this is usually the name of the profile (this element is called envelop name in IEC 62361-100). This name will be the starting "xs:element" of the schema, and this element will be a sequence of "Root Class"

"xs:element(s)".

- Schema version

When all schema parameters are entered: click on “OK” button, and this will display the “Root Class” window:

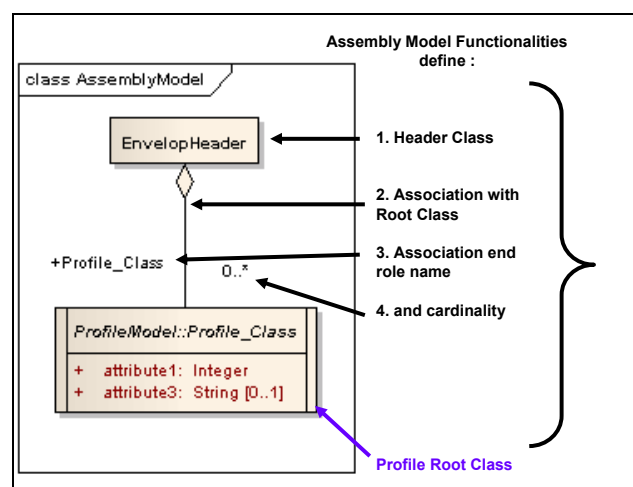


The “Root Class” window let you define some of Assembly level functionalities: the association between Header and the root class(es):

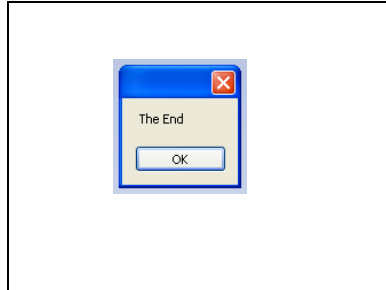
- association end role name,
- association end cardinalities.

Enter the cardinality for this Root Class and click “OK”. If there are several Root Classes, the same number of “Root Class” windows will be displayed.

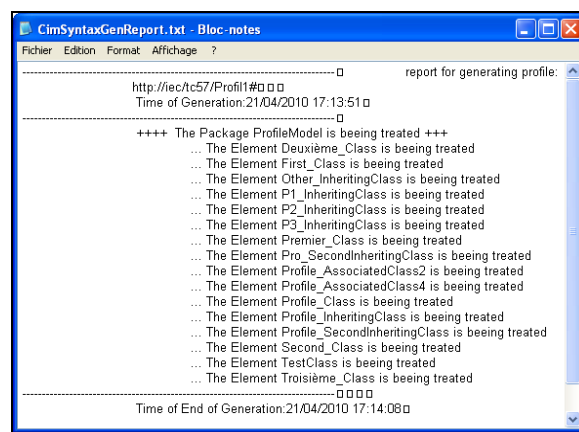
NOTE: CimSyntaxGen Add-In doing a kind of Assembly level functionalities that could have been done at UML Assembly modelling level, like defining a header (Envelop) and its relationships with root classes. In UML assembly model it will look that this:



When all RootClass windows have been taken care of, the XSD generation process is launched and when it is finished, the “End” window is displayed:



The XSD file has been created alongside with a log file named “CimSyntaxGenReport.txt”:



NOTE: the parameters of for the schema characteristics window are placed in the configuration file (see corresponding section).

11.Generation of WG16 Style XSD

11.1.Overview

WG16 ESMP style XSDs are conforming to

- Option rules given by IEC 62361-100:
 - no envelop name,
 - elements order is not alphabetical but specified by the "AttributeOrder" feature in CimConteXtor.
- Rules defined by IEC 62325-450:
 - Use of CodeLists instead of enumerations.

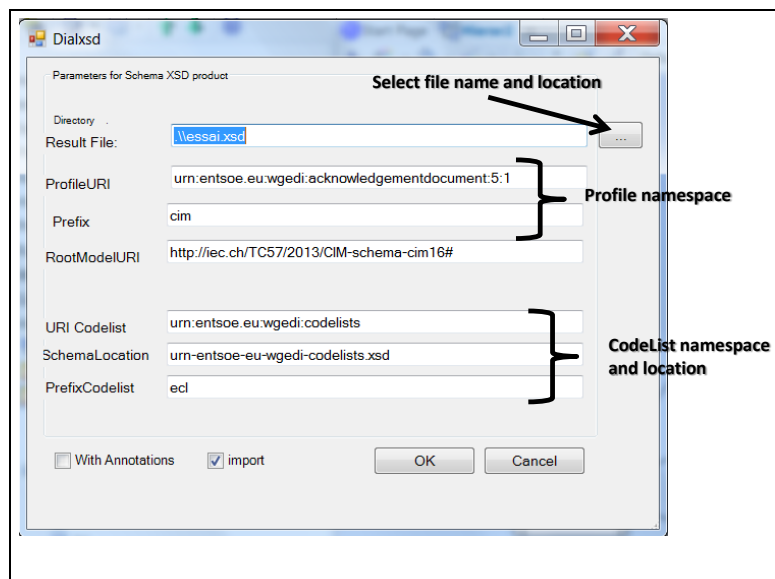
11.2.Root Class checking

Before launching the XSD generation, verify that you have at least one Root Class in the profile package or in its sub-packages. See above sections 10.2 and 10.3.

11.3.Select Assembly Model package and launch XSD WG 16

Select the UML Assembly Model package from which you want to generate an XSD and select "XSD WG16". By doing this a first window is displayed. This window let you define your XML Schema characteristics:

- File name and location



- Profile namespace: this will enable to define both the profile namespace and the target name.
 - Profile URI
 - Profile prefix (not used for ESMP)
- Root Model URI: this is the URI of the Root Model. It is used to specify on which class a profile class is based on in the schema this will be used by the sawsdl attribute.
- CodeList namespace and location:
- CodeList URI
- CodeList prefix
- CodeList schema location
- WithAnnotations: if checked, the schema will have annotations (UML element description), otherwise no,
- Import: if checked, CodeList schemas will be imported, if not CodeList Schemas will be included.

11.4.ENTSO-E parameters

The user shall then provide the following information:

- Result File: the name of the XSD file to be generated (there is a new dialog box to state in which folder the file is to be registered).
- ProfileURI: this is the standard namespace, and it is composed as follow:

urn:iec62325.351:tc57wg16:<process>:<document>:<version>:<release>

where:

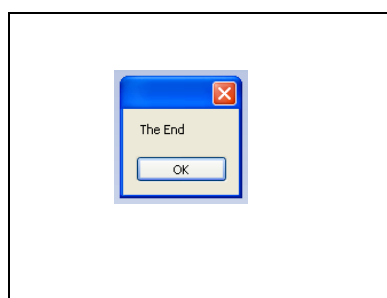
1. iec62325.351 shall be the stem of all European style market profile XML schema namespaces.
2. tc57wg16 identifies the organisation or group of organisations within IEC that own the object being referenced. In the case of TC57 this shall be the WG16.
3. <process> identifies the specific process where the object is situated, e.g. the part of the IEC 62325 standards in which the XML schema is defined, e.g. 451-1, 451-2, 451-3, etc.
4. <document> identifies the electronic document schema.
5. <version> identifies the version of the document schema.
6. <release> identifies the release of the document schema.

- Prefix: blank value.
- RootModelURI: <http://iec.ch/TC57/2013/CIM-schema-cim16#>.
- URI Codelist: the URI of the codelist to be used, i.e. urn:entsoe.eu:wgedi:codelists.
- SchemaName: the filename of the codelist, i.e. urn-entsoe-eu-wgedi-codelists.xsd.
- PrefixCodelist: the prefix used in the schema for the codelist, i.e. cl.
- The case “import” enables to generate a schema with the “import” option of the codelist, if the case is not selected, an xsd with the “include” option of the codelist is generated.
- Then click “OK” to generate the xsd; at the end a dialog box “End” will be displayed.

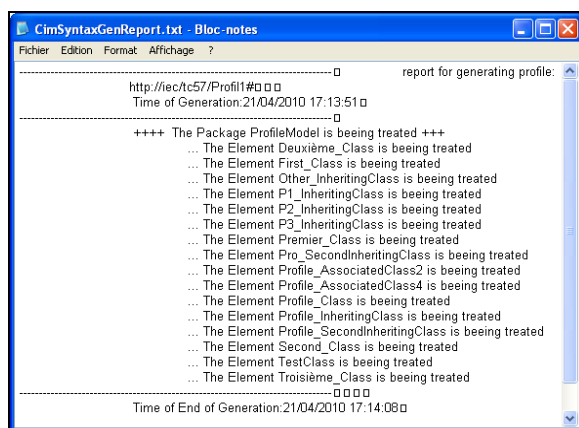
11.5.Generation process

When all schema parameters are entered: click on “OK” button,

the XSD generation process is launched and when it is finished, the “End” window is displayed:



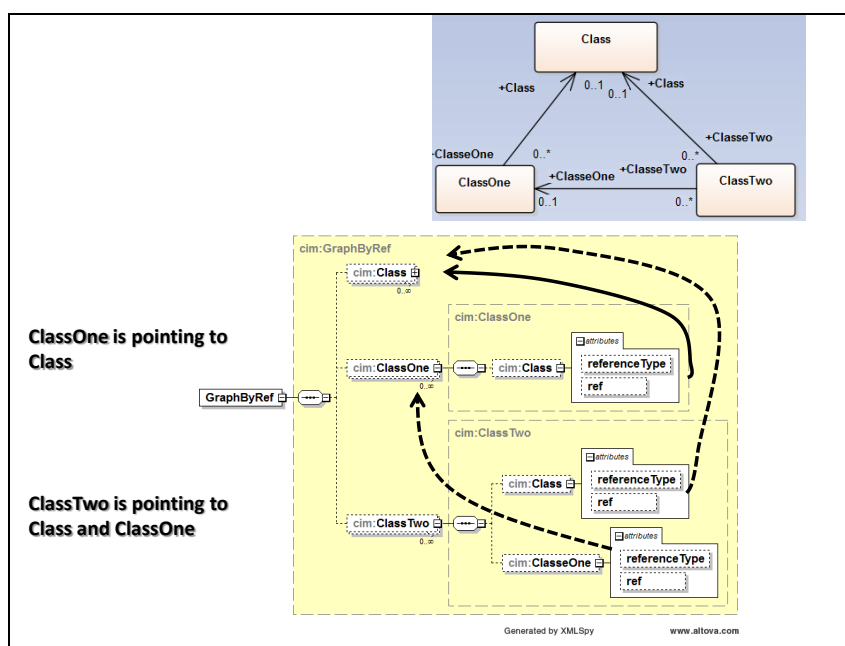
The XSD file has been created alongside with a log file named “*CimSyntaxGenReport.txt*”:



NOTE: the parameters of for the schema characteristics window are placed in the configuration file (see corresponding section).

