

CIMCONTEXTOR MANUAL

v11.6.0 | 10 September 2026

For CimConteXtor software version 3.3.10

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

CimContextor manual.....	1
TABLE OF CONTENTS.....	3
EXECUTIVE SUMMARY	12
LICENSING	13
Notice.....	13
1. INSTALLATION AND OVERVIEW	14
1.1. Platform Architecture.....	14
1.2. Downloading “CimConteXtor”	14
1.3. Before installing “CimConteXtor”	14
1.4. Installation	14
1.5. Overview.....	15
1.6. License	15
1.7. Resource’s directory	16
1.8. Configuration	16
1.8.1. Installing and using the CimConteXtor Add-in without an external configuration file	16
1.8.2. Import of a custom configuration.....	16
1.8.3. Changing the configuration.....	17
1.8.4. Export of a configuration	17
Configuration parameters.....	18
2. BEFORE USING CIMCONTEXTOR	18
2.1. Get some ideas about UML modelling and profiling methodology.....	18
2.2. Be aware of naming rules	18
2.3. Define your Project Environment specific elements	19
2.4. Define package template.....	21
2.5. Define datatypes package template and dependencies.....	21
2.6. Define “IsBasedOn” Package Dependencies	21

2.7.	Do not forget	24
3.	CIMCONTEXTOR PRESENTATION.....	25
3.1.	General	25
3.2.	“About” Menu	26
3.3.	“Options” Menu	26
4.	"CREATE PROFILEPACKAGE" WITH CIMCONTEXTOR	29
5.	CREATE “ISBASEDON ELEMENTS” WITH CIMCONTEXTOR	30
5.1.	IsBasedOn Drag and Drop function	30
5.2.	Qualify an “IsBasedOn” element Name	33
5.3.	“IsBasedOn” relation in EA.....	34
6.	CREATE “ISBASEDON” PRIMITIVES WITH CIMCONTEXTOR	34
6.1.	Overview.....	34
6.2.	Drag and drop a Primitive, create a Primitive	34
6.3.	Drag and Drop a Primitive, create a CIMDatatype	35
6.4.	Qualify the IsBasedOn CIMDatatype	36
6.5.	Copy Parent Element.....	36
6.6.	Define value space restrictions	36
7.	CREATE “ISBASEDON ENUMERATION” WITH CIMCONTEXTOR	39
7.1.	Reminder	39
7.2.	Create an <<enumeration>>	39
7.3.	Create an “IsBasedOn” Enumeration with <i>CimConteXtor</i>	40
7.4.	Copy Parent Element.....	41
7.5.	Selecting “EnumeratedLiterals”	41
7.6.	Editing an IsBasedOn enumeration:	41
8.	CREATE “ISBASEDON DATATYPES” WITH CIMCONTEXTOR	41

8.1.	Qualify the IsBasedOn Datatype	42
8.2.	Copy Parent Element.....	42
8.3.	Define attribute cardinality	43
8.4.	Define CIMDatatype value space	43
8.5.	Define default or fixed value.....	43
8.6.	Define Datatype value domain	44
9.	CREATE “ISBASEDON COMPOUND” WITH CIMCONTEXTOR.....	44
10.	CREATE “ISBASEDON CLASS” WITH CIMCONTEXTOR	44
10.1.	Overview.....	44
10.2.	Select Information for the IsBasedOn class (Options Menu).....	45
10.3.	Copy parent class.....	45
10.4.	Copy profile class Notes, Tagged Values, and Constraints	46
10.5.	Edit profile class Notes and Tagged Values of the profile (IsBasedOn) class	47
10.6.	Edit profile class Constraints	47
10.6.1.	Overview.....	47
10.6.2.	Edit a new constraint from scratch:.....	48
10.6.3.	Use of constraint template	50
10.6.4.	Use of predefined constraints:.....	50
10.7.	Edit profile class Stereotype	52
10.8.	Define profile class as a Root class	52
10.9.	Define profile class as an Abstract Class	52
10.10.	Inherits	52
10.11.	Define Attributes for Profile Class (Attribute Menu)	53
10.12.	Select attributes.....	53
10.13.	Define attribute’s cardinality (optional or mandatory)	54
10.14.	Define attribute’s default or fixed value	54
10.15.	Edit attribute type:.....	55
10.16.	Edit attribute stereotype	56

10.17.	Edit attribute constraint	57
11.	MODIFY AN ISBASEDON ELEMENT	59
12.	CREATE ASSOCIATIONS FOR PROFILE CLASSES: USE OF “EDIT CONNECTORS”	59
12.1.	Overview	59
12.2.	Edit connectors functions	60
12.3.	Select profile class associations	61
13.	REFINE PROFILE CLASS ASSOCIATIONS FOR WG13 OR GRAPH PROFILE STYLE	62
13.1.	Overview	62
13.2.	Copy Tagged Values or Notes	63
13.3.	Edit Profile Association Tagged Values or Notes	63
13.4.	Association Stereotype	63
13.5.	Restrict Association cardinality	64
13.6.	Qualify association end role name	64
13.7.	Managing end role name qualifier	65
13.8.	Select association direction	65
13.9.	Save refined association	66
13.10.	Edit a modified association	67
13.11.	Create associations	67
14.	REFINE PROFILE CLASS ASSOCIATIONS FOR HIERARCHICAL PROFILE STYLE	68
14.1.	Overview	68
14.2.	Copy Tagged Values or Notes:	68
14.3.	Edit Profile Association Tagged Values or Notes	69
14.4.	Association Stereotype	69
14.5.	Association Constraint	69

14.6.	Manage Qualifier	70
14.7.	Select association type	70
14.8.	Restrict Association cardinality	71
14.9.	Qualify association end role name	72
14.10.	By Ref association	73
14.11.	Save refined association	73
14.12.	Make multiple profile associations basedOn the same Information Model association	74
14.13.	Edit a modified association	75
14.14.	Create associations	75
14.15.	XOR constraint on Associations	76
15.	MODIFY PROFILE CLASS ASSOCIATIONS	78
16.	USE INHERITANCE IN PROFILE FOR ISBASEDON CLASSES	78
16.1.1.	Remember:	78
16.2.	CimConteXtor inheritance possibility	80
16.3.	Good practice	80
16.4.	Create an “ <i>IsBasedOn</i> ” class that inherits from another “ <i>IsBasedOn</i> ” class	80
17.	USE INHERITANCE IN PROFILE FOR ASSOCIATIONS	84
17.1.	Reminder	84
17.2.	Create “Associations” for an “ <i>IsBasedOn</i> ” class that inherits from another “ <i>IsBasedOn</i> ” class	85
17.3.	RolesAutoNaming	90
18.	ASSEMBLY MODEL	93
18.1.	Overview	93
18.2.	Attribute Order for Property Grouping	95
18.3.	Property Grouping	97
18.4.	Attribute order for syntax generation	98
19.	UTILITIES MENU	98

19.1.	Check profiling	98
19.2.	IntegrityCheck.....	100
19.3.	ESMPIntegrityCheck	103
19.4.	CGMESIntegrityCheck	103
19.5.	Profile building helps	104
19.6.	Use of "ReplaceSimpleFloatByFloat"	105
19.7.	Use of LocalizeDataTypes	106
19.8.	Use of "CreateGlobalProfile".....	109
19.9.	CreateGlobalProfile for a package	110
19.10.	CreateGlobalProfile for a diagram	110
19.11.	Use of "CopyAllNotes"	110
19.12.	Use of "RecoverProfileDatatypes"	111
19.13.	Use of "MakeMembersMandatory"	112
20.	COPY SUPERCLASS CONNECTIONS	113
21.	XSD SYNTACTIC GENERATION	115
21.1.	Root Classes	115
21.2.	Use of "Union" and inheritance.....	116
21.3.	Element Order (Attribute Order)	117
22.	CONFIGURATION FILE	117
22.1.	Overview	117
22.2.	AppSettings parameters	118
22.3.	DataProfiles parameters	119
22.4.	DataQualifier parameters.....	120
22.5.	DataConstraint parameters.....	121
22.6.	DataClassifierConstraint parameters	121
22.6.1.	Overview.....	121
22.6.2.	Classifier constraint as XML facets	122

22.7.	Classifier constraint as XML predefined datatypes	124
23.	MODEL VALIDATION	127
23.1.	Introduction	127
23.2.	Model loading	128
23.2.1.	Database Connection	128
23.3.	Configuration	129
23.3.1.	Configuration Dialog	129
23.3.2.	Import of Configuration	130
23.3.3.	Export of configuration	131
23.3.4.	Logging	131
23.4.	Validation report	132
23.5.	Validation rules	133
23.5.1.	Scope	133
23.5.2.	Code	133
23.5.3.	Rules for Classes	134
23.5.4.	Rules for packages	136
23.5.5.	Rules for attributes	136
23.5.6.	Rules for operations	138
23.5.7.	Rules for associations	138
23.5.8.	Rules for dependencies	139
23.5.9.	Rules for diagrams	139
23.5.10.	Excluded JCleanCIM rules	140
23.6.	Validation test package	140
23.7.	Known problems and how to solve them	141
23.7.1.	Model Inconsistencies	141
A.	ANNEX A: MODELLING METHODOLOGY SUMMARY	142
B.	ANNEX B: INFORMATION MODEL UML MODELLING	143
B.1.	UML Class Modelling	143
B.2.	UML element attached information	144
B.3.	UML stereotype examples	145
B.4.	Information Model Naming Rules	146
B.5.	Information Model modelling rules	146
B.6.	Datatype Types and UML modelling rules	147
	Domain Datatypes	148

B.7.	Primitive value space restrictions.....	149
B.8.	Enumeration value space.....	150
B.9.	CIMDatatype “IsBasedOn” process.....	151
B.10.	Compound	152
C.	ANNEX C: PROFILE UML MODELLING RULES	152
C.1.	Profiling rules:	152
C.2.	“IsBasedOn” for Profile Classes.....	152
C.3.	“IsBasedOn” for Profile class attributes	154
C.4.	“IsBasedOn” for Profile Class associations.....	154
C.5.	Profile naming rules	155
D.	ANNEX D: PROFILE RULES FOR INHERITANCE	156
D.1.	Inheritance principles.....	156
D.2.	Profile level class inheritance rules.....	157
D.3.	Profile level “IsBasedOn” class inheritance rules	157
D.4.	Profile level attributes inheritance rules	160
D.5.	Profile level association inheritance rules	162
D.6.	Use of Association CHOICE Constraint with inheritance rules	164
E.	ANNEX E: INFORMATION MODEL EXTENSION	167
E.1.	General	167
E.2.	Custom Information Model extensions	167
E.2.1.	General.....	167
E.2.2.	Adding an extension class.....	168
E.2.3.	Adding an extension attribute	168
E.2.4.	Adding an extension enumerated literal.....	169
E.2.5.	Adding an extension association between extension and IM classes.....	169
E.2.6.	Adding an extension association between two IM classes.....	170
F.	ANNEX F: CREATEGLOBALPROFILE DESCRIPTION	171
F.1.	Starting CreateGlobalProfile.....	171
F.2.	CreateGlobalProfile for a package	172

- F.2.1. Package CreateGlobal Profile rules172
 - F.2.2. Example of rules for a package CreateGlobalProfile.....173
- F.3. CreateGlobalProfile for a diagram.....174**
 - F.3.1. Diagram CreateGlobalProfile rules174
 - F.3.2. Example of results175
- G. ANNEX G: ADAPTING PROFILE TO A NEW CIM VERSION177**
- H. ANNEX H: ENHANCED DRAG AND DROP FOR ISBASEDON CLASSES178**
- I. ANNEX I: INTEGRITYCHECK MESSAGES179**
 - I.1. CGMES Integrity Check List.....179
 - I.2. Integrity Check Message list181
 - I.3. ESMP Integrity Check Message List.....183

Executive Summary

CimConteXtor is an Enterprise Architect (EA) Add-In. The main purpose of it is to create a profile in EA with a profile being a subset of an existing Canonical Common Information Model (CIM). This sub-setting can be done only according to some rules. CimConteXtor has implemented some rules based on those proposed by UN/CEFACT¹ Core Component Technical Specification and some of those adopted by IEC² TC 57 (like IEC 62325-450) for the electricity market.

Although originally, CimConteXtor has been developed to build CIM based profiles, it can be used with any Information Model as long as it is available in EA.

“Enterprise Architect” is a tool that is developed and distributed by Sparx Systems.

“UCAIug CIM Modeling Guide”, available for public use by UCA International Users Group, <https://cim-mg.ucaiug.io/latest/>, was created by CIM Model Managers.

CimConteXtor 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 CimConteXtor 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 : United Nation - Center for Trade Facilitation and e-Business

² [IEC](#): International Electro-technical Commission – Standard body for electro-technical domain

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 CimConteXtor

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. Installation and overview

1.1. Platform Architecture

The CimConteXtor 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.

1.2. Downloading “CimConteXtor”

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

- *CimConteXtorSetup_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.
- *CimConteXtor_Manual_v11-6-0.pdf*: A manual with the necessary documentation. This is the current document the user is reading.
- *cimcontextor_SourceCode-zip*: A package with the source code.

1.3. Before installing “CimConteXtor”

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.

1.4. Installation

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

1.5.Overview

“CimConteXtor” is an “Enterprise Architect” Add-In. “Enterprise Architect” (EA) is primarily a UML modelling tool.

The main purpose of CimConteXtor is to create a profile in EA. A profile that is a subset of an existing CIM model. This sub-setting is done according to a methodology (“IsBasedOn”) based on UN/CEFACT Core Component Technical Specification and adapted by IEC/TC57 for its own purposes (see annexes of this document to see the methodology rules).

The models are stored in “EA” format and can be exported by the EA export feature as an “XMI” file.

To use CimConteXtor, one should be familiar with:

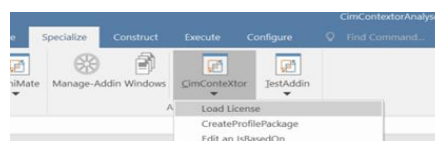
- UML class diagram modelling,
- Enterprise Architect application,

and with rules for creating a CIM profile based on an Information Model in UML form (the best is to be aware of UN/CEFACT Core Component Technical Specification CCTS and IEC/TC57 62325-450) and look at the annexes of this document.

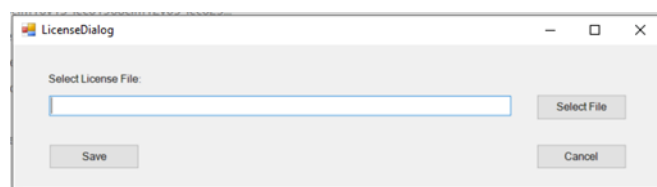
1.6.License

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

After installing CimConteXtor, you can 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 CimConteXtor resources directory (see below). 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. Uploading a valid license file is required in order to continue to work. For this, please use the license included in the installation package.

A warning mechanism starts informs the user 30 days before the expiry date of the license how many days are still left till a renewal of the license is necessary.

1.7.Resource's directory

The CimConteXtor Add-in uses the resources directory of which the directory path is set up by the following pattern

```
C:\Users\<Windows User>\AppData\Roaming\ENTSO-E\CimConteXtor\Resources
```

whenever there are data to store internally, and a user is not asked for a location. From version 3.2.4 of CimConteXtor the usage of this resource's directory has been minimised. The placeholder <Windows User> stands for the logged in Windows user.

1.8.Configuration

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

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

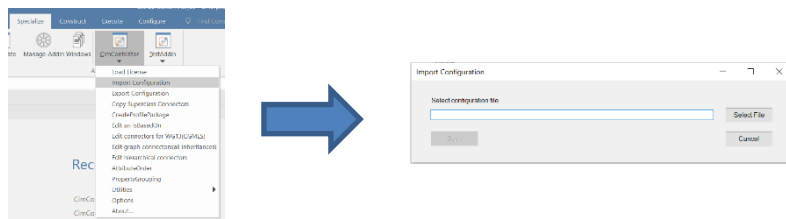
The following actions described in the subchapters below are possible:

1.8.1. Installing and using the CimConteXtor Add-in without an external configuration file

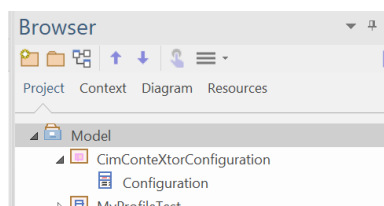
The Add-in is installed and can be used right away. The default configuration is internal and cannot be changed by a user.

1.8.2. 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, CimConteXtorConfiguration:



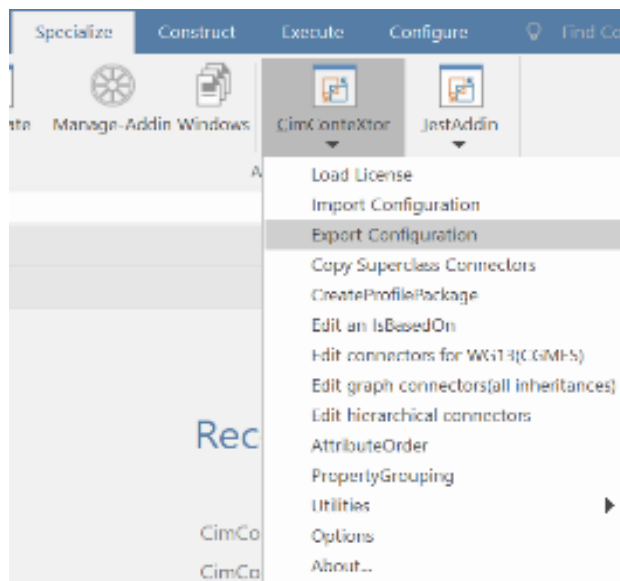
When loading the XML configuration, a quick XML validation routine checks the configuration file. If there is an error in the XML a message is displayed telling the possible reason (e.g. an XML element name is misspelled and which element).

1.8.3. 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 above) or an update of the configuration, if there exists already a custom configuration.

1.8.4. 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 you to check the default configuration.

Configuration parameters

For details about the configuration parameters, see the chapter [Configuration](#) below.

2. Before using CimConteXtor

2.1. Get some ideas about UML modelling and profiling methodology

To understand how to use CimConteXtor, read first Annexes [A](#), [B](#), [C](#) and [D](#) of this document. These annexes give you some principles on UML modelling and how it is used when doing profile according to a methodology based on UN/CEFACT Core Component Technical Specification.

2.2. Be aware of naming rules

Component naming is something very important when doing modelling in general and UML modelling in particular. CCTS gives a lot of naming rules based on ISO/IEC 11179 standard. IEC TC 57 has borrowed some of these rules for its UML information model element naming. So before using CimConteXtor (see section 23.4 naming rules), be aware:

- of the use of naming rules for Class, Attribute, Association end role, Primitive, Enumeration, Datatype and Compound names,
- that, at Information Model level, class names MUST be unique,

- that, at Profile level, class names MUST be unique (but they could have the same name as one at Information Model level). If various classes are “IsBasedOn” the same Information Model Classes, use an appropriate qualifier to distinguish them,
- that Primitive, Datatype, enumeration and Compound names SHOULD be unique in all the EA project name space if they are used across all the model levels (Information Model and Profiles). To have uniqueness, use of qualifiers is required when appropriate to distinguish them (See section “Define datatype template and dependencies”).
- that association end role names between two given classes MUST be unique (use qualifiers as appropriate).

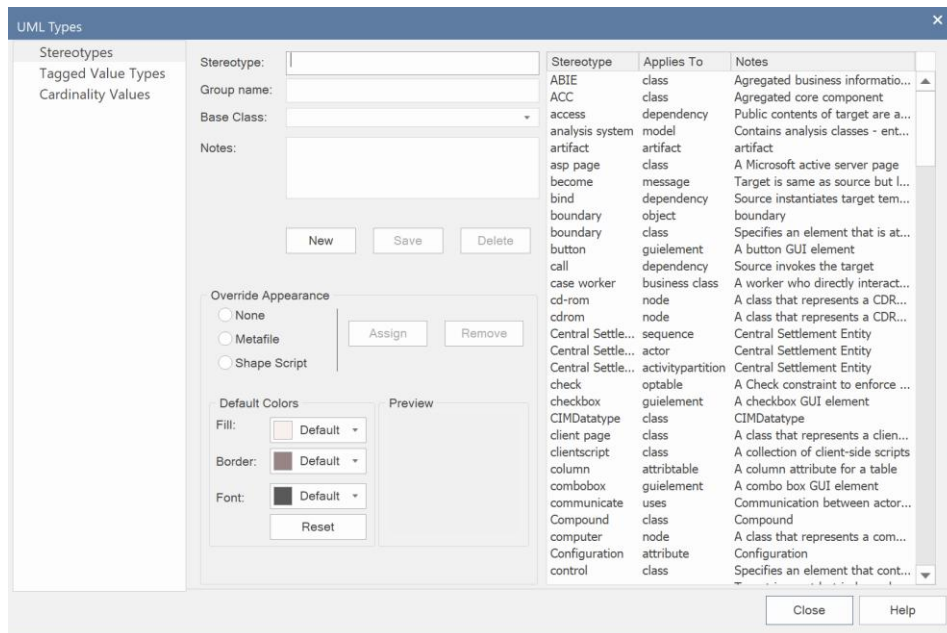
2.3. Define your Project Environment specific elements

Define all your specific UML project environment elements in Enterprise Architect before using CimConteXtor:

- Stereotype for each component category: class, attribute, association,
 - For example, the stereotype *nc* used to identify the ENTSO-E Network Code Profiles CIM extensions.
- Stereotype for data typing (especially if you are starting your own information model): Datatype, Primitive, Enumeration, Compound,
- Tagged Values names,
- Cardinality values.

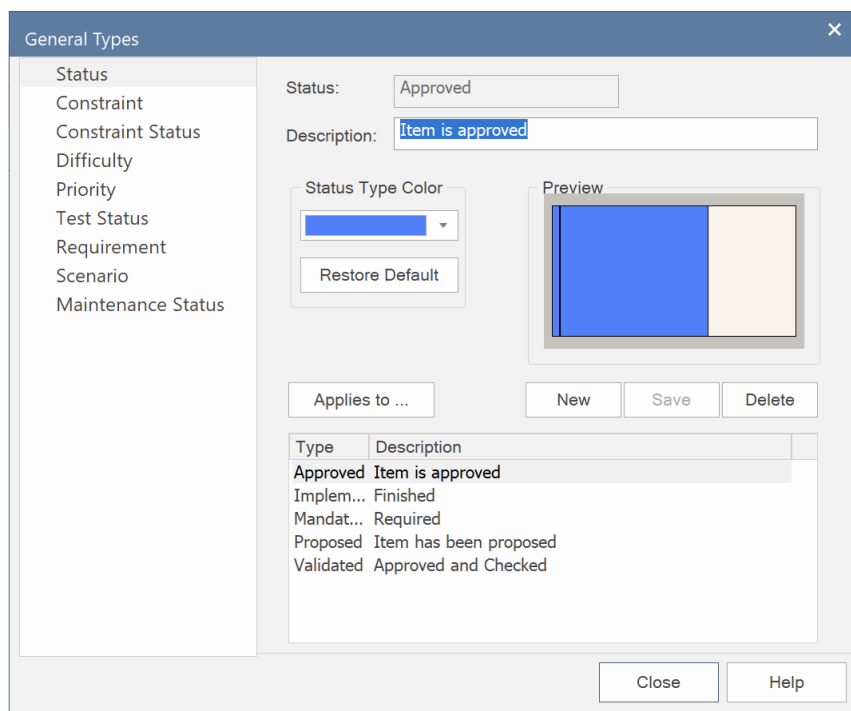
Use for that the usual EA interface:

Settings --> UML Types



Define constraint types if needed. Use for that the usual EA interface:

Settings --> General Types



2.4. Define package template

Define your package template, this is very important for syntactic model generation (see CimSyntaxGen user manual), especially for assembly and syntactic level generation:

- Put Information Model and Profiles in different packages,
- Define “*IsBasedOn*” package dependencies (see “*Packages Template and dependencies*” section below).

2.5. Define datatypes package template and dependencies

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

1° If you want profile standalone packages, 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 “*DomainProfile*” package. This implies too that all names are unique on the project name space (use of qualifier is required when doing an “*IsBasedOn*”).

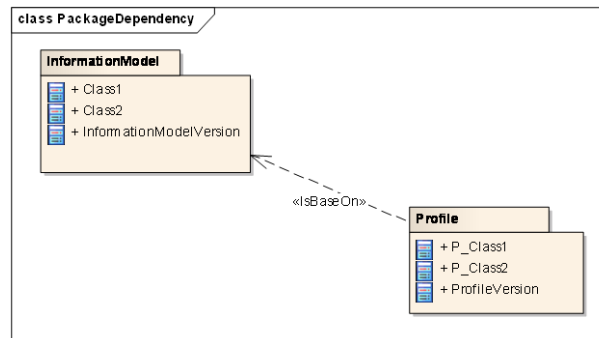
2.6. Define “*IsBasedOn*” Package Dependencies

“*IsBasedOn*” Package dependencies specify a hierarchy and must be defined at the time you work out Package Template definition (see use of “*CreateProfile*” Menu).

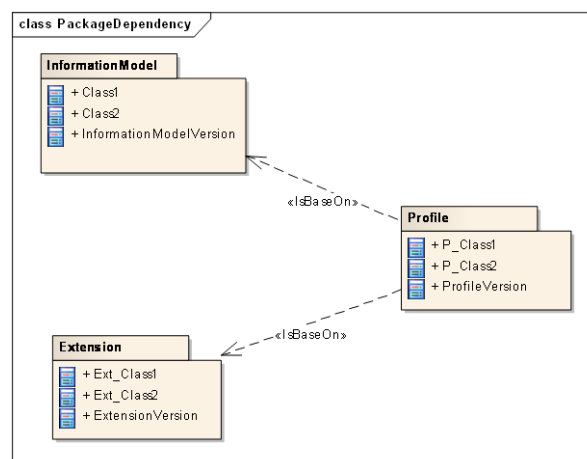
The “*IsBasedOn*” hierarchy could be of different kinds:

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.



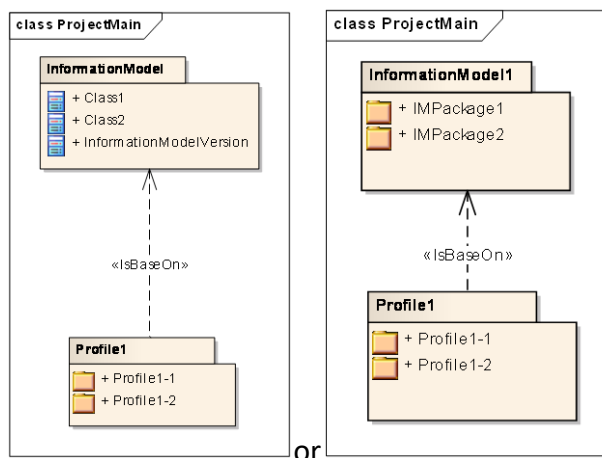
A profile could be based on several packages (example Information Model and Extensions):



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

Profiles rules:

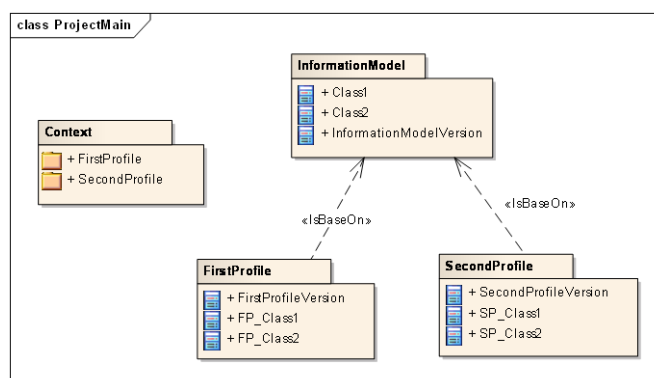
- All profile classes are in sub-packages,
- all profile classes names are unique in the context of the profile package (a profile sub-package class could not have the same name as another class of the sub-package, but also as another class in another different profile sub-package,
- associations between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



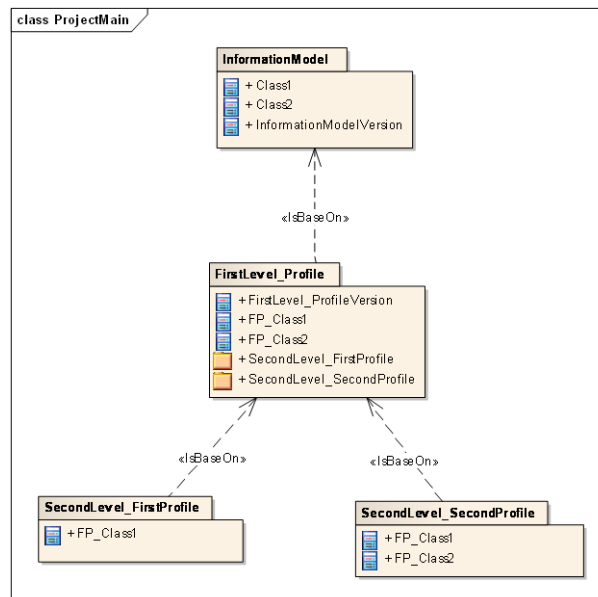
4° Sub-packages classes are based on the package classes

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 organised in sub-packages.

Sub-Profile packages are contained by the profile package.

This case is like the following one.



2.7. Do not forget

Do not forget:

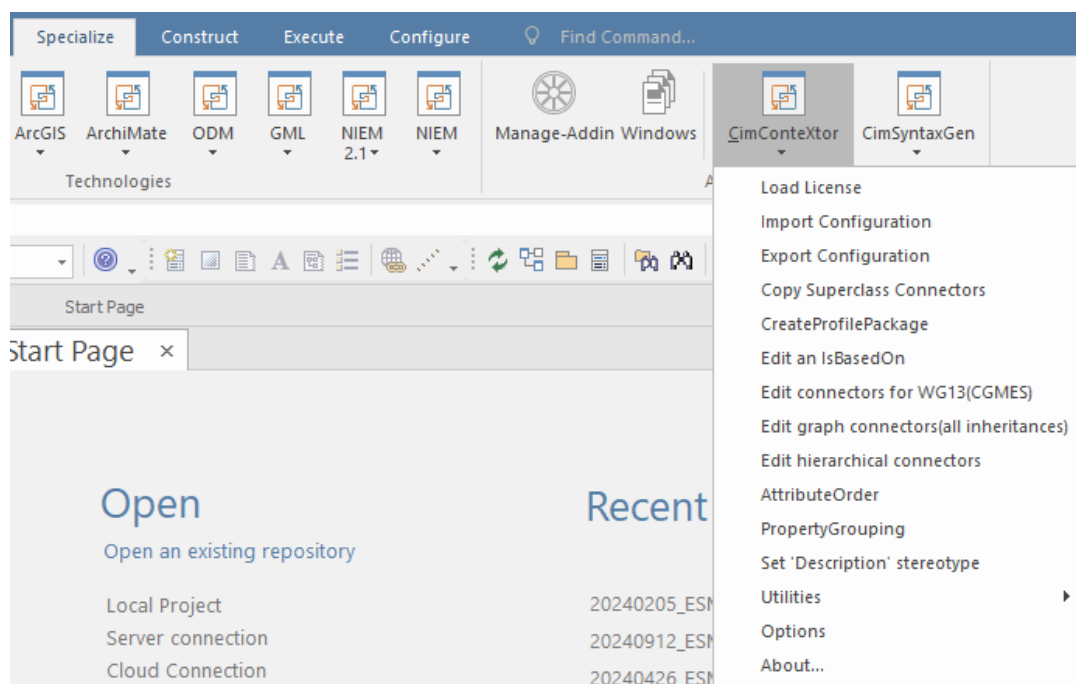
- That the information model must exist before using *CimConteXtor*, and that this information Model must have been built according to some UML modelling rules (see annexes A, B, C and D),
- What will be the target for the syntactic model generation: XSD, RDFS, others...
- What are the package dependencies,
- Where you define your datatypes,
- That, at profile level, super classes, if no instances are expected, MUST be “*Abstract*”,
- That creating a profile is a process with several steps:
 1. Define project environment: Stereotypes, Tagged Values, Constraint types,
 2. Define package Template (Information Model level, Profile, Datatypes packages),
 3. Specify “*IsBasedOn*” dependency between Information packages and Profile packages (use the “*CreateProfile*” or “*CreateGlobalProfile*” function),
 4. Create profile (or “*IsBasedOn*”) datatypes,
 5. Create profile (or “*IsBasedOn*”) classes,

6. Use profile to generate assembly model if required (Example: IEC 62325-450),
7. Use profile or assembly model to generate syntax models (see “*CimSyntaxGen User Guide*”).

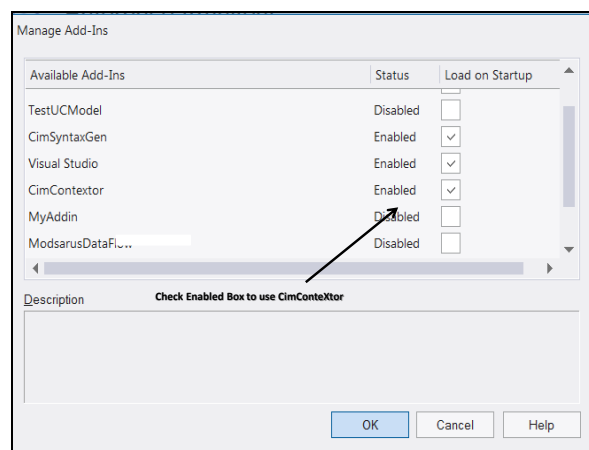
3. CimConteXtor Presentation

3.1.General

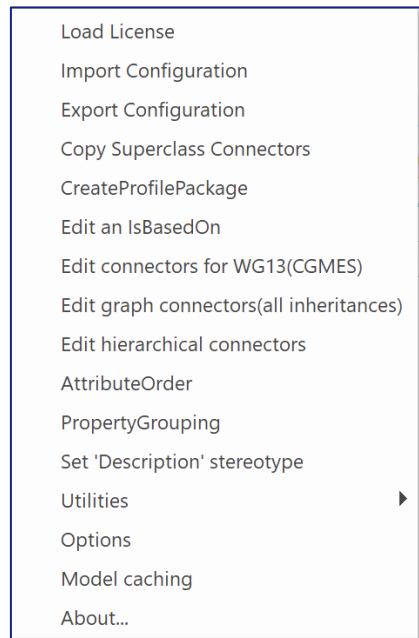
CimConteXtor is managed as an EA Add-In. It is referenced in the EA tool bar:



The CimConteXtor 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 *CimConteXtor*’s checkbox:

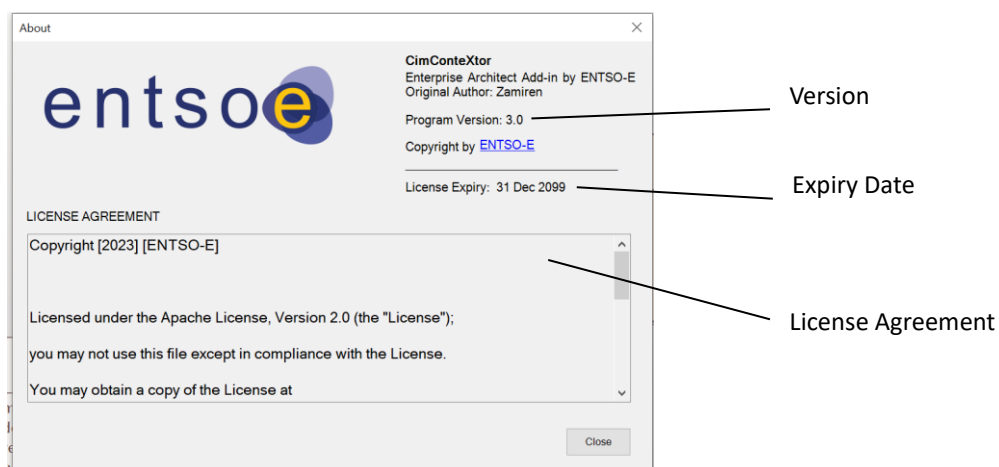


CimConteXtor has a base functionality of making an “*IsBasedOn*” Class by "dragging and dropping" an Information Model Class. But it has some other functionalities that are given by the *CimConteXtor* Menu items:



3.2.“About” Menu

The “About” menu gives information on “CimConteXtor” version and validity:

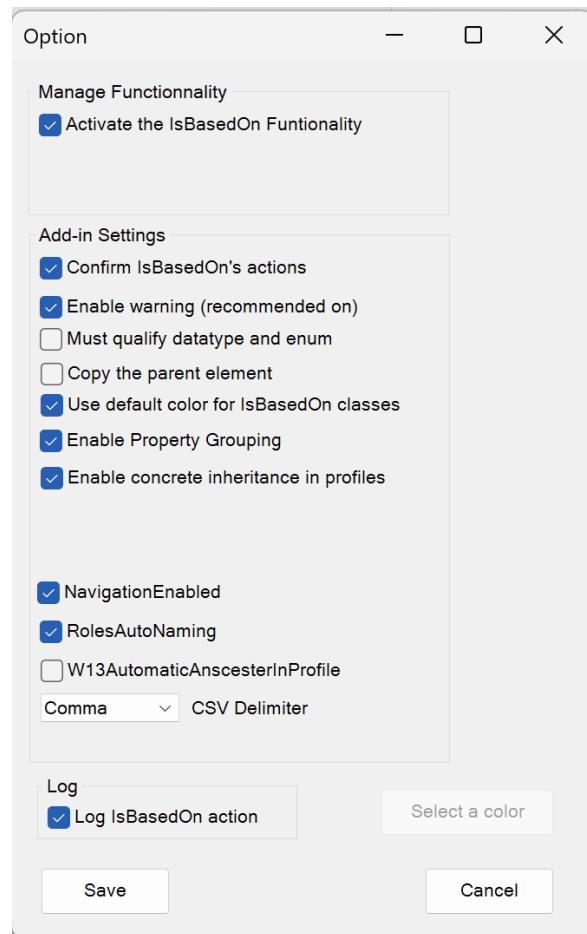


3.3.“Options” Menu

The “Options” menu lets you:

- Enable or disable *CimConteXtor* Add-In functionality.
- Confirm *IsBasedOn* Actions. When unchecked, each drag and drop action will only enable "*IsBasedOn*" function, no class copy is allowed in a diagram. Unchecking is interesting when starting a profile.
- Enable or Disable warnings.
- Force the use of qualifiers for primitives, datatypes, enumerations and compounds when an "*IsBasedOn*" is made on one of these elements. When checked this option will force the use of a qualifier: this is useful when you want reusability of all datatypes across profiles. Uncheck this option when you target a profile standalone package that includes the primitives, datatypes, enumerations and compounds it is using. But in this case be sure that attribute typing does not use primitives, datatypes, enumerations or compounds that belong to another package.
- Copy the parent element. This option fix what is the default for copying parent element when doing an "*IsBasedOn*" action. If unchecked, by default the parent class will not be copied in the current diagram.
- Fix the class box background colour.
- Enable property Grouping. When checked these options allows "PropertyGrouping" in Assembly Model.
- Enable concrete inheritance in Profiles. When checked, this option allows superclass in a profile hierarchy to be concrete.
- Navigation Enabled. This option let connectors in a hierarchy to be drawn as an arrow instead of an aggregation.
- RolesAutoNaming. This option allows change of end role names when an association of a super class in the information model is used as a native association for a profile class that *IsBasedOn* a subclass of the superclass.
- WG13AutomaticAncestorInProfile. This option allows drag and dropping of a class with all its ancestors in the diagram (option for WG13 and CGMES profiling style).
- Trace all actions in a log.

Note: When working with ESMP the "NavigationEnabled" should not be checked in the figure below.



- The *"CreateProfilePackage"* is used to create a Profile Package and associated diagram with the right dependencies.
- The *"Edit IsBasedOn"* is used when you need to modify an *"IsBasedOn"* class that has been created with the *"Drag and Drop"* feature of CimConteXtor (See *"Edit IsBasedOn"* sections).
- The *"Edit Connectors"* is used whenever you want to specify or modify the associations of an *"IsBasedOn"* class (See section 12 *"Edit Connectors"* function).
- There are three different kinds of Edit Connectors:
 1. Edit Connectors for WG13. This menu optimizes the presentation of connections between classes for WG13 profile design (The profile keeps the inheritance path of the information model, and at a class level there could be only the native associations).
 2. Edit Graph Connectors. This menu is used in the design of complex graph profiles (any kind of inheritance could use at profile level).
 3. Edit Hierarchy Connectors. This menu is used for the design of hierarchical profiles (like WG14 or WG16 ones).