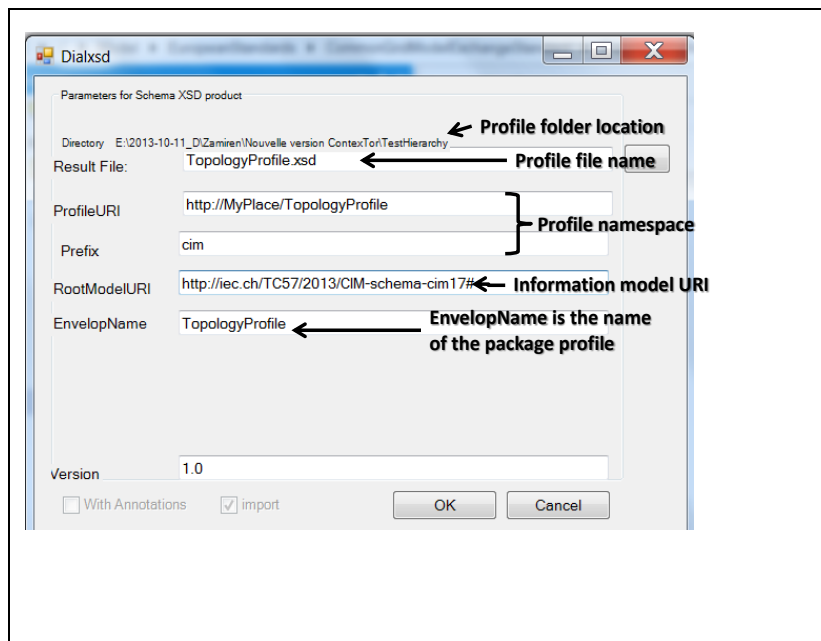
12.XSD by Ref

For a graph profile, *XSDbyRef* generates an XML Schema instead of an RDF Schema. So, the trick is to be able to express in a hierarchy something that is a graph. To do that, one of the IEC 62361-100 structure is used: the "byRef" design that is a pointer to another object and this pointer ID is usually the mRID of the object. So, each class of the graph profile (in the example Class, ClassOne and ClassTwo) is mapped as an xml element of the profile element name (in the example "GraphByRef"). Then, each oriented association is mapped as an element (with a ref attribute) of the class that is the origin of the oriented association. For example: ClassOne is the origin of the association with Class, so ClassOne will have a Class (end role name) element with an xml "ref" attribute (see IEC 62361-100 for more explanation):



The use case is the output of network profiles as XML file.

When selecting the "XSDByRef" menu on a package, the following window will pop up:



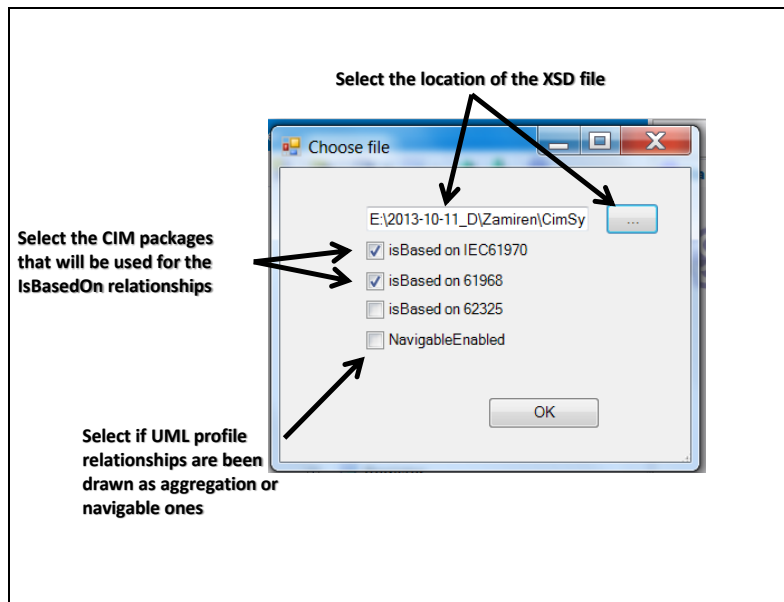
When all fields have been completed, the OK button is pressed, the processing is launched and an XSD file is generated.

13.XSD to Prof Menu

"XSDToProf" feature is used to map XSDs that are conforming to IEC 62361-100 and are using CIM for their semantics. The use case is to get XSDs corresponding UML profiles.

To use "XSDToProf" one must have an EA project with the CIM. To store the result (UML profile) in EA, it is better to have a package dedicated to profile packages.

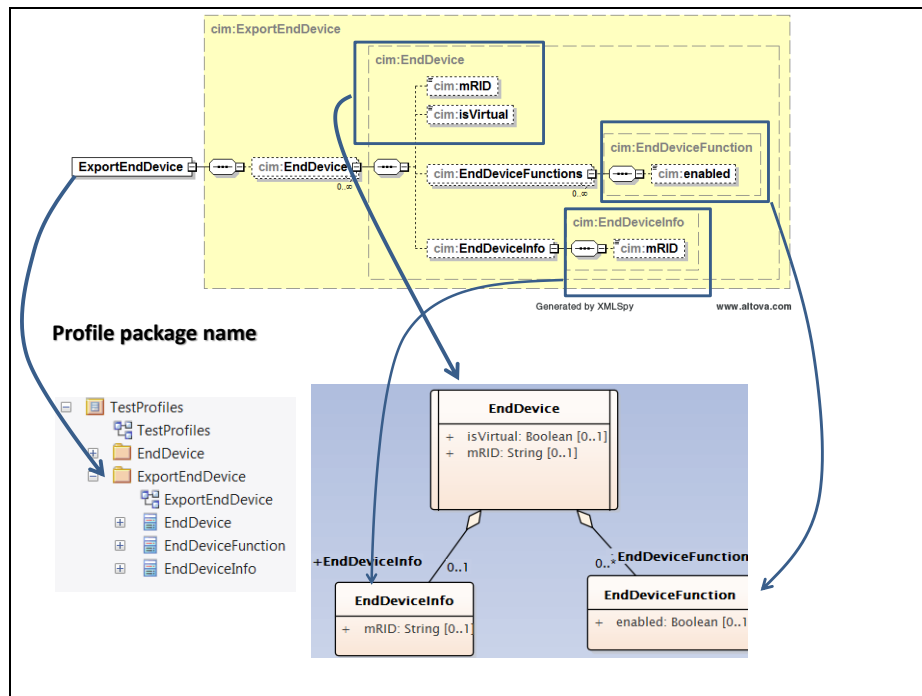
Select this upper package and launch "XSDToProf" menu. A window will open:



Select the xsd you want to map to UML, by opening the directory browser: it gives the location of the xsd file. Specify CIM packages that will be used to define IsBasedOn relationships. Select the type of drawn relationships (navigable or aggregate) in the UML Profile. By clicking OK, the mapping process will start and will end with the opening of an "END" pop up window.

The result is the following, under the package that was used to launch "XSSToProf":

- a new package has been created with the name of the schema top level element (here ExportEndDevice),
- UML classes have been created (EndDevice, EndDeviceInfo and EnDeviceFunction) with the appropriate attributes and relationships (including IsBasedOn relationships with CIM elements),
- A diagram has been created with the classes.



14. How to use CimSyntaxGen for HTML documentation generation

14.1. Overview

This feature allows you to generate an HTML document that describes all the elements of a given package with one associate UML diagram. It is a simple feature that is designed to handle simple package with a diagram that shows all classes.

Requirement: to be able to generate HTML documentation, the selected package should:

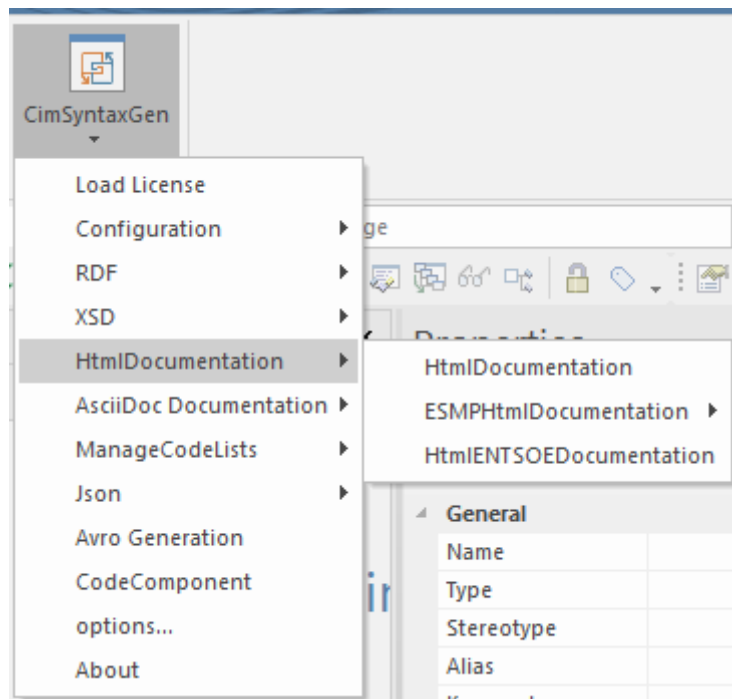
- be self-describing (i.e. the package must not rely on elements described in another package, otherwise the hyperlinks will not be active),
- have in the first diagram, all the classes and elements that are of interest (be sure to put this diagram in the first place).

14.2. Select profile package

Select the profile package from which you want to generate HTML documentation, and check that the most significant diagram is put in the first place.

14.3. Select HTML generation

Go to the EA “Add-Ins” Menu, select “CimSyntaxGen” then select one of the “htmlDocumentation” items:



- HtmlDocumentation
- ESMPHtmlDocumentation
- HtmlIENTOEDocumentation

15. Generic HtmlDocumentation generation

15.1. Overview

When selecting the generic HtmlDocumentation, you will be prompted with the usual Windows “save file as”. Select the folder where you want to put the HTML file. Give a name for the HTML file and “save”.

15.2. HTML file

In fact, in the chosen folder, two items are created:

- The HTML file whose name is the name given in the above step with the “.html” extension.

- One folder whose name is the name given in the above step appended with “_Images”. This folder contains all the images of the package UML diagrams. The image format is “png” and the name of each image is the name of the corresponding diagram.

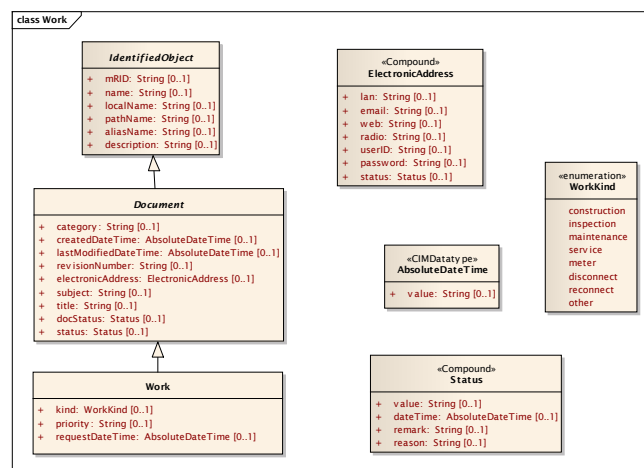
The HTML file gives you

1. Concrete Classes
2. Abstract Classes
3. Enumerations
4. Compound Types
5. Datatypes

In each category, items are put in alphabetical order.

15.3.HTML generation example using the CIM to build a Work Profile

- File name: “WorkProfile”
- EA Profile Package name: “Work”
- EA “Work” Profile Package Diagram name: “Work”
 - The diagram shows one concrete class (Work) that inherits from two abstract classes (IdentifiedObject and Document), along with appropriate Datatypes (AbsoluteDateTime), Compounds (Status and ElectronicAddress) and Enumeration (WorkKind).



HTML output:

- Image Folder name: “WorkProfile_Images”
Image file name: “Work.png”
- HTML file name: “WorkProfile.html”

- Example for Concrete Class:

Concrete Classes			
Work			
Document used to request, initiate, track and record work. This is synonymous with Work Breakdown Structure (WBS), which is traversed through the (currently informative) recursive association of Work. Note that the work name is equal to the WBS name, which is given in the inherited "name" attribute.			
Native Members			
kind	0..1	WorkKind	Kind of work.
priority	0..1	String	Priority of work.
requestDateTime	0..1	AbsoluteDateTime	Date and time work was requested.
Inherited Members			
category	0..1	String	see Document
createdDateTime	0..1	AbsoluteDateTime	see Document
docStatus	0..1	Status	see Document
electronicAddress	0..1	ElectronicAddress	see Document

- Example for Abstract Classes:

Abstract Classes			
Document			
Parent class for different groupings of information collected and managed as a part of a business process. It will frequently contain references to other objects, such as assets, people and power system resources.			
Native Members			
category	0..1	String	Utility-specific categorisation of this document, according to their corporate standards, practices, and existing IT systems (e.g., for management of assets, maintenance, work, outage, customers, etc.).
createdDateTime	0..1	AbsoluteDateTime	Date and time that this document was created.
docStatus	0..1	Status	Status of this document. For status of subject matter this document represents (e.g., Agreement, Work), use 'status' attribute. Example values for 'docStatus.status' are draft, approved, cancelled, etc.
electronicAddress	0..1	ElectronicAddress	Electronic address.
lastModifiedDateTime	0..1	AbsoluteDateTime	Date and time this document was last modified. Documents may potentially be modified many times during their lifetime.

- Example for Enumeration:

Enumerations	
WorkKind	
Kind of work.	
construction	
inspection	
maintenance	
service	
meter	
disconnect	
reconnect	
other	

- Example for Compound:

Compound types

ElectronicAddress

Work

Electronic address information.

lan	0..1	String	Address on local area network.
email	0..1	String	Email address.
web	0..1	String	World Wide Web address.
radio	0..1	String	Radio address.
userID	0..1	String	User ID needed to log in, which can be for an individual person, an organisation, a location, etc.

Example for CIMDatatypes:

CIMDatatypes

AbsoluteDateTime

Work

Date and time as "yyyy-mm-ddThh:mm:ss.sss", which conforms with ISO 8601. UTC time zone is specified as "yyyy-mm-ddThh:mm:ss.sssZ". A local timezone relative UTC is specified as "yyyy-mm-ddThh:mm:ss.sss-hh:mm". AbsoluteDateTime can be used both for calendar time, e.g. 2007-02-07T10:30, and for relative time, e.g. 10:30.

value	0..1	String	String representation of date and time, refer to description of the class.
-------	------	--------	--

16. How to use CimSyntaxGen for HTML ENTSOE Documentation generation

This HTML documentation generation is used to do output for CGMES profiles. The main difference with the generic output (see above section) is given by parameters of the configuration file (see configuration section):

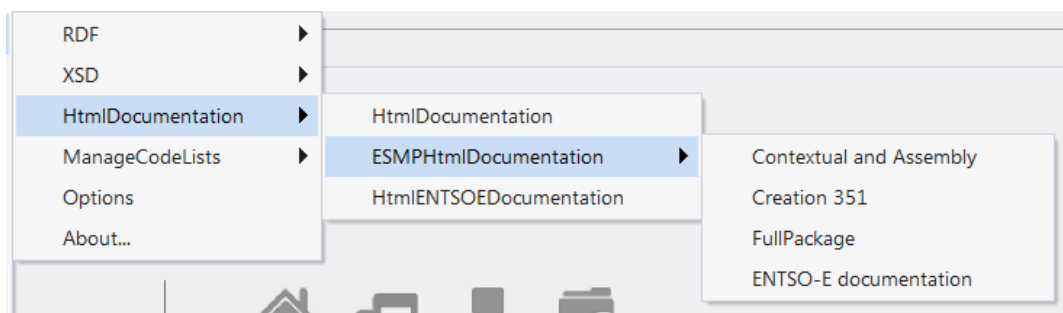
- UML diagrams could be exported or not in the documentation depending on if "EntsoeKeepUMLDiagrams" has been checked or unchecked in the configuration file.
- Hyperlinks between elements could be exported or not in the documentation depending on if "EntsoeKeepHyperlinks" has been checked or unchecked in the configuration file.
- A class inheritance path could be outputted if "EntsoeKeepInheritancePath": has been checked in the configuration file.
- For the dynamics package where images of explanations have been entered, those images could be exported if "EntsoeKeepClassDiagrams" has been checked in the configuration file.

There is also one difference, notes that have been attached to classes in the profile are exported in the description part of the class but highlighted in blue.

17. How to use CimSyntaxGen for ESMP HTML documentation generation

Overview

The goal of the documentation generation is to generate template that will be used to build IEC ESMP standards or ENTSO-E documentation, so the html generation is different from the above generations. Due to the structure of the ESMP profiling where there are several levels of modeling, the ESMP HTML generation is going to consider these different levels:



Four different documentation outputs are available:

- Generation of both Contextual and Assembly documentation template
- Generation of 62325-351 package
- Full package of multiple contextual and assembly models
- ENTSO-E documentation

As a basic rule, in a given document package, the Contextual Model shall be always before the Assembly Model

17.1. ENTSO-E Documentation

The generated html file outlines are as follows:

- Contextual model
 - Diagram of the contextual model
 - IsBasedOn dependency
- Assembly model
 - Diagram of the assembly model
 - IsBasedOn dependency

- List of classes: first the root class and then the other classes by alphabetic order. For each class, the attributes are ordered as per business requirements, i.e. as they will be listed in the XML schema. The associations between classes are also described and the order ranking is also provided.
- List of datatypes: the list of datatypes used within the document is provided, the order is alphabetical with first the compounds and then the CIM datatypes. In addition, for the CIM datatypes based on a codelist, the name of the codelist is provided.

17.2.Part 351 IEC standard generation

The “part 351” documentation, i.e. IEC 62325-351, is generated from the UML package “ESMPClasses” in the package “IEC62325-351”.

Once the “ESMPClasses” package is selected, select the option “*Creation 351*” and the process will be launched.

17.3.Conceptual and assembly models IEC standard generation

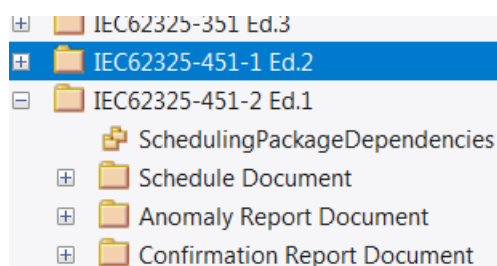
The conceptual and assembly models documentation for a document is generated from the document package, e.g. for the acknowledgement document by selection the UML package “Acknowledgement Document”.

NOTE: the order in the package is important, first the contextual document model and then the assembly document model, otherwise the generated document is not good.

Once the package is selected, select the option “*Contextual and Assembly*” and the process will be launched.

17.4.Set of conceptual and assembly models IEC standard generation

The conceptual and assembly models’ documentation for a set of documents is generated from the “standard” package, e.g. for the IEC 62325-451-2 by selection the UML package “IEC62325-451-2”. This will generate the documentation for the “Schedule Document”, the “Anomaly Report Document” and the “Confirmation Report Document”.



NOTE: Once the html file is generated, some updates are to be carried out to have an appropriate MS Word file as per the requirements of IEC, in particular, tables and figures numbering, standard styles to be used, etc. This is done by applying a Word macro.

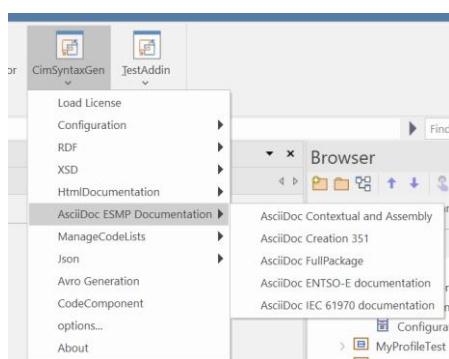
18.How to use CimSyntaxGen for AsciiDoc documentation generation

18.1.Overview

AsciiDoc is a markup language written in human-readable, plain text format. It supports all the structural elements necessary for writing notes, documentation, articles, books, eBooks, slideshows, web pages, technical manuals and blogs. The language can be used to produce a variety of presentation-rich output formats including HTML, PDF, EPUB, DocBook, man page.