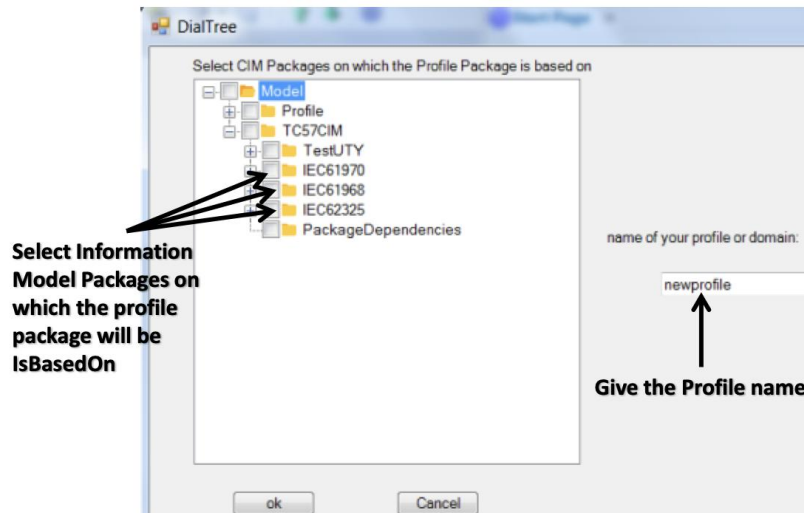
- The “*AttributeOrder*” is used at two levels: at Profile level to define the order of properties used for property grouping and at Assembly level to define the order of elements output in an XML schema (See section 18.2 “*AttributeOrder*” sections).
- The “*PropertyGrouping*” is used when you need to generate an assembly model with grouping of properties (See section 18.3 “*PropertyGrouping*” sections).
- The “*Utilities*” are used:
 - To change some CGMES old profile design, especially in the use of datatypes,
 - To create a profile for an all-Information Model package or diagram,
 - To recopy all information model notes in a profile,
 - To make all attribute types of a profile to be the ones of a domain profile,
 - To make all attributes of profile classes mandatory,
 - To check if a profile is conforming with the profiling rules. Currently, depending on the kind of profiles (IEC, CGMES and European Market style, there are different checking (See “*IntegrityCheck*” section 19.2).

4. Create “ProfilePackage” with CimConteXtor

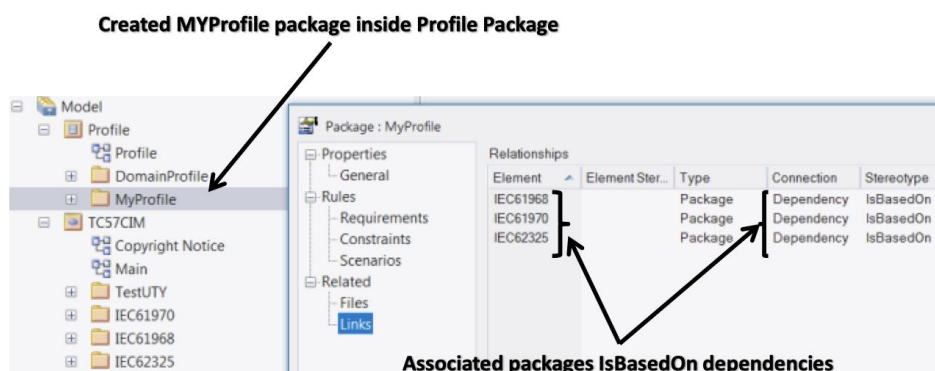
The “*CreateProfilePackage*” Menu is used to create a Profile package with its “*IsBasedOn*” package dependencies (usually Information Model packages).

To create a package profile inside another package, select the containing package and launch the CimConteXtor “*CreateProfilePackage*” Menu. This will open a window:

CreatePackageProfile



Enter the name of the profile and select the packages of the information model that will be used as the basis for the profile. Saying "OK" will create a profile package "based on" the selected packages:



5. Create "IsBasedOn Elements" with CimConteXtor

5.1. IsBasedOn Drag and Drop function

The basic function of *CimConteXtor* is to create "*IsBasedOn*" elements. This is done by dragging and dropping an element (primitive, enumeration, datatype, compound and class) in a profile diagram. There are two ways of using this function:

- either directly, EA drag and drop function are bypassed and copying a class in another diagram is not possible, the "*IsBasedOn*" function is always active. This behaviour is selected when the "*Confirm IsBasedOn's action*" box is checked in the *CimConteXtor* "Options" Menu.

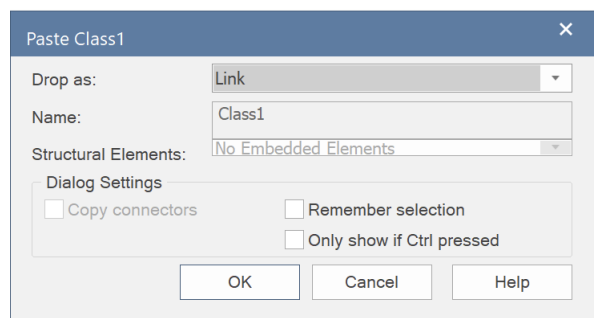
- Or indirectly, *CimConteXtor* asks if you want to bypass EA drag and drop function or if you want to use it. This will let you copy a class in a diagram or do an “*IsBasedOn*”. This behaviour is selected when the “*Confirm IsBasedOn’s action*” box is unchecked in the *CimConteXtor* Options Menu.

When you drag and drop an Information Model element,

- if the direct mode is selected (“*Confirm IsBasedOn’s action*” box is checked), the “*IsBasedOn*” window will appear directly (step 3),
- if the indirect mode is selected (“*Confirm IsBasedOn’s action*” box is unchecked), three windows will be displayed successively (step 1, 2 and 3):

1° a first prompt window is displayed:

This is the usual EA prompt window to ask if you want to show the existing element in a new diagram or make an instance or make a child element. If you want to create a profile element based on the information class you drag and drop, click on “OK”



2° a second prompt window appears:

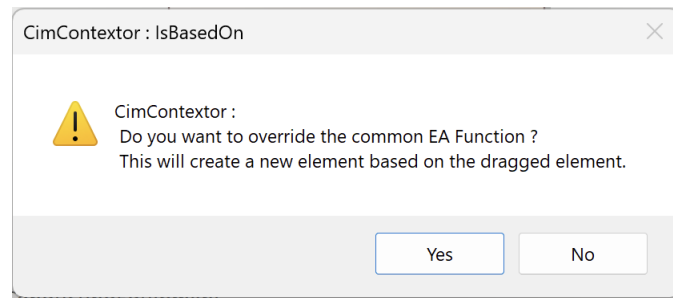
Note: if the window does not appear, check if *CimConteXtor* Add-In has been enabled (see Manage Add-Ins function In EA) and that the “Activate the *IsBasedOn* functionality” box is checked in the *CimConteXtor* “Options” window.

This pop-up window asks you if you want to make:

- a drag and drop of the element in the diagram (same as EA function),
- or to make an “*IsBasedOn*” element based on the one you are dragging and dropping.

If you want:

- to drag and drop the element, select “NO”,
- to make an “*IsBasedOn*” element, select “OK”.

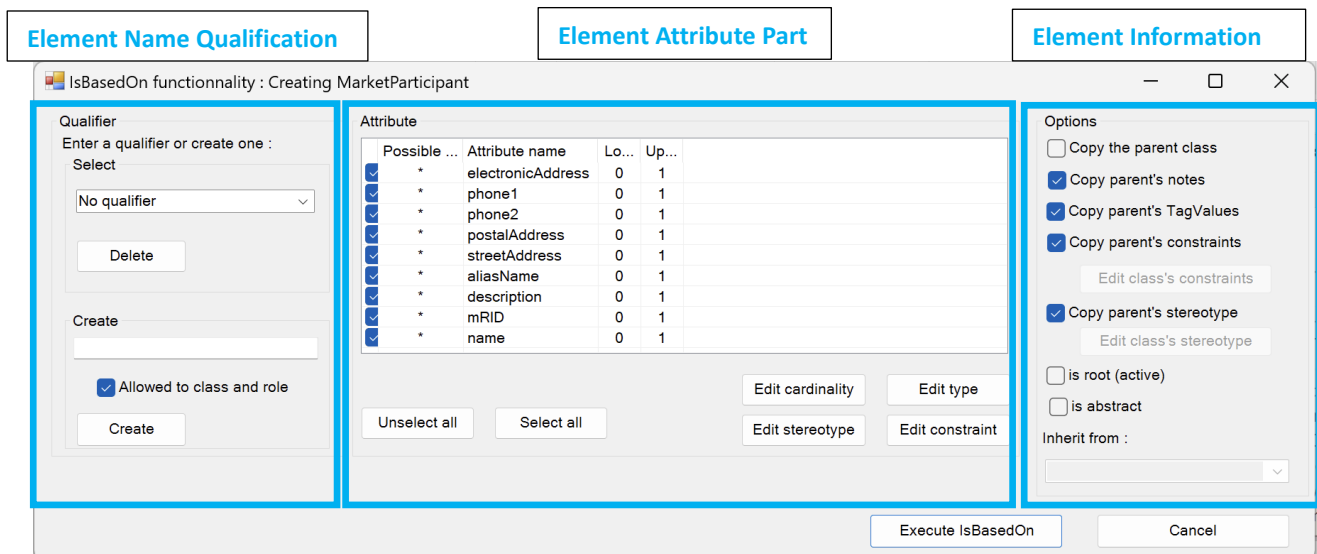


3° the “IsBasedOn” window appears:

But depending on the element (primitive, enumeration, datatype, compound, class) you have dragged and dropped, the “IsBasedOn” window will appear and have some differences according to the element that it is “IsBasedOn”.

This window lets you qualify the “IsBasedOn” (or profile) element and refine its information and attributes. Basically, this window has three parts:

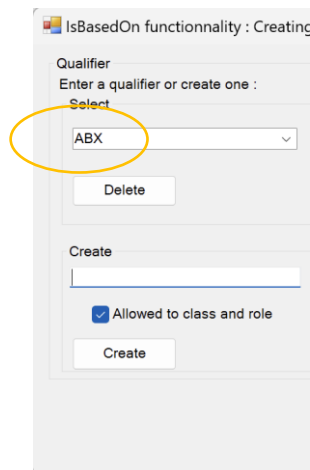
- one for giving a qualifier, on the left side,
- one for profile element information, on the right side,
- one for attribute selection and restriction, in the middle.



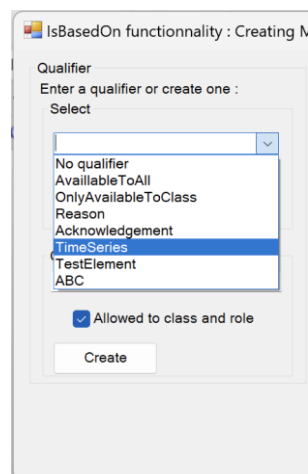
5.2. Qualify an “IsBasedOn” element Name

Two ways:

- either you want to give a qualifier straightaway: write the name in the qualifier field directly,



- or you want to use an already defined qualifier that you could find and or insert into a list. Select the name in the drop-down menu.



Note: you could manage your list by entering new qualifiers in the create field and click create. You could also say that this qualifier will be entered in the qualifier class and role list, if the “Allowed to class and role” box is checked, otherwise it will belong only to the list of the element you are working on.

Note: you could specify a predefined list of qualifiers by using the "Configuration.xml" file of CimConteXtor (see configuration file section). The “Configuration.xml” can be exported, edited and

imported (see 1.8). The configuration editor implemented in CimSyntaxGen will be also included in CimConteXtor in a future release.

Now depending on the element that is the parent element of the *IsBasedOn* one, the window parts for information and attributes will have some differences.

5.3. “*IsBasedOn*” relation in EA

The “*IsBasedOn*” relation between two elements (Classes, Primitives, Datatypes, Compounds) could always be checked in EA:

- Open the “*Properties*” windows of the element
- Go to the “*link*” tab
 - If there is an “*IsBasedOn*” relation, it will be marked with a “dependency” connection and a “*IsBasedOn*” stereotype.
- Go to the “*General*” tab, and select “*Tags*”
 - If there is an “*IsBasedOn*” relation, there will be a “GUIDBasedOn” TaggedValue that gives the value of the GUID of the information model element.

6. Create “*IsBasedOn*” Primitives with CimConteXtor

6.1. Overview

There are two cases for using the “*IsBasedOn*” feature on a Primitive:

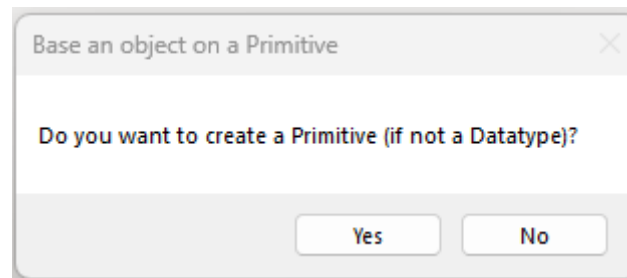
- Create a Profile Primitive,
- or create a CIMDatatype whose value space is a restriction of the primitive value space.

Example of creation:

- a String profile primitive *IsBasedOn* the String primitive of the information model.
- a String value space restricted to 35 characters.

6.2. Drag and drop a Primitive, create a Primitive

So, when the drag and drop element is a <<Primitive>>, a window open to ask if you want to create a profile Primitive (Yes) or if you want to create a Datatype (No) that is based on the Primitive but defines a restricted value space.



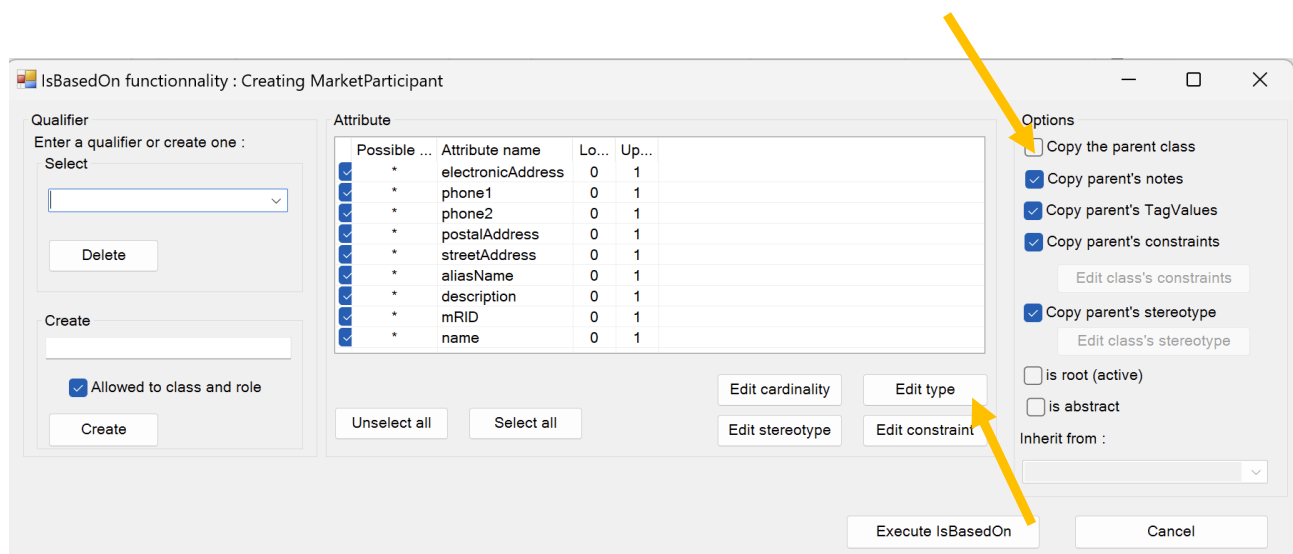
If you choose "Yes", a profile Primitive will be created.

6.3. Drag and Drop a Primitive, create a CIMDatatype

If you choose "No", it means that you want to define a value space that is a restriction of the value space of the Primitive. In this case a <<CIMDatatype>> with an attribute "value" is created as the "IsBasedOn" element:

- The type of the attribute "value" is identical to the <<Primitive>> it is based on: if the <<Primitive>> is String, the type will be String and so on,
- The value space of the attribute "value" is a subset of the primitive value space. This subset is specified by defining restrictions on the primitive value space. These restrictions are standardized as "facets", and these facets depend on the primitive.

The "IsBasedOn" window for a CIMDatatype basedOn a Primitive will be shown like this:



6.4. Qualify the IsBasedOn CIMDatatype

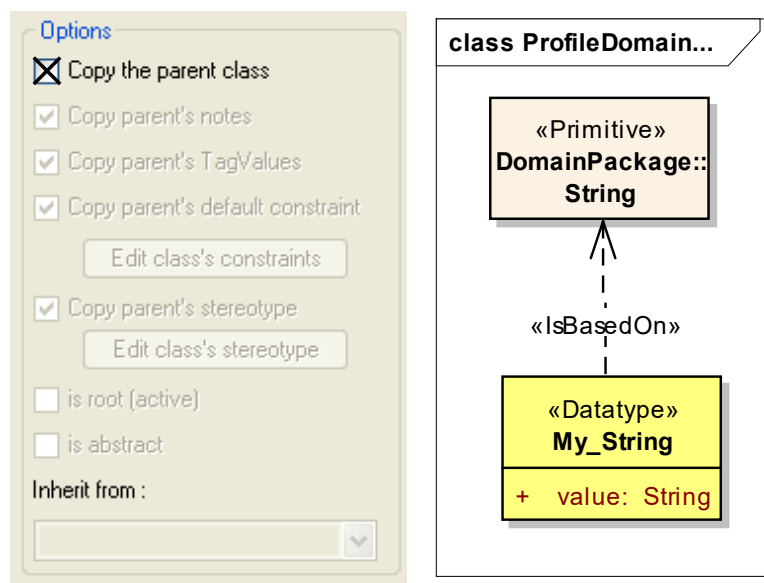
See section 5.2 for use and management of qualifiers.

- If the “*Must qualify datatype and enum*” box of the “*Options*” Menu has been checked, the “*IsBasedOn*” Datatype name must have the Primitive name prefixed by a Qualifier. *CimConteXtor* will force you to enter a qualifier for this Datatype.
- If the box is unchecked, the datatype could have the same name as the primitive, but this is not a good practice at all.

Remember that if you want to be able to reuse any primitive, datatype, enumeration or compound in any profile, their names must be unique in the Information Model and Profiles name space.

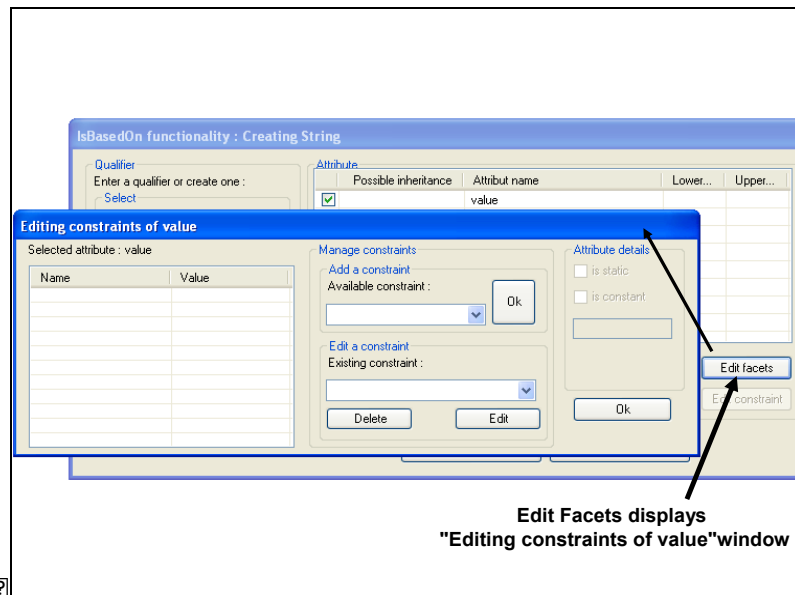
6.5. Copy Parent Element

Selecting “*Copy parent’s class*”, allows the inclusion of the Information Model Primitive along with the corresponding “*IsBasedOn*” datatype, and shows the “*IsBasedOn*” dependency link between the two elements:

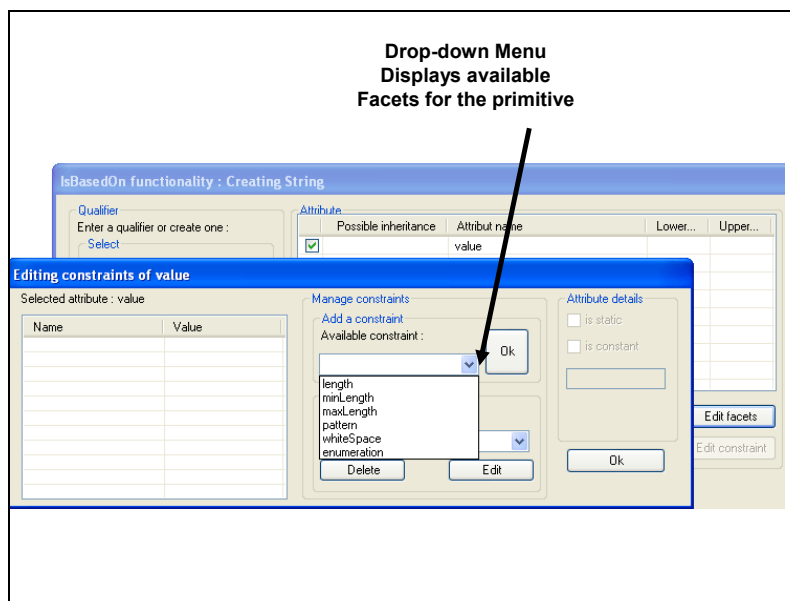


6.6. Define value space restrictions

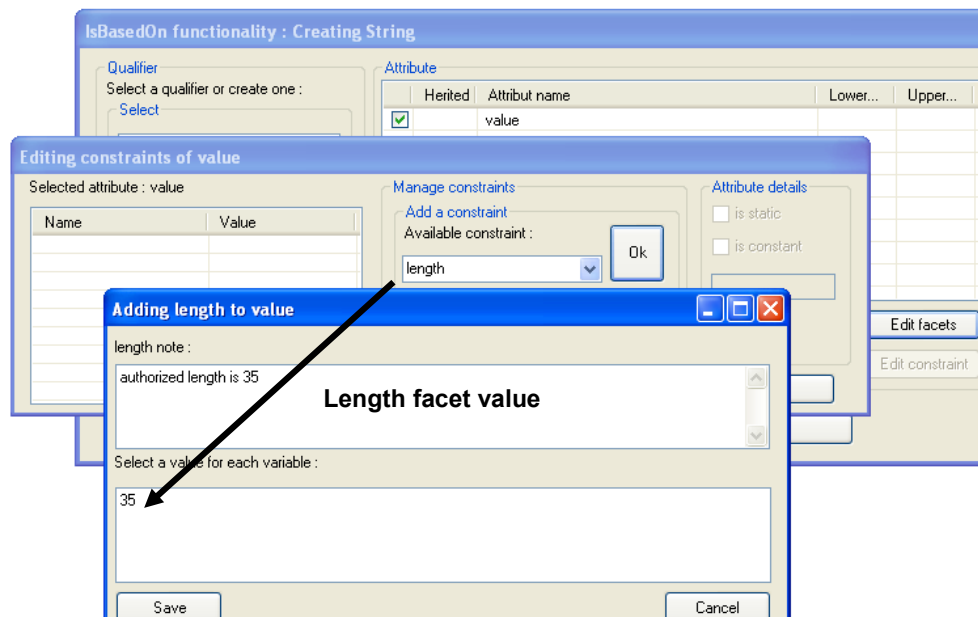
Selecting “*Edit constraints*” lets you define all restrictions (or facets or constraints) on the value space of the primitive. A new window appears to express constraints:



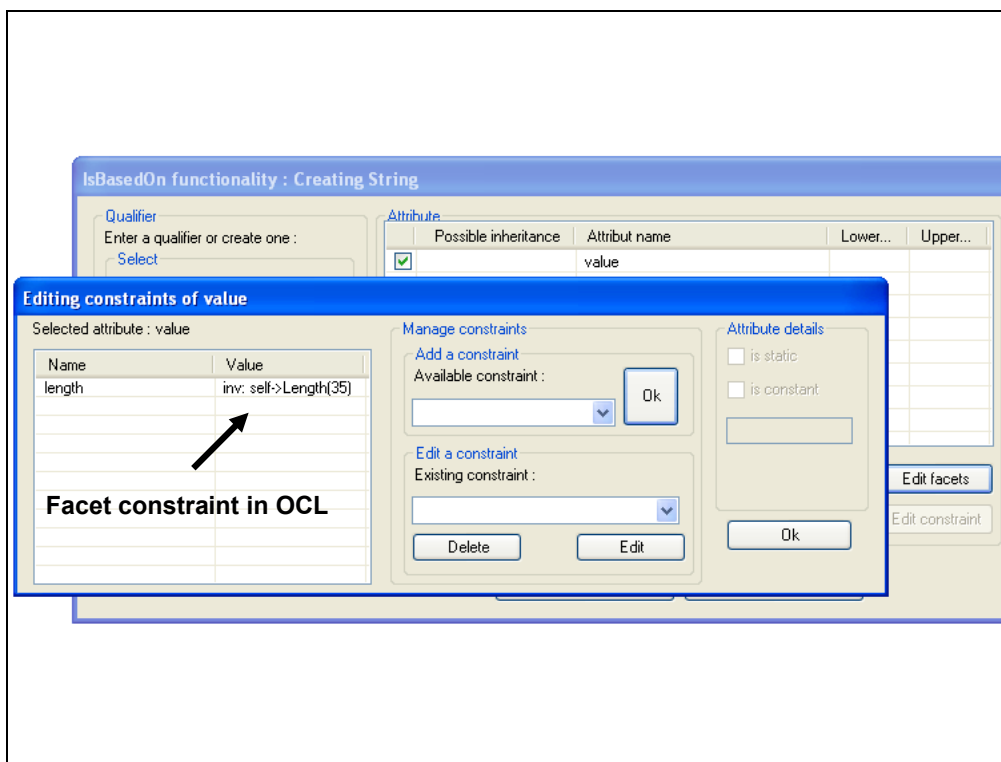
To add a facet, click on the arrow of the drop-down menu of “Available constraint”:



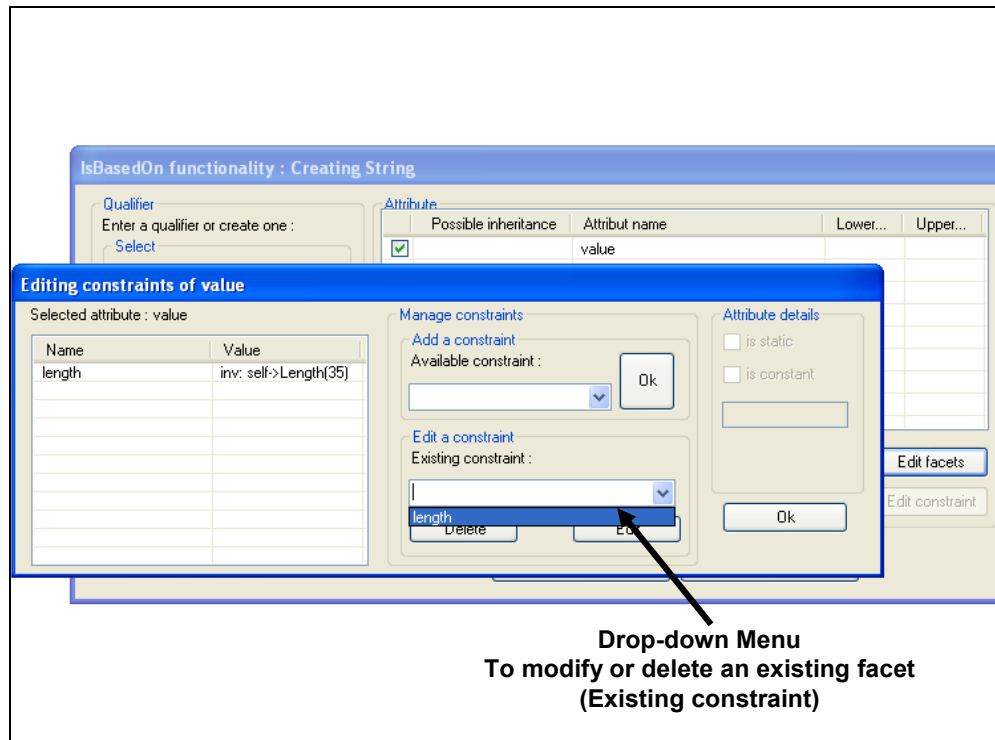
This will show all the available facets (or constraints) for the primitive (here a String). Select a facet and click on “OK” button, this will display the facet window:



Enter the value for the given facets and optionally enter facet notes. Then “Save”. The Constraint window will appear, and will show the facet with its value (note that the facet value is entered as an OCL constraint, but this is done automatically by *CimConteXtor*):



The existing facet is now also part of the drop-down menu of the existing constraint: this will enable you to modify or delete this existing constraint:



7. Create “IsBasedOn Enumeration” with CimConteXtor

7.1. Reminder

- All <<enumeration>> names follow the UML naming rule that specifies that the enumeration name is ended by the term “Kind”.

NOTE: There are several standard CIM classes that do not comply with this naming convention. This is an instance where backwards compatibility supersedes a modelling rule.

- All <<enumerations>> are “IsBasedOn” either a <<Primitive>>, or a <<CIMDatatype>> “IsBasedOn” a <<Primitive>> or another <<enumeration>>. This makes a hierarchy of elements.

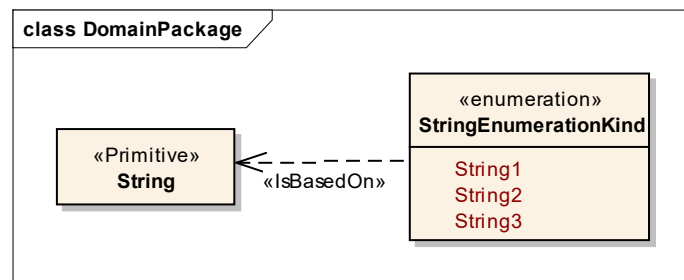
7.2. Create an <<enumeration>>

This is done with usual EA tools, by selecting the enumeration box in the EA “Class Menu” of the “Tool Box” and by dragging and dropping it in a diagram.

NOTE: Avoid using a self-defined stereotype with the name “enumeration” or “Enumeration”. The Enterprise Architect provides in the class diagram a pre—defined data type “enumeration”. A self-defined stereotype with this name might cause unpredictable side effects.

Depending on the kind of <<enumeration> you are doing, you must make a dependency link stereotyped with “IsBasedOn” between this <<enumeration>> and the <<Primitive>> (or <<CIMDatatype>>) it is based on.

This is done with usual EA functions not with CimConteXtor



7.3.Create an “IsBasedOn” Enumeration with CimConteXtor

This function is only to create an “IsBasedOn” <<enumeration>> on an existing <<enumeration>>.

When the drag and drop element is an <<enumeration>>, it means that you could restrict the list of “enumeratedLiterals” of the <<enumeration>>. The “IsBasedOn” window for an enumeration will appear like this:

The screenshot shows the 'IsBasedOn functionality: Creating PhaseCode' dialog box. It has several sections:

- Qualifier:** A dropdown menu set to 'No qualifier' and a 'Delete' button.
- Create:** A text input field and a checkbox labeled 'Allowed to class and role' which is checked, with a 'Create' button below.
- Attribute:** A table with columns 'Possible ...', 'Attribute name', 'Lo...', and 'Up...'. It contains a list of attribute names: ABCN, ABC, ABN, ACN, BCN, AB, AC, BC, and AN. Each row has a blue selection icon in the 'Possible ...' column and '1' in the 'Lo...' and 'Up...' columns. Below the table are 'Unselect all' and 'Select all' buttons.
- Options:** A section with several checkboxes: 'Copy the parent class' (unchecked), 'Copy parent's notes' (checked), 'Copy parent's TagValues' (checked), 'Copy parent's constraints' (checked), 'Copy parent's stereotype' (checked), 'is root (active)' (unchecked), and 'is abstract' (unchecked). There are also buttons for 'Edit class's constraints', 'Edit class's stereotype', and 'Inherit from :'. At the bottom are 'Execute IsBasedOn' and 'Cancel' buttons.

- Qualify the IsBasedOn enumeration (see section 5.2 for use and management of qualifiers):
 - If the “Must qualify datatype and enum” box of the “Options” Menu has been checked, the “IsBasedOn” enumeration name will have the name of the enumeration

it is based on prefixed by a Qualifier. *CimConteXtor* will force you to enter a qualifier for this enumeration.

- If the box is unchecked, the enumeration could have the same name as the enumeration it is “IsBasedOn”, but this is not a good practice at all.

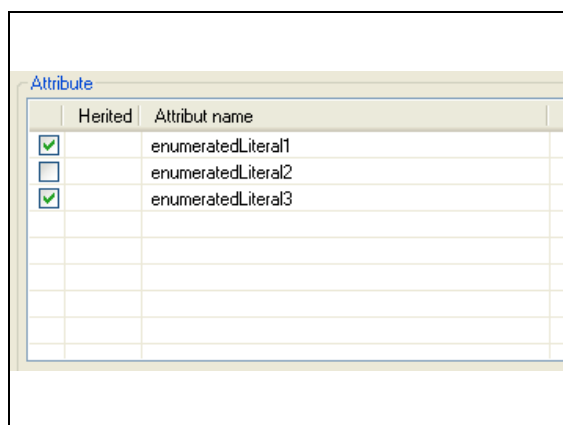
Remember that if you want to be able to reuse any primitive, datatype, enumeration or compound in any profile, their names must be unique in the Information Model and Profiles name space.

7.4.Copy Parent Element

Selecting “*Copy parent’s class*”, allows the inclusion of the Information Model enumeration along with the corresponding “*IsBasedOn*” enumeration, and shows the “*IsBasedOn*” dependency link between the two elements:

7.5.Selecting “EnumeratedLiterals”

Select the “*enumeratedLiterals*” for the “*IsBasedOn*” enumeration



7.6.Editing an IsBasedOn enumeration:

If you need to change some choices on an IsBasedOn enumeration or update the profile enumeration according to Information Model new version, you could use the Edit an IsBasedOn.

Note: in this version do not use the EditIsBasedOn on profile UnitSymbol enumeration neither on Currency enumeration. Delete the profile enumeration and redo a drag and drop of the CIM enumeration.

8. Create “IsBasedOn Datatypes” with CimConteXtor

When the drag and drop element is a <<CIMDatatype>>, it means that you want to restrict the value space, the value domain or both of the <<CIMDatatype>>. The “*IsBasedOn*” window for a <<CIMDatatype>> will be displayed as follows:

IsBasedOn functionality: Creating ActivePower

Qualifier
Enter a qualifier or create one :
Select
No qualifier
Delete

Attribute

Possible inheritance	Attribute name	Lower...	Upper...
☒	value	0	1
☒	unit	0	1
☒	multiplier	0	1

Unselect all Select all Edit cardinality Edit facets Edit stereotype Edit constraint

Options
☐ Copy the parent class
☒ Copy parent's notes
☒ Copy parent's TagValues
☒ Copy parent's constraints
☒ Copy parent's stereotype
☐ is root (active)
☐ is abstract
 Inherit from :

Create
☒ Allowed to class and role
 Create

Execute IsBasedOn Cancel

8.1. Qualify the IsBasedOn Datatype

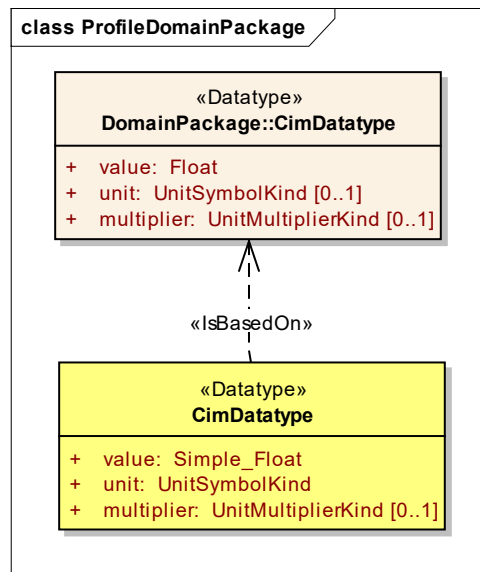
To qualify the name of a datatype see section 5.2 (use and management of qualifiers):

- If the “*Must qualify datatype and enum*” box of the “*Options*” Menu has been checked, the “*IsBasedOn*” datatype name will have the name of the datatype it is based on prefixed by a Qualifier. *CimConteXtor* will force you to enter a qualifier for this datatype.
- If the box is unchecked, the datatype could have the same name as the datatype it is “*IsBasedOn*”, but this is not a good practice at all.

Remember that if you want to be able to reuse any primitive, datatype, enumeration or compound in any profile, their names must be unique in the Information Model and Profiles name space.

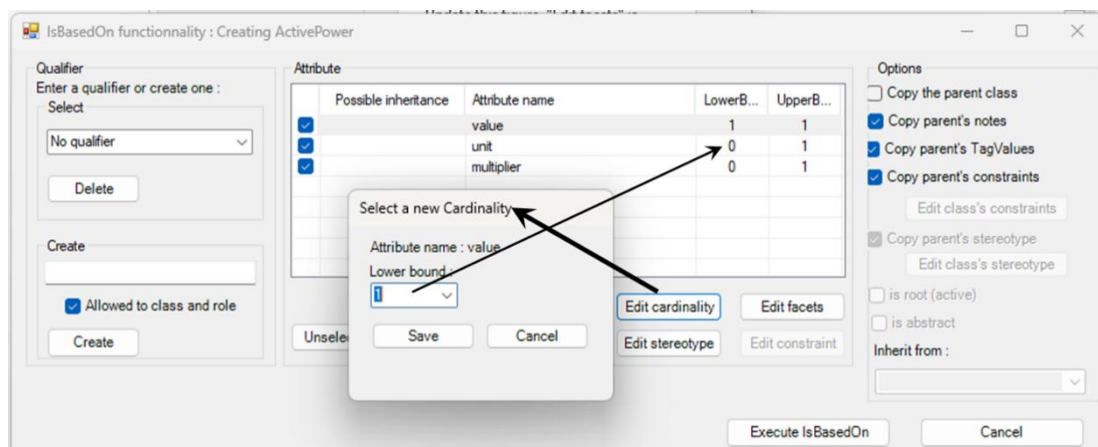
8.2. Copy Parent Element

Selecting “*Copy parent's class*”, allows the inclusion of the Information Model enumeration along with the corresponding “*IsBasedOn*” enumeration, and shows the “*IsBasedOn*” dependency link between the two elements:



8.3. Define attribute cardinality

Select an attribute and click on the “Edit cardinality” button. This will display the “cardinality” window. Select appropriate cardinality and save:



8.4. Define CIMDatatype value space

The value space could be defined as described in section 6.6 (define value space restriction).

8.5. Define default or fixed value

To define default or fixed value, just use the usual EA class property window.

8.6. Define Datatype value domain

Select one of the other datatype attributes (other than attribute “value”), and refine if necessary, its cardinality and type, as done for the attribute “value”.

9. Create “IsBasedOn Compound” with CimConteXtor

When the drag and drop element is a <<Compound>>, it means that an “IsBasedOn” <<compound>> class should be created. Apart from the fact that a <<compound>> class has no associations, the basic “IsBasedOn” process is nearly the same as the one for a class: see next section on “Create IsBasedOn Class with CimConteXtor”.

The only difference with class “IsBasedOn” process is the use of qualifiers:

- If the “Must qualify datatype and enum” box of the “Options” Menu has been checked, the “IsBasedOn” compound name will have the name of the compound it is based on prefixed by a Qualifier. CimConteXtor will force you to enter a qualifier for this compound.
- If the box is unchecked, the datatype could have the same name than the datatype it is “IsBasedOn”, but this is not a good practice at all.

Remember that if you want to be able to reuse any primitive, datatype, enumeration or compound in any profile, their names must be unique in the Information Model and Profiles name space.

10. Create “IsBasedOn Class” with CimConteXtor

10.1. Overview

When the drag and drop element is a class, it means that an “IsBasedOn” class should be created, and that the properties of this class are going to be restricted.