18.2.Document generation

The generation can only be applied on packages and is started by the 'Ascii ESMP Documentation' menu:



The generation works analogously to the HTML documentation generation. Diagrams are stored as pictures in a subfolder 'image' created during generation.

NOTE: The generated AsciiDoc file refers to these pictures in this folder, i.e., changing the location of this folder requires an update of the references in the AsciiDoc document.

19.How to use CimSyntaxGen to Manage CodeList

19.1.Description

This feature is used for managing ESMP enumerations and XSD CodeLists. The "ESMPEnumerations" package contains all the enumerations used within ENTSO-E.

All the schema developed in the framework of the European style market profile are using external XML documents to provide the list of codes to be used in the various enumerations.

The following XSD documents are to be used:

- "*urn-entsoe-eu-wgedi-codelists.xsd*": the XSD to be used with all the CIM XSD.
- "*urn-entsoe-eu-local-extension-types.xsd*": the XSD to be used with all the CIM XSD.
- "*etso-code-lists*": the XSD to be used with all the ENTSO-E nonnamespace XSD.

This module of CIMSyntaxGen enables:

- to import an existing *urn-entsoe-eu-wgedi-codelists.xsd* in a package.
- to generate from the *ESMPEnumerations* package the three codelists (previously described) and the associated documentation.

Import

In order to carry out the imports, the following steps are to be carried out:

- Create a new package in the UML model with an associated class diagram.
- Select this new package and right click on the menu to select an "*ImportCodeLists*" in the "*ManageCodeLists*" menu,
- A dialog box will open, enter in the "*StdCodeLists*" field the path of the codelist to be imported,
- Click "OK" and the codes defined in the file "*StdCodeLists*" are imported.

The import function is to be carried out on an empty package; however, an empty "class" diagram is to be included in the package before running the import function.

The enumerations generated shall not be copied into the *ESMPEnumerations* package, as each enumeration has a unique GUID and enumeration is referred to, based on this GUID, in the *EMSPClasses* package.

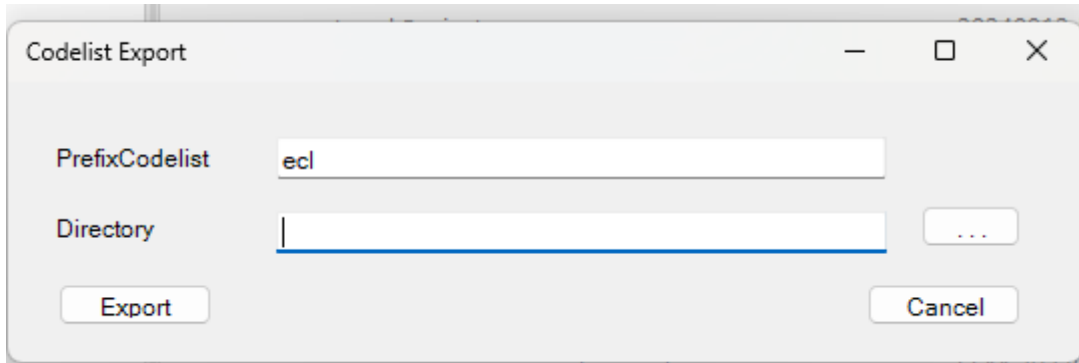
Only the attributes of enumeration could be copied from the import codelist in the *ESMPEnumerations* package.

19.2.Generation of the codelists and documentation

To carry out the export, the following steps are to be carried out:

- Select an Enumeration package and right click in the menu to select "*ExportCodeLists*".
- A dialog box will open.

- Enter the prefix to be used in the codelist (by default it is given by the configuration file profdata name="CodeListPrefix").
- Select the directory where the codelists shall be exported.
- Select "Export".



20.JSON Schema export

20.1.Overview

The export in JSON schema is done according to IEC62361-104-TS-Draft. This is still a draft. The mapping is like what has been done with IEC 62361-100 (XML NDR).

The mapping is done for different versions of the JSON schema specification and for two kinds of UML profiling: WG19 style and WG16 style.

- WG19 style: could have several root classes, use of special constructs like "Ref", "Union", "XOR", sorting xml elements in alphabetical order, etc.
- WG16 style: have only one root class.

NOTE: In the JSON implementation, the order of the attributes does not follow the order of the XML implementation.

20.2.JSON schema versions

JSON Schema specifications have several versions:

- <http://json-schema.org/draft-07/schema#>
- <http://json-schema.org/draft/2019-09/schema#>
- And a last version <http://json-schema.org/draft/2020-12/schema>

The supported versions are the first two ones.

NOTE: except draft-07, all schema versions are not yet supported by all validation tools.

20.3.UML profile styles

There are different UML profile designs:

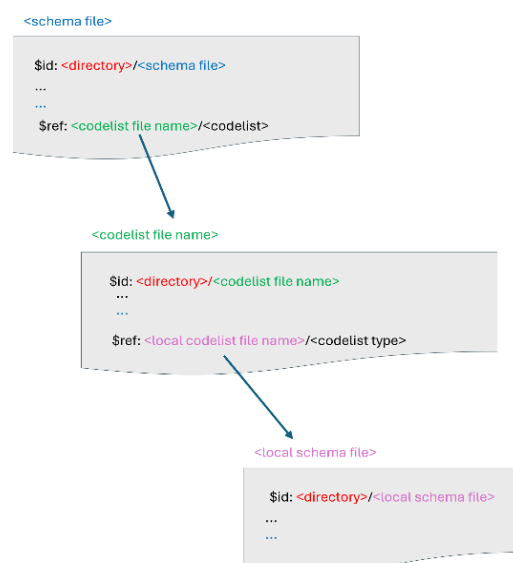
- According to WG14 design (called here WG19 design), that is mapped to the basic of IEC 62361-104.
- According To WG16 ESMP design, that is mapped to some alternate mapping and the use of external codelists.

So, the two mappings are provided.

20.4.Connectivity between Schema, Codelist and External Codelist files

There is a logical connection between the JSON schema file of a profile, the codelist file it refers to and an optional external/local codelist schema file. For the generation of the basis template of the external/local codelist schema file see 20.5.

The following figure depicts the connectivity of these schema files.



Due to the lack of a comprehensive namespace functionality of JSON, contrary to XSD schemas, the JSON schema files contain references to the directories and file paths of schemas they refer to.

20.5.External CodeList Management

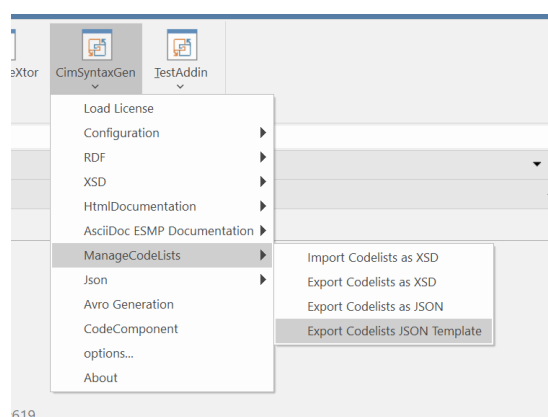
The ability to refer to external CodeLists guarantees that the CodeLists are always up to date and could be extended locally.

For instance, ENTSO-E manages the standardized lists as XML schemas which are available on ENTSO-E site. ENTSO-E provides a package with XML instance files defining the ENTSO-E CodeLists and a sample local extension to these. The latter enables one to define local codes to be used based on bilateral agreements with other parties, e.g. based on specific market rules for a local market.

The JSON schema mapping offers a way to have the same kind of feature as for XSD.

CimSyntaxGen provides a functionality to generate a sample external codelist JSON schema file that can be used as template to ease the manual editing of such a local codelist schema file.

NOTE: The generation is based on a package containing elements with the stereotype 'enumeration' and name ending 'Typelist' or 'typelist'.



The export dialogue needs an output directory (will be created if not existent) and a name of the template file.

The following figure depicts an excerpt of the template file:

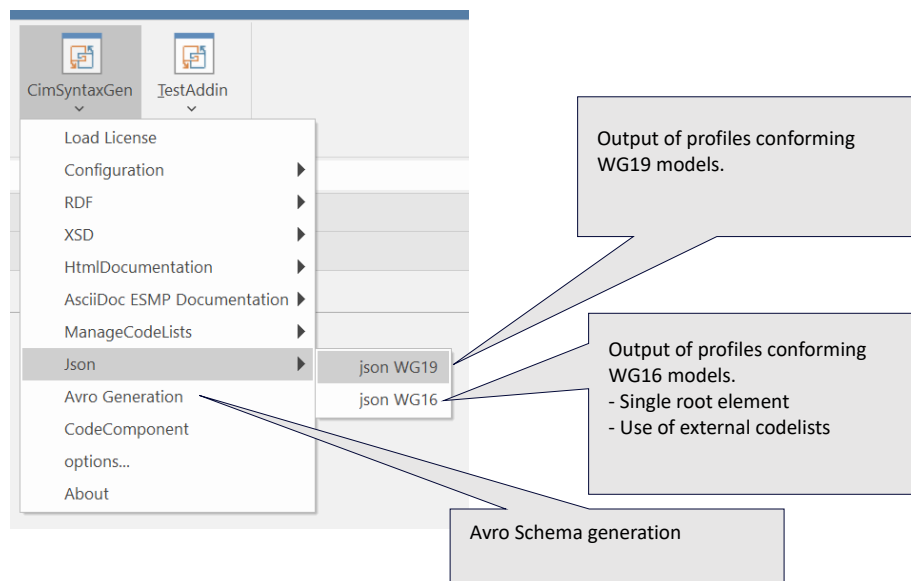
```
{
  "$id": "",
  "$schema": "http://json-schema.org/draft-07/schema#",
  "title": "",
  "description": "",
  "namespace": "",
  "definitions": {
    "CodeLists": {
      "$ref": "#"
    },
    "LocalAllocationModeTypelist": {
      "id": "#LocalAllocationModeTypelist",
      "title": "",
      "description": "",
      "type": "string",
      "enum": [
        "A09"
      ]
    },
    "LocalAnalogTypelist": {
      "id": "#LocalAnalogTypelist",
      "title": "",
      "description": "",
      "type": "string",
      "enum": [
        "A01"
      ]
    },
    "LocalAssetTypelist": {
      "id": "#LocalAssetTypelist",
      "title": "",
      "description": "",
      "type": "string",
      "enum": [
        "A09"
      ]
    },
    "LocalAuctionTypelist": {
      "id": "#LocalAuctionTypelist",

```

20.6.Launching Json schema export

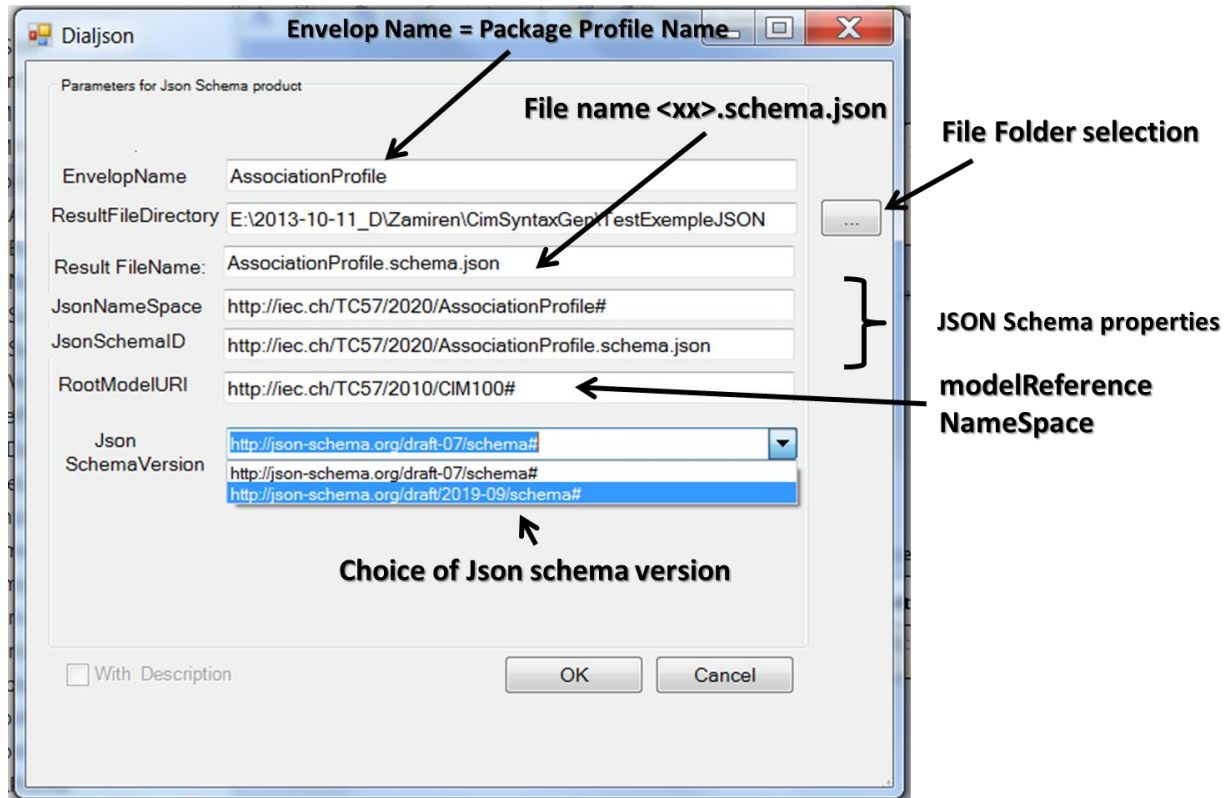
To launch JSON Schema export, select a profile package and then "Json" on the menu. Two choices are offered:

- Export for WG19 style UML profiles
- Export for WG16 ESMP style UML profiles.



21.Generation of JSON Schema WG19 Style

When JSON/WG19 is selected, the first window is popping up:



Fill the given fields:

1. The envelop name is the package name of the profile,
2. Choose the result Json schema file location,
3. By default the File name is the Envelop name appended with ".schema.json"
4. Edit the profile "Json schema namespace",
5. By default the profile "Json schema ID" is the base URI of the "JsonSchemaNamespace" appended by ".schema.json"
6. Give the URI of the Information Model
7. Choose the Json Schema specification version the export will comply with two choices:
 - Draft-07
 - Draft 2019-09.
8. Click OK

A dialog pops up to specify how many root classes you are allowing in the instance. Fill MinOccur and MaxOccur fields and click OK. And end windows will appear and click OK to stop the process.

22.Generation of JSON Schema WG16 Style

22.1.Launching Json schema export

When JSON/WG16 is selected, the first window is popping up:

Parameters for Json Schema Generation of WG16

EnvelopName	Model
Canonical Base URI	http://iec.ch/TC57/2013/
JsonSchemaID	file:///C:/Users/amrodriguez/Desktop/JSON/Model.schema.json
RootModelURI	http://iec.ch/TC57/2013/CIM-schema-cim16#
Json Schema Version	http://json-schema.org/draft-07/schema#
JsonNamespace	file:///iec.ch/TC57/2013/Model#

Output

ResultFileDirectory	C:\Users\amrodriguez\Desktop\JSON	...
Result FileName:	Model.schema.json	

Codelist

CodeList	Schema File Name	C:\TestCIMContextor\codelist	...
Local CodeList	Schema File Name	C:\TestCIMContextor\localCodelist	...

Generate Cancel

Fill the given fields:

1. The envelop name is the package name of the profile,
2. The canonical Base URI of the schema.
3. The "Json schema ID" can be specified here in the dialog or specified in the configuration.
4. Give the URI of the Information Model
5. Choose the Json Schema specification version the export will comply with, two choices:
 - Draft-07
 - Draft 2019-09.
6. Edit the profile "Json schema namespace",
7. ResultFileDirectory specifies the directory where the output file is stored.
8. Result FileName specifies the file name of the output.
9. Select location of Standard Codelists.
10. Select location of local Codelists XSDs.
11. Click OK

22.2.CodeList management

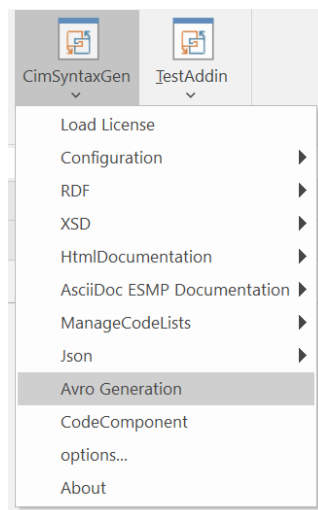
To be able to use external codelists with Json schema, the location for Json codelist schema must be imposed at location that could be reached by any profile JSON schema.

23.Avro Schema export

23.1.Overview

There is not yet an official specification giving the mapping of CIM UML artefacts to AVRO schema artefacts (like the mapping for XML and JSON schema). However, ENTSO-E provides a specification according to which the AVRO schema generation was implemented.

The generation is started by selecting the menu entry 'Avro Generation':



After entering the export directory and filename and running the generation, the AVRO schema file is exported.

NOTE: AVRO schema files have the extension ".avro".

23.2.Codelist

AVRO schema does not allow the use of external CodeLists, because there is no mechanism like JSON Pointer. Therefore, the mapping of an element of a CodeList is done in the AVRO schema at the requested level in the form of an array.

NOTE: AVRO schema name for enums must match the "name" syntax starts with [A-Za-z_], subsequently contains only [A-Za-z0-9_]. So, a code cannot start with a number. A codelist that includes such code must be changed. For the converter, such codes were prefixed with "z_" to keep

track of the code. Example in EntsoE "StandardReasonCodeType" there is a code whose value is "999", it has been changed to "z-999".

24.Code Component export

24.1.Overview

The menu allows generation of a Core Components delivery package for WG16 ESMP Code Components.

The Core Component delivery package is a zip file, which includes:

- a manifest xml file (conforming to the IEC Manifest XSD) that describes:
 - the publication(s) from where the CodeComponents are defined,
 - the CodeComponent file(s) description,
 - the History file(s) describing the changes which have been considered in the associated package, since the last IEC publication (at least).
- an IEC copyright XSD file.
- an IEC Manifest XSD file.
- the code component file(s) extracted from the IEC publication(s) (typically XSD file, XML file, SNMP MIB file ...),

All these files must be generated before the launch of CoreComponent menu, except the XML IEC copyright and Manifest files that are generated at the time of the export.

24.2.IEC Copyright files

The XSD IEC Copyright file ("*IECCopyright.xsd*") is located in:

"C:\Users\<UserName>\AppData\Roaming\ENTSO-E\CimSyntaxGen\Ressources\".

This XSD is used to generate the specific IEC Copyright xml file (" for the given Code Component export. And this xml file is located also in CimSyntaxGen\Resources folder.