There are two behaviours for this IsBasedOn drag and drop feature:

- The **basic one (described here in section 10)**, where the result of the drag and drop is a profile class. This is the default feature.
- The **enhanced one (described in annex H)**, where the result of the drag and drop is a profile class with all its super classes. This is used for example in WG13 or CGMES profiling style. This feature is activated by the checking of the " WG13AutomaticAncestorInProfile" in the configuration file.

Reminder: classes have two kinds of properties:

- the attributes (simple properties),

- the associations (complex properties).

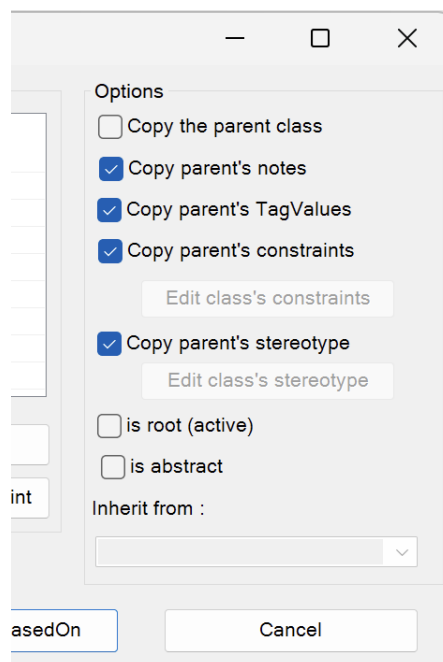
The drag and drop function will let you qualify the class name, define its information and define its attributes based on the parent class. It will not perform class association definition: for this definition the “*Edit connectors*” function will be used (see “*Edit Connectors*” section).

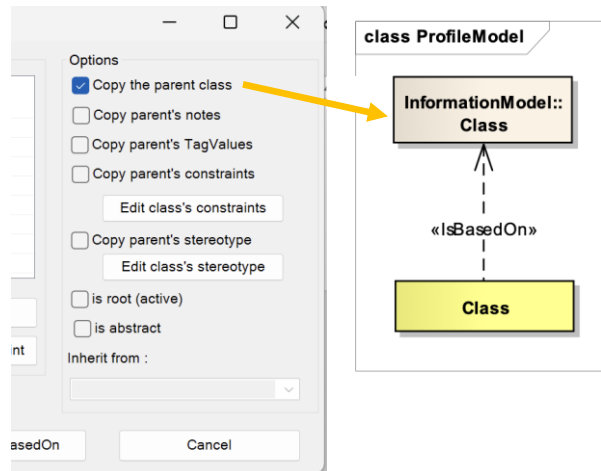
10.2. Select Information for the IsBasedOn class (Options Menu)

The “*Options*” Menu allows you to define all “*IsBasedOn*” class information: Notes, TaggedValues, Constraints, Stereotype, isRoot Class, isAbstract Class.

10.3. Copy parent class

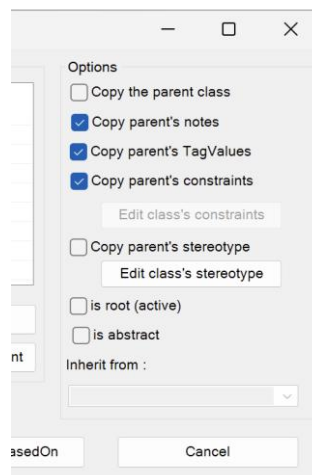
Copy the information Model class in the profile diagram: this option allows the inclusion in the profile diagram of the Information Model class along with the corresponding profile class, and shows the “*IsBasedOn*” dependency link between the two classes:



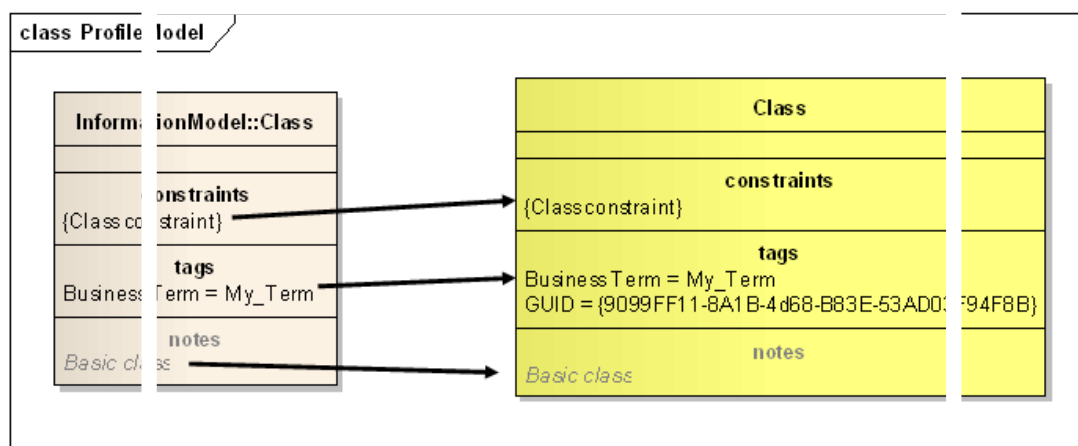


10.4. Copy profile class Notes, Tagged Values, and Constraints

Copy data from the Information Model Class. This option (checking appropriate boxes) allows the copying of all information attached to the Information Class in the Profile Class:



The copied information will be displayed in EA diagram if appropriate EA feature visibility is selected:



Note: the GUID (internal unique identifier attributed by EA for a Model element) TaggedValue is always added to a profile element.

10.5.Edit profile class Notes and Tagged Values of the profile (IsBasedOn) class

To Edit new Notes and select other Tagged Values for the IsBasedOn Class: just use the usual EA class property window.

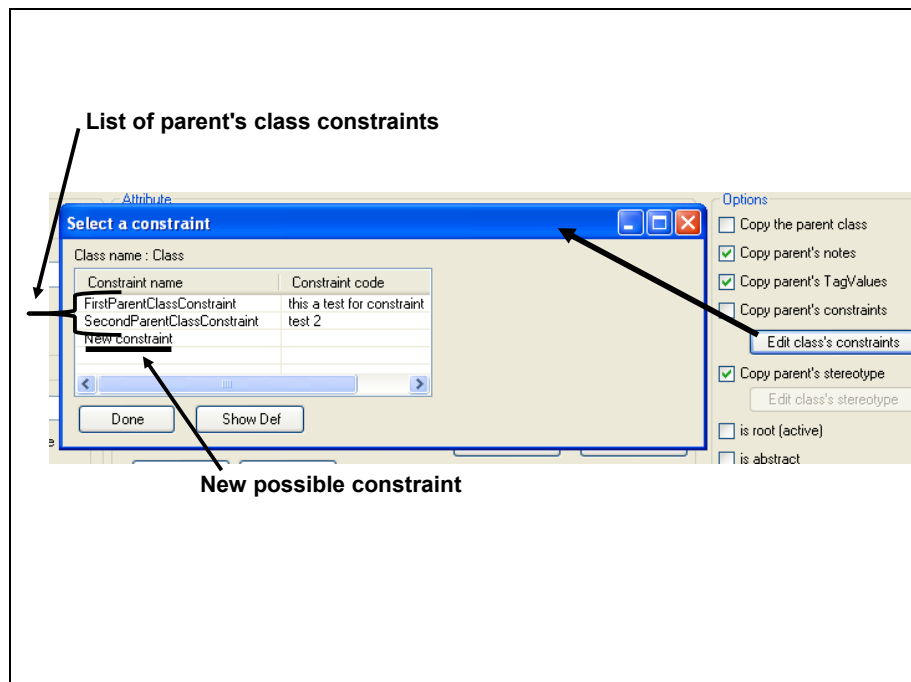
10.6.Edit profile class Constraints

10.6.1.Overview

A profile class might have constraints:

- a copy of parent class' constraints, a selection of parents' class constraints, and use of any of them with or without modification,
- new constraints.

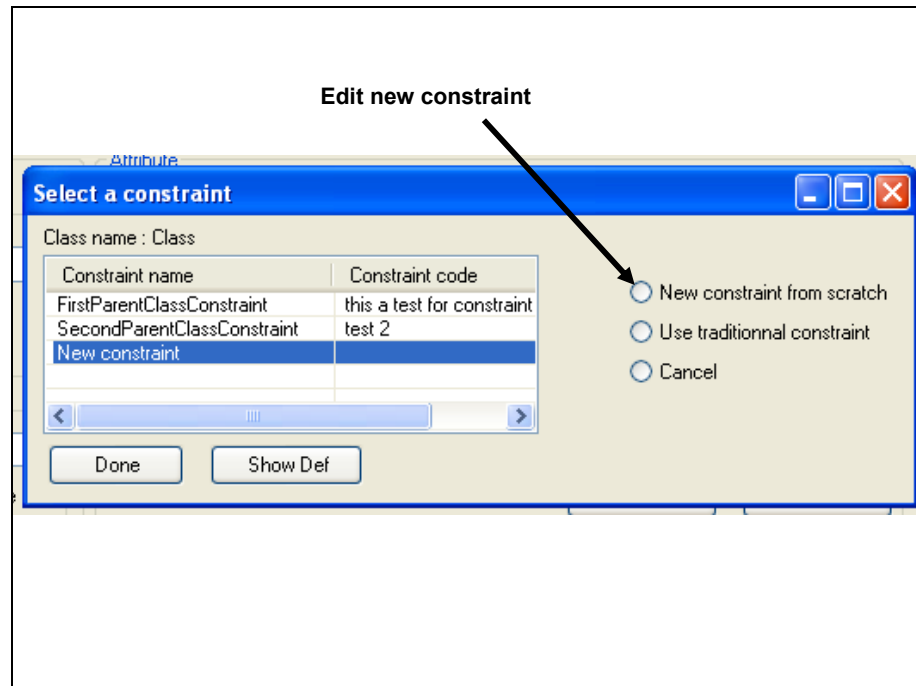
To edit constraints for the Profile Class: click on the “*Edit Class's constraints*” button, a new pop-up window appears and displays a list that includes copy of parents' class constraints (if they are available) and potential new constraint:



To Edit a new constraint for the profile class, select “*New constraint*” (it will be highlighted in blue), the window changed and offers you a choice for constraint editing: edit a new constraint from scratch or use a predefined constraint.

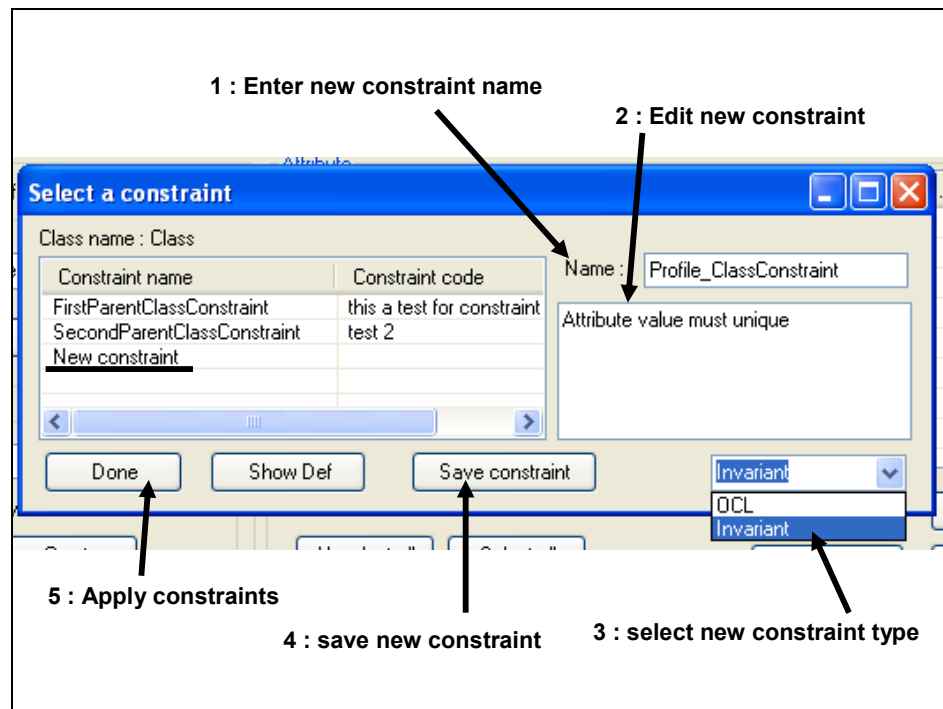
10.6.2.Edit a new constraint from scratch:

Select “*New constraint from scratch*”; this changes the window as follows:

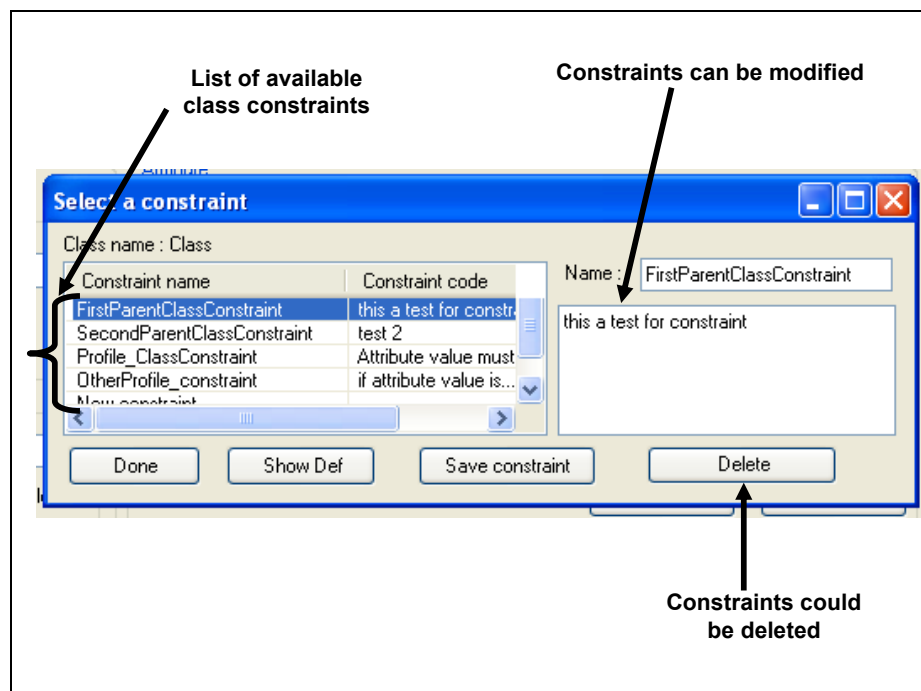


To edit the new constraint:

1. First enter the name of the constraint,
2. Edit the constraint,
3. Select the constraint type (the list of constraint type is managed by EA Menu and is configurable, by default it is OCL or INVARIANT). Usually, the type will be the language in which the constraint is expressed,
4. Click on “*Save constraint*” button,



5. Then you can add other constraints and manage the list of constraints.

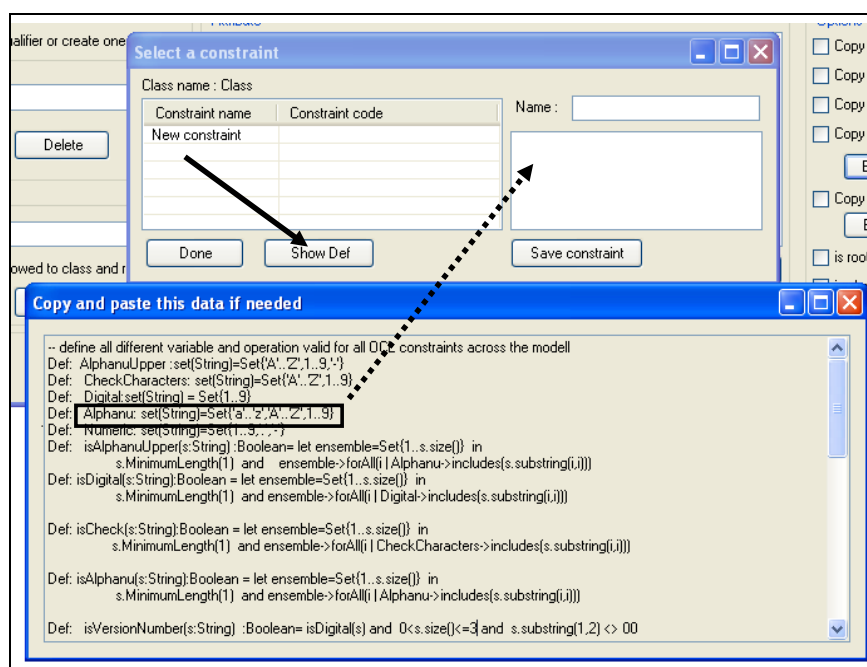


Constraints could be modified and deleted.

Finally, apply the constraints to the profile class by clicking on "Done".

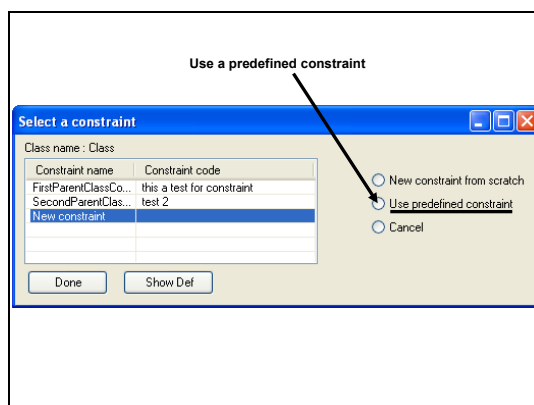
10.6.3. Use of constraint template

Constraint templates may be edited in the configuration file of *CimConteXtor*. Then this template could be used to edit new constraints. These templates could be accessed through the “Show Def” button, which will display in a “Copy and paste this data if needed” window, where the constraint template could be copied and pasted:

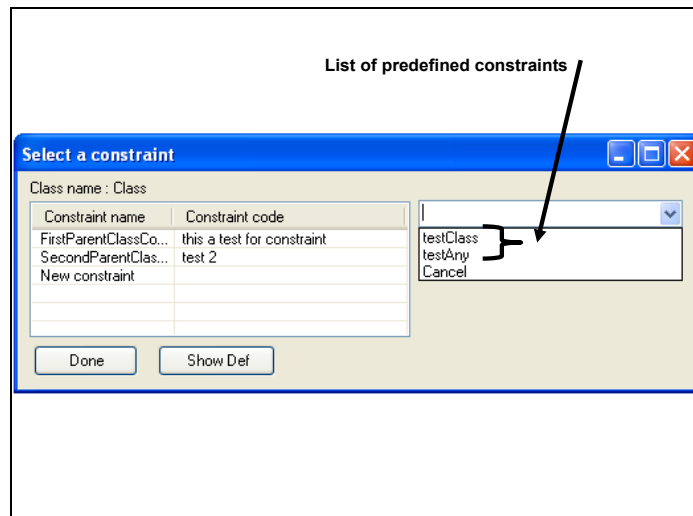


10.6.4. Use of predefined constraints:

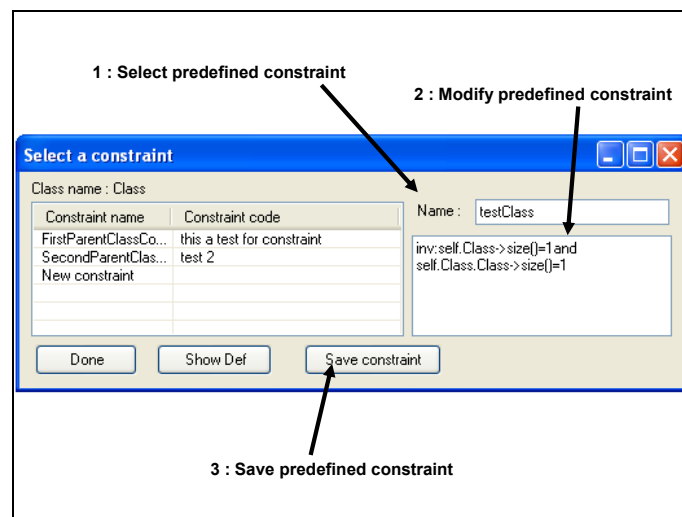
Predefined constraints are edited in the configuration file of *CimConteXtor*. To use them, select “Use predefined constraint”:



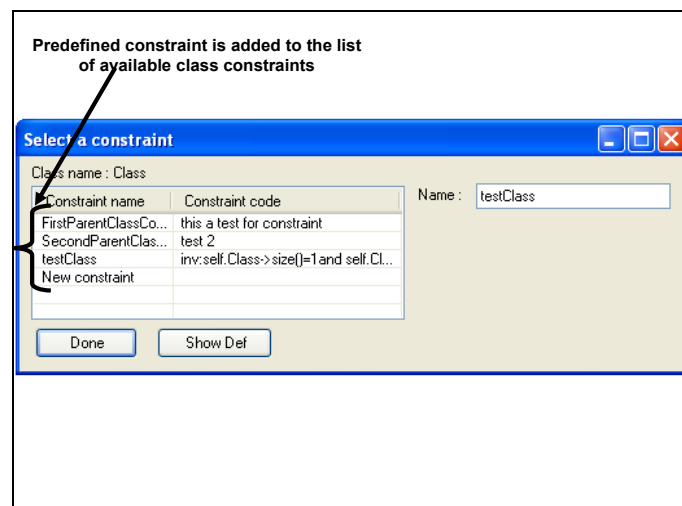
The following window is displayed, and a drop-down menu gives the list of predefined constraints:



Select a predefined constraint, modify it if appropriate and save:

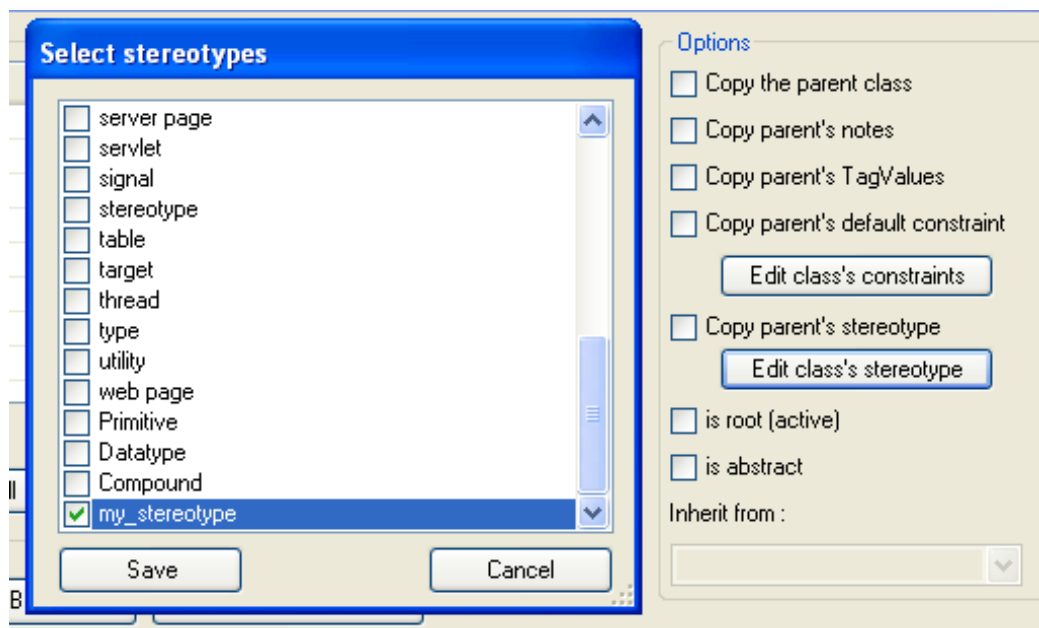


The predefined constraint is added to the list of class constraints:



10.7. Edit profile class Stereotype

To edit a new stereotype for the Profile Class: do not select “Copy parent’s stereotype” in the “Options Menu”. By doing that, the “Edit Class’s stereotype” button is made available. By clicking on it, a new pop-up window appears that displays all possible class stereotypes:



Select a stereotype

Note: When making a contextual model «IsBasedOn» ESMP, the “Copy parents stereotype” must be unchecked and the stereotype «ABIE» must be added.

10.8. Define profile class as a Root class

Select the “is root (active)” box, if the class is a root class for an XSD implementation. (note, the CimConteXtor “is root (active)” is displayed in the EA class property window as an “Is Active” property and not as an “Is Root” one).

10.9. Define profile class as an Abstract Class

Select “is abstract” box, if the profile class is abstract (see “Use of inheritance for profile class” section).

10.10. Inherits

This option is explained in the “Use of inheritance for profile class” section.

10.11. Define Attributes for Profile Class (Attribute Menu)

All the attribute definitions are in the attribute part of the “*IsBasedOn*” window:

Herited	Attribut name	Lower...	Upper...
☒	attribute1	0	1
☒	attribute2	0	1
☒	attribute3	0	1

Unselect all Select all Edit cardinality Edit type Edit stereotype Edit constraint

10.12. Select attributes

To select attributes that will become properties of the profile class, check appropriate attribute’s box:

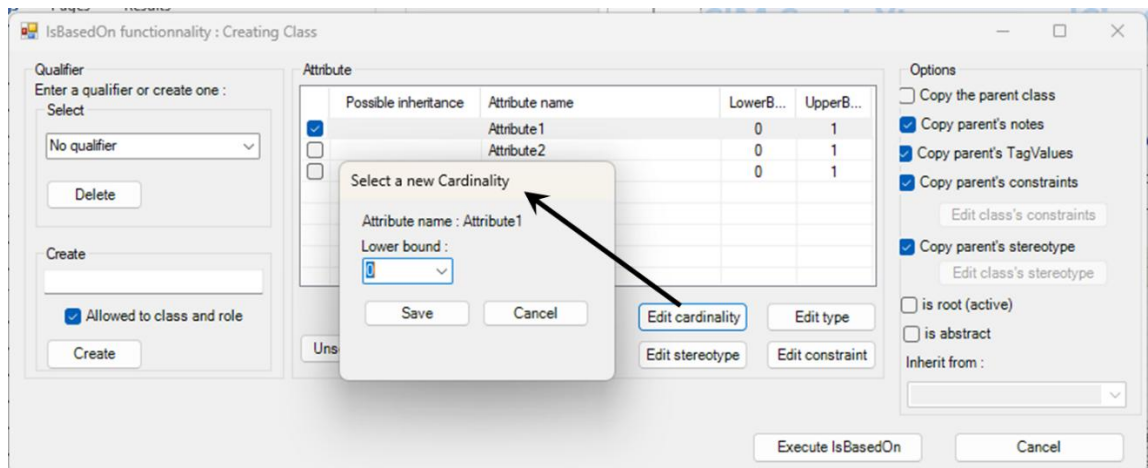
Herited	Attribut name	Lower...	Upper...
☒	attribute1	0	1
☐	attribute2	0	1
☐	attribute3	0	1

Unselect all Select all Edit cardinality Edit type Edit stereotype Edit constraint

Note: attributes that are mandatory cannot be deselected.

10.13. Define attribute's cardinality (optional or mandatory)

To refine an attribute's cardinality, click on the attribute whose cardinality you want to refine (it will be highlighted in blue), then click on "Edit cardinality" button, the "Edit cardinality" window will be displayed:

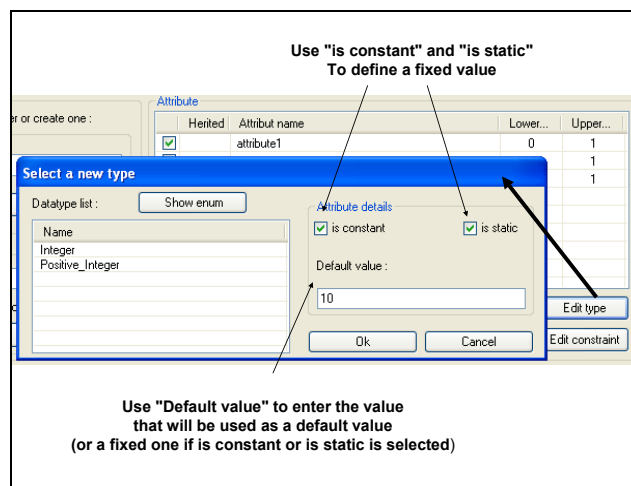


For attributes, the only allowed cardinalities are optional and mandatory: mandatory being a more restricted cardinality than the optional one.

10.14. Define attribute's default or fixed value

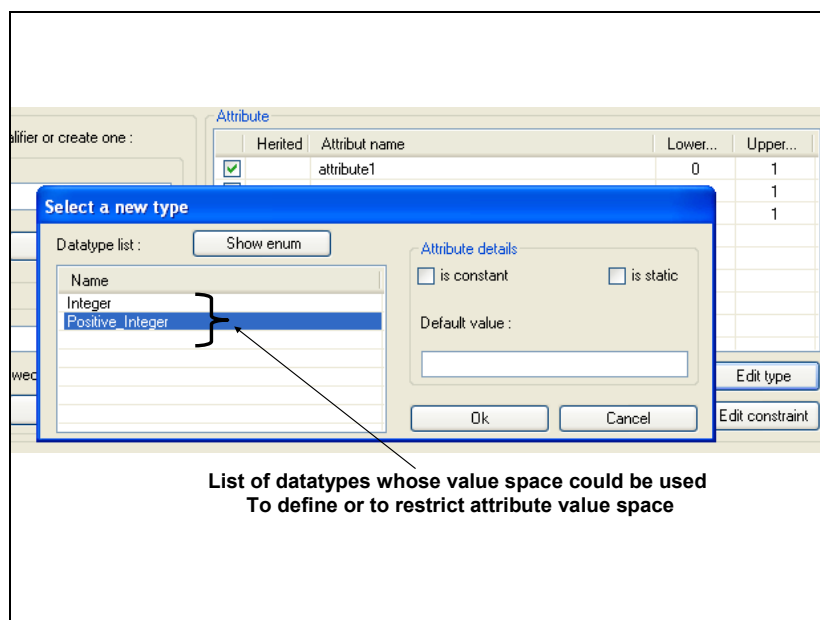
To define default or fixed value for an attribute, click on the "Edit type" button, then the "Edit type" window will be displayed. Then you can set up:

- a default value by entering a value in the "default value" field,
- **or** a fixed value if there is a value in the "default value field" and the "is constant" and "is static" boxes are selected.

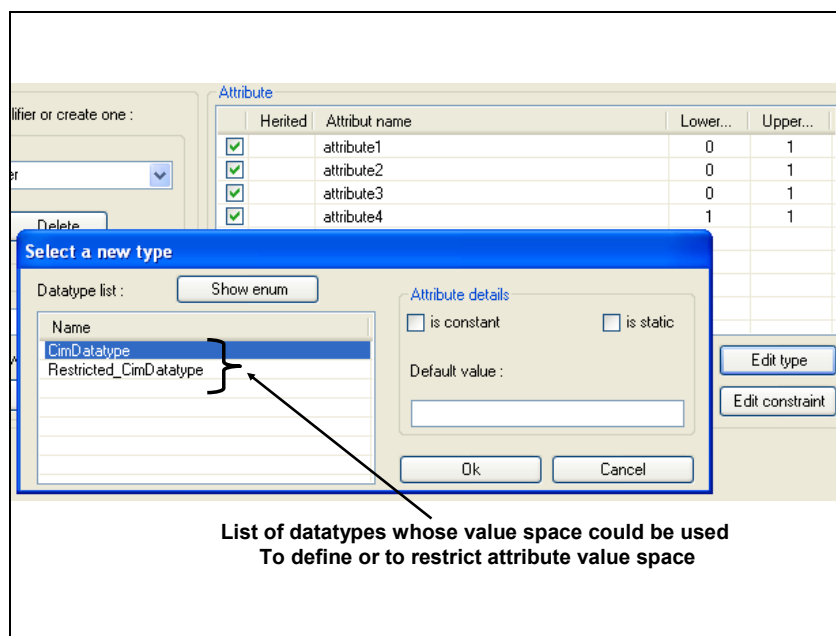


10.15.Edit attribute type:

To define the value space (and value domain if appropriate) of an attribute, select the attribute (it will be highlighted), then click on the “Edit type” button, the “Edit Type” window will be displayed:

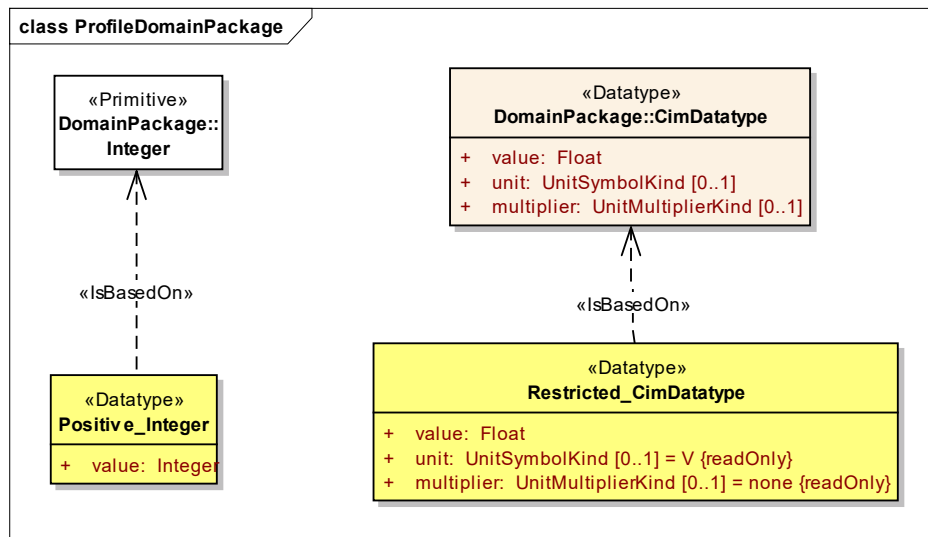


OR



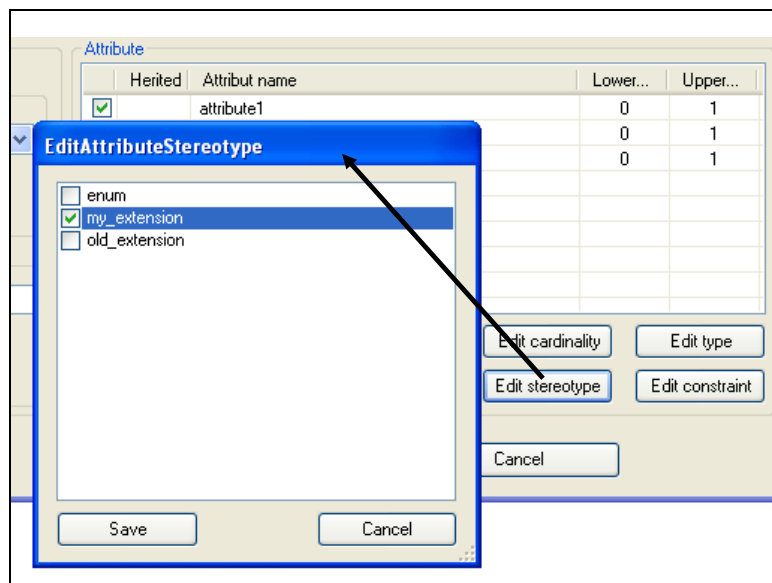
A datatype list is displayed and shows which **already existing** datatypes could be used to restrict the attribute’s value space, and/or its value domain. Select the appropriate datatype.

Remember: datatypes are organized in a hierarchy of “*IsBasedOn*” datatypes, and the list will give an idea of the hierarchy, and thence all the possible restrictions. *The first datatype is always the least restricted one.*



10.16.Edit attribute stereotype

To edit attribute’s stereotype, select the attribute, then click on “*Edit stereotype*” button, the “*EditAttributeStereotype*” window will be displayed. The list of all available attribute’s stereotypes will be shown, check the appropriate stereotype and “*Save*”:

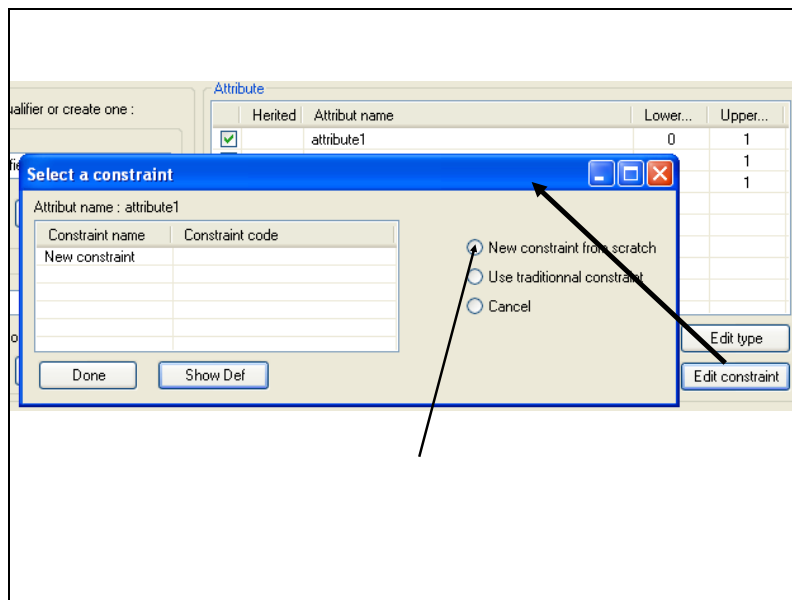


10.17. Edit attribute constraint

Remember, make difference between:

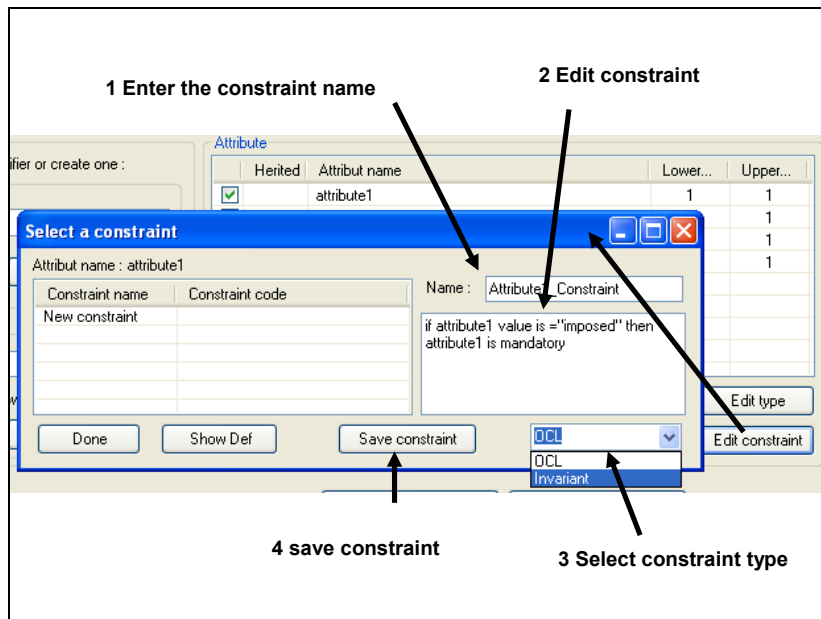
- the “*Edit facets*” that is used to restrict the value space of a datatype,
- the “*Edit type*” that is used to select the type of an attribute,
- and “*Edit constraint*” that is used to express attribute’s constraint (example the value of the attribute should be unique).