24.3.Delivery package name

The delivery package is standardized as follow:

{RefStandard}.{CodeComponentName}.{VersionStateInfo}.{LightFull}{PublicationStage}.zip

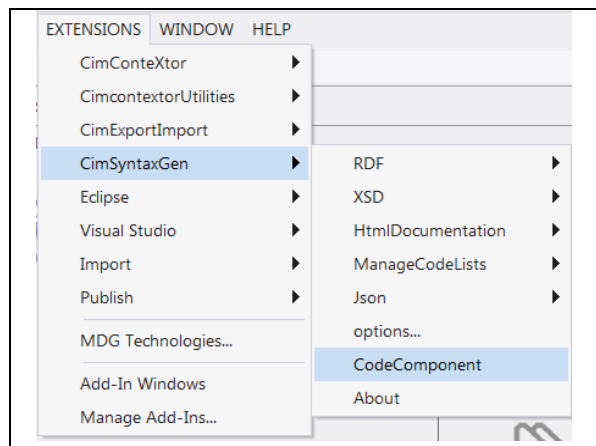
Where:

- RefStandard is the IEC standard name in the form of:

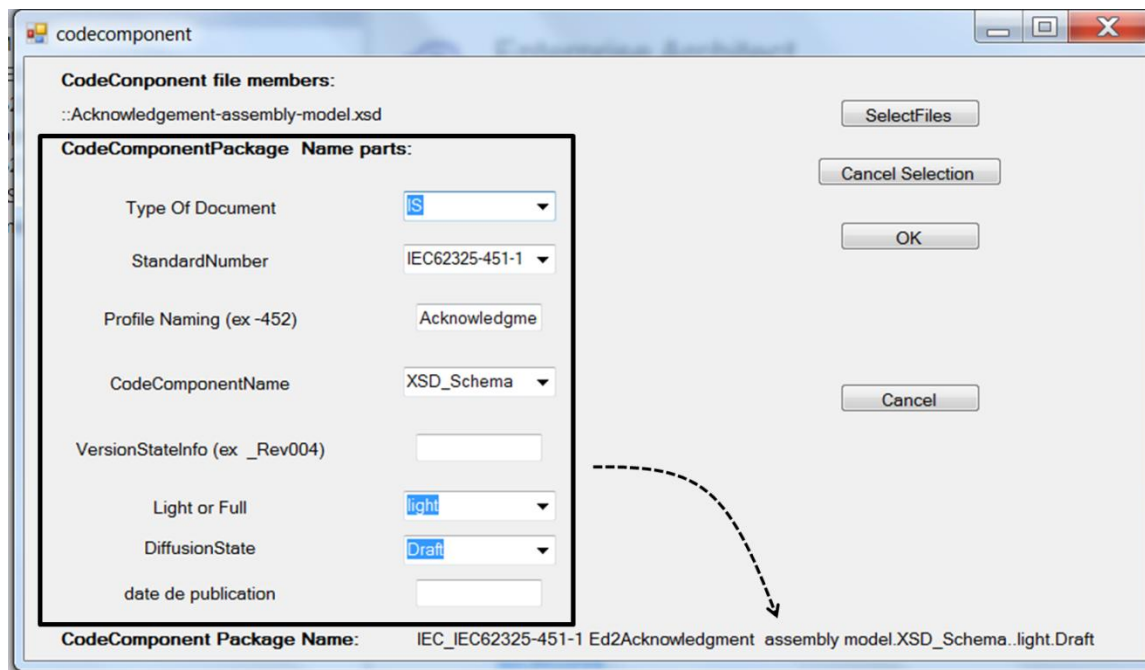
- `{IEC{StandardType}_{StandardNumber}.{PublicationYear}{EditionInfo}}`
- StandardType = { _TR | _TS } or empty if IS,
- StandardNumber = 5 digits IEC numbering,
- PublicationYear = 4 digits' year number,
- (optional) EditionInfo = _ed{Version}.{Amendment},
- Example: "IEC_TR_61850-90-4.2013_ed1.0",
- CodeComponentName: designates a part of the parent IEC deliverable which is of type Code Component ("XSD-Schema, RDF Schema"),
- VersionStateInfo: Code component version,
- LightFull: indicates whether the content included in the considered file, fully reflects the Code component content (full) or reflects just a part of it (light).
- PublicationStage: reflects the IEC stage of the considered code component. Typically = {Draft} or nothing if this is an official publication.

24.4.Launching CodeComponent

Select CodeComponent menu:



A windows opens:



25.CimSyntaxGen configuration file

25.1.Managing configuration file

See 5.6.

25.2.Overview

The goal of this file is to provide parameters for *CimSyntaxGen* that could be tailored by the user to fulfill its requirements. It means that the user could change some parameters and add new ones.

The file comes with two different kinds of parameters:

- **appSettings:** this gives the state of elements used in the Option Menu or by some processes,
- **dataProfiles:** this gives schema parameters generation or Html documentation generation parameters.

25.3.AppSettings parameters

"AppSettings" parameters are defined as xml element called "configuration" with a name and a value. Example:

```
<configuration name="Xmlbase" value="Checked" />
```

Some of those parameters are used for defining the Option Menu where the user can check or uncheck parameter boxes. The "AppSettings" give the default value for the option parameters: checked or unchecked boxes:

The 'Option' dialog box contains the following settings:

- AppSettings:**
 - ☒ Log
 - ☒ Xmlbase
 - ☒ StereotypeNameSpaces
 - ☐ ModeBatch
 - ☐ SawsdlNoPack
- DataProfile:**

DataProfile	Stereotype	HtmlDoc	RDFS
☐	European	☒	☒
☐	Entsoe	☒	☒
☐	Operation	☒	☒
☐	Abstract	☐	☒
☐	ShortCircuit	☒	☒
☐	Description	☒	☒
- Buttons:** new, delete
- Logging:**
 - ☒ Logging On/Off
 - Log File Directory: C:\Users\lovene\AppData\Roaming\ENTSO-E\CimS
- Buttons:** Save, Cancel

These parameters are:

- Log: enable or not the log of processing results,
- Xmlbase: enable the use of an Xmlbase element in the schemas,
- StereotypeNameSpaces: enable the use of specific namespaces for the stereotyped elements,
- ModeBatch: Batch mode processing enabled,
- SawsdlNoPack: output or not of packages name in the sawsdl attribute in case of XSDByRef generation,
- dataProfile: not use,
- Html → check or uncheck is given by profstereo parameter,

- RDFS➔ check or uncheck is given by profstereo parameter,

These parameters values are updated when doing a "Save" in the Options Window.

There are some more parameters that are not related to the Option Menu; they are not shown, but they drive CimSyntaxGen behavior:

- *VersionConfig*: version of the configuration (updated manually),
- *ImportCodeList*: if check CodeList schemas are imported in the profile schema, otherwise they are included,
- *Esmf*: deprecated,
- *ModeTest*: reserved for debugging,
- *EntsoeKeepInheritancePath*: used for Entsoe Html Documentation generation, if checked enable the output of inheritance path field for a class,
- *EntsoeKeepUMLDiagrams*: used for Entsoe Html Documentation generation, if checked enable the output of UML diagrams,
- *EntsoeKeepClassDiagrams*: used for Entsoe Html Documentation generation of "Dynamics package", if checked enable the output of Notes diagrams,
- *EntsoeKeepHyperlinks*: used for Entsoe Html Documentation generation, if checked enable the output of hyperlinks in the documentation,
- *EntsoeManageCodeLists*: make menu for manage Code list,
- *NavigationEnabled*: defines if the relation for hierarchy class is shown as an arrow or an aggregation,
- *XSDByRef*: is XSDByRef menu is activated or not,
- *SelectedCanonicalPackagesForXsdToProf*: defines what are the information model packages the imported schema UML mapping will be based on,
- *ImageExt*: gives the image format of the exported diagram files; could be .emf, bmp, .jpg, .gif, .png, .tga.

25.4.DataProfile parameters

These parameters are specific for profile management; they are called "*dataprofile*". There are four kinds of parameters that are used to drive schemas and documentation generation:

- **profdata**: these are simple parameters that have a name and a value, example:

```
<profdata name="DefaultXSDFilePath" value=".\essai.xsd"/>
```

- **profstereo** these are complex parameters that have a name and some other items, example:

```
<profstereo name="ShortCircuit" prefix="cim" uri="http://iec.ch/TC57/2013/CIM-schema-cim16#" html="Checked"
rdfs="Checked" />
```

- **shacldata**: these are simple parameters used for constraints export and that have a name and a value, example:

```
<shacldata name="descriptionAttributeCardinality" value="This constraint validates the cardinality of the
property (attribute)." />
```

- **jsondata**: these are simple parameters used for json export and that have a name and a value, example:

```
<jsondata name="JsonSchemaVersions" value="http://json-schema.org/draft-07/schema#|http://json-
schema.org/draft/2019-09/schema#" />
```

25.5.Profdata parameters

ProfData General:

These parameters are used for the edition of the header of the xsd file.

- *DefaultXSDFilePath*: default xsd file location in the directory,
- *CurrentXSDFilePath*: current xsd file location,
- *DefaultTargetNamespace*: default target namespace for the schema,
- *ProfileNamespace*: URI of the profile namespace
- *TargetNamespace*: value of the target namespace
- *DefaultProfileNamespace*: profile default namespace.

ProfData XSD WG19

- *DefaultRootModelURI*: default value for the URI of the model used as the model on which the profile is based on,
- *RootModelURI*: value of the URI of the model used as the model on which the profile is based on,
- *Prefix*: value of the prefix for the profile namespace,
- *EnvelopName*: name of the profile,
- *Version*: version number for this profile.

ProfData XSD WG16

- *URICodelist*: current value of the Codelist URI, Codelist namespace,
- *PrefixCodelist*: prefix for the code list namespace,
- *URISchemaLocation*: codelist schema location,

- *CodeListLocalLocation*: codelist local extension schema location,
- *CodeListLocation*: value="urn:entsoe.eu:wgedi:codelists"
- *ExtensiontXSDFilePath*: current extension xsd file location.

ProfData RDFS

- *ListStereoNamespace*: list of stereotypes used in the profile, example "Entsoe|ShortCircuit|Operation|Abstract"
- *ListStereoNamespaceExportable*: list of stereotypes that will be used to be exported or not in an equipment profile rdf schema, example "ShortCircuit|Operation"
- *EntsoeEquipmentProfile*: name of the package for the equipment profile,
- *EntsoeDataTypesDomain*: name of package where there are profile datatypes,
- *EntsoeExpStereo*: name of the stereotype used to specify special processing for element having this stereotype,
- *PackageExtension*: name of the extension package,
- *EntsoeVersion*: name of the version class for EntsoE profiles.

ProfData XSDToProf

- *ListSingularName* value="Status|Address|StreetAddress" For XSDToProf use of singular name.

ProfData for Copyright

- *CCTypeOfDocument*: type of document (IS/TS/TR)
- *CCStandardNumber*: standard reference (62325, 61970, 61968, 61970-600 ed1, 61970-600 ed2)
- *CCProfileName*: profile name
- *CCCodeComponentName*: kind of component (RDFS_Schema, XML Schema)
- *CCDiffusionState*: diffusion level (Draft/IEC/ENTSOE/ENTSOEdraft/others)
- *CCVersionStateInfo*: version of component
- *CCselectedProfiles*: name of the file selected for Copyright information
- *CCLightFull*: component with or without elements description (full/light)
- *CCPublicationDate*: date of component generation.