To edit an attribute constraint, select the attribute (it will be highlighted), then click on the “*Edit constraint*” button, the “*Edit Constraint*” window will be displayed. To define a new constraint, click on the “*New constraint from scratch*”:



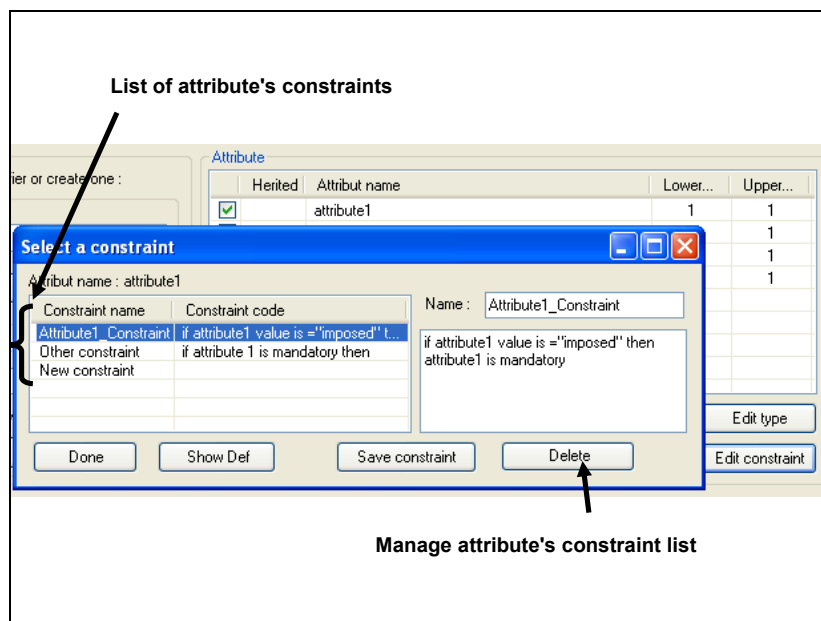
Define attribute new constraint:

1. Enter a name for the constraint
2. Edit constraint
3. Select constraint type



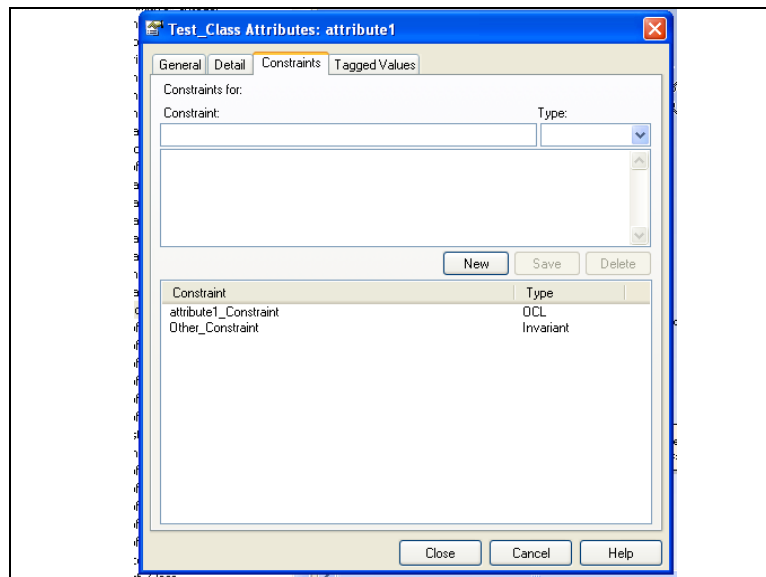
4. Then "Save".

The constraint has been added to the list of the possible constraints for the attribute:



Attribute constraints could also be done using constraint template and predefined constraints. These functions are the same as those used for class constraint editing.

The attribute constraints could be seen in the corresponding EA attribute property window:



11.Modify an IsBasedOn Element

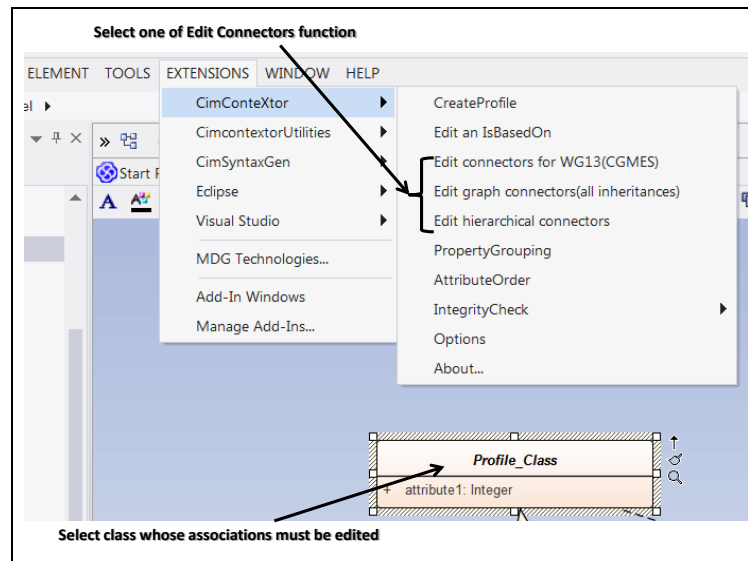
To modify an “*IsBasedOn*” (or profile) element (enumeration, datatype, compound or class) uses the “*Edit an IsBasedOn*” function.

12.Create Associations for profile classes: use of “Edit Connectors”

12.1.Overview

Once profile classes have been edited, then associations between them can be specified. Here we describe associations editing between profile classes, in the case of profile classes that stand by themselves: this means that the inheritance functionality is not used and so that there are no profile super classes. (See Inheritance section if you want to use inheritance in your profile and have profile super classes).

To specify associations for a profile class, click on the class and then go to the “*CimConteXtor Menu*” and select one of the “*Edit connectors*” functions:

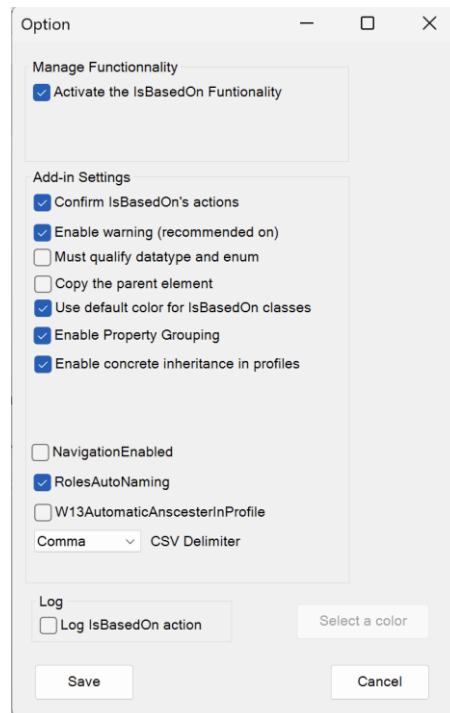


12.2.Edit connectors functions

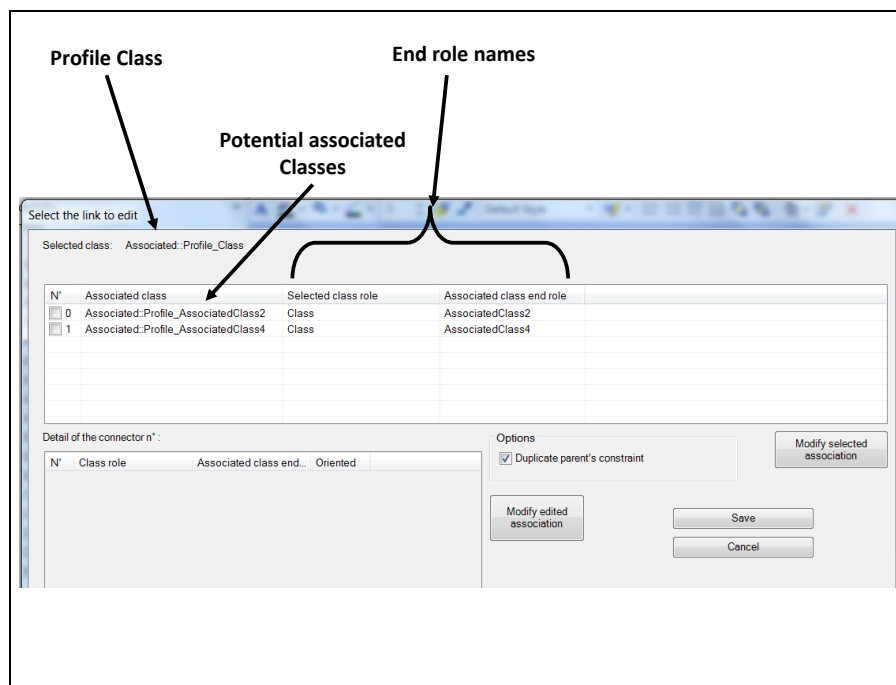
There are three different "Edit Connectors" functions for three different profile targets:

- two (Edit Connectors for WG13 and Edit Graph Connectors) targets graph profiles (no hierarchy):
 - simple graph according to IEC 61970-452 or CGMES,
 - complex graph.
- One targets hierarchical profiles (according for examples to IEC/TC57/WG14 profiles or IEC/TC57/WG16 European style Market Profiles whose syntactic output is XSD).

Warning: if you want to target hierarchical profiles, first go to the CimConteXtor Options Menu and if "NavigationEnabled" box is checked, unchecked it, and click 'Save':

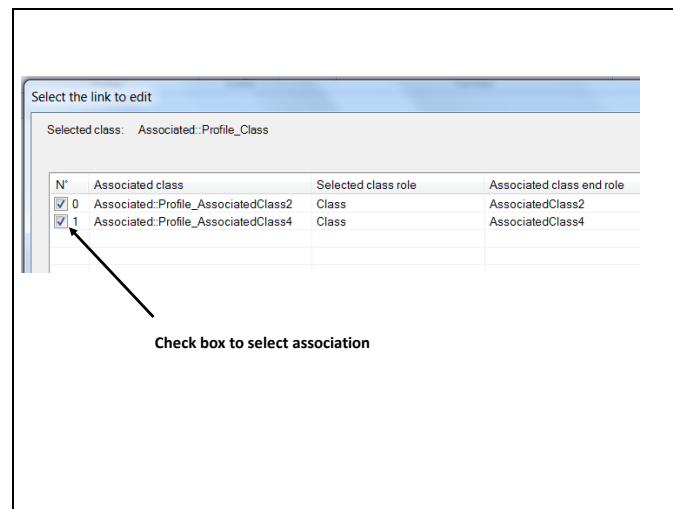


When you select one edit connector function, then a general "Edit connectors" window is displayed, whatever function (WG13, Graph or Hierarchical Edit Connectors) is selected:



12.3. Select profile class associations

This is done by checking the association or connector box.



If you do not need to refine the association, then click on “Save” button: this will draw the lines that show associations between the selected profile class and the associated classes (and create the association in the profile model).

13.Refine profile class associations for WG13 or Graph Profile style

13.1.Overview

When you have used the WG13 or graph Edit Connectors function, and you have selected an association, then if you click on “Modify Selected Association” button, an “EditingAssociation” window pops up and allows the refinement of the “IsBasedOn” association as follows:

Select the link to edit

Selected class: Associated:Profile_Class

N°	Associated class	Selected class role	Associated class end role
0	Associated:Profile_Associat...	Class	AssociatedClass2
1			

Editing AssociationClass - Profile_AssociatedClass2

Class

Parent's role : Class

class role

No qualifier

Class

☐ Is Direction End

Profile_AssociatedClass2

Parent's role : AssociatedClass2

Other class role

No qualifier

AssociatedClass2

☐ Is Direction End

Options

☒ Copy parent's TaggedValues

☒ Copy parent's notes

Stereotype

☒ Copy Parent's stereotype

Cardinality

Selected class cardinality : 0 * Other end cardinality : 0 *

Manage qualifier

Save Cancel

There are several parts in this “*Modify Selected Association*” window:

- Selected class role
- Other end role
- Options
- Stereotype
- Cardinality
- Manage qualifier

13.2.Copy Tagged Values or Notes

If boxes in the “*Options*” part of the window are selected, the parent association Tagged Values and Notes will be copied in the “*IsBasedOn*” association.

13.3.Edit Profile Association Tagged Values or Notes

To edit new Tagged Values or new Notes, use the usual EA association Properties window.

13.4.Association Stereotype

To copy the parent association stereotype check “*Copy Parent’s stereotype*” box.

To edit a new stereotype, uncheck the “Copy Parent’s stereotype” box and pick up a stereotype in the drop-down menu.

The screenshot shows a dialog box titled "Editing AssociationClass - Profile_AssociatedClass2". It has several sections: "Class" (Parent's role: Class, class role: No qualifier, Class: Class, Is Direction End: unchecked), "Profile_AssociatedClass2" (Parent's role: AssociatedClass2, Other class role: No qualifier, AssociatedClass2: AssociatedClass2, Is Direction End: unchecked), "Options" (Copy parent's TaggedValues: checked, Copy parent's notes: checked), "Stereotype" (Copy Parent's stereotype: unchecked, dropdown menu open showing "No Stereotype", "subscribe", and "No Stereotype"), "Cardinality" (Selected class cardinality: 0, Other end cardinality: 0), and "Manage qualifier". There are "Save" and "Cancel" buttons at the bottom.

13.5.Restrict Association cardinality

To restrict association ends cardinalities, enter or select proper values in each field of either of the association end sides.

The screenshot shows the same dialog box as before, but with the "Cardinality" section highlighted. The "Selected class cardinality" is 0 and the "Other end cardinality" is 0. An arrow points to the "Other end cardinality" field with the label "Restrict Multiplicity".

13.6.Qualify association end role name

Association end role names could be qualified:

- either to express a more refined semantics of the end role name,

- or to distinguish end role name when there are multiple associations between two classes: in this case qualifiers are mandatory.

Enter qualifiers or select qualifiers in the drop-down menus for each end role name:

Editing AssociationClass - Profile_AssociatedClass2

Class
Parent's role : Class
class role : My

Profile_AssociatedClass2
Parent's role : AssociatedClass2
Other class role : New

Options
☒ Copy parent's TaggedValues
☒ Copy parent's notes

Stereotype
☒ Copy Parent's stereotype
No Stereotype

Use Qualifier if appropriate

☐ Is Direction End ☐ Is Direction End

Cardinality
Selected class cardinality : 0 *
Other end cardinality : 0 *

Manage qualifier

Save Cancel

13.7.Managing end role name qualifier

Use the "ManageQualifier" button to enter new qualifiers (See also section 5.2.).

Editing Association - Class - Profile_AssociatedClass2

Parent's role : Class
class role : No qualifier

Profile_AssociatedClass2
Parent's role : AssociatedClass2
Other class role : No qualifier

Options
☒ Copy parent's TaggedValues
☒ Copy parent's notes

Stereotype
☒ Copy Parent's stereotype

Manage Qualifier

Create
☒ Allowed to class and role
RoleQual
Create qualifier

Delete
No qualifier
Delete qualifier

Close

Manage qualifier

13.8.Select association direction

The association must be oriented for WG13 style design (The association will be drawn in UML diagram as an arrow). To do that, select the "IsDirectionEnd" box, on the side you want to have the arrow:

13.9. Save refined association

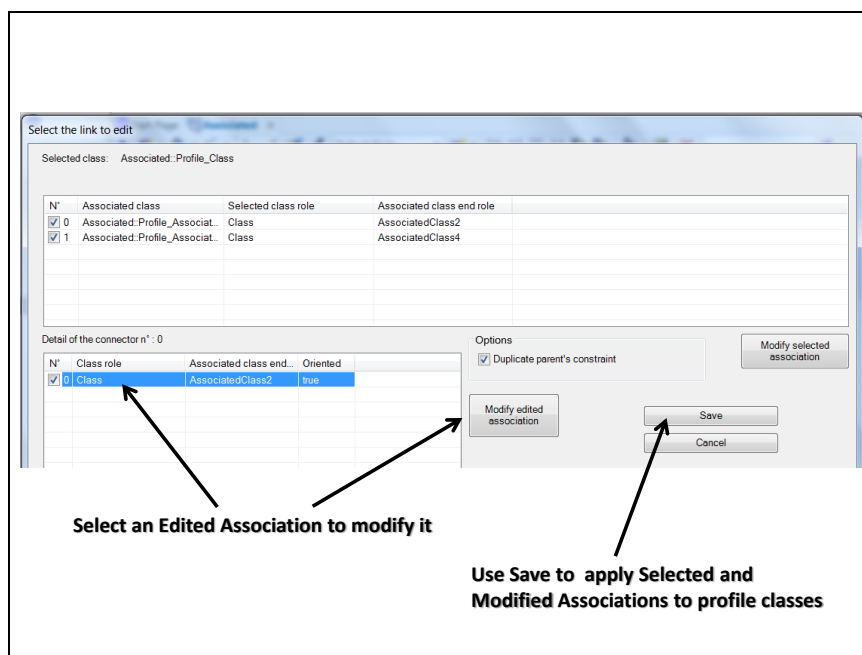
When you click on the “Save” button of the “*Modify Selected Association*” window, the edited association appears in the “*Detail of connector*” window part, where it shows that the association has been refined.

13.10.Edit a modified association

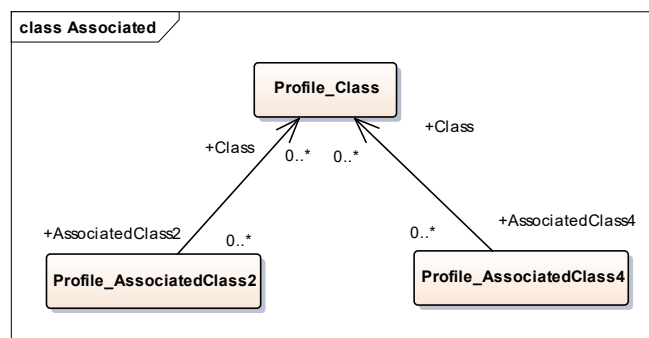
If you want to reedit a modify association, select the modify association in the “Detail connector” part and click on the “Modify Edited Association”, this will display again the “Editing Association” window that will let you modify the modified association characteristics.

13.11.Create associations

To create the associations between a class and other ones, click on the “Save” button, this will draw associations in the diagram:



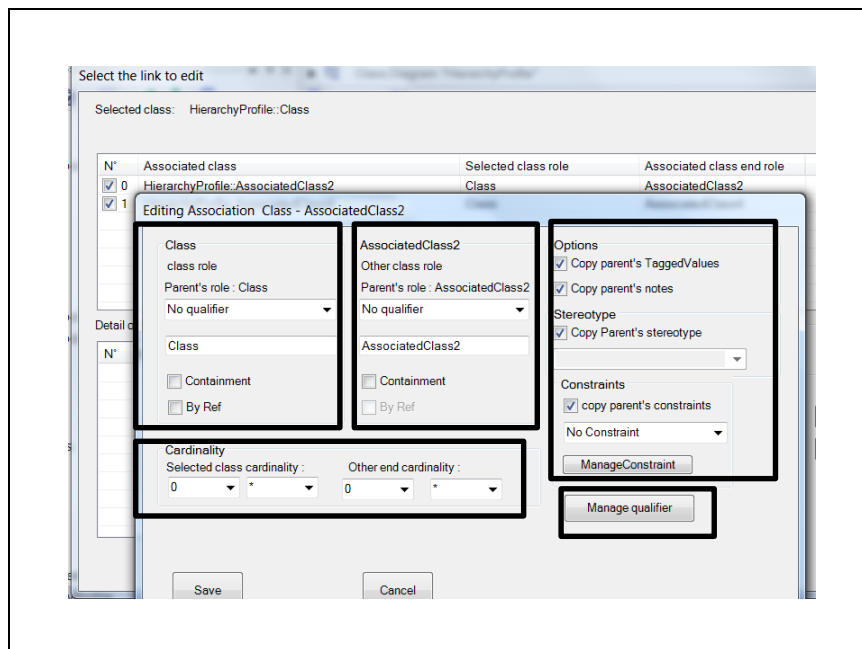
The result will be:



14. Refine profile class associations for Hierarchical Profile style

14.1. Overview

When you have used the Edit Hierarchical Connectors function, and you have selected an association, then if you click on "Modify Selected Association" button, an "EditingAssociation" window pops up and allows the refinement of the "IsBasedOn" association as follows:



There are several parts in this "Modify Selected Association" window:

- Selected class role
- Other end role
- Options
- Stereotype
- Cardinality
- Manage qualifier

14.2. Copy Tagged Values or Notes:

If boxes in the "Options" part of the window are selected, the parent association Tagged Values and Notes will be copied in the "IsBasedOn" association.

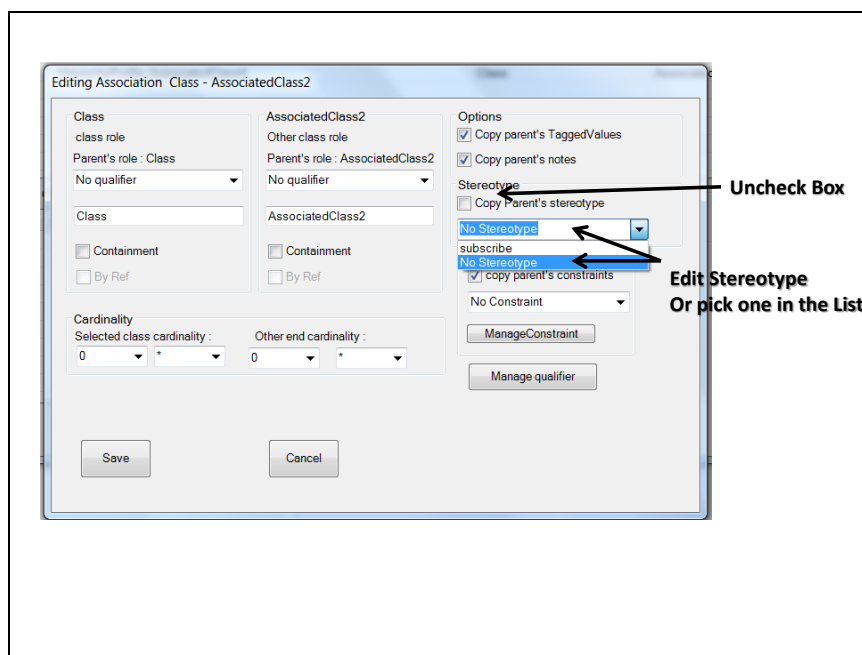
14.3.Edit Profile Association Tagged Values or Notes

To edit new Tagged Values or new Notes, use the usual EA association Properties window.

14.4.Association Stereotype

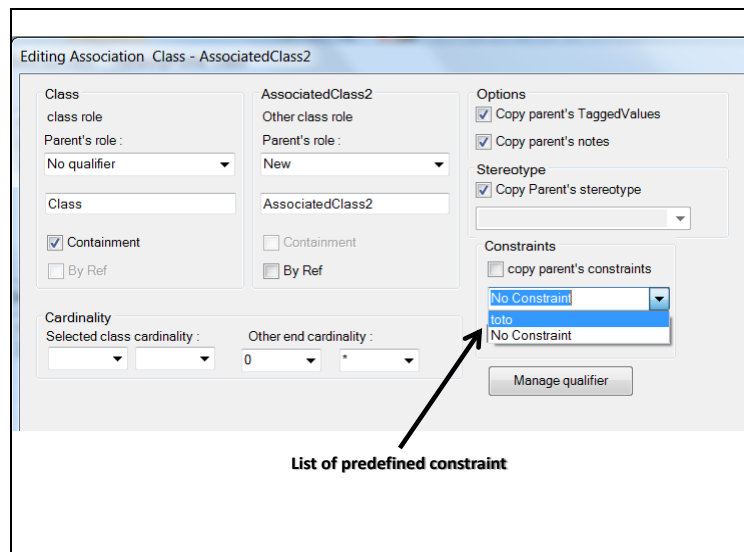
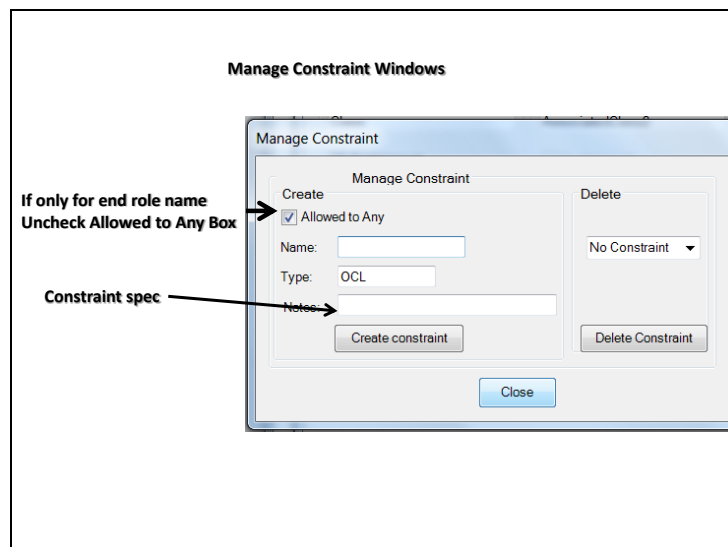
To copy the parent association stereotype check *"Copy Parent's stereotype"* box.

To edit a new stereotype, uncheck the *"Copy Parent's stereotype"* box and pick up a stereotype in the drop-down menu.



14.5.Association Constraint

See section 10.6 for using constraints already defined by the config file or by the "ManageConstraint" button. To attach a constraint to the association, get the list of constraints and select one. If you need to define a new (not existing in the list) use the "ManageConstraint" button to edit a constraint that will be added to the list.



14.6.Manage Qualifier

See section 5.2

14.7.Select association type

The association must be restricted to a containment (usually aggregation is used). To do that, select the containment box:

- This will turn down the containment end side cardinality, because usually the cardinality on this end is 1 (but it could be set up if needed),
- And usually, the containment end side role name is blanked, but it could be kept if needed; so if you do not need it, remove the end role name.

Remove end role name on containment side if not required

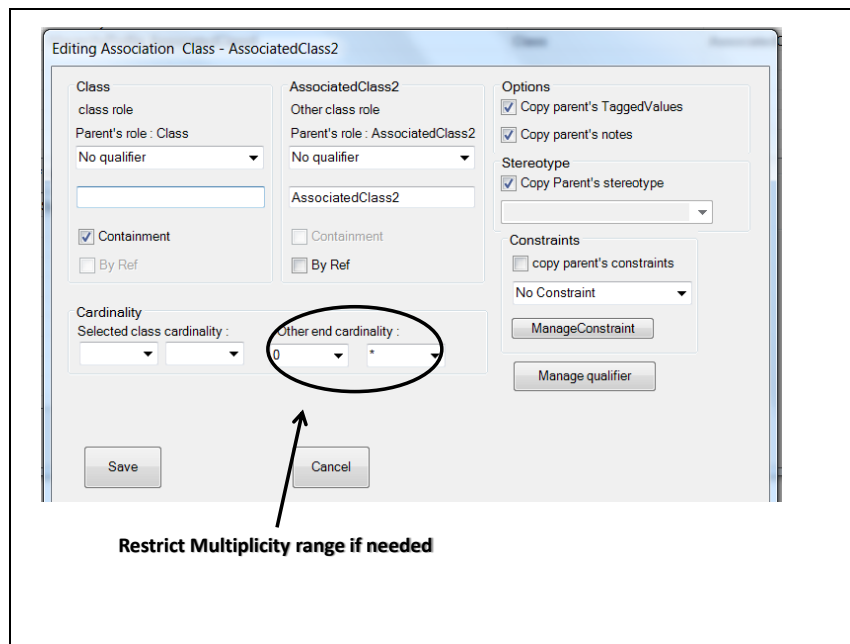
Choose Containment side

Usually the containment end role multiplicity is not shown

Note: in case the association is based on an information model super class, but is used with a subclass, then changing end role name to subclass name is possible if RolesAutoNaming option is checked (See inheritance section).

14.8.Restrict Association cardinality

To restrict association ends cardinalities, enter or select proper values in each field of either of the association end sides.

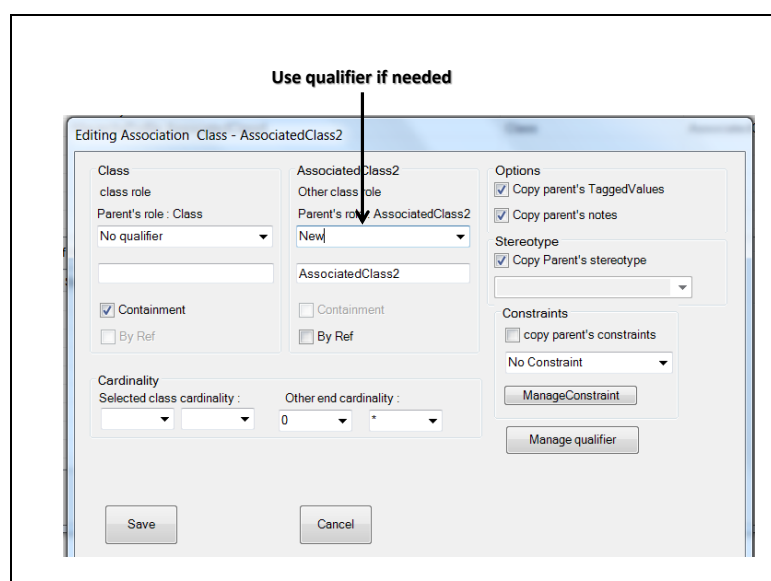


14.9. Qualify association end role name

Association end role names could be qualified:

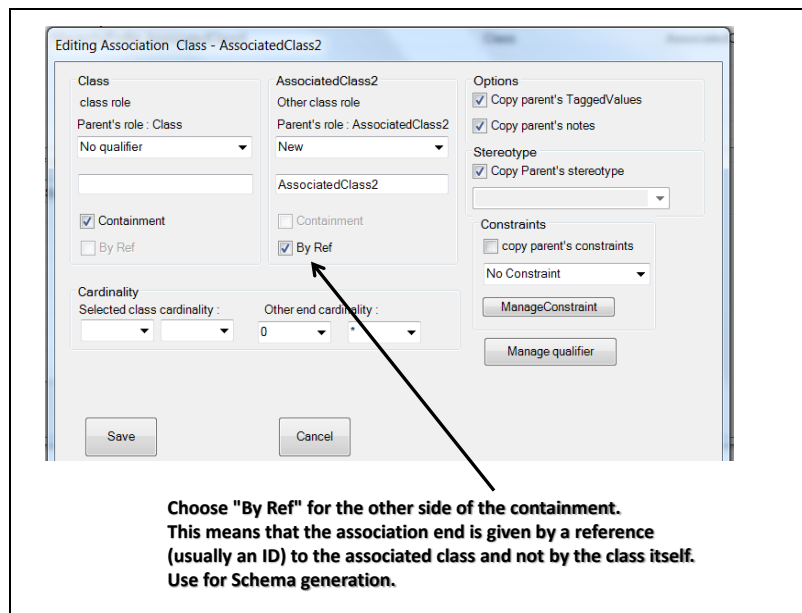
- either to express a more refined semantics of the end role name,
- or to distinguish end role name when there are multiple associations between two classes: in this case qualifiers are mandatory.

Enter qualifiers or select qualifiers in the drop-down menus for each end role name:



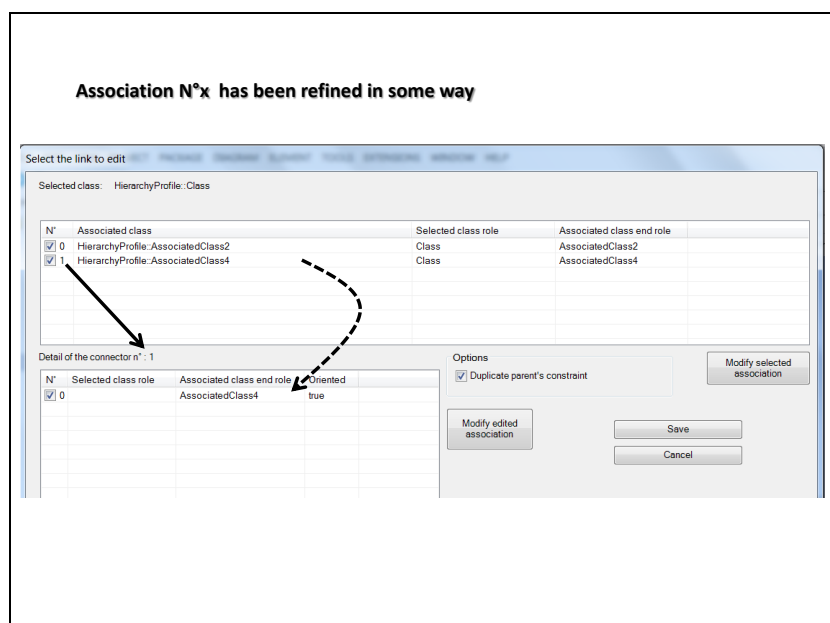
14.10.By Ref association

Sometime, especially with XML Schema, when a profile class is aggregated to several other classes, it might be interesting to use a pointer to it instead of having a duplication of its instances. To enable that, check the “By Ref” box:



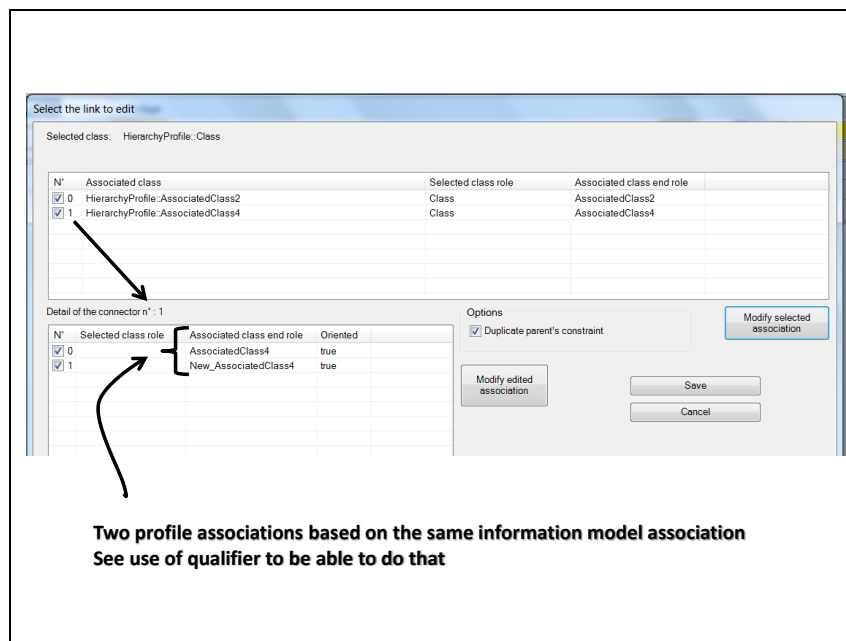
14.11.Save refined association

When you click on the “Save” button of the “Modify Selected Association” window, the edited association appears in the “Detail of connector” window part, where it shows that the association has been refined.

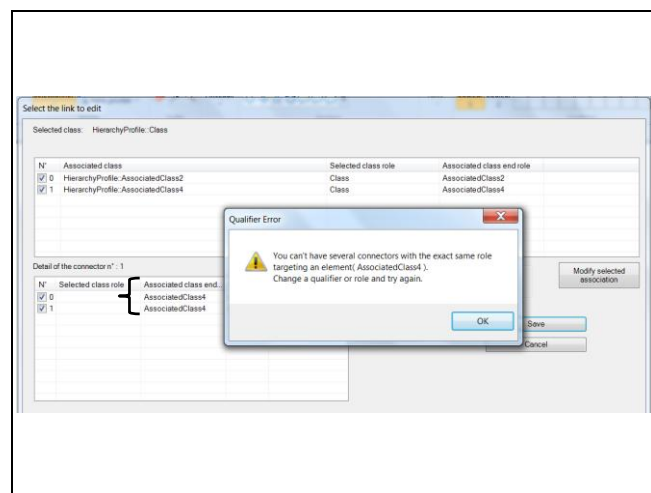


14.12. Make multiple profile associations based on the same Information Model association

If you need to have, between two classes, several associations that are based on the same parent association, redo all the steps of the “*Modify Selected Association*” function. When you save this new association, it will appear in the “*Detail of the connector N°*” window part alongside with the other modify associations based on the same one:



But when you do several associations based on the same one, you need to qualify the end role names to distinguish the associations. If you do not do that, *CimConteXtor* will remind you to do it when you try to create the associations in the profile by clicking on the “*Save*” button:

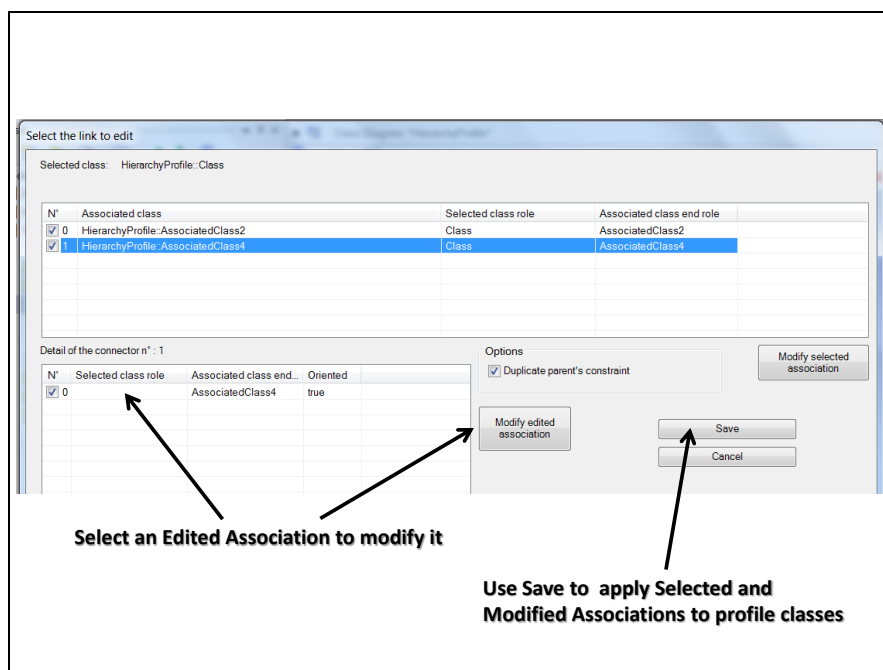


14.13.Edit a modified association

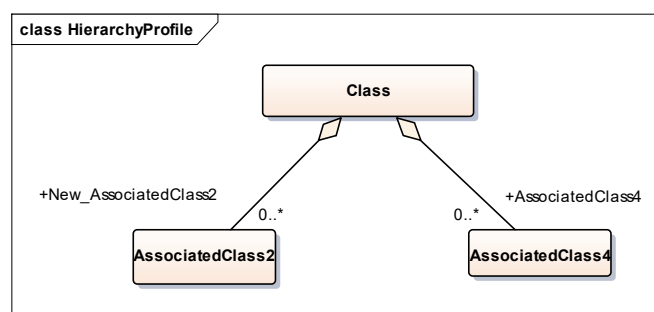
If you want to reedit a modify association, select the modify association in the “Detail connector” part and click on the “Modify Edited Association”, this will display again the “Editing Association” window that will let you modify the modified association characteristics.

14.14.Create associations

To create the associations between a class and other ones, click on the “Save” button, this will draw associations in the diagram:



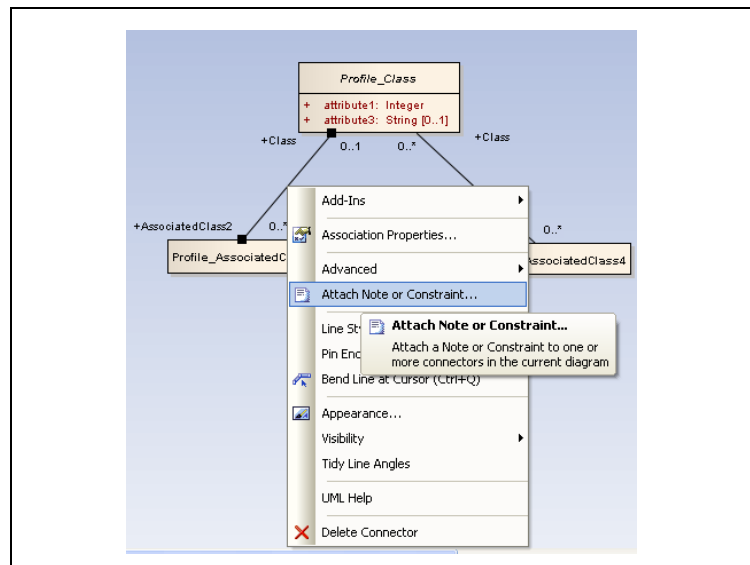
Save will display associations in the diagram:



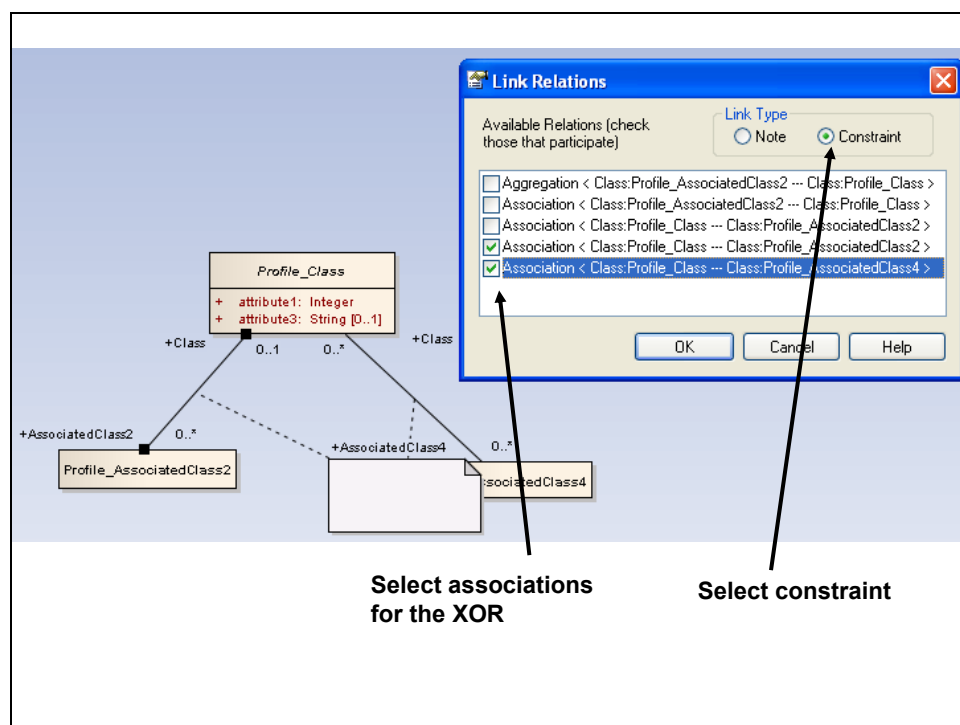
14.15.XOR constraint on Associations

“XOR” is a constraint that says that only one of the associations should be used at a time in an instance.

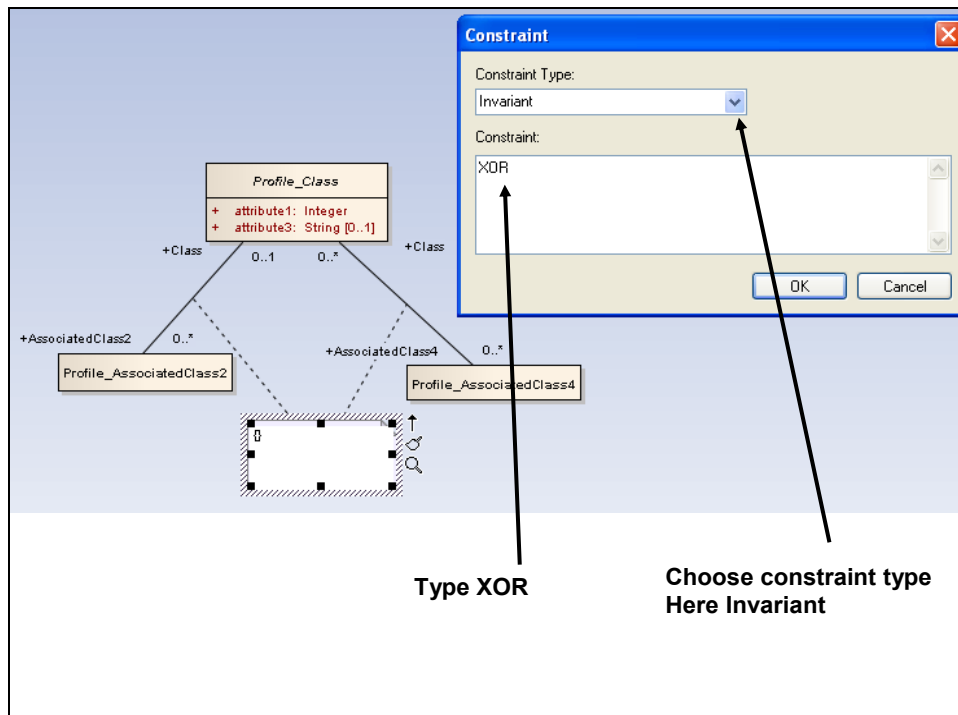
To specify make a “XOR” constraint between associations, **use the usual EA function**. Select an association, then right click on it to have the Menu, and select “Attach Note or Constraint”:



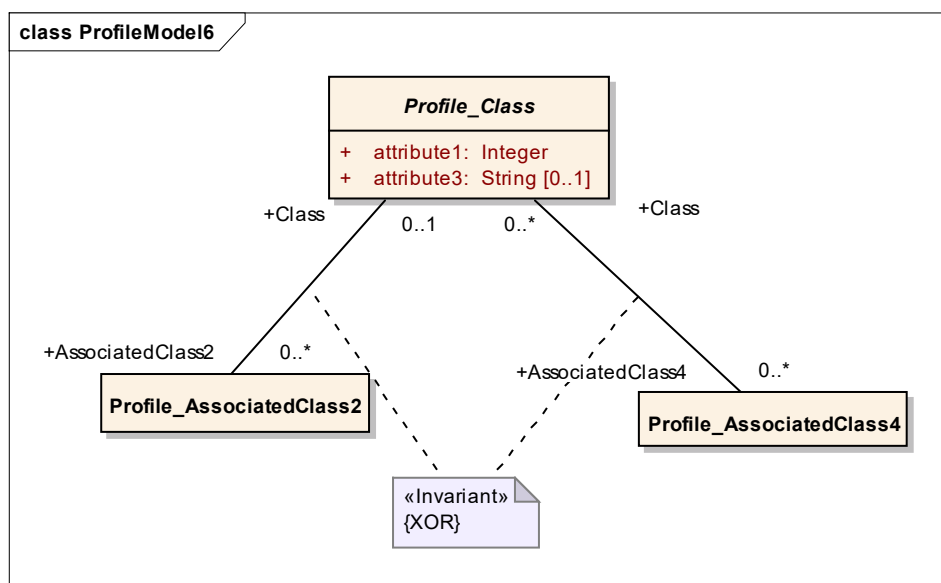
This will display the “Attach Note or Constraint” window. Select constraint, and select associations involved in the common XOR constraint:



Click “OK”, this will display an “Edit constraint” window. Select ‘Invariant’ for type, and enter “XOR” in the constraint field (XOR term is mandatory, enter it exactly like this):



By clicking on “OK” the attach constraint will be displayed in the diagram:



WARNING: the attached XOR constraint note must be recreated each time the “Edit connectors” is reused.

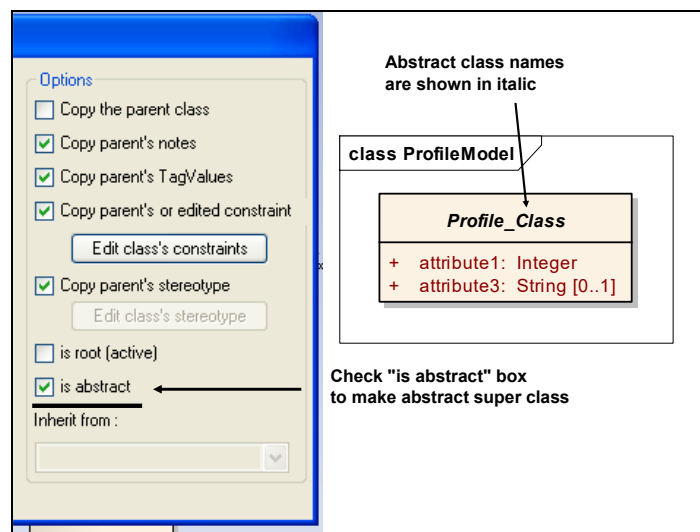
15. Modify profile class associations

Select profile class and use the appropriate “Edit Connectors” function.

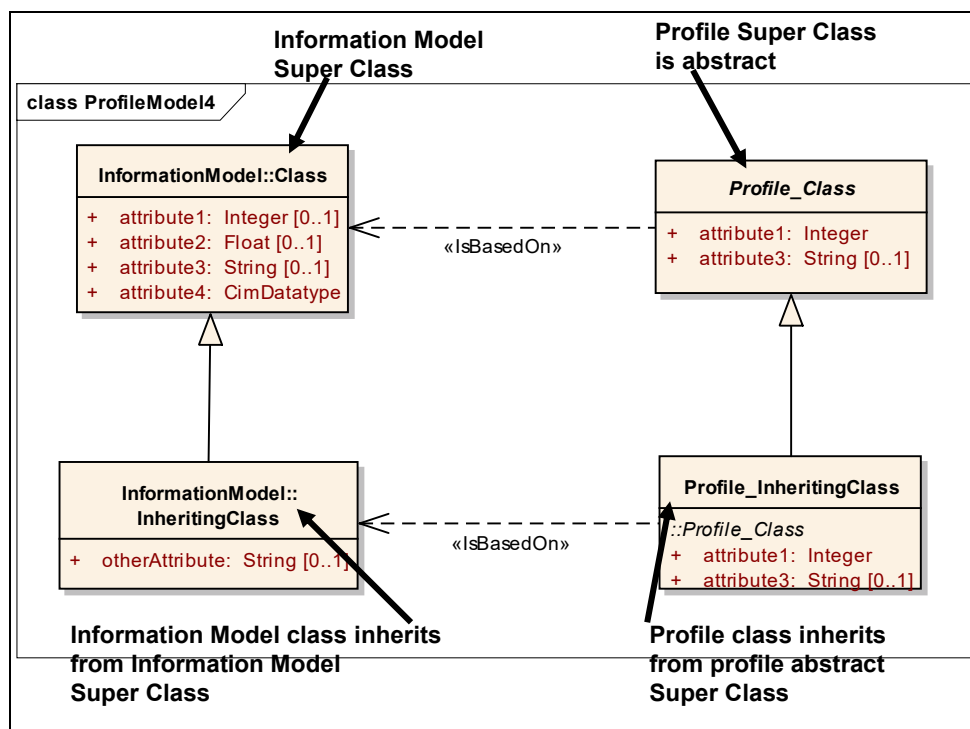
16. Use Inheritance in profile for IsBasedOn classes

16.1.1. Remember:

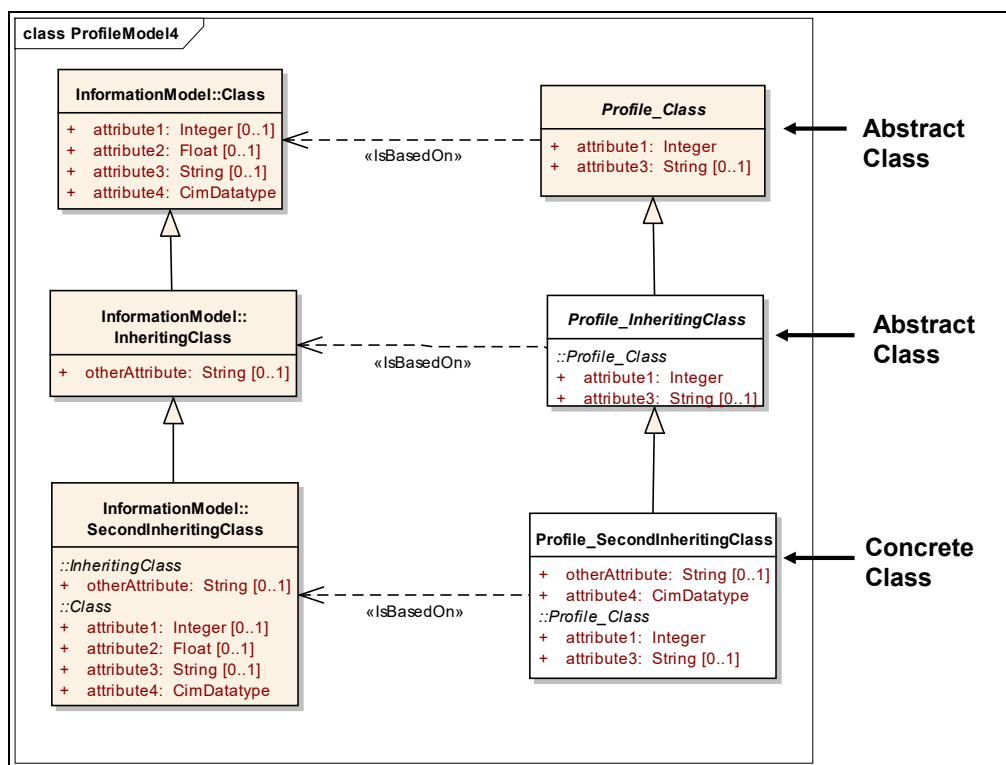
1. In a profile, inheritance can be used, but the good practice is to make all the profile super classes ABSTRACT.



2. In a profile, inheritance can be used between profile classes ONLY if these classes are “IsBasedOn” Information Model classes that have themselves an inheritance relationship:



3. In a profile, inheritance could be recursive, in this case all super classes are abstract and only the last class that inherits is concrete



16.2.CimConteXtor inheritance possibility

By default, a profile class can only inherit from a profile superclass if their inheritance relationship reflects a direct inheritance relationship in the corresponding information model. But this could be removed, by unchecking the "*EnabledIntermediaryInheritance*" element in the configuration file.

16.3.Good practice

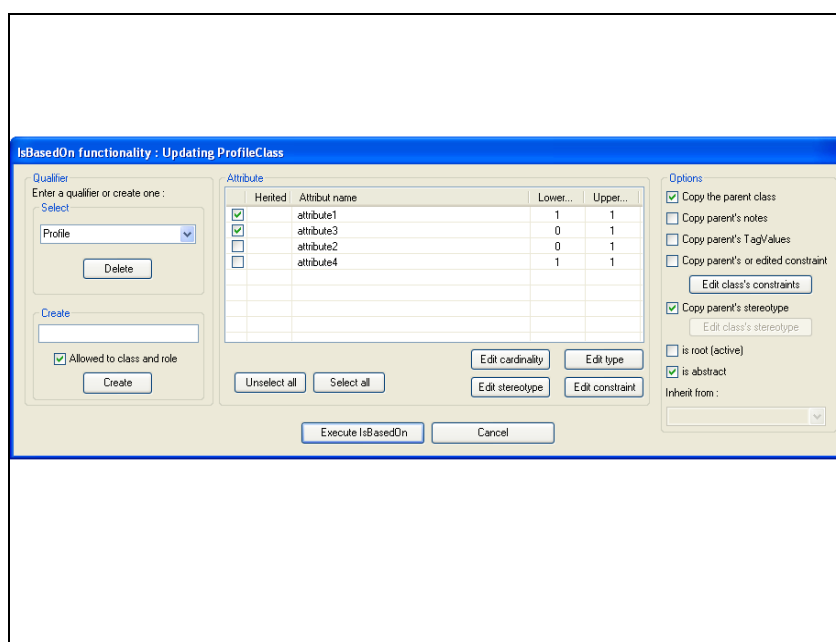
A good practice for doing a profile with super classes is to create first all the profile super classes by using the *CimConteXtor* "drag and drop function" as described in section 10 (create "*IsBasedOn* classes with *CimConteXtor*").

But do not forget to check the "is abstract" box for the class that is "IsBasedOn". In UML, when the class is abstract, its name appears in Italic in the diagram.

16.4. Create an "IsBasedOn" class that inherits from another "IsBasedOn" class

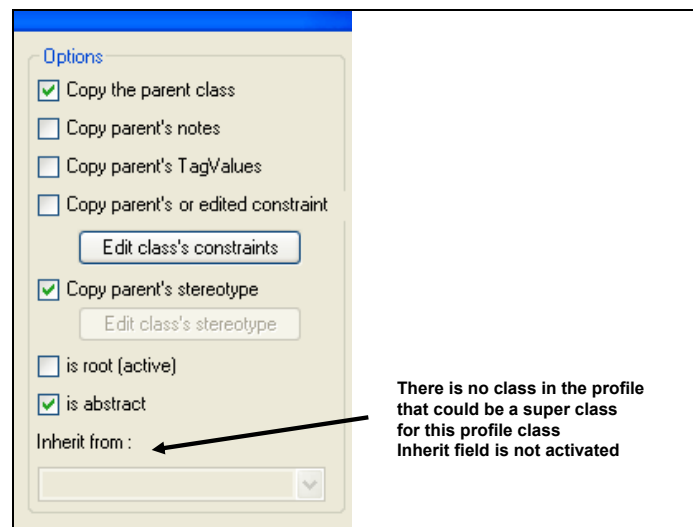
To create the "*IsBasedOn*" class use the drag and drop function, several successive windows will be displayed:

- the usual EA window for making a link, or an instance or a child (select a link),
- then the *CimConteXtor* window that asks if you want to override EA function (say yes),
- and at last, the *CimConteXtor* "Edit *IsBasedOn*" window.

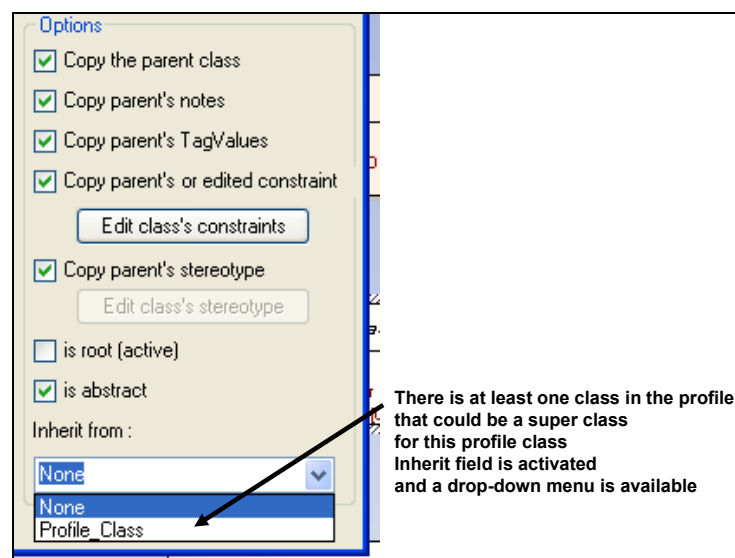


CimConteXtor is looking if there is a possible profile superclass for this "*IsBasedOn*" profile class:

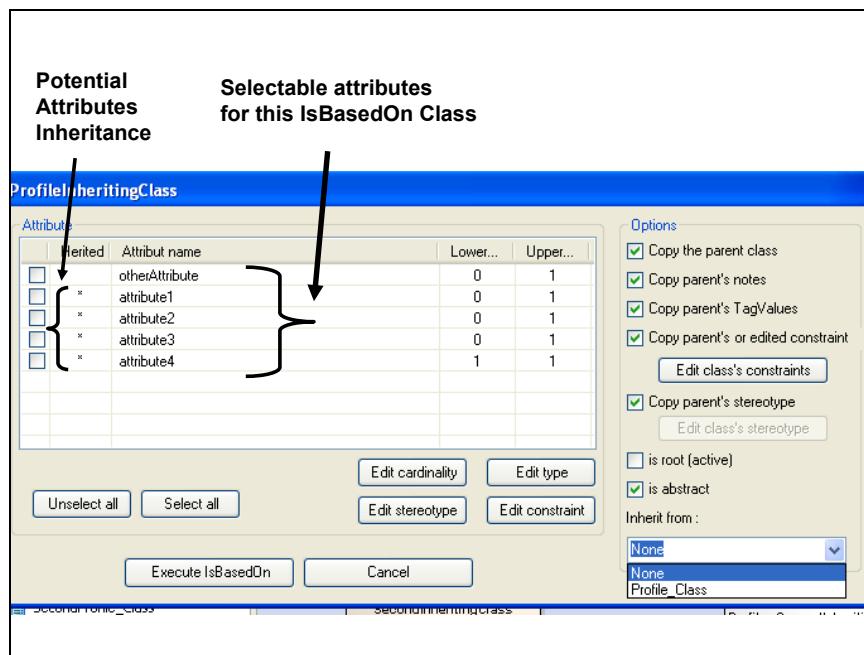
- If there is no possible super class, the “*Inherit from*” field is not shown:



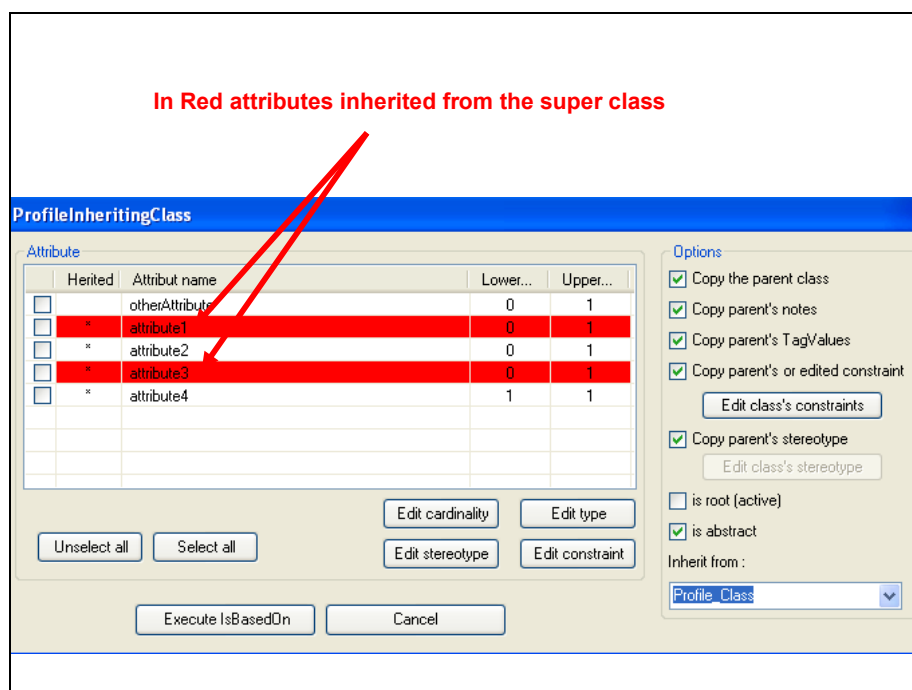
- If there is a possible super class, the “*Inherit from*” field is shown and a drop-down menu shows which classes it could inherit from:



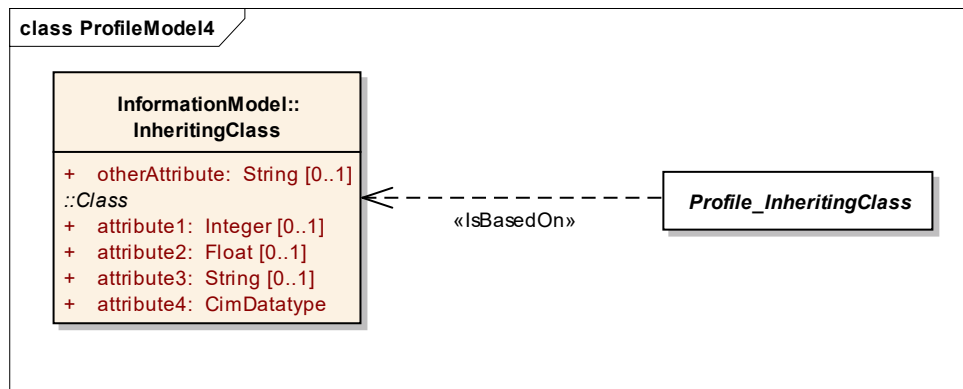
The attribute part of the “*IsBasedOn*” window will show all possible attributes for this “*IsBasedOn*” class and all possible inheritance for attributes.



Here, because you want to create an “*IsBasedOn*” class that inherits, select the profile class that will be the superclass for your “*IsBasedOn*” class. By doing this the attribute part of the “*IsBasedOn*” window will highlight in red all the attributes that are inherited from the superclass and that you cannot uncheck now when you do your attribute selection.



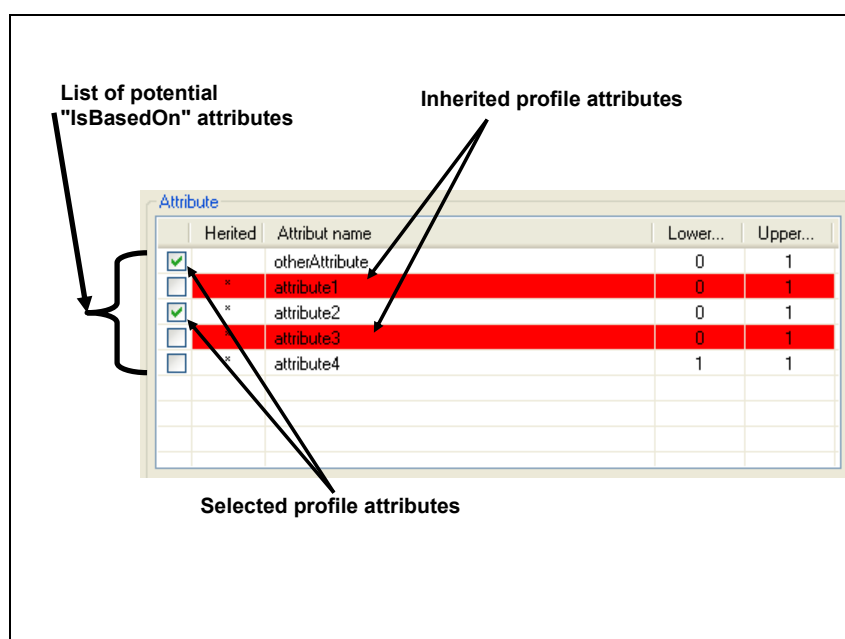
Some explanations in the example shown:



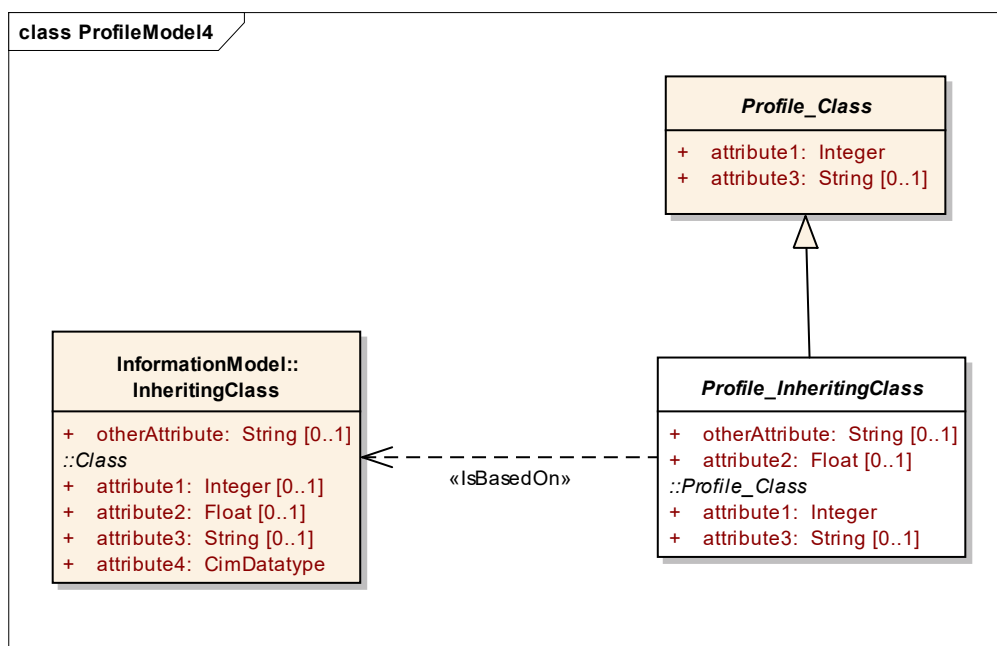
“Profile_InheritingClass” is “IsBasedOn” the Information Model “InheritingClass”. So, potentially it could have all the attributes of “InheritingClass”:

- “otherAttribute” (intrinsic “InheritingClass” attribute),
- “attribute1” (“InheritingClass” attribute that was inherited from Information Model “Class”),
- “attribute2” (InheritingClass attribute that was inherited from Information Model “Class”),
- “attribute3” (InheritingClass attribute that was inherited from Information Model “Class”),
- “attribute4” (InheritingClass attribute that was inherited from Information Model “Class”),

If “Profile_InheritingClass” inherits from “Profile_Class” that has two attributes “attribute1” and “attribute3”, it could be able to select the remaining attributes “otherAttribute”, “attribute2” and “attribute4”. Suppose we select “otherAttribute” and “attribute2”, then “Profile_InheritingClass” will have four attributes: two coming through inheritance and two selected from the “IsbasedOn”:



Then when the “*Execute IsBasedOn*” will be processed, the EA diagram will appear like this:



17. Use Inheritance in profile for Associations

17.1. Reminder

To use inheritance for association editing in a profile, all involved profile classes, concrete and super classes must have been created first in the profile: only associations between profile classes could be edited. The rules expressed in the above section (“*Use Inheritance in profile for IsBasedOn classes*”) are a prerequisite:

- In a profile, inheritance can be used, but in this case **all super classes MUST be ABSTRACT (if not agreed differently by the design of the profile)**,
- In a profile, inheritance can be used between profile classes ONLY if these classes are “*IsBasedOn*” Information Model classes that have themselves an inheritance relationship,
- In a profile, inheritance could be recursive, in this case all super classes are abstract and only the last class that inherits is concrete.

17.2. Create “Associations” for an “*IsBasedOn*” class that inherits from another “*IsBasedOn*” class

The use of inheritance between two profile classes implies good understanding of what it means for profile association's creation, as explained in annex D “*Profiles rules for inheritance*” and especially in its part “*Profile level association inheritance rules*”. At a profile class level, we have two kind of associations:

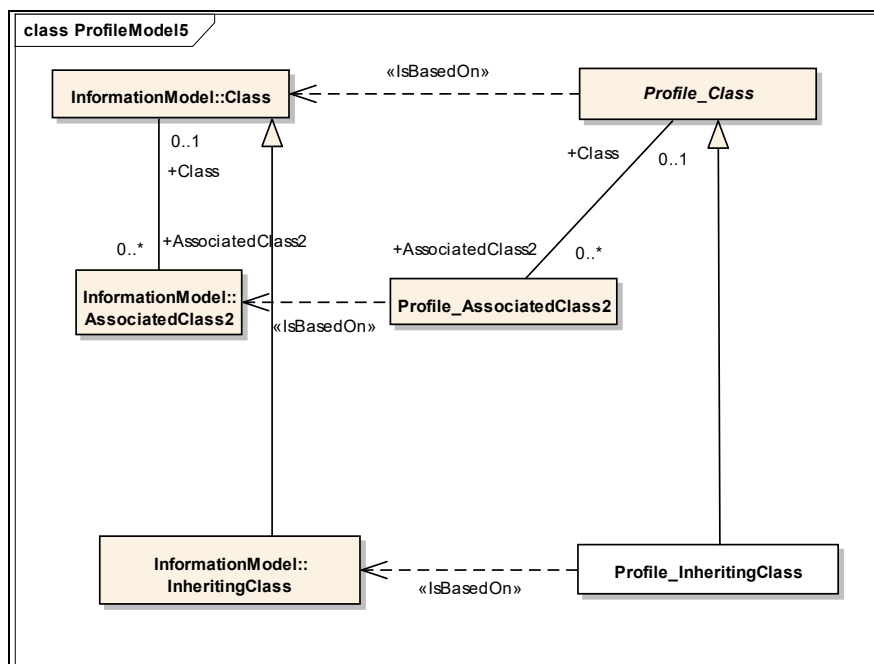
- the associations that are inherited from an abstract profile class,
- the associations that are “*IsBasedOn*”.

Let's take an example, where we have:

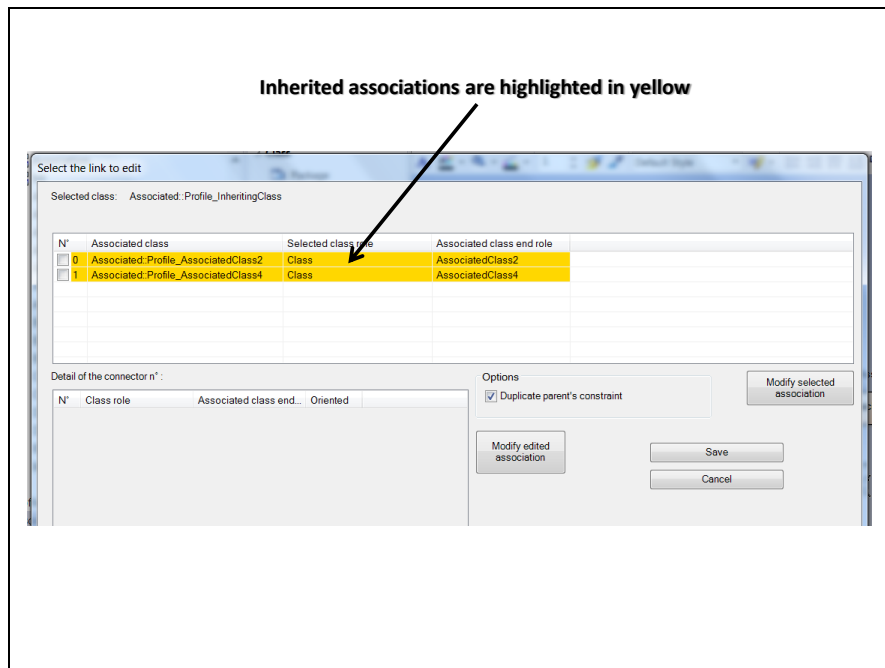
- An abstract profile class named “*Profile_Class*” that have an association with another profile class named “*Profile_AssociatedClass2*”
- A profile class named “*Profile_InheritingClass*” that inherits from “*Profile_Class*”.

“*Profile_InheritingClass*”

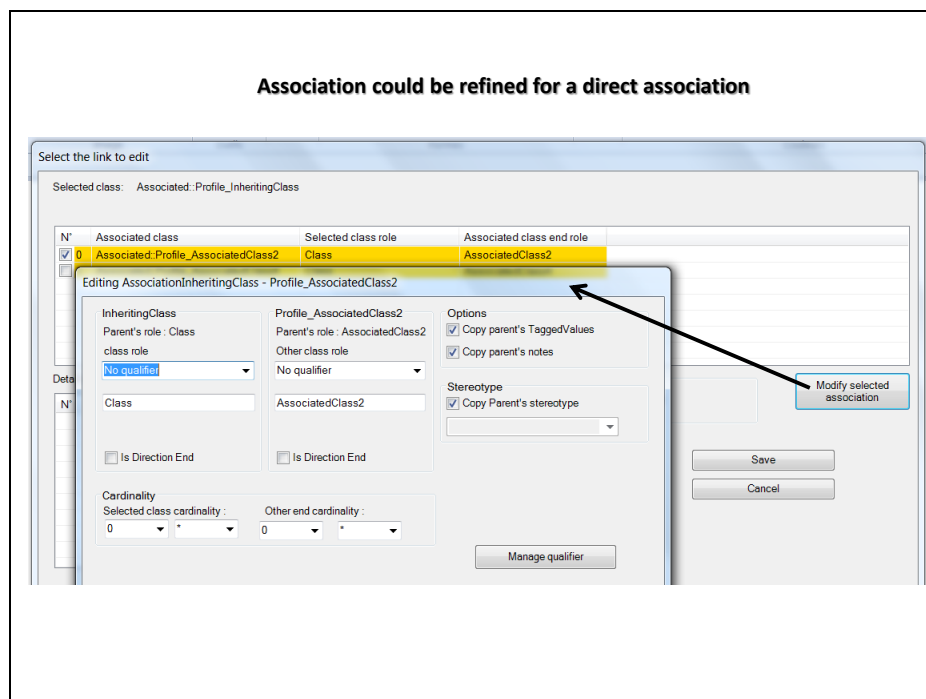
- inherits the association with “*Profile_AssociatedClass2*”, this association has two end role names “*Class*” and “*AssociatedClass2*”,
- but potentially could also have direct associations with “*Profile_AssociatedClass2*”. But in this case, the end role names MUST be qualified.



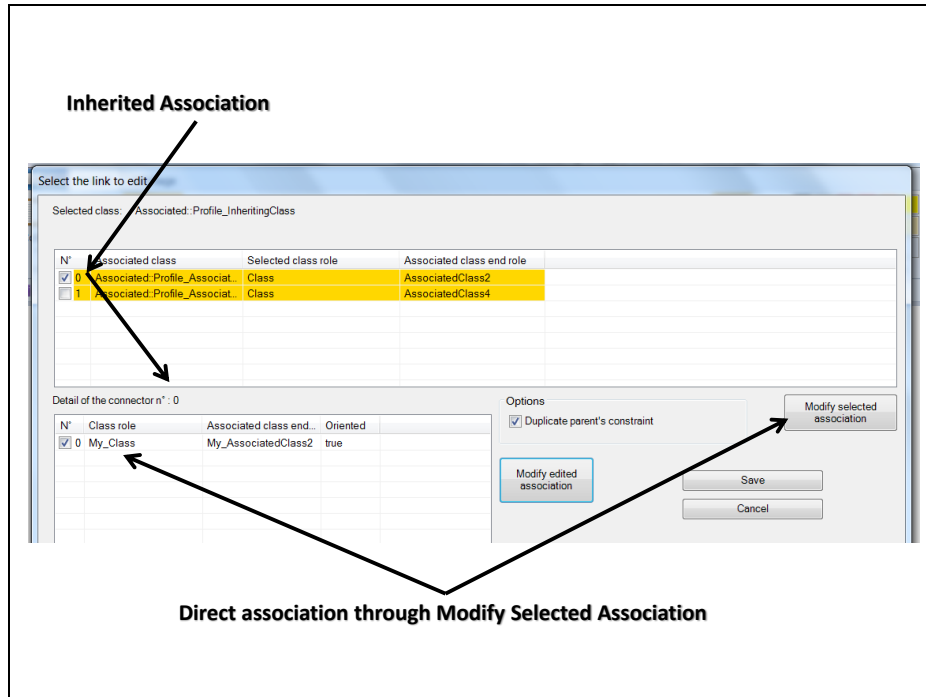
Selecting “*Profile_InheritingClass*” and selecting “*Edit connectors*” will display the “*Edit connectors*” window. But this time, a new information will be shown: inherited associations that are highlighted in yellow colour:



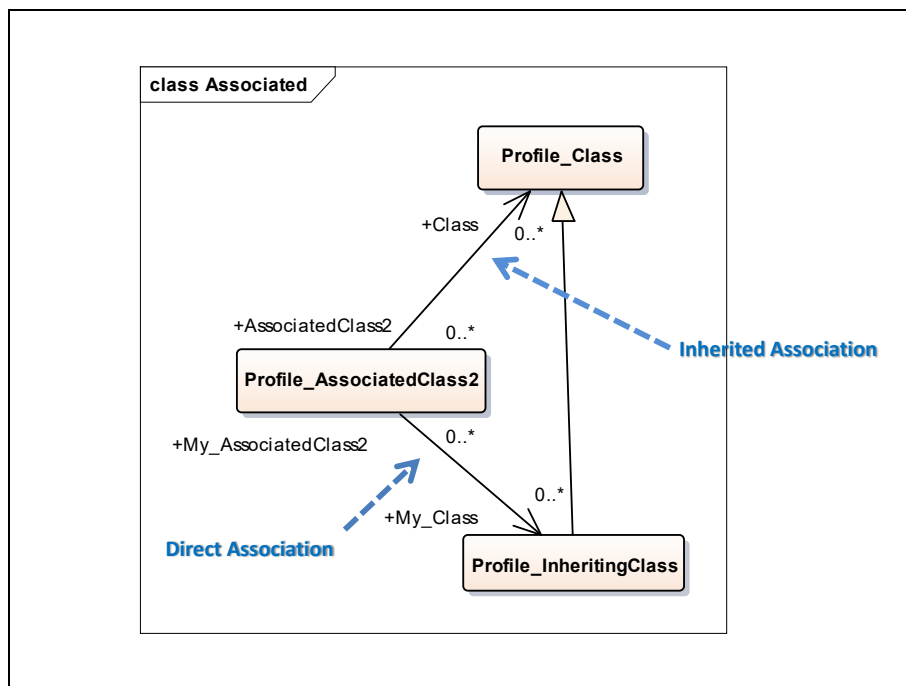
If we want to have also a direct association between “*Profile_InheritingClass*” and “*Profile_AssociatedClass2*”, we must use the “*Modify Selected Association*” feature because we need to qualify end role names.