ProfData CodeComponent

- *CCSelectedFiles*: name of files in the CodeComponent zip file
- *CCPackageName*: name of the CodeComponent zip file.

25.6.Profstereo parameters

These parameters are for Stereotypes processing for RDFS processing and HTML documentation processing. The template for "*profstereo*" is:

- Name: stereotype name,
- Prefix: prefix for the stereotype namespace,
- Uri: URI of the stereotype namespace,
- Html: says if the stereotype is used for Html documentation,
- Rdfs: says if the stereotype is used for RDFS generation.

25.7.SHACL parameters

These parameters (name and value) are used for RDFS SHACL constraints export and gives either the definition of a constraint or the message that will be generated when the constraint is validated against an instance.

- *descriptionAttributeCardinality*: description used for a SHACL attribute cardinality definition (example: "This constraint validates the cardinality of the property (attribute).").
- *messageAttributeCardinality*: description of the message that will be delivered in case the SHACL attribute cardinality rule fails (example: "Cardinality violation. Upper bound shall be 1 | Cardinality violation. Missing required property (attribute). | Cardinality violation. Upper bound shall be 1 or missing required property (attribute)").
- *descriptionAssociationCardinality*: description used for a SHACL association multiplicity definition (example: "This constraint validates the cardinality of the association at the used direction.").
- *messageAssociationCardinality*: description of the message that will be delivered in case the SHACL association multiplicity rule fails (example: "Cardinality violation. Upper bound shall be 1. | Cardinality violation. Missing required association. | Cardinality violation. Upper bound shall be 1 or missing required association.").
- *descriptionValueType*: description used for a SHACL ValueType definition (example: "This constraint validates the value type of the association at the used direction.").
- *messageValueType*: description of the message that will be delivered in case the SHACL ValueType rule fails (example: "One of the following does not conform: 1) The value type shall be IRI; 2) The value type shall be an instance of the class: {the IRI of the class}.").
- *descriptionDatatype*: description used for a SHACL DataType definition (example: "This constraint validates the datatype of the attribute").
- *messageDatatype*: description of the message that will be delivered in case the SHACL DataType rule fails (example: "The datatype is not literal, or it violates the xsd datatype. | The datatype is not IRI (Internationalized Resource Identifier) or it is an enumerated value

which is not part of the enumeration. | Blanknode (compound datatype) Violation. Either it is not a BlankNode (nested structure, compound datatype) or it is not the right class").

25.8.JSON parameters

The parameters (name and value) are used for JSON Schema export:

- *LastJsonExportedFile*: URI of the last exported Json schema (use for Avro schema converter)
- *ForWg*: JSON style export (wg19 or wg16)
- *CanonicalBaseURI*: URI of the canonical model use for the profiles (use for modelReference: example <http://iec.ch/TC57/2020/>).
- *JsonNameSpace*: Json schema namespace keyword (example: <http://iec.ch/TC57/2020/CIM-JsonProfiles/EndDeviceEvents#>).
- *JsonSchemaID*: ID of Json schema (used for \$id keyword, example: <https://iec.ch/TC57/2020.EndDeviceEvents.schema.json>).
- *JsonSchemaVersions*: Json schema specification the Json schema export is conformed to (possible value: <http://json-schema.org/draft-07/schema#>|<http://json-schema.org/draft/2019-09/schema#>).
- *JsonSchemaVersion*: Json schema specification used for the export (example: <http://json-schema.org/draft-07/schema#>).
- *CodeListModelURI*: URI of the model the codelists are referring (used for modelReference in Codelists, example: <http://iec.ch/TC57/2010/62325-351-Ed3#>).
- *JsonCodeListSchemaLocation*: Location of the standard XSD codelists that will be used for Json Standard codelist export.
- *JsonLocalCodeListSchemaLocation*: Location of the local XSD codelists that will be used for Json Local codelist export.

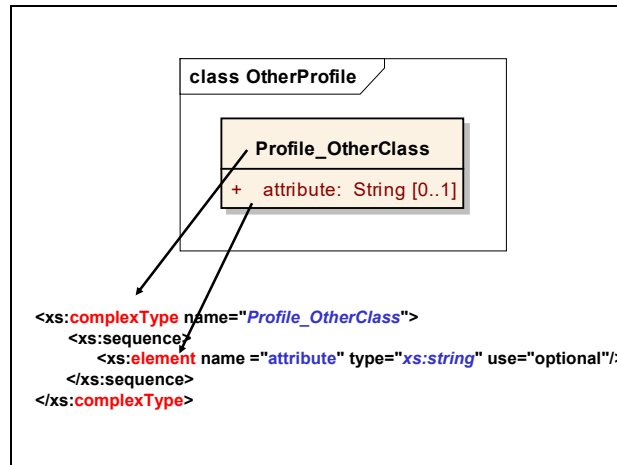
26.UML/XSD generation principles

26.1.XSD generation basic principles

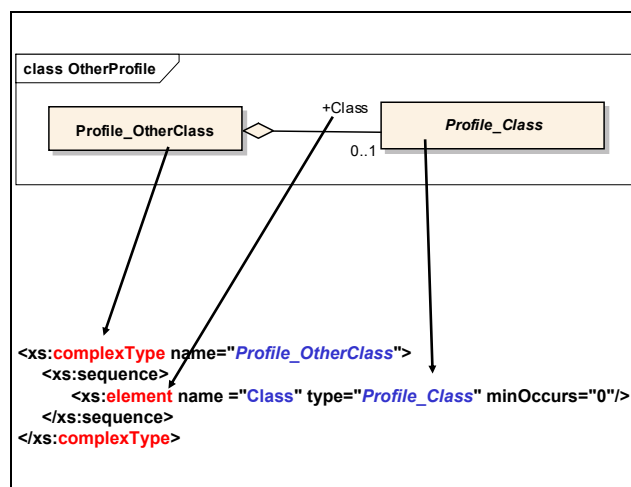
The basic principles of XSD generation are:

- Each UML class is generated as an XML "*complexType*" declared globally, and its name is the UML class name,
- Each UML class attribute is generated as an XML local element inside the

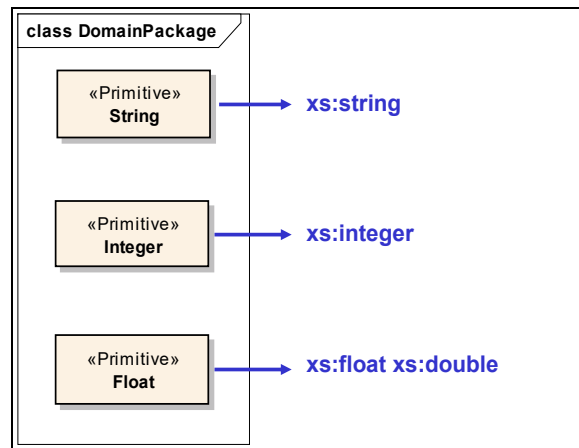
corresponding class associated with "complexType", whose name is the UML attribute name and whose type name is the UML attribute type name:



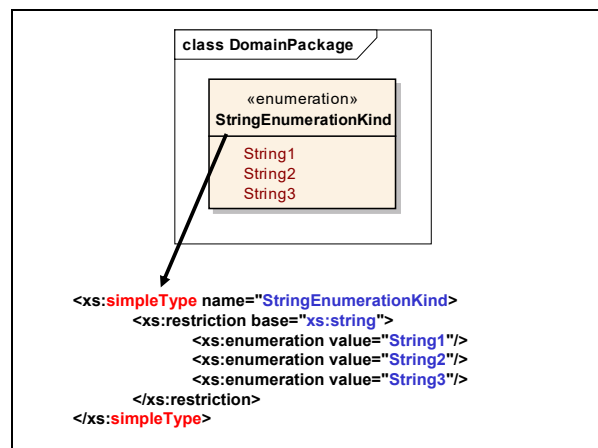
- Each UML class aggregation is generated as an XML local element inside the corresponding class associated "complexType", whose name is the UML aggregation end role name and whose type name is the UML associated class name:



Each UML Primitive is mapped to an XML datatype or a union of XML datatypes:



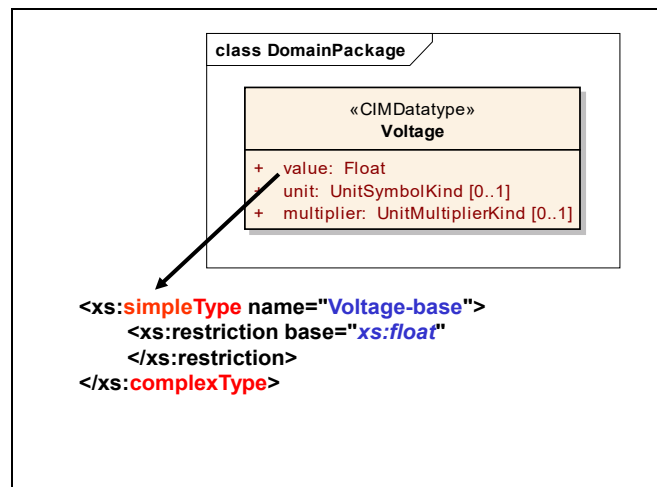
Each UML enumeration is mapped to a `simpleType` that is an enumeration restriction based on an XML primitive (usually an `xs:string`) and whose name is the UML enumeration name :



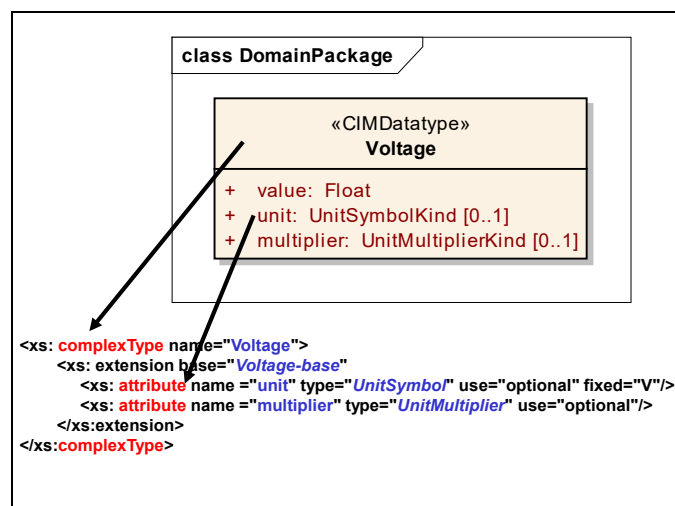
26.2.CIMDatatype generation principles

UML CIMDatatype generation are a special use case:

- Each UML CIMDatatype value attribute is mapped to a *simpleType* whose name is the name of the UML CIMDatatype concatenated to an hyphen followed by the term *base*:

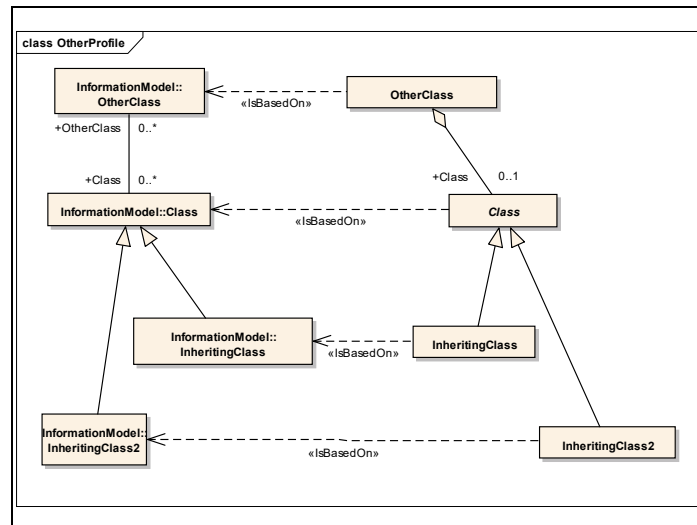


- Each UML CIMDatatype is then mapped to a "complexType", whose name is the UML CIMDatatype name. This complex type is an extension of "simpleType" defined before, where all other UML datatype attributes (different from the attribute "value") are mapped to XML attributes:

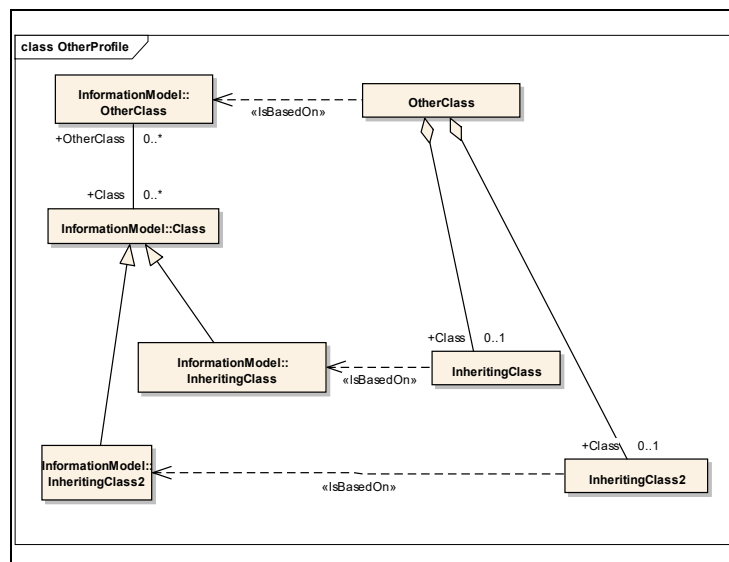


27.Subclass inherited association and XSD generation principles

In case of inheritance, dealing with profile subclass inherited association end role name when generating an XSD requires special mapping. In a straight application of the rule for association end role name mapping there could be an element name duplication. This could happen in the following examples:



or



The standard XSD association mapping would be :

```
<xs:complexType name="OtherClass">
  <xs:sequence>
    <xs:element name="Class" type="InheritingClass"/>
  </xs:sequence>
</xs:complexType>
```

In XML, a schema could not have two elements with the same name but with different types.

To avoid invalid XSDs, new mapping rules must be used, and these rules depend on the terms that are used for the UML end role name. There are three cases:

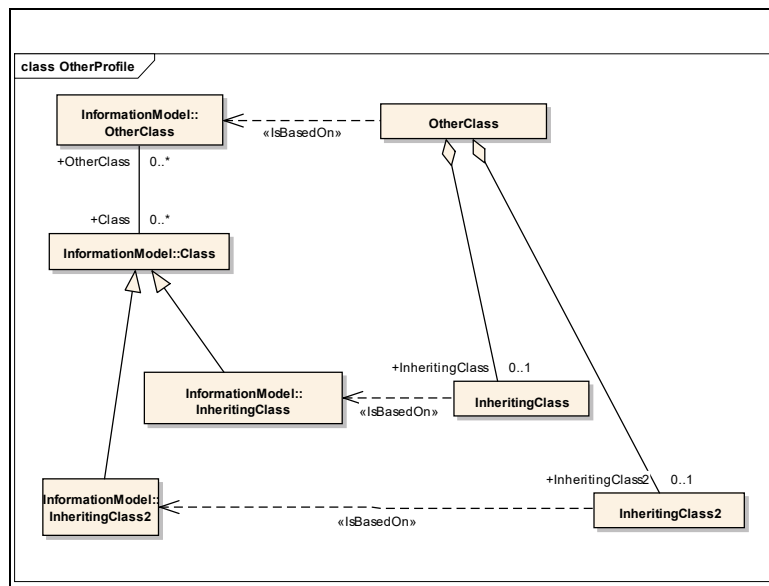
1. End role name matches the associated abstract super class name,
2. End role name matches a qualifier term concatenated to an underscore that prefixes the associated abstract super class name,
3. End role name has no relation to the associated abstract super class name.

First case : end role name matches associated super class name

In this case the mapping is: for each association with a sub-class, the association will be mapped to an element whose name will be the UML sub-class name (instead of the super class name). In the example “*Class*” is the end role name and the super class name is “*Class*”, then the XSD generation will produce:

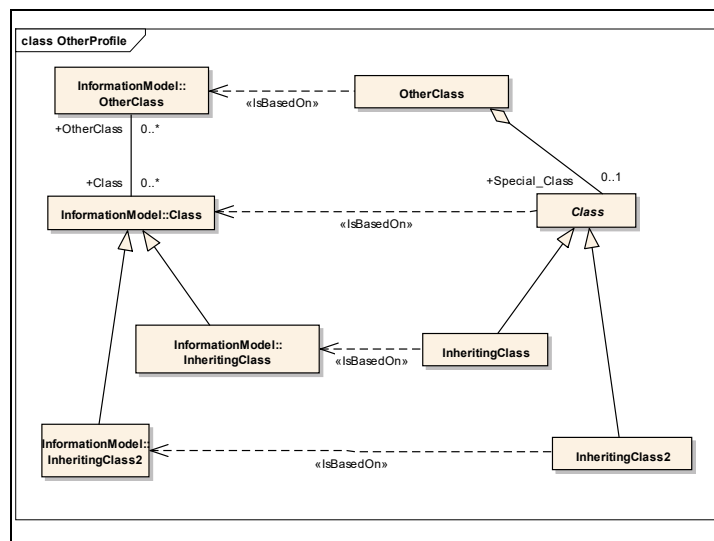
```
<xs:complexType name="OtherClass">
  <xs:sequence>
    <xs:element name="InheritingClass" type="InheritingClass"/>
    <xs:element name="InheritingClass2" type="InheritingClass2"/>
  </xs:sequence>
</xs:complexType>
```

This in fact is equivalent to the following UML diagram where the end role names would have been changed to be the same as those of the subclasses:



Second case : end role name is a qualifier, and underscore and super class name

At the information model level and/or profile level the end role name could be qualified and thus have a qualifier term.



In this case the mapping is: for each association with a sub-class, the association will be mapped to an element whose name will be the qualifier term followed by and underscore followed by the UML sub-class name (instead of the qualifier term, the underscore and the super class name). Example end role name is "*Special_Class*", super class name is "*Class*", then the XSD generated will be:

```
<xs:complexType name="OtherClass">
```

```
<xs:sequence>
```

```
<xs:element name="Special_InheritingClass" type="InheritingClass"/>
```

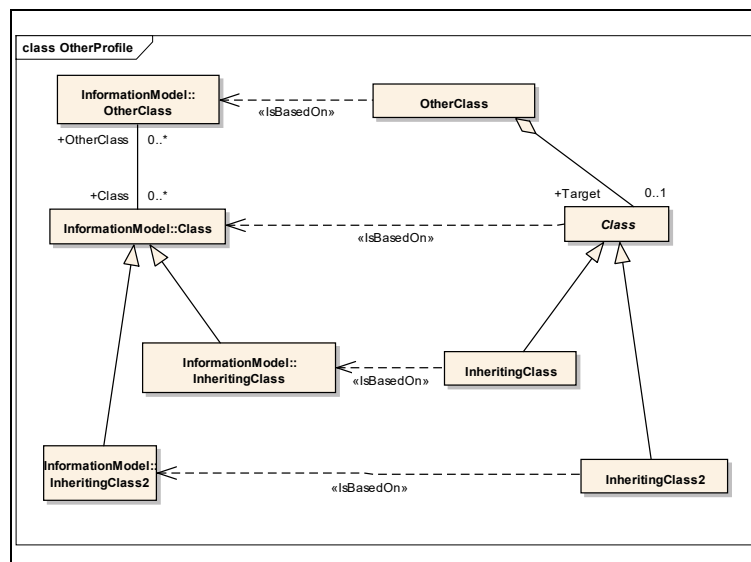
```
<xs:element name="Special_InheritingClass2" type="InheritingClass2"/>
```

```
</xs:sequence>
```

```
</xs:complexType>
```

Third case: end role name has no relation with super class name

At Information Model level, end role name has nothing to do with the super class name.



In this case the mapping follows this rule: for each association with a sub-class, the association will be mapped to an element whose name will be the end role name followed by and underscore followed by the UML sub-class name (instead of the qualifier term, the underscore and the super class name). Example end role name is “Target”, super class name is “Class”, then the XSD generated will be:

```
<xs:complexType name="OtherClass">
```

```
<xs:sequence>
```

```
<xs:element name="Target_InheritingClass" type="InheritingClass"/>
```

```
<xs:element name="Target_InheritingClass2" type="InheritingClass2"/>
```

```
</xs:sequence>
```

```
</xs:complexType>
```