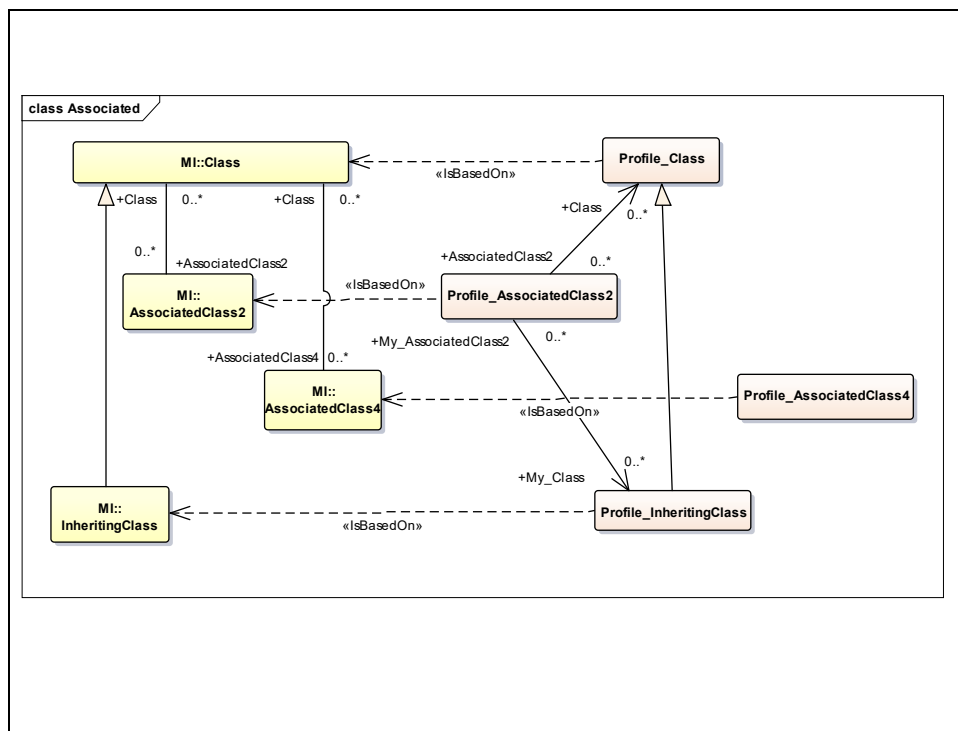
Then save will display the modified association in the “Detail Of Connector” of the “Edit connectors” window:



Click on “Save” will show association in the diagram:



Let's continue with another example:



Here, there is a new profile class named “*Profile_AssociatedClass4*”. There could be an association with “*Profile_InheritingClass*”. How it will be shown?

The association between “*Profile_InheritingClass*” and “*Profile_AssociatedClass2*” is inherited so it is highlighted in yellow and there is also a potential direct “*IsBasedOn*” association between “*Profile_InheritingClass*” and “*Profile_AssociatedClass4*”

**Direct association between
Profile Inheriting_Class and
Profile_AssociatedClass2
Checked box**

**Inherited association between
Profile Inheriting_Class and
Profile_AssociatedClass2
Yellow highlight**

Select the link to edit

N°	Associated class	Selected class role	Associated class end role
☒ 0	Associated: Profile_AssociatedClass2	Class	AssociatedClass2
☐ 1	Associated: Profile_AssociatedClass4	Class	AssociatedClass4

Detail of the connector n°:

N°	Class role	Associated class end...	Oriented

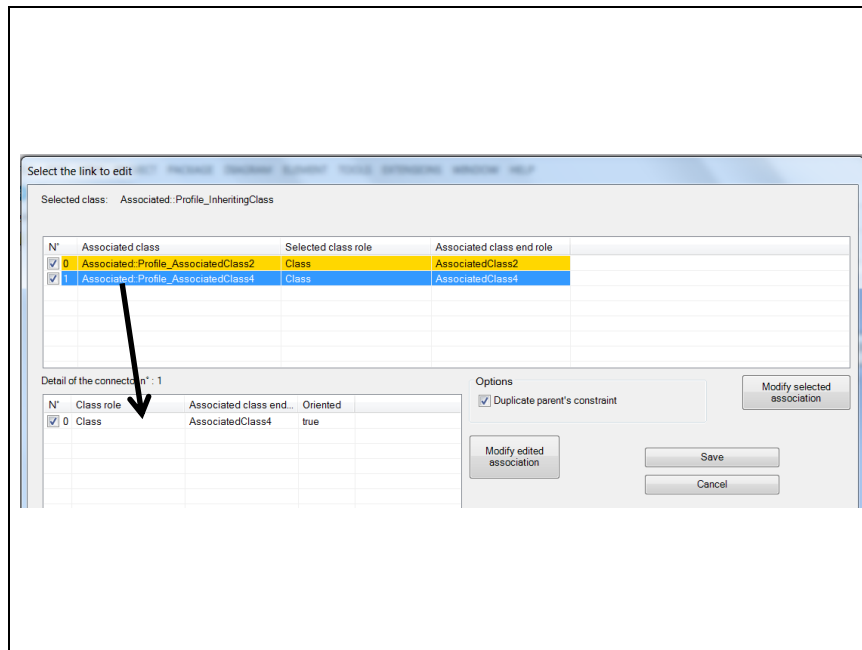
Options

☒ Duplicate parent's constraint

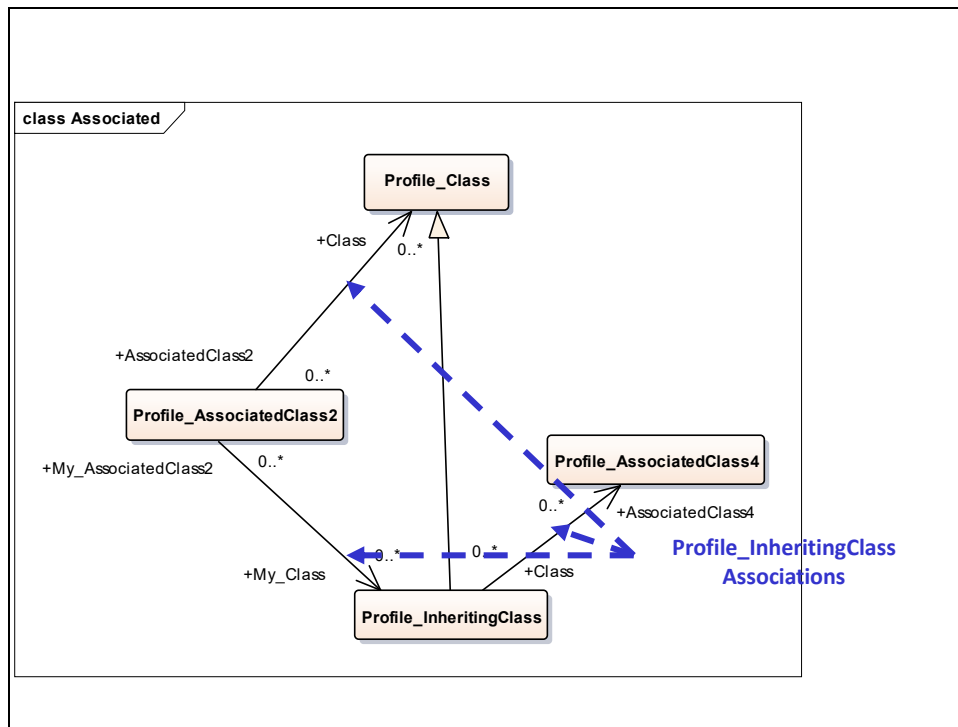
Buttons: Modify selected association, Modify edited association, Save, Cancel

**Possible association between
Profile Inheriting_Class and
Profile_AssociatedClass4**

This association could be duplicated directly, or it could be modified if needed:

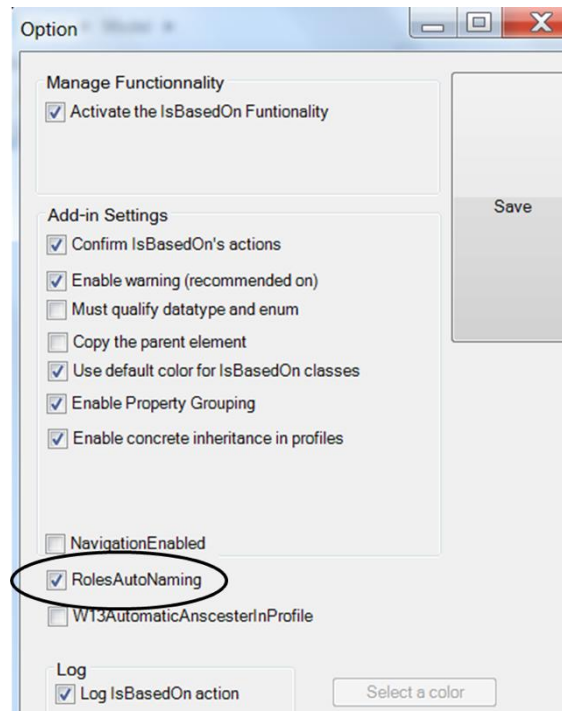


Click on “Save” will show associations in the diagram:



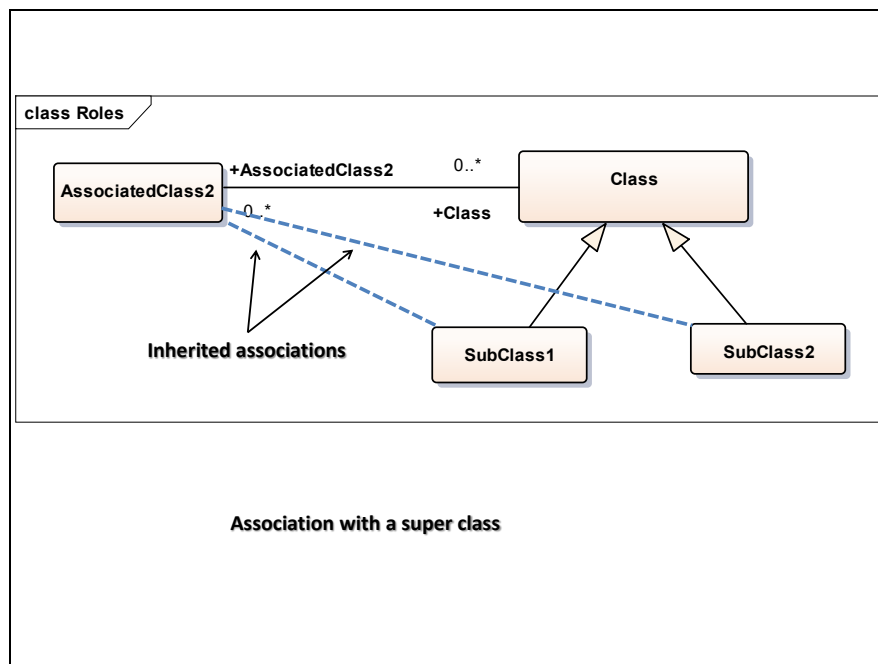
17.3.RolesAutoNaming

This option is used with the "EditHierarchicalConnectors" Menu. To allow this option, the "RolesAutoNaming" box of the CimConteXtor Option Menu must be checked.

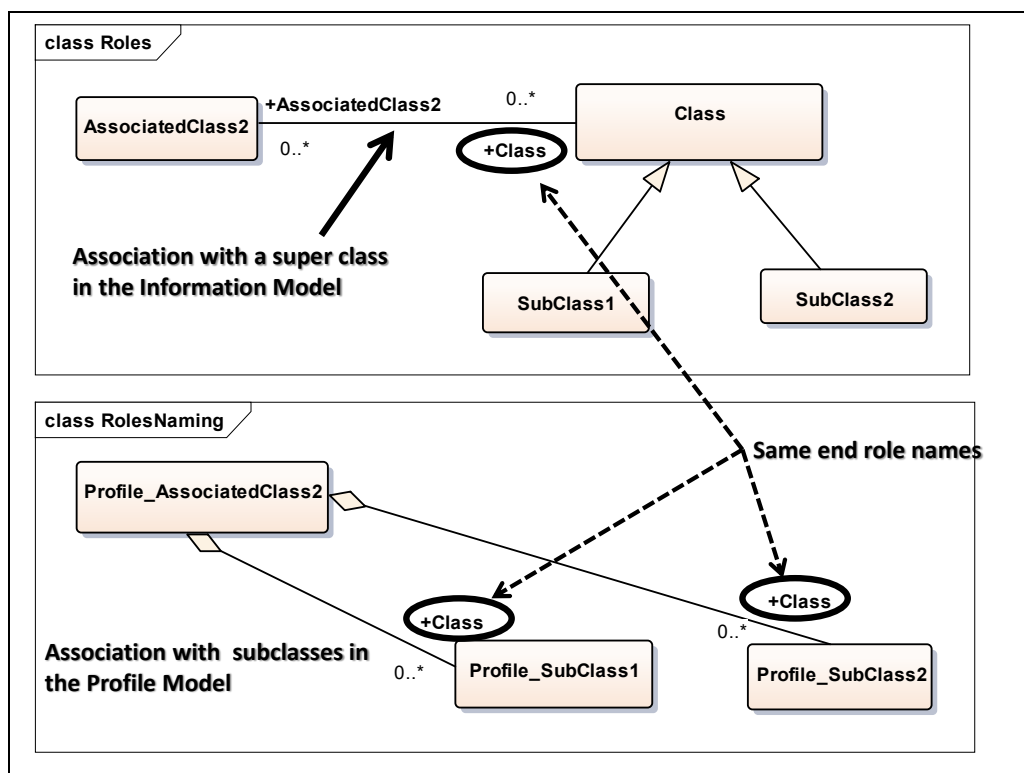


When this option is useful? Let's take an example.

If we have a class (*AssociatedClass2*) that has an association with another class (*Class*) that acts as a superclass for classes (*SubClass 1 and 2*), this means that because of the inheritance, *AssociatedClass2* is also associated with *SubClass1* and *SubClass2*.



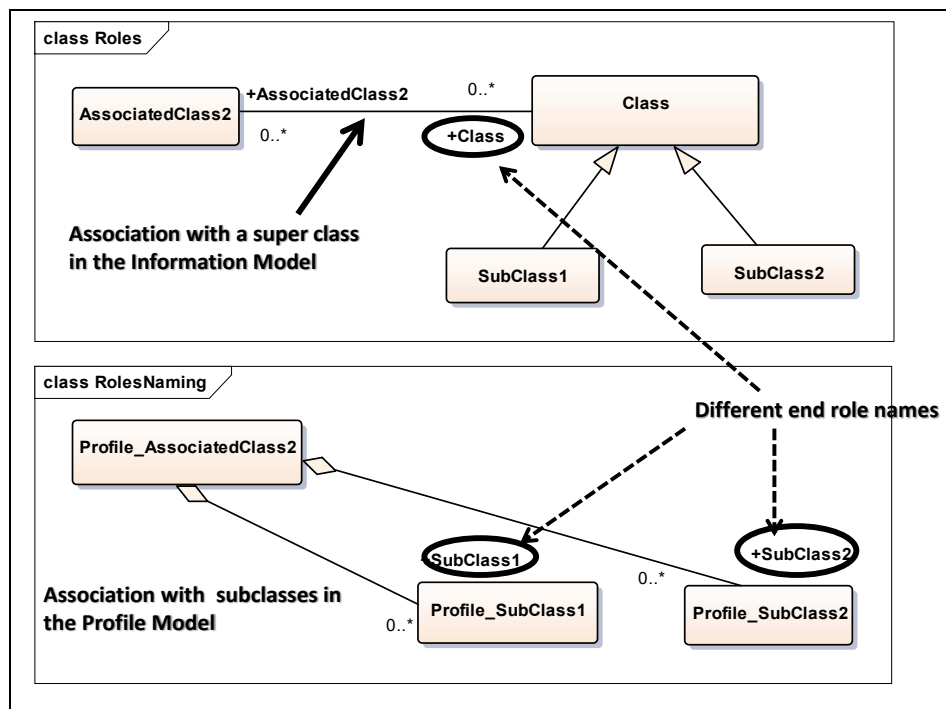
When doing a profile, in a hierarchy mode, the associations selected could be between the associated class and the subclasses as in the following diagram. Note that in this case, in the profile, the end role name on the side of the subclasses will be "class" as specified in the information model.



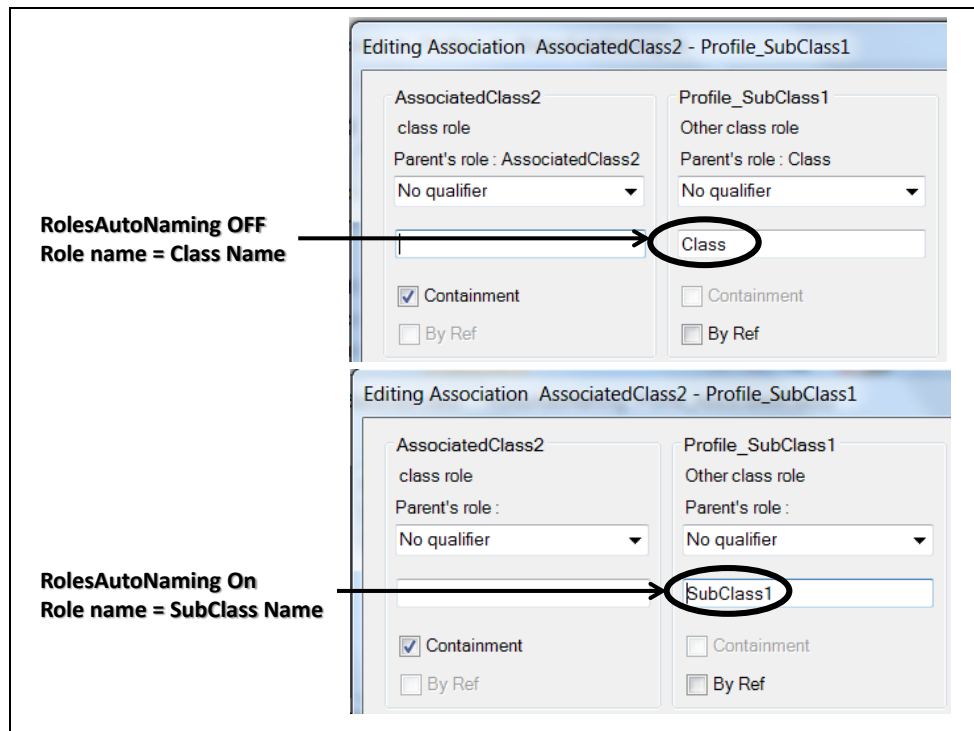
There are two drawbacks in this design:

- When using a superclass, it is to mutualize properties. But in fact, do we want to say that *AssociatedClass* has a "class" property (end role name) with *Class* and *SubClass*, or do we want to say that *AssociatedClass* has a "class" property with *Class* and a "subclass" property with *SubClass*? So, depending on the interpretation, we could take one or the other.
- When targeting a hierarchy profile, where the intended mapping is XSD, having same end role name with two different classes (*SubClass1* and *SubClass2* will lead to a wrong XSD. That is why in IEC 62321-100, which specifies how the mapping is done, there is a changing name rule. But making the change at mapping level results in having different names in UML and XSD.

So, the *RolesAutoNaming* provide a way, if needed, to be able to change end role names at UML profile level.

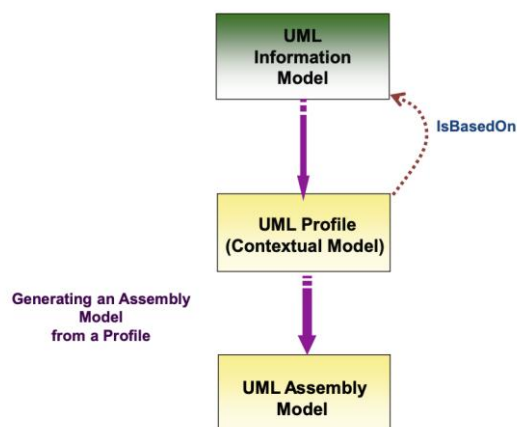


When using "*RolesAutoNaming*", changed name will appear in the *EditingAssociation* windows:



18.Assembly Model

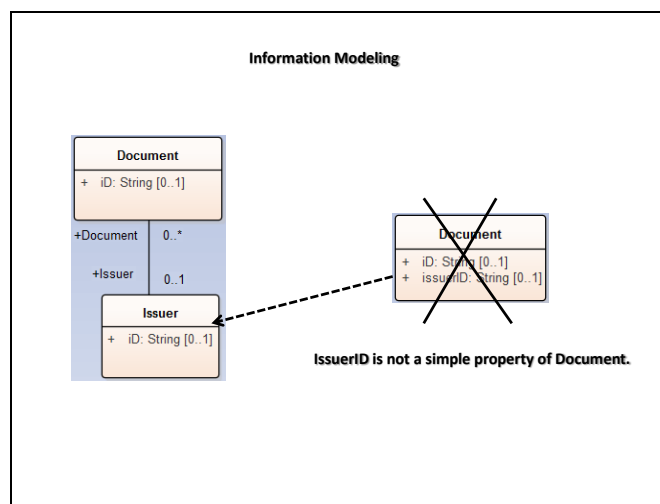
18.1.Overview



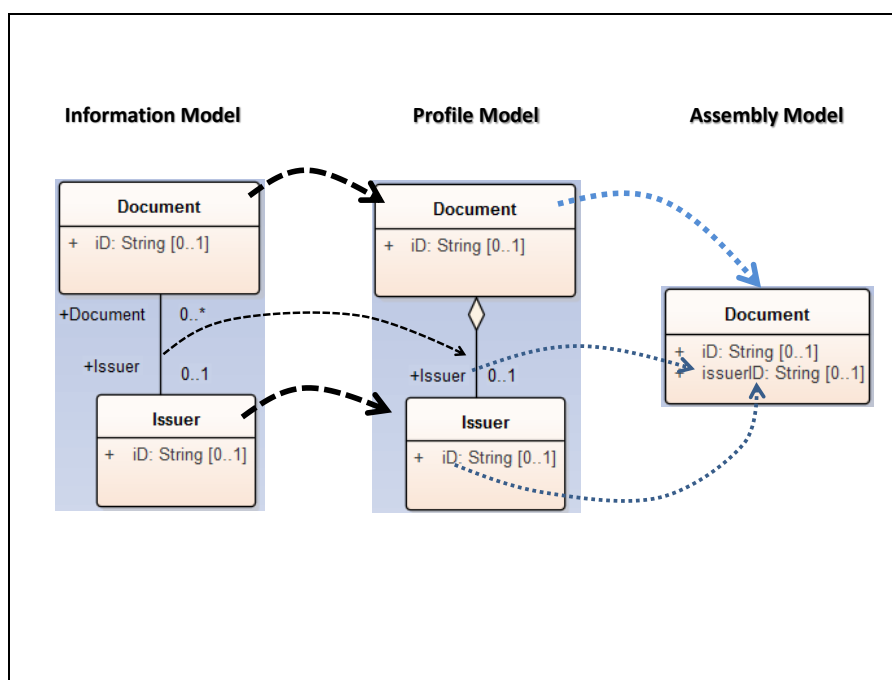
An Assembly model has several purposes:

- To add to the profile model a header,
- To realize "Property Grouping".

Here we just define how to generate an Assembly model that is used to specify the "*Property Grouping*". In the design of information model, the rule is to have in each class, only the properties that belong to this class. Example:



But for different reasons, at profile level, requirements have been made to regroup properties so that a class could have, as simple properties, properties that are coming from other classes. Here is an example:

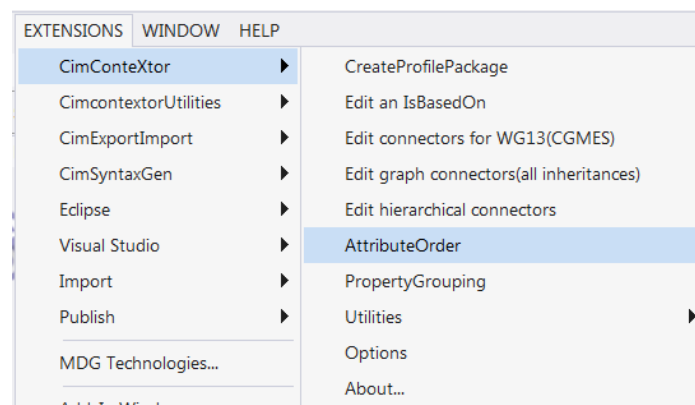


In this example, at the information model level we have two classes (Document and Issuer). At hierarchical profile level, Document is the upper hierarchy class and Issuer a leaf class. The assembly level is going to group properties in one class. There are rules for enabling property grouping between classes:

- Properties of a leaf class could be aggregated in an upper class only if the association multiplicity is 0..1 or 1 (no property grouping if multiplicity 0 or 1..n).
- The name of the aggregated property is "*Issuer.iD*", concatenation of:
 - Name of the end role name (here Issuer),
 - A dot,
 - Name of the attribute (here iD).
- Property grouping could be iterating.

18.2.Attribute Order for Property Grouping

Before doing the property grouping itself, one need to define in which order the items (attribute and associated class attribute) part of the grouping will be grouped. This is done by using "AttributeOrder" of the CimConteXtor Menu:



The attribute order feature is to be applied to each class of the profile. The result will generate a tagged value called "*ESMPRG*" for each attribute and association.

To order the attributes and the associations in the UML profile class diagram, the following steps are to be made:

1. Select one of the classes to be ordered.

Note: it is highly recommended to start from the "leaf" classes up to the root class.

2. Right click, select "Specialize", then "CimConteXtor" and "AttributeOrder".
3. A dialog box will appear:

Attribute order for Class

Items to be ordered for potential property grouping:

- **Class attributes: attribute1 and attribute2**
- **AssociatedClass2 attributes: firstA and secondA (because AssociatedClass2 end role multiplicity is 0..1)**
- **AssociatedClass4 will not be grouped because its end role multiplicity is 0..n**

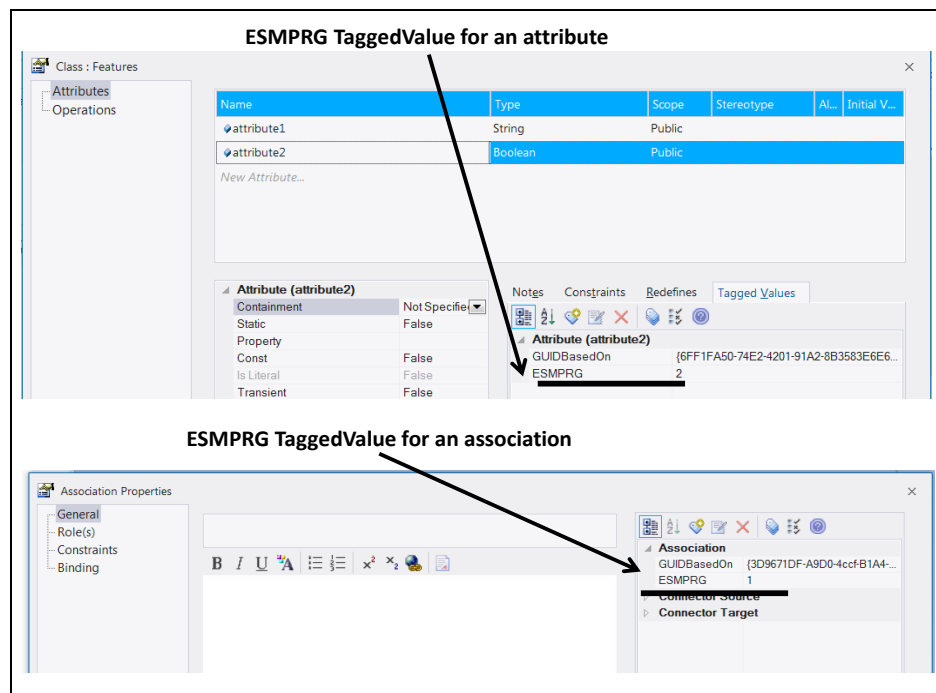
4. Use the “up” or “down” button to order each attribute of the class (element type “A”) or the association end role name for the associated classes (element type “C”). It is recommended that the associations of multiplicity 0..* be put at the end of the list of attributes, because they will not be grouped.
5. When the order is fine, click on “ok” to end the dialog and save the configuration.

Attribute order for Class

Using up or down button, properties have been ordered:

- **Class attribute1**
- **AssociatedClass2 attributes: firstA and secondA**
- **Class attribute2**
- **AssociatedClass4 will not be grouped because its end role multiplicity is 0..n**

With the change made, the attributes in a class will respect the ESMPRG tagged value.



In case the attributes have not been ordered, there will be an error message at the following step, i.e. PropertyGrouping.

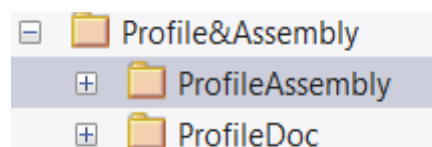
18.3. Property Grouping

The “*PropertyGrouping*” option is to be used to generate from a property model the associated assembly model.

All the classes associated with a multiplicity of 0..1 or 1..1 to a class are inserted within this latter class, and the order is provided by the “*AttributeOrder*” operation.

To generate the assembly model, the following steps are to be made:

1. Define an Assembly model package in the browser. This package must be inside a package where the Profile (here ProfileDoc) Model is defined:



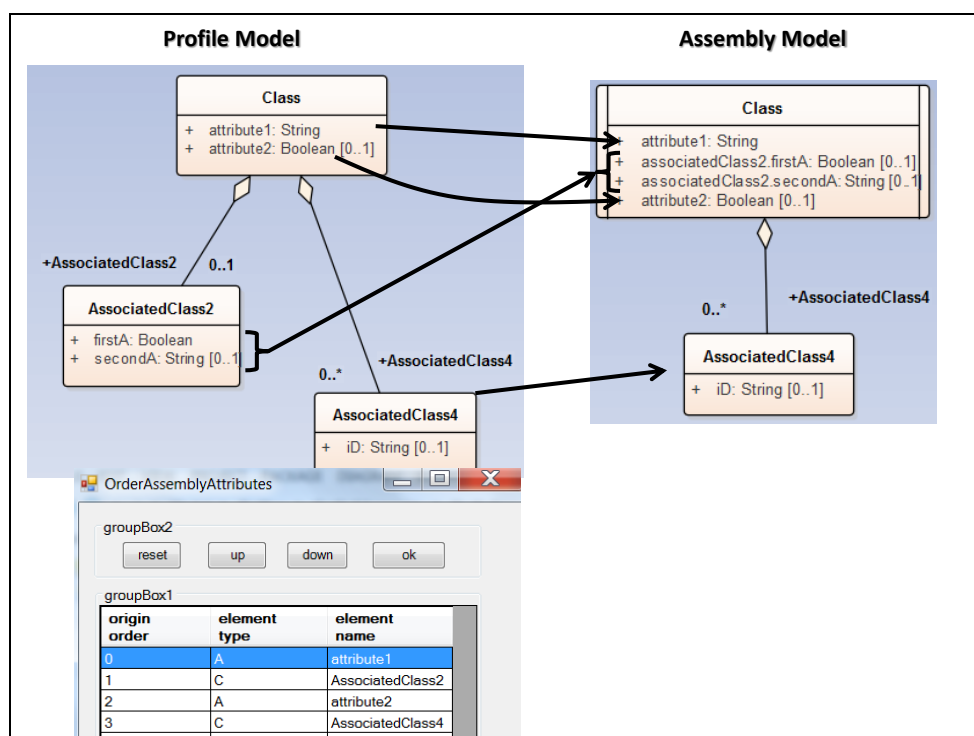
2. In the package browser, select the assembly package to be generated.

Note: if the assembly package is not empty, all its content will be deleted.

3. Right click, select “Extension”, then “CimConteXtor” and “PropertyGrouping”.

The generation process starts, and a confirmation of the package to be grouped will be asked; click “Ok” when requested.

At the end of the process, the assembly diagram is displayed.



It is recommended to “arrange” the display (position of the classes, position of the association, etc.) to enhance the visual aspect. It is also recommended to check that all the attributes are in the appropriate order; otherwise, the steps described are to be resumed.

Note: the process will always put the assembly package before the contextual package. You will have to change this order before generating documentation.

18.4.Attribute order for syntax generation

At this stage an "OrderAttribute" must be now done on the Assembly Model Classes to define the XSD mapping.

19.Utilities Menu

"Utilities" Menu offers several features to check if profiles are correct according to profiling rules and to help profile building.

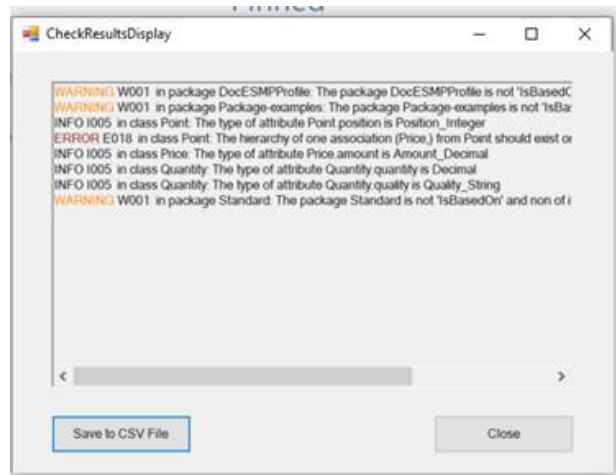
19.1.Check profiling

Check features of the "Utilities" menu are used to verify if a UML model follows rules for UML modelling. Currently, depending on the modelling context, there are three different checks:

- General integrity check,

- Integrity check for European Style Market Profile (ESMP),
- Integrity check for CGMES profiles.

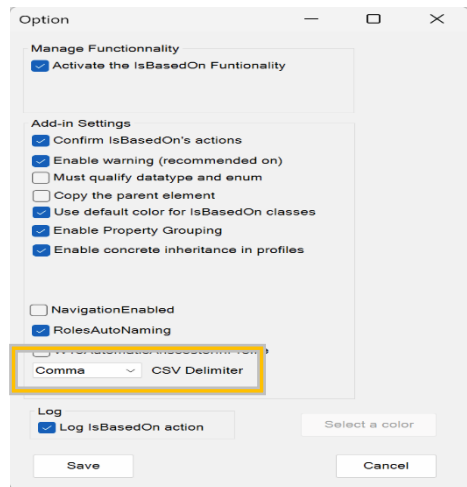
The results of the General and ESMP integrity checks are shown in a result window:



whose content can be exported into an Excel Sheet (CSV file):

	A	B	C	D	E	F
1	General Integrity Check					
2	Severity	Code	Message	Package	Class	Attribute
3	WARNING	W001	The package DocESMPProfile is not 'IsBasedOn' and non of its parent packages!			
4	WARNING	W001	The package Package-examples is not 'IsBasedOn' and non of its parent packages!			
5	INFO	I005	The type of attribute Point.position is Position_Integer			
6	ERROR	E018	The hierarchy of one association (Price) from Point should exist or reflect upperlevel hierarchy!			
7	INFO	I005	The type of attribute Price.amount is Amount_Decimal			
8	INFO	I005	The type of attribute Quantity.quantity is Decimal			
9	INFO	I005	The type of attribute Quantity.quality is Quality_String			
10	WARNING	W001	The package Standard is not 'IsBasedOn' and non of its parent packages!			
11						
12						

The delimiter can be defined by a listBox in the Option window. The default is the comma:



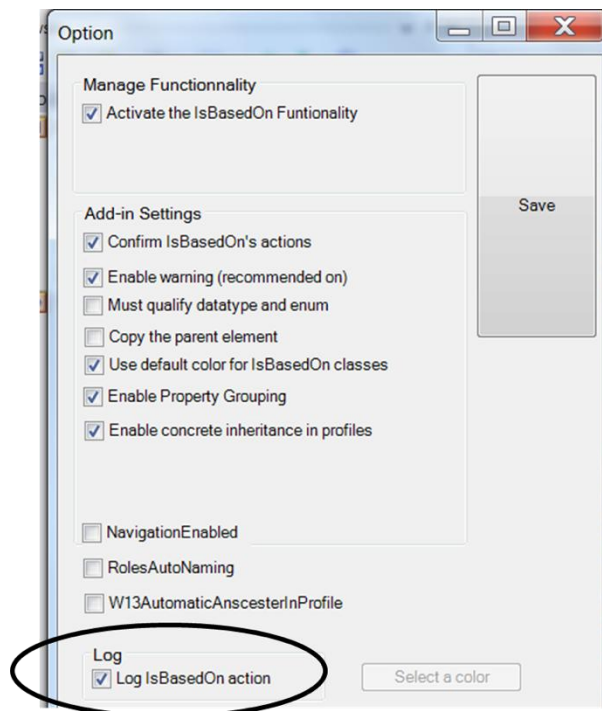
19.2.IntegrityCheck

IntegrityCheck checks the possible errors in editing a new profile. It checks the basic errors, without specific rules used by different profiling styles (CGMES, ESMP...). The result can be exported as CSV file (the delimiter is configurable via the Options window, see above).

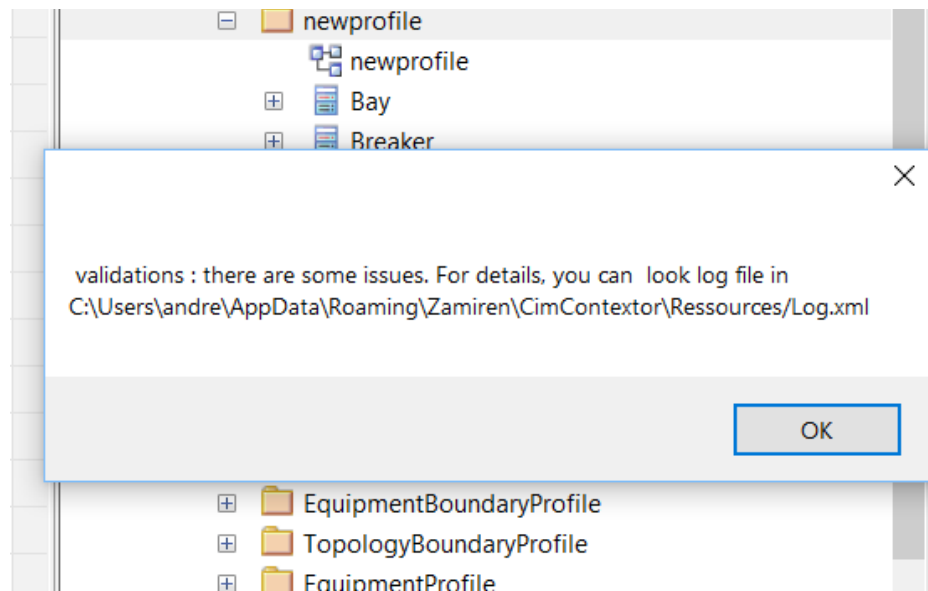
The errors are stored in the log file which is in

`"Users\<user name>\AppData\Roaming\ENTSO-E\CimConteXtor\Resources\Log.xml"`

Do not forget to select the Log option in the menu options:



The location of the log file is displayed when the checking is achieved in case there are errors for instance.



Here is an example of an excerpt of the results:

```
<add-inLogs>

  <logDate date="11-21-2018">

    <log time="23:37:56" username="ANDREDELL2\andre" ConfigSaved="Element:EnableESMPHierarchy Value:Unchecked" />

    <log time="23:37:56" username="ANDREDELL2\andre" IntegrityCheck=" info:  ++++++  the package newprofile is beeing
checked for consistency ++++++" />
```

```
<log time="23:37:56" username="ANDREDELL2\andre" IntegrityCheck=" ***** entree dans verf_element
Bay*****" />

<log time="23:38:06" username="ANDREDELL2\andre" IntegrityCheck=" issue: get IBO element for Bay La reference d'objet
n'est pas definie Ã une instance d'un objet." />

<log time="23:38:06" username="ANDREDELL2\andre" IntegrityCheck=" ***** entree dans verf_element
Breaker*****" />

<log time="23:38:06" username="ANDREDELL2\andre" IntegrityCheck="info: the type of attribute Breaker.normalOpen is
Boolean" />

<log time="23:38:07" username="ANDREDELL2\andre" IntegrityCheck="info: the type of attribute Breaker.open is Boolean" />

<log time="23:38:07" username="ANDREDELL2\andre" IntegrityCheck=" ***** entree dans verf_element
ConductingEquipment*****" />

<log time="23:38:07" username="ANDREDELL2\andre" IntegrityCheck="info: the type of attribute
ConductingEquipment.aggregate is Boolean" />

<log time="23:38:07" username="ANDREDELL2\andre" IntegrityCheck="info: the type of attribute ConductingEquipment.mRID is
String" />

<log time="23:38:07" username="ANDREDELL2\andre" IntegrityCheck=" ***** entree dans verf_element
EquipmentContainer*****" />
```

The syntax of a log line is:

```
<log time=xxx username=yyy IntegrityCheck= "a line of information"
```

A line of information begins with either:

- "Info:", if it is an info
- "issue:", if it is an issue

Each checking of an element begins with:

```
<log time=xxx username="yyy" IntegrityCheck=" ***** entree dans verf_element zzz*****" />
```

The text after "info:" or "issue:" is self-explanatory. For instance, in the following example:

```
IntegrityCheck=issue: get IBO element for Bay La reference d'objet n'est pas definie Ã une instance d'un objet."
```

The text which comes from a code exception in local language means that the object is not based on an Information Model object.

It is impossible to tell if the type of an object is correct, so this type is printed as info line and may be checked visually:

```
<log time="23:38:07" username="ANDREDELL2\andre" IntegrityCheck="info: the type of attribute
ConductingEquipment.mRID is String" />
```

See H for the list of Messages

19.3.ESMPIegrityCheck

This program does all the checks of “IntegrityCheck” while adding some additional checking adapted to the rules followed by European Style Market profiles (see IEC62325-450).

Some of these rules are:

- The existence of a regional contextual profile (IEC62323-351) from which all the other profiles are derived,
- All association must have an end role of type “shared”,
- All classes have an additional stereotype “ABIE”,
- All identifiers of the type of an attribute must be defined in a special datatype domain shared by all profiles and different from the CIM,

The location and the syntax of the log file are identical as the “IntegrityCheck”'s:

- either in "User/AppData/Roaming/Zamiren/CimConteXtor/Ressources/Log.xml"
- or in the same folder as the EA project, if there is a local config file in that folder.

See Annex I for the list of Messages

19.4.CGMESIntegrityCheck

This program checks the profile (graph) according to the practice of CGMES. In the log file, programming or system issues (like access to files) and not profiling issues may be specified.

The results of checking the profile can be found at the same location as the log file under the name "CheckProfileErrorFiles.csv". This is a spreadsheet with the following columns:

Message:a self-explanatory message concerning a found issue,

Package:the checked package,

Class:the checked class,

Attribute:the checked attribute,

Association:the checked association,

CIMText:content on the CIM side,

ProfileText:content on the profile side.

Here is an example of the resulting file:

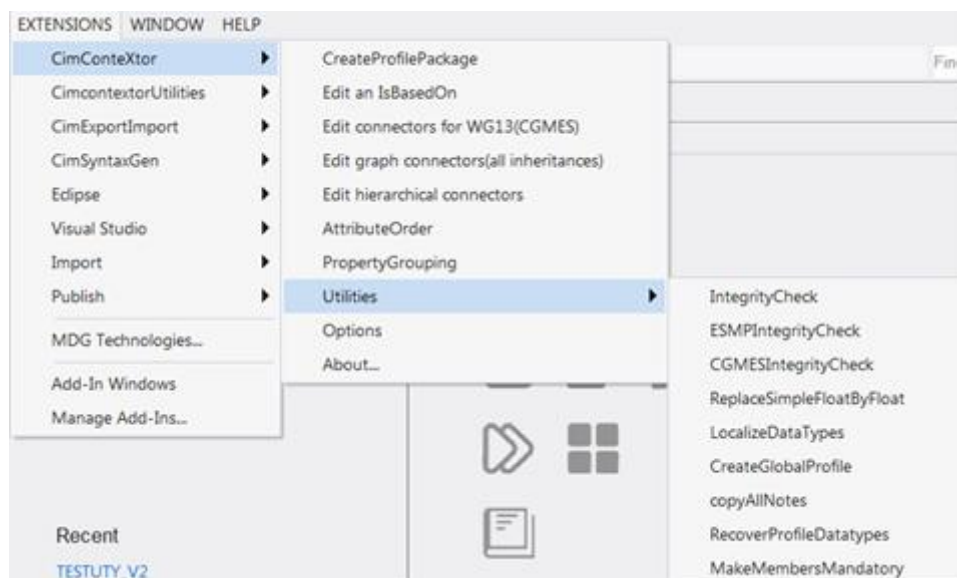
Message	Package	Class	Attribute	Association	CIMText	ProfileText
Info	Check the Profile Graph2Profile	CIMVersion=62325CIM03v01	date=2018-10-10	Profile Version=Euml	from CIMVersion=Buml	
HasNote	Graph2Profile					Beginning of the note ="mon profil
NotSameNote	Graph2Profile	A1				79
HasAttachedNote	Graph2Profile	A1				Beginning of the note ="note de classe
NotSameCardinality	Graph2Profile	A1	at1		for CIM attribute LowerBound=0	LowerBound=1
NotNaturalDirection	Graph2Profile	A1		b2 -> a3	b2[0..1]	a3[0..1]
NotSameNote	Graph2Profile	A3				77
NotSameCardinality	Graph2Profile	A3		a3 -> b2	b2[0..1]	a3[0..1]
NotNaturalDirection	Graph2Profile	A3		a3 -> b2	b2[0..1]	a3[0..1]
NotNaturalDirection	Graph2Profile	A3		b2 -> a3	b2[0..1]	a3[0..1]
NotConsistantEnumerationValues	Graph2Profile	ActivePower	multiplier		for CIM initial value=	for profilemultiplier initial value=22
NotSameNote	Graph2Profile	B1				83
NotSameNote	Graph2Profile	B2				82
NotSameAbstract	Graph2Profile	B2			Abstract=0	Abstract=1
HasNonAttachedNote	Graph2Profile					Beginning of the note ="note dissociée

See H for the list of Messages

19.5.Profile building helps

Utilities offers many functions to help profiles building:

- *"ReplaceSimpleFloatByFloat"* enables the change of CIMDatatype called *"Simple_Float"* for a *"Float"* Primitive in CGMES older profiles,
- *"LocalizeDataTypes"* allows older version of CGMES profiles to have their own specific domain package, instead of sharing a common one,
- *"CreateGlobalProfile"* enables the creation of a profile from an information model package or a diagram, (this is different from *"CreateProfile"* feature described in section [CreateProfile](#)),
- *"CopyAllNotes"* is a function that replaces all profile element description (notes) by the Information Model element description (useful in case of a new CIM version),
- *"RecoverProfileDatatypes"* checks if datatypes used as profile attribute type are profile datatypes or not. When it is not the case, it corrects the profile and if needed create the profile datatypes needed,
- *"MakeMembersMandatory"* makes every attribute in a profile class mandatory.



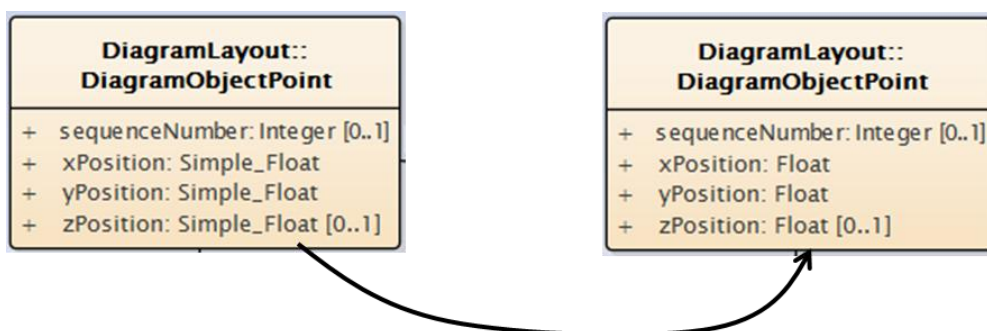
19.6. Use of "ReplaceSimpleFloatByFloat"

CIMDatatype "Simple_Float" was used to specify the value space of a single precision floating point number, while Primitive "Float" in CIM is specifying the value space of a floating-point number (simple or double precision...).

CIM wants to change the definition of Float to be a simple precision value space.

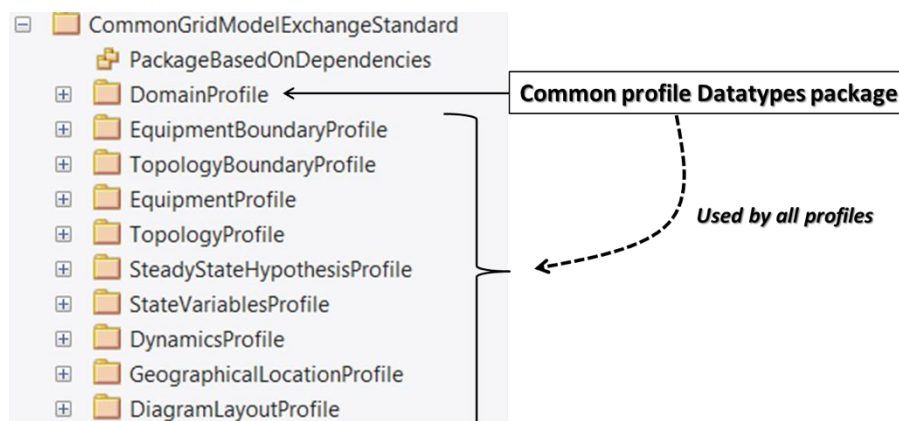
To keep existing profiles with this new definition, this utility feature replaces Simple_Float datatype by Float in a given profile package.

Select a profile package, launch the ReplaceSimpleFloatByFloat, the attribute datatype will be changed in every class of a package.

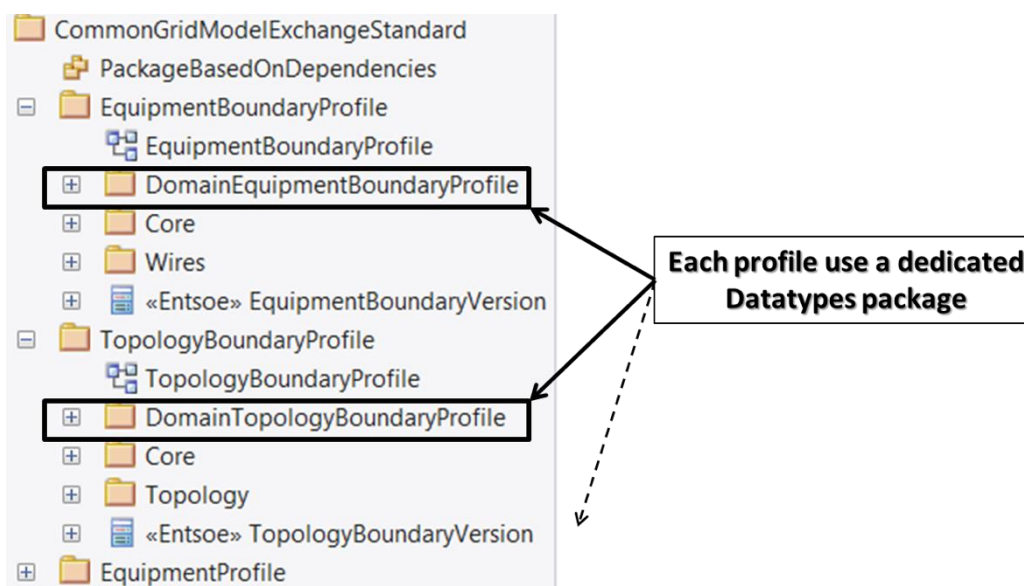


19.7. Use of LocalizeDataTypes

As stated in section **Error! Reference source not found.**, there are many ways to define the profile package structure. For example, all the profiles could share a common Domain package that holds all the profiles datatypes, this was the case for CGMES:

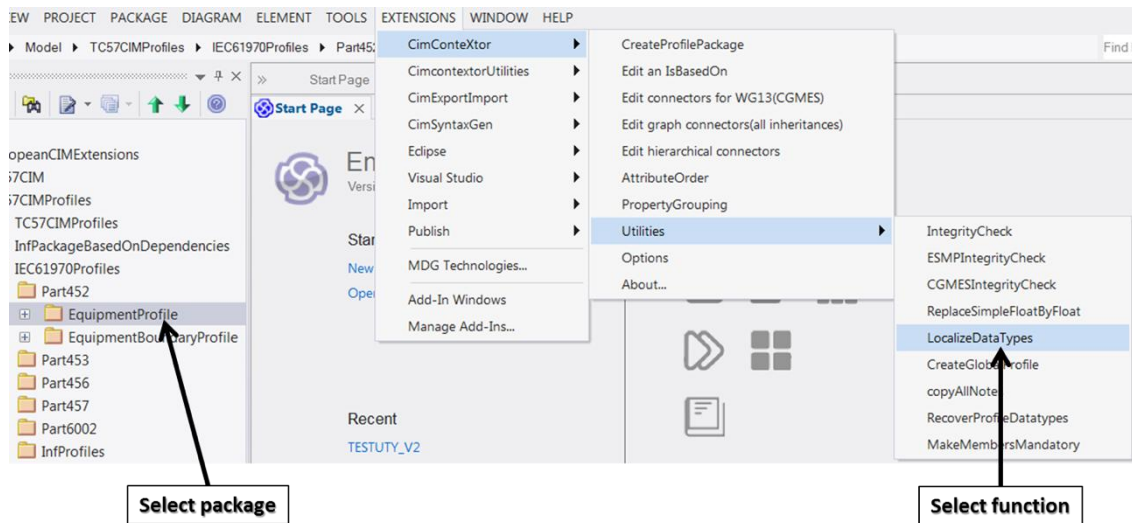


But another design could have been to have one domain package for each profile, like this:



The LocalizeDataTypes Utility feature is a mean to move from a one shared domain package to one domain package per profile.

The way to change the structure is to select a profile package, then select the "LocalizeDataTypes" feature.

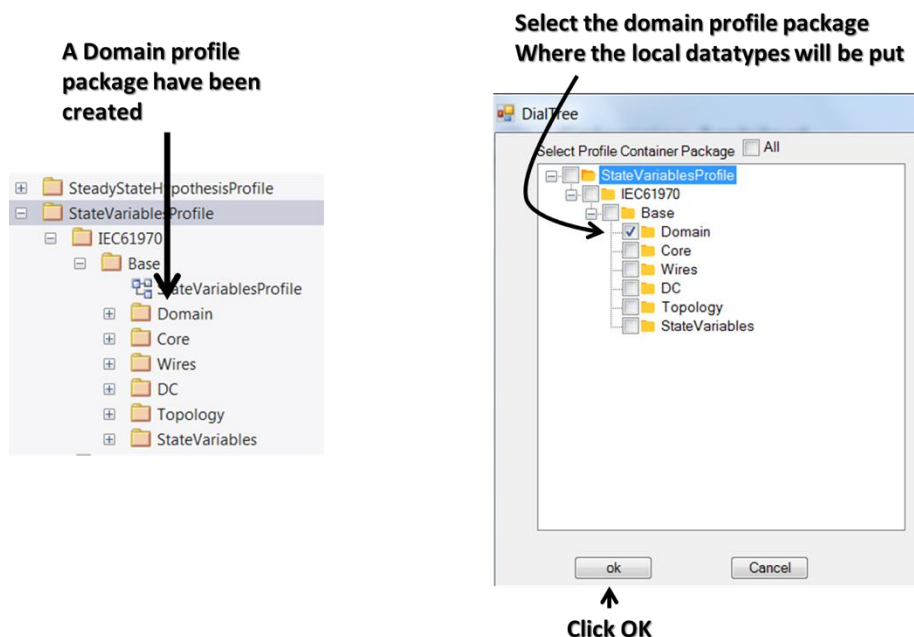


CimConteXtor will create a dedicated domain profile package and change the type of each profile class attributes to match the profile domain datatypes. But depending of the profile package structure, there are several cases:

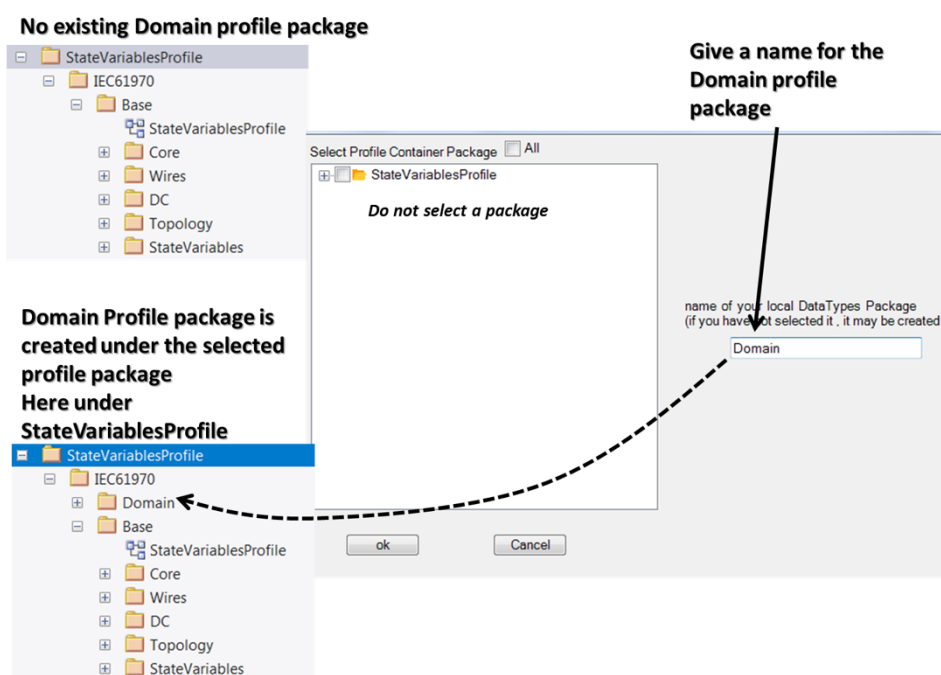
- The domain profile package has been created, with a given name, in that case you just want to design this package to have the local datatypes,
- The domain profile package has not been created:
 - Either you give its name, and the domain profile package will be created with this name under the profile selected package for the LocaliseDatatypes,
 - Or the domain profile package will be created with under the profile selected package for the LocaliseDatatypes and its name will be the name of the profile (example EquipmentBoundaryProfile) prefixed by Domain

So, when the LocaliseDatatypes is launched on a profile package, a window open:

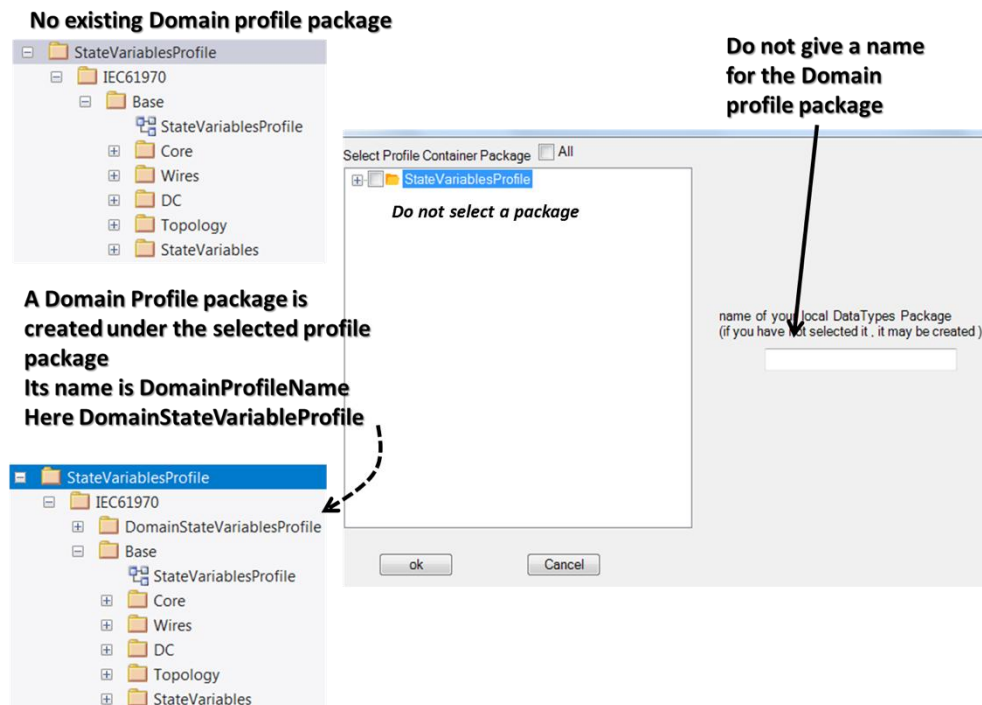
- First case, a Domain profile package exists: select this package:



- Second case: you want to give a name to the local domain profile package. Give the name in the window (do not select a package):



- Third case, you do not give a name to the local profile package, just click OK. The name of the package will be the ProfilePackage Name preceded by Domain:

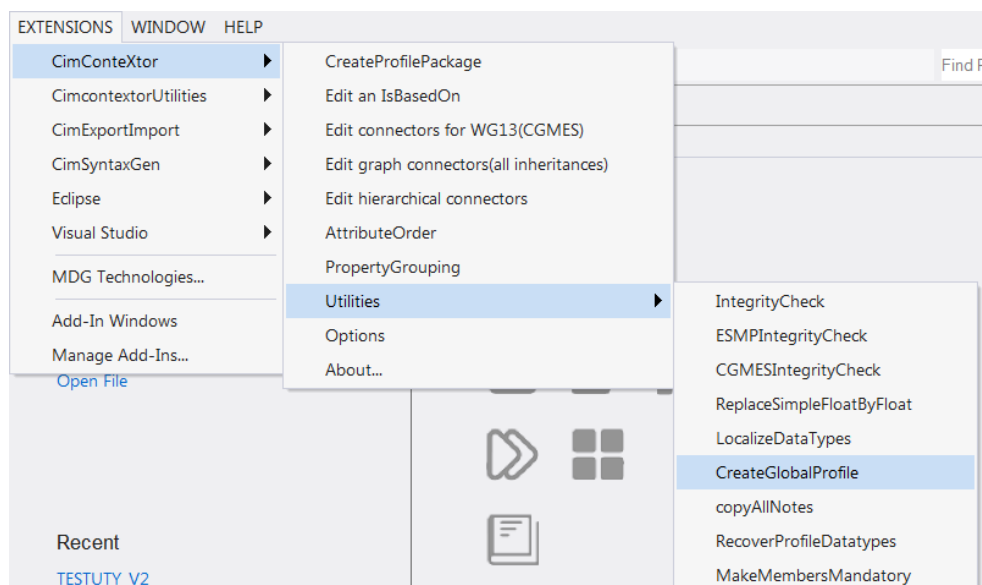


After clicking OK, a domain profile package is created if necessary, and all the local datatypes are put in this domain profile package.

19.8. Use of "CreateGlobalProfile"

"Utilities/CreateGlobalProfile" is different from the "CreateProfilePackage" (see section 4). The second one is used to create the template for the package profile, when the first one is used to create also profile classes.

The starting point for "CreateGlobalProfile" is either a package or a diagram. So, select either a package or a diagram and then launch the Utilities/CreateGlobalProfile:



19.9.CreateGlobalProfile for a package

If you have selected a package and launched the command CreateGlobalProfile, see section 28.1 [AnnexF](#).

19.10.CreateGlobalProfile for a diagram

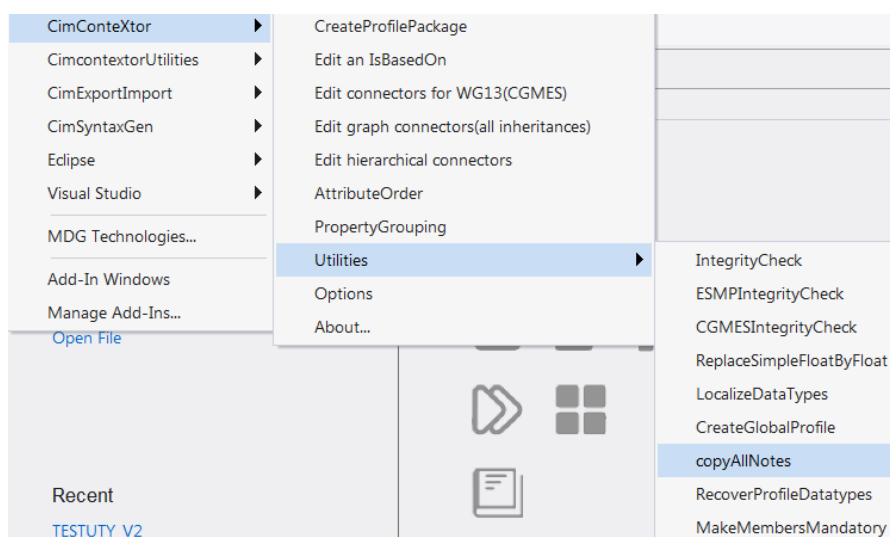
If you have selected a diagram and launched the command CreateGlobalProfile, see section 28.2 [AnnexF](#).

19.11.Use of "CopyAllNotes"

CopyAllNotes is used to update the description (notes) of profile elements (class, attribute, end role name). This happens when a new version of the information model is delivered, and where some descriptions have been changed: so, it is a way to keep profile description with information model ones.

Note: this supposed that the description in the profile must be the same as the one in the information model.

Select a profile package and launch the "*Utilities/CopyAllNotes*":

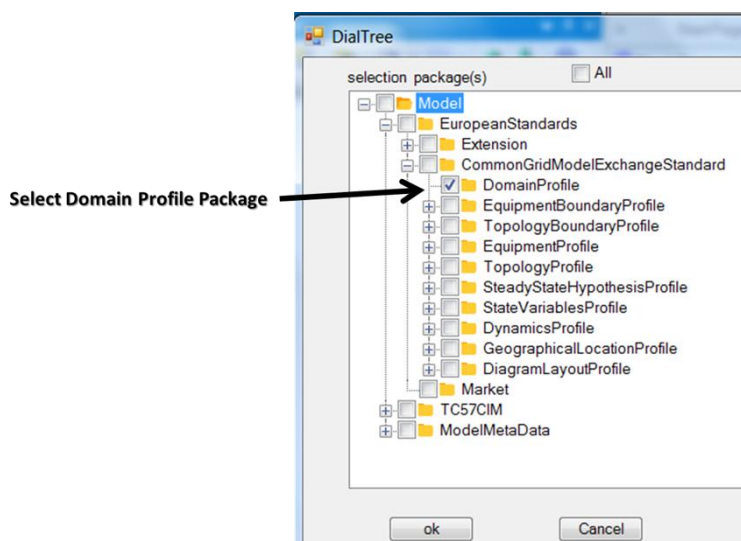


After the end of the process, the profile element descriptions have been updated.

19.12. Use of "RecoverProfileDatatypes"

"*RecoverProfileDatatypes*" is used to check if the datatype used as the type of a profile attribute is belonging to the right domain package: usually it should belong to a domain profile and not to the Information Model one. If the datatype is not pointing to the right profile package, it makes the change if the datatype exists, or it creates the datatype in the domain profile package and then it makes the change.

Select a package where you want to verify datatypes, you will be prompted by a dialog box, asking you to select the domain profile package that holds the right datatypes for the given profile:

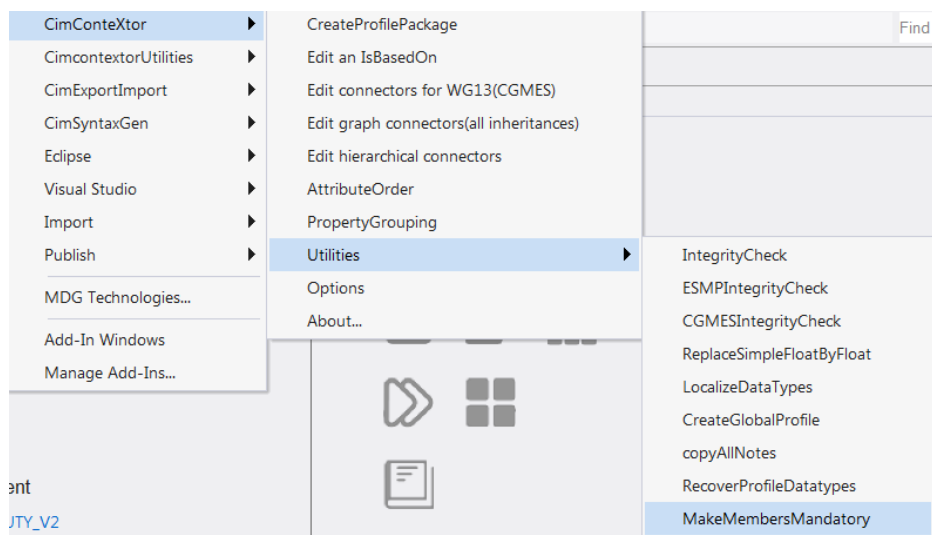


Click "OK" and the process will end: wrong datatypes pointing has been changed, and missing profile datatypes have been created.

19.13. Use of "MakeMembersMandatory"

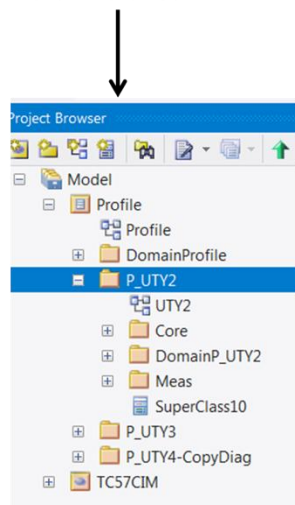
MakeMembersMandatory is a feature that enables the profile class attribute cardinality to be changed from optional to mandatory. This is useful, in case most of the attributes of the profile are "mandatory", adjustment could be done after by using the *EditIsBasedOn* feature.

Select a profile package and launch the Utilities/MakeMembersMandatory:

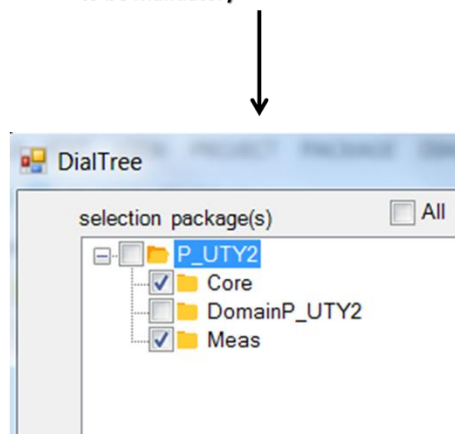


You will be prompted by a dialog box, asking you on which package or sub package(s) you want to run the feature:

Select profile package
In the EA browser



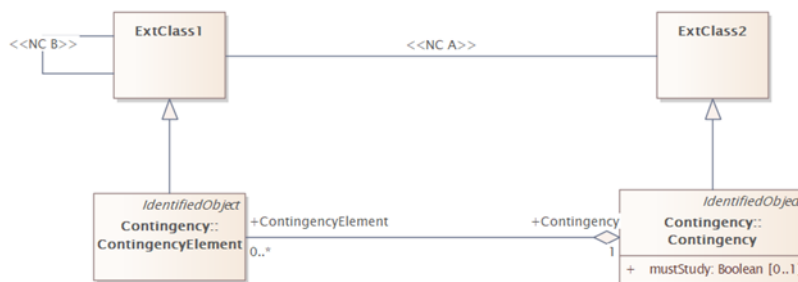
Select Profile Package(s) where you
want to have all attributes cardinality
to be Mandatory



Then click on "OK", and at the end of the process, every attribute's cardinality is mandatory in the selected package(s).

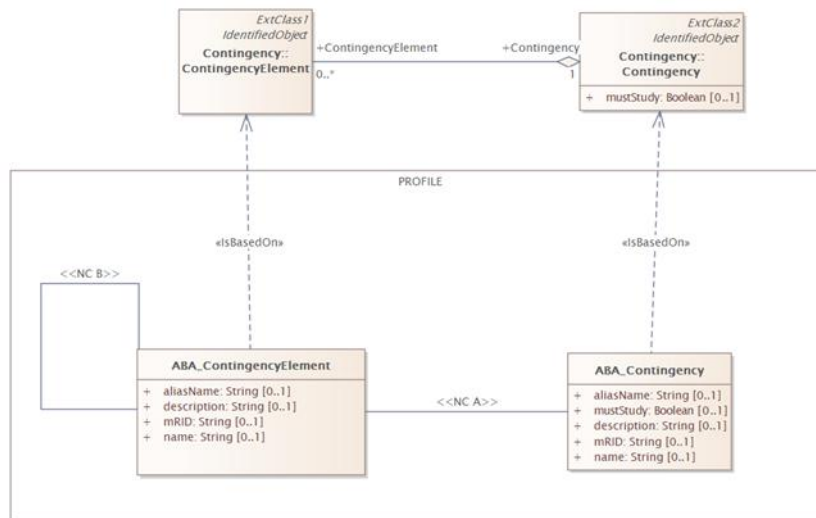
20.Copy Superclass Connections

In case there are CIM classes that have 'not based on' parent classes like depicted in the following figure

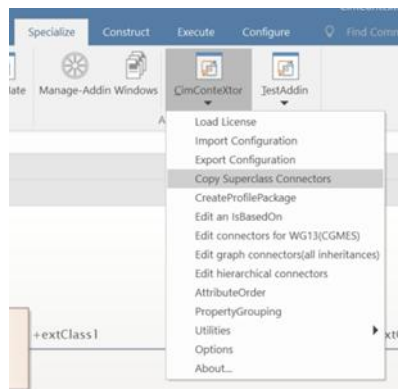


The classes ExtClass1 and ExtClass2 are not part in CIM, i.e. these are extended classes and are not 'IsBasedOn' classes. The CIM classes ContingencyElement and Contingency inherit from these classes.

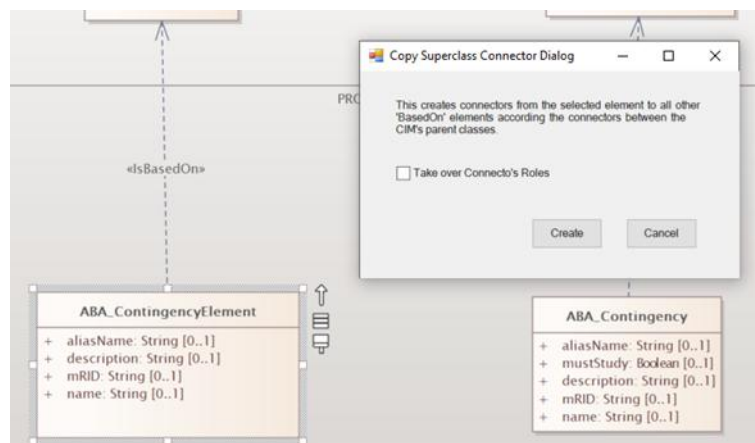
The functionality *Copy Superclass Connections* copies the two connectors (stereotyped for instance by 'NC A' and 'NC B') down when extending the CIM classes by 'IsBasedOn' classes like in the following picture



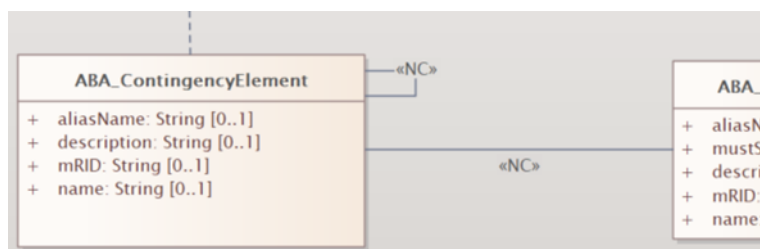
At first you select the 'IsBasedOn' class of which you would like to take over all associations of CIM parent classes. Clicking on the menu item in the ribbons menu



or by a right click and selecting the menu item from the context menu shows up the following pop-up window

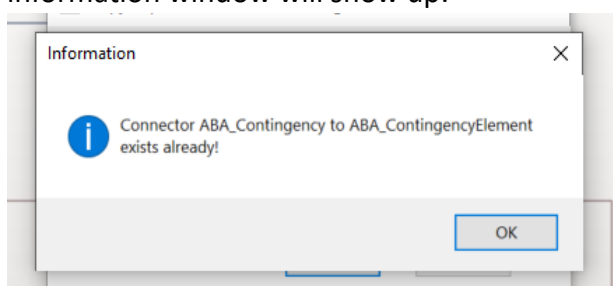


When clicking on “Create” the associations are copied:



There two special cases to keep in mind:

1. When using this mechanism on a class of which the associations exist already, an information window will show up:



2. The newly created associations obtain during the copying process a tagged value 'GUIDBasedOn' with the GUID of the association of the superclasses. This enables the mechanism to recognise the existence of the association by the GUID of the tagged value in case of repeated copying.

21.XSD Syntactic Generation

CimConteXtor does not do syntactic generation: to do that, you could use another Add-in “*CimSyntaxGen*”, which among others performs RDF Schemas and XSD schema generation (see *CimSyntaxGen* user’s manual). But for syntactic generation, there is some information that must be given, and UML could be used to do that: here are some examples.

21.1.Root Classes

When the syntactic target generation is XML Schema (see *CimSyntaxGen* user manual), you must define before which classes will be taken as « *Root Classes* », and therefore will be the starting point of the hierarchies for the XML Schema. To do that you must check the “*Is Active*” box in the “*Details*” window of the class properties.

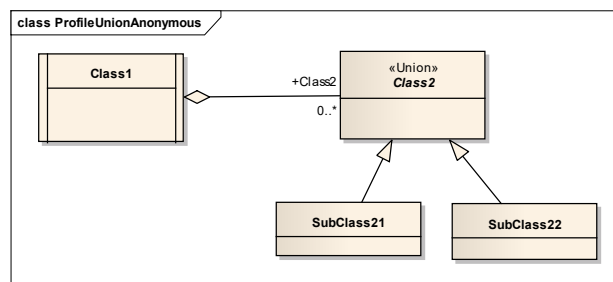
Note: there could be several “Root” classes in the profile, but each hierarchy under these “Root” classes must be independent.

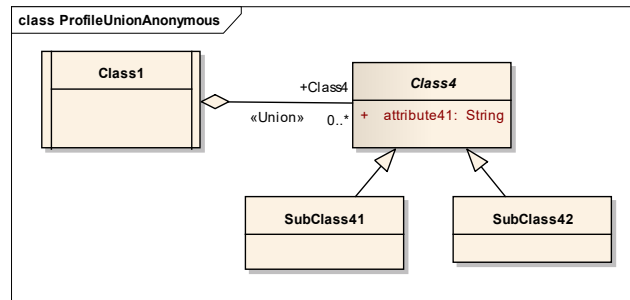
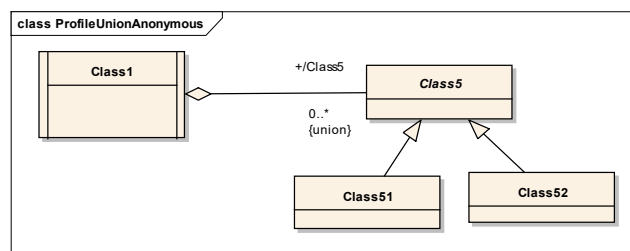
21.2. Use of “Union” and inheritance

Union could be used to express, in a short cut, the fact that an aggregation between a class and an abstract superclass implies that all aggregation between the class and the subclasses are considered in the XSD.

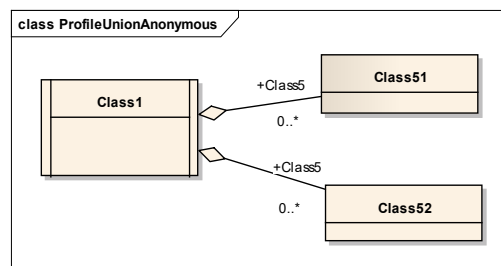
Union could be seen several ways and may be defined at several levels:

- The superclass could be marked with a Union stereotype,
- The aggregation could be marked with a Union stereotype,
- The aggregated end role name could be marked using a {union} constraint.





Note: the mapping to XSD is different than the following representation (see CimSyntaxGen).



21.3.Element Order (Attribute Order)

The order of the XML elements is driven by the attribute order specification.

22.Configuration File

22.1.Overview

The goal of configuration is to provide parameters for CimConteXtor that could be tailored by the users to fulfil their requirements. It means that the user could change some parameters and add new ones.

The configuration is coming with five different kinds of parameters:

- **appSettings**: this gives the state of elements in the Option Menu,

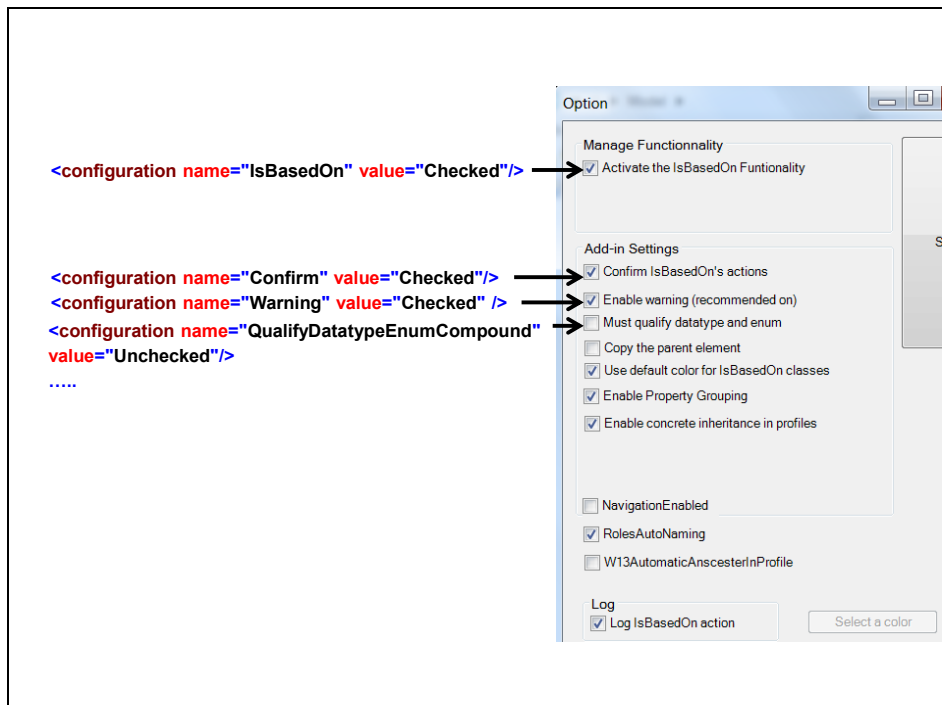
- dataProfiles: this gives a list of stereotypes used in a profile, or what is the profile domain package name,
- dataQualifier: this gives a list of predefined qualifiers that could be used by some given UML element,
- dataConstraint: this gives a list of constraints that could be used by some given UML element,
- dataClassifierConstraint: this gives a list of constraints that are used to restrict Primitive value space.

22.2.AppSettings parameters

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

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

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



These parameters are (see section 3.3 for the meanings):

- IsBasedOn

- Confirm
- Warning
- QualifyDatatypeEnumCompound
- CopyParentElement
- ConfigColor
- EnablePropertyGrouping
- EnableConcreteInheritanceInProfiles
- NavigationEnabled
- AutomaticChangeOfRoleName
- WG13AutomaticAncestorInProfile
- Log

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 CimConteXtor behavior:

- *EnabledIntermediaryInheritance*: this will enable the use of a partial hierarchy path. It means that a class could inherit from any of the inheritance hierarchy class: this will be shown in the inherit box of the EditIsBasedOn windows,
- *EnableESMPHierarchy*: this a parameter for the ESMPIntegrityCheck menu, that check if the hierarchy of packages and dependencies is good,
- *EnableCheckAttributeIdentifier*: for IntegrityCheck also. When this parameter is set, Integrity check verify that the attribute type is conforming to the type of information model one,
- *SimpleProfiling*: for future use. If set, it prohibits the use of qualifiers.
- *TimeAnalysis*: for development only. If set, it allows performance analysis.

22.3.DataProfiles parameters

These parameters are specific for profile management, they are called "profdata". They are currently two of them:

ListStereoNamespace: used to handle list of stereotypes used by CGMES, list for IntegrityCheck. Current values: " Entsoe|ShortCircuit|Operation|Abstract",

EntsoeDataTypesDomain: gives the name of profile domain package for CGMES profiles.

ProfilesPackage: gives the name of the package that contains profile packages and Domain package.

```
<profdata name="ListStereoNamespace" value="Entsoe|ShortCircuit|Operation|Abstract"/>
```

```
<profdata name="EntsoeDataTypesDomain" value="DomainProfile"/>.
```

```
<profdata name="ProfilesPackage" value="Profile" />
```

22.4.DataQualifier parameters

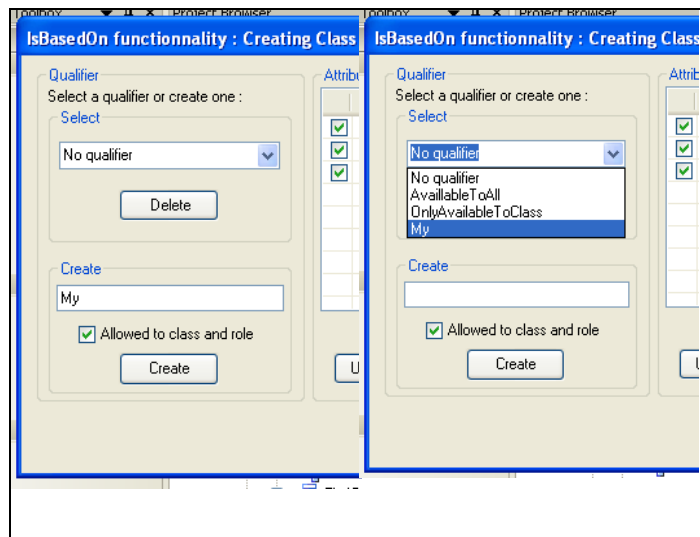
These parameters give the list of predefined "qualifiers" for given UML elements, they are called "qualifier", have a name and have an assignment to UML elements given by the value of "AllowedTo". Example:

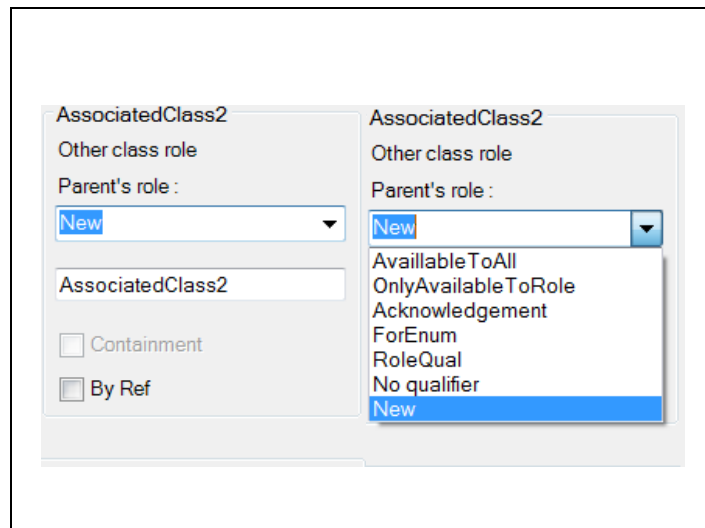
```
<qualifier name="Available" allowedTo="any"/>
```

The list of "allowedTo" value:

- Any: qualifier could be used by class, datatype and end role name,
- Class: qualifier could be used only by class,
- Datatype: qualifier could be used only for datatypes,
- Role: qualifier is only for end role name.

The list of qualifiers is used by the drop-down menu of the qualifier box of either the "EditIsBasedOn" class (class and datatype) windows or the "ModifySelectedAssociation" windows:





The list is dynamic, because when you use the manage qualifier feature for associations or create qualifier feature for classes, when save, the created qualifier is added to the configuration file.

Note: check the config file and make your own qualifiers if needed.

22.5.DataConstraint parameters

These parameters called "*constraint*" give predefined constraint for class (including datatype) and association. These constraints will appear in the drop-down menu.

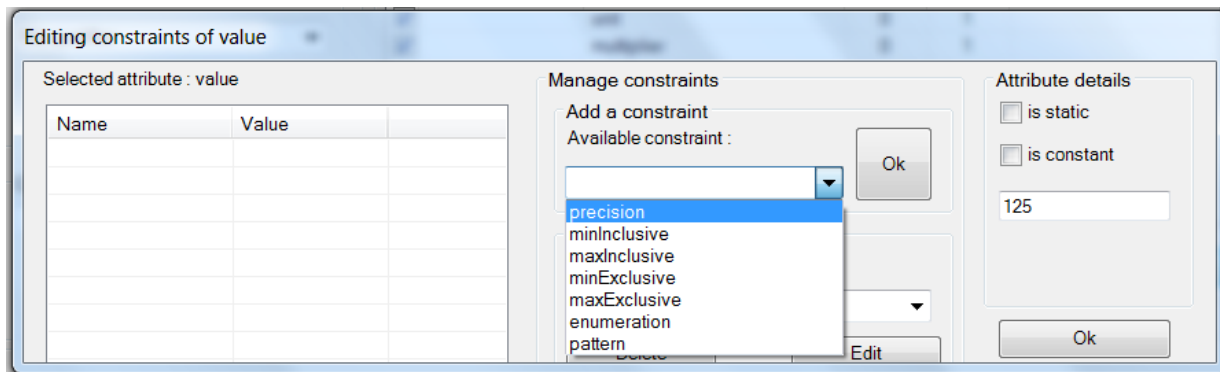
<constraint name="testDatatype" type="OCL" allowedTo="datatype" notes="OCL constraint">.

- name: name of the constraint
- type: OCL or INV
- allowedTo: any, class, datatype, attribute and role
- notes: constraint expression

22.6.DataClassifierConstraint parameters

22.6.1.Overview

These parameters called "*classifierconstraint*" define a value space constraint for a given <<Primitive>>. They can only be defined in the configuration file. They will appear in the drop-down menu of an "*EditFacet*" window of a CIMDatatype "value" attribute.



There are different classifier constraints:

1. Constraints that are XML datatype facets,
2. Constraints that are predefined XML value space (example token...).

A data classifier is defined by the following element:

- name: name of the facet, or name of the value space,
- type: type of constraint defined in the value element (OCL or INV),
- allowedTo: primitive on which the constraint applied,
- variable type: type of the variable that will be used in the constraint expression (any, Numeric, an xml primitive, or DEFINED)
- variable list: if the variable type is "DEFINED", this is the list of possible values
- comment: a comment on the constraint
- constraint expression: usually an OCL constraint that takes a parameter (\$N\$, \$P\$, \$I\$ or \$C\$) to output the right expression with the parameter entered by the user.

DataClassifier example:

```
<classifierconstraint
  name="minInclusive" → xml minInclusive facet
  type="OCL" → type of output constraint
  allowedTo="DateTime" → datatype it is applied to
  variableType="Integer" → the value that will be entered will be an integer
  variableList="" → reserved for DEFINED variable Type
  comment="" → any comment for this constraint
  inv: self->minInclusive($N$) → template for the constraint expression
</classifierconstraint>
```

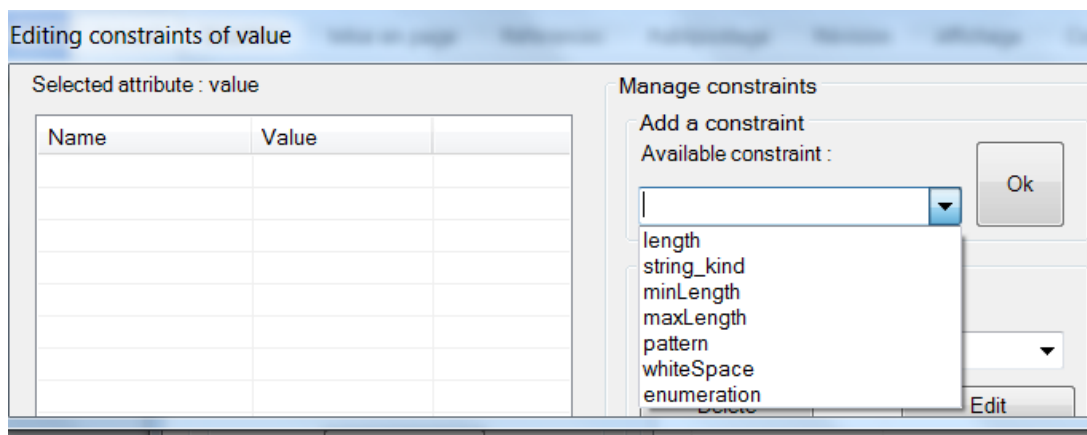
22.6.2. Classifier constraint as XML facets

These constraints give the parameters to define a restriction on the value space of a given primitive, to allow the user to define the value of the restriction and to allow CimConteXtor to generate the corresponding expression in OCL. How does it work?

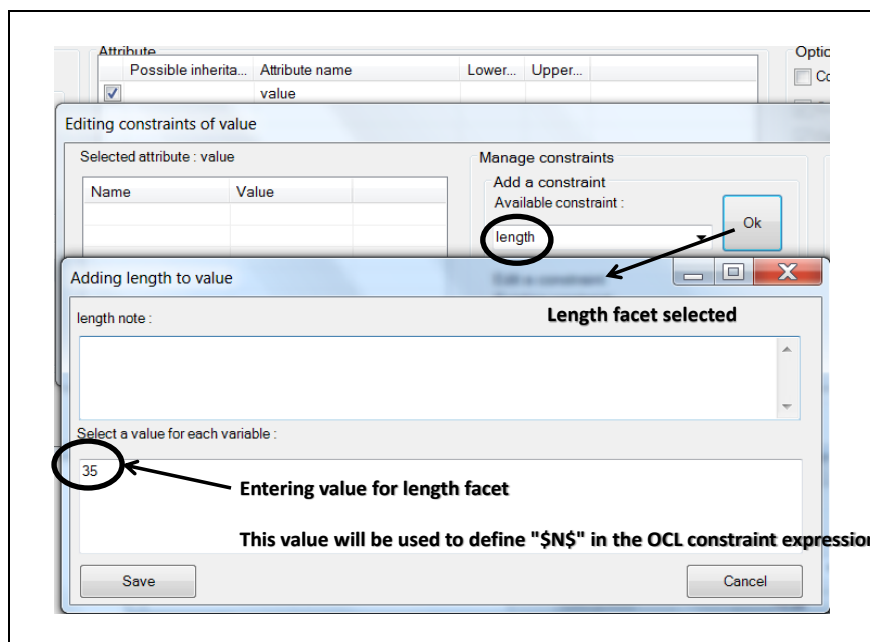
Example of a "length" restriction for String. The following classifier constraint is in the configuration file:

```
<classifierconstraint name="length" type="OCL" allowedTo="String" variableType="Numeric"
variableList="" comment="">
  inv: self->Length($N$)
</classifierconstraint>
```

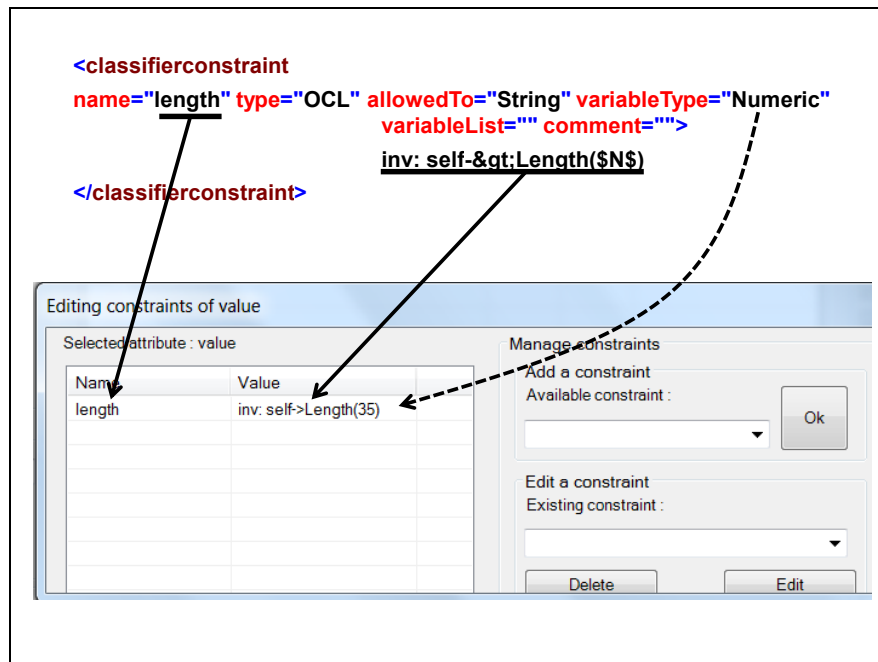
When the "EditFacet" of a value attribute is activated, the following window appears, and in the drop-down menu of the available constraint field there is the "length" facet:



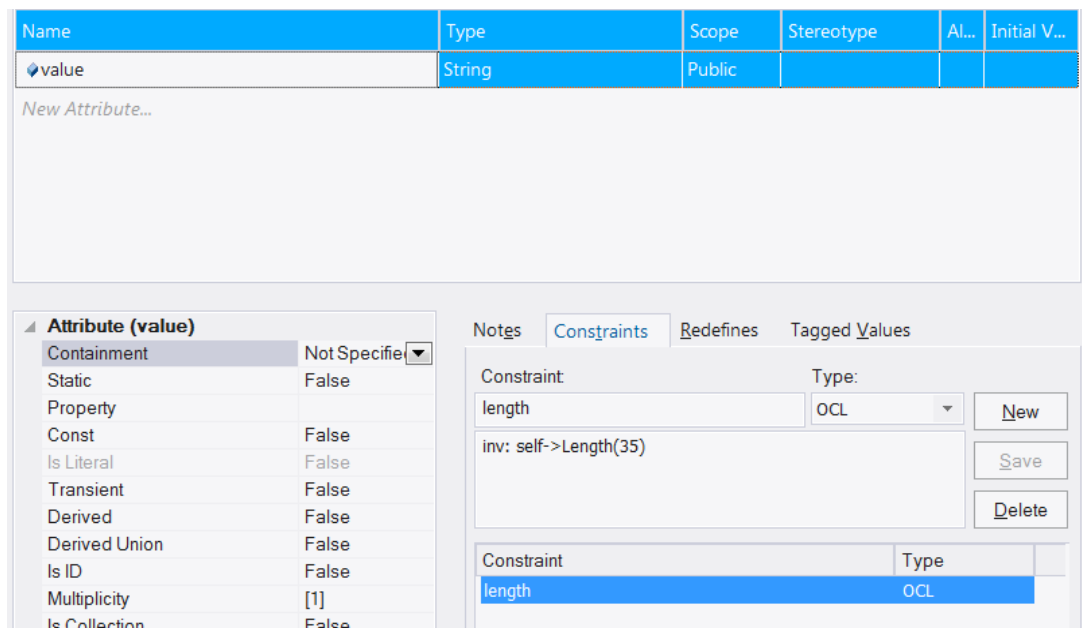
Selecting "length" and make "Ok" will open a new window:



By entering a value (here a numeric value), the restriction is saying that the length of the string will be this value (35 in the example). By making a "Save", this will enter the restriction and generate the OCL constraint (according to the template define in the constraint classifier: " inv: self->Length(\$N\$)" = "inv:self->Length(35)").



This constraint is now visible in EA datatype properties window:



The list of constraint classifier is that of xml datatypes.

22.7.Classifier constraint as XML predefined datatypes

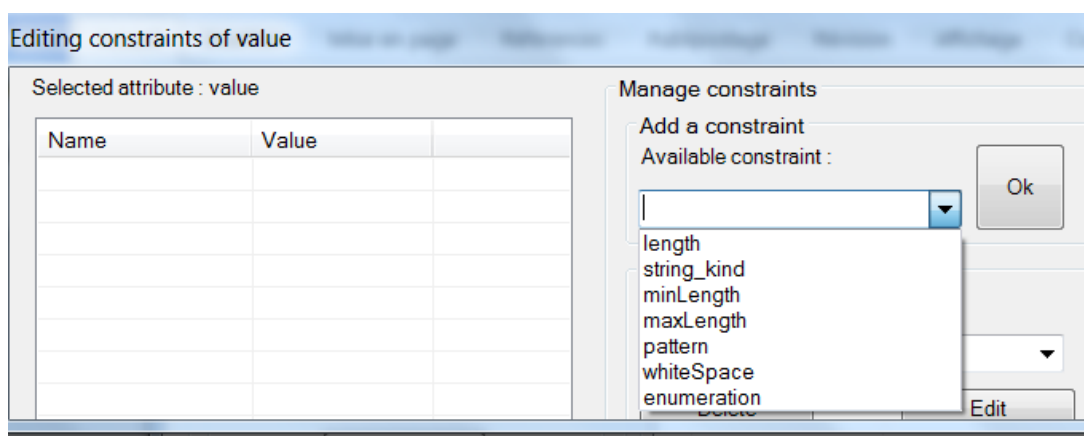
These constraints give the parameters to select a predefined named xml value space that restricts already the value space of a given primitive. This will allow the user to use predefined value space,

which may also be restricted. CimConteXtor will generate the corresponding value space as a name. How does it work?

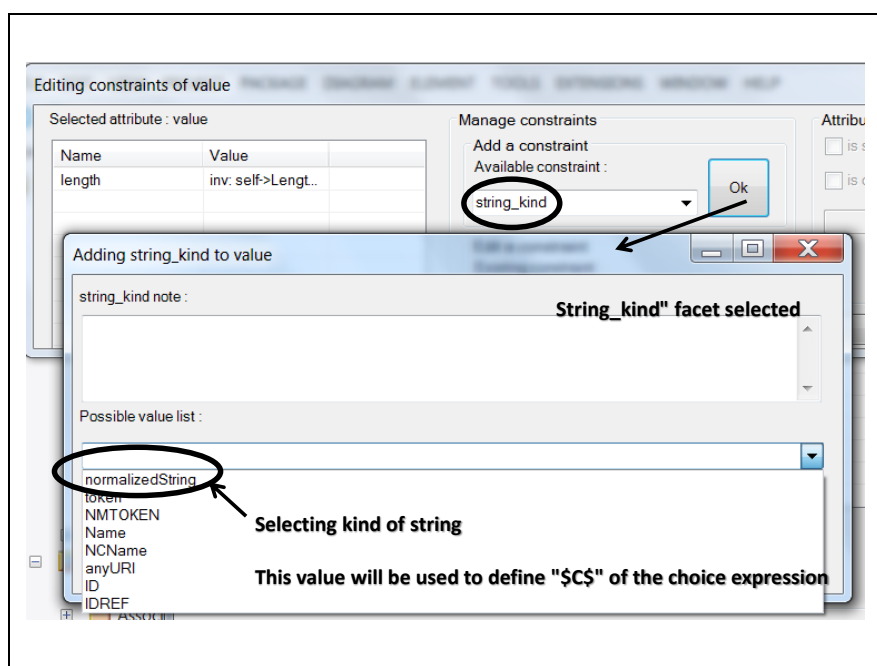
Example of a "string_kind" restriction for String. The following classifier constraint is in the configuration file:

```
<classifierconstraint name="string_kind" type="INV" allowedTo="String" variableType="DEFINED"
variableList="normalizedString,token,NMTOKEN,Name,NCName,anyURI,ID,IDREF" comment="">
choice=$C$
</classifierconstraint>
```

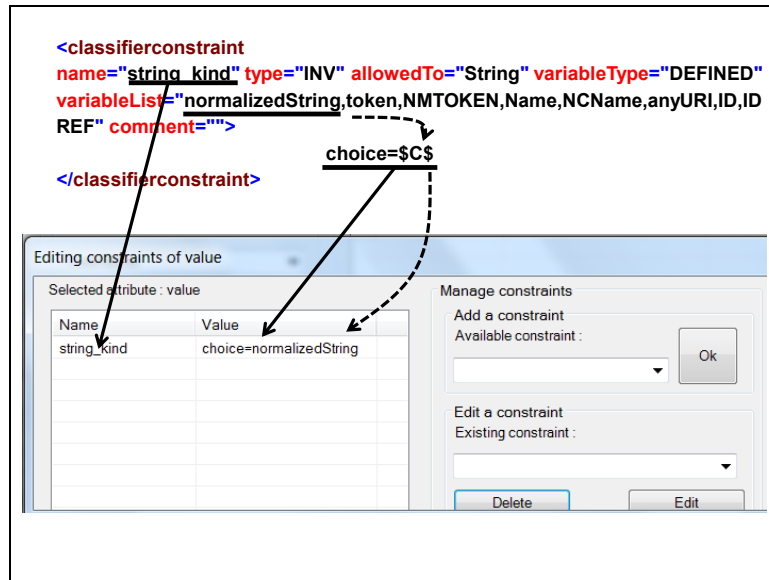
When the "EditFacet" of a value attribute is activated, the following window appears, and in the drop-down menu of the available constraint field there is the "string_kind" facet:



Selecting "string_kind" and make "Ok" will open a new window:



By selecting one of the values (here a normalizedString), the restriction is saying that the value space is that of an XML datatype (normalizedString in the example). By making a "Save", this will enter the selected value and generate a choice constraint (according to the template define in the constraint classifier: "*choice=\$C\$*" = "*choice=normalizedString*").



Predefined XML value spaces exist for:

- String: normalizedString, token, NMTOKEN, Name, NCName, anyURI, ID, IDREF
- Integer: long, int, short, byte, unsignedLong, unsignedInt, unsignedShort, unsignedByte.

23. Model validation

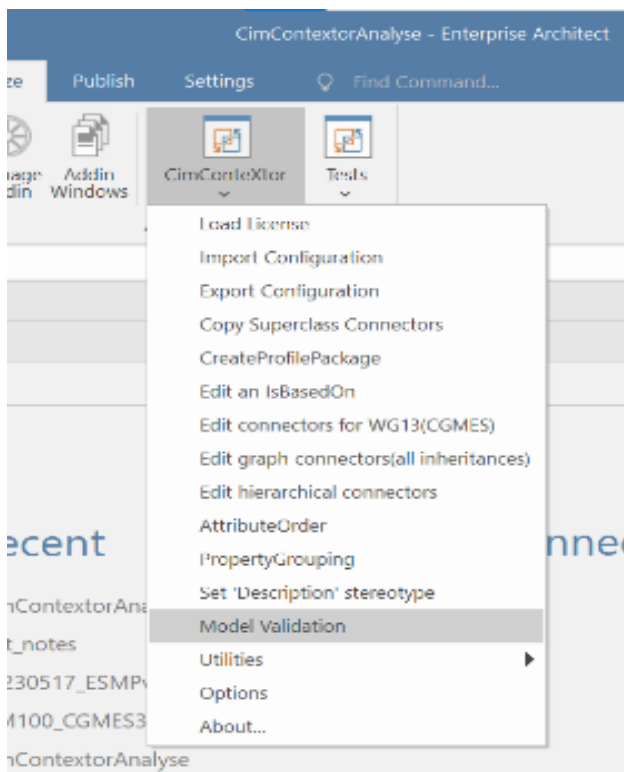
23.1. Introduction

The CimConteXtor validation mechanism allows the model to be validated against the guidelines and recommendations for using the UML to construct and specify CIM and IEC61850 models. It is predicated on the validation capabilities of JCleanCIM.

The validation mechanism comprises

- Loading the model into memory to accelerate the validation process
- Maintaining the model validation configuration
- Validating different modelling elements: Packages, Elements (Classes), Attributes, Connectors (Associations and Dependencies), Operators and their Parameters and Diagrams.
- Providing a report which can be exported as CSV file with possibility to choose the delimiter.

The validation dialog is started in EA by the corresponding menu item of the CimConteXtor menu:



23.2. Model loading

Generally, to load a model from EA can be done in several ways (from fastest to slowest):

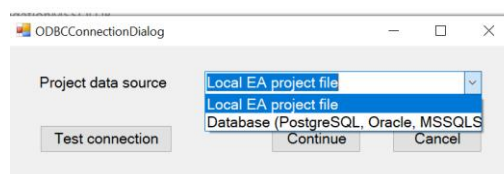
- Directly from the local *.eap file as MS Access database (database connection) or from a central database by SQL queries.
- Exporting the model as XMI file, reading the XMI file and parsing its content.
- Using the EA API.

The quickest way to build an in-memory model is by using SQL queries to load data from the database. This can be done in a generic way to provide the possibility to use this mechanism also in CimSyntaxGen. Therefore, the codebase for loading model is provided as separate library for CimConteXtor and CimSyntaxGen.

In contrast to JCleanCIM, the in-memory modeling elements are independent from the validation mechanism and specifics of CIM and related standards.

23.2.1.Database Connection

Even in the case of a local EA project file (*.eap, *.eapx), which is a local MS Access database in form of a single file, the EA project data is always kept in a database. Access to the database is defined by an ODBC Data source connection. The model loading mechanism is using the EA API internal connection string which contains all necessary information to use the ODBC connection. In case of a local file the handling of this information is differs from the one of databases of a DBMS system. Therefore, the user has to select the project data source (local file or DBMS system) before the model loading can start. The following figure shows the dialog for selection:



The connectivity to the database is tested by the button “Test connection”.

NOTE: This will test whether the connection to the database can be opened. This not a guarantee that the model loading mechanism has valid access to the tables.

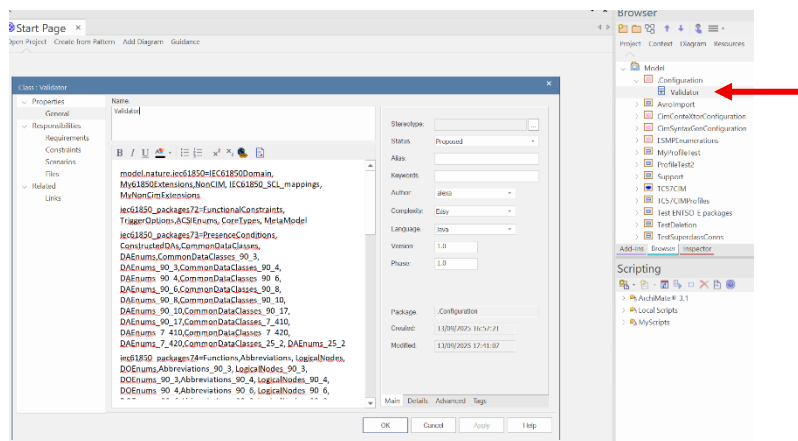
Since the access to the database is done by just using the DSN (Data Source Name) of the ODBC connection which already holds all information for the database access inclusive user and password, no additional data has to be entered. This mechanism has been tested with a PostgreSQL, an Oracle and a MSSQL Server database.

By clicking on the button “Continue” the model loading mechanism starts with a progress bar. The project data source selection is stored in the configuration (see below) and is pre-filled for future use.

23.3.Configuration

The configuration can be created and maintain either by the configuration dialog or by importing a (java) properties file.

In any case, the configuration is stored in the EA project’s root model in the package “Configuration” as Notes in the element ‘Validation’:

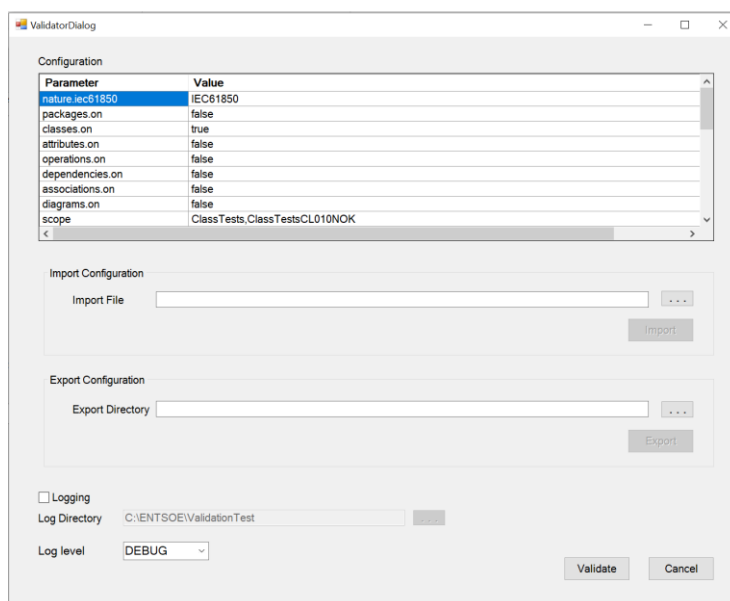


NOTE:

- This package and its content is excluded from the validation processing.
- It is not recommended to modify the Notes field of the Validation element. This might cause side effects leading to errors when validating.
- If the Validation element is deleted, it will be newly created when the validation dialog is opened.

23.3.1.Configuration Dialog

The following screenshot depicts the configuration dialog:



The upper part lists the configuration parameters with their values. Per default there are parameters without values present. These parameters are all that the validation mechanism recognizes and there is no possibility to enter new parameters by editing this parameter list.

However, the values of the parameter can be edited. If a parameter's value field contains a comma-separated list of values, then the list is parsed and internally used as list of values, e.g. a list of package names.

NOTE: *Newly entered values are only stored when a validation is started.*

23.3.2.Import of Configuration

To import a configuration the file must be select by clicking on the corresponding button with dots on the right-hand side of the import text field. When a file is selected, the import button becomes enabled.

NOTE: *The import will erase an existing configuration completely and store the imported one.*

The configuration file must be in the Java properties file format. Comments (indicated with a Hash character '#'), tabulators and empty lines are omitted. Line breaks ('\n') at the end of a line are recognized.

It is recommended at first to export the configuration and then to edit the exported file to make sure, the parameters are correctly specified. In contrast to JcleanCIM the parameters don't have the qualifier 'validation'. As the configuration is already validation specific, such a qualifier would be redundant.

E.g. In JCleanCIM you might have 'validation.iec61850.packages' and in CimConteXtor you must provide 'iec61850.packages'.

23.3.3. Export of configuration

To export a configuration the button on the right-hand side of the export text field must be clicked to open the directory selection dialog. The exported file name's format is:

Validation_<YYYYMMDD>.<HHmm>.properties

with YYYY standing for the year, MM for the month and DD for the day. 'HHmm' is the time in hours (24 hours format) and minutes.

E.g.: Validation_20250912_1707.properties

NOTE: Several values of a value field of a parameter are exported as comma-separated list. There are no line breaks and comments exported.

23.3.4. Logging

The configuration dialog also allows for switching on/off logging into a log file. The log file name's format is analogous to the configuration export file:

ValidationLog_<YYYYMMDD>.<HHmm>.log

with YYYY standing for the year, MM for the month and DD for the day. 'HHmm' is the time in hours (24 hours format) and minutes.

E.g.: ValidationLog_20250911_2302.log

There is the possibility to configure the log level. Since logging reduces the performance of the validation mechanism a log level with higher priority, e.g. ERROR, minimizes writing information into the log and will have just a small impact on the validation's performance.

The log levels are (from higher priority to lower priority):

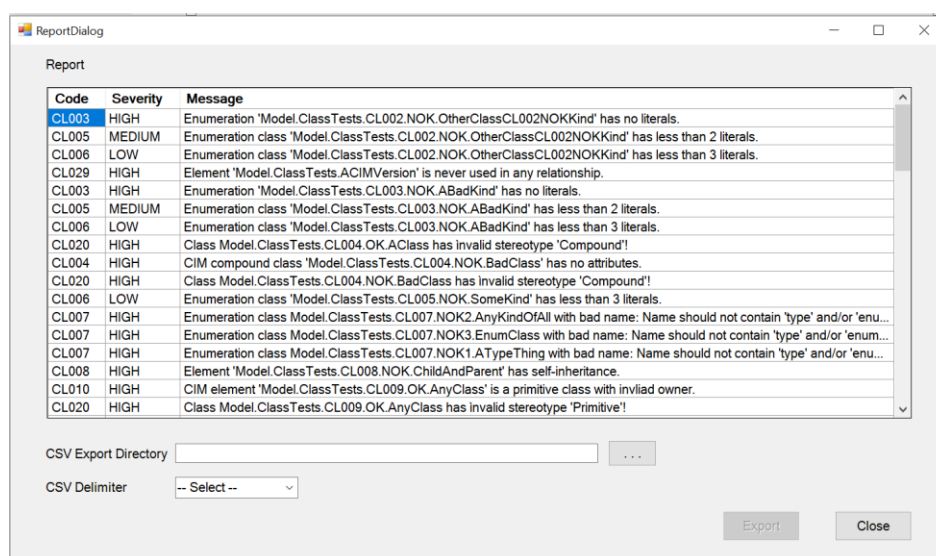
- FATAL
Errors with immediate shutdown of the CimConteXtor.
- ERROR
Errors that enable to close the CimConteXtor.
- WARN
Warnings.
- INFO
Information that might be hints for error situations or necessary amendments of the configuration.
- TRACE
Information that supports to understand difficult error situations.

- DEBUG

Provides a lot of information for understanding the process and error situations. Helpful in case of errors to support the developer in bug-fixing.

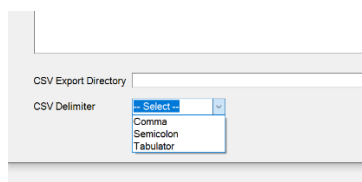
23.4.Validation report

After running the validation a report dialog opens automatically with a list of validation rules violations. The following screenshots shows this dialog:



Each violation is listed by code, severity (high, medium, low) and a message. To export the report as CSV file the button on the right hand side of the export directory text field has to be used to open the directory selection dialog.

The delimiter for the CSV file can be selected by the combo box. There are 3 types: Comma, Semicolon and Tabulator:



The exported file name's format is:

Report_<YYYYMMDD>.<HHmm>.csv

with YYY standing for the year, MM for the month and DD for the day. 'HHmm' is the time in hours (24 hours format) and minutes.

E.g.: Report_20250910_1016.csv

The csv file contains a header row with Code, Severity and Message.

23.5.Validation rules

The validation mechanism is based on validation rules and is implemented via an assertion mechanism to allow for a DSL (domain specific language) approach in future enabling to specify the rules easily without implementation effort.

Due to these assertions the rules are formulated in a command (positive) way, e.g. 'Class name starting with upper case', i.e., the class name as to follow this rule in contrast to 'Class name starting with lower case' as in JCleanCIM.

Therefore, the rules specify what has to be. A violation leads to an error.

23.5.1.Scope

Some rules apply in general, some apply to CIM models only, and some apply to IEC61850 models only.

The validation mechanism covers seven kinds of UML elements:

- Attributes
- Associations
- Classes
- Packages
- Diagrams
- Dependencies (hand-drawn)
- Operations (for UML of IEC61850 only)

The implemented validators deal with rules for

- Model consistency.
- CIM/IEC61850 UML naming and design rules..
- Potential model editor errors.
- Identifying remains from Rose or other imported models.
- Illegal and redundant UML constructs that the EA in some situations allows.

23.5.2.Code

Each rule is identified by a code set up by an abbreviation of the modelling element and a sequential 3-digits number, e.g. CL003

The abbreviations are as follows

- Class: CL
- Attribute: AT
- Association, Dependency: AS
- Operation, Parameter: OP
- Package: PK
- Diagram: DI

23.5.3.Rules for Classes

Code	Rule	Severity
CL001	CIM classes with expected elements only	High
CL002	Expected connectors only	High
CL003	Enum class with literals	High
CL004	CIM compound class has attributes	High
CL005	Enum class with more than one literal	High
CL006	Enum class with more than two literals	High
CL007	Enum class with valid name	High
CL008	Class without self-inheritance	High
CL009	CIM primitive class without attributes	High
CL010	CIM primitive class with valid owner	High
CL011	Class without duplicate inherited attribute names	High
CL012	Class without duplicate own or inherited association end names	High
CL013	Class without self-dependency	High
CL014	Class with leaf property not set	High
CL015	Class with root property not set	High
CL016	Class with persistent property not set	High

CL017	Class with no multiple superclasses	High
CL018	Class with superclasses from allowed owner only	High
CL019	Class is not association class	High
CL020	Class with allowed stereotype only	High
CL021	CIM class with no old datatype stereotype	High
CL022	CIM class used for attributes and has no associations	High
CL023	CIM class used for attributes and has no subclasses	High
CL024	CIM class used for attributes and has no superclasses	High
CL025	CIM class is not abstract	High
CL026	CIM class has no operations	High
CL027	CIM class has no explicit dependencies	High
CL028	Class without nesting through attribute	High
CL029	CIM class is used in relationships	High
CL030	Class with allowed tag names	High
CL031	CIM datatype class with valid attributes	High
CL032	Class with valid characters in name	High
CL033	CIM class name starting with upper case	High
CL034	CIM class is singular	High
CL035	IEC61850 LN class in correct group	Medium
CL036	IEC61850LN class with correctly formed name	High
CL037	Enum class with all codes	High

CL038 Enum class without duplicate codes High

CL039 No class with the same name High

23.5.4. Rules for packages

Code	Rule	Severity
PK001	Package with no self-dependency	High
PK002	Package with allowed stereotypes	High
PK003	Top level package with version class	High
PK004	IEC61850 package with alias as title	High
PK005	Package with allowed tag names	High
PK006	Package with valid characters in name	High
PK007	Package names are unique	High

23.5.5. Rules for attributes

Code	Rule	Severity
AT001	Enum literals with enum stereotype	High
AT002	Attribute with valid multiplicity	High
AT003	CIM attribute is optional	High
AT004	CIM attribute is public	High
AT005	Attribute that is static and const	High
AT006	CIM attribute that is not static and not const without initial value	High
AT007	Attribute is const and static	High

AT008	Attribute with allowed stereotypes only	High
AT009	Attribute that is enums and in enumerated class	High
AT010	CIM attribute that should not be replaced with association	High
AT011	Attribute whose type is not informative	High
AT012	Attribute with allowed tag names	High
AT013	IEC61850 attribute with existing sibling	High
AT014	CIM attribute without flag in name	Medium
AT015	CIM attribute with valid characters in name	High
AT016	IEC61850 attribute with valid characters in name	High
AT017	IEC61850 DO attribute with correct full name	High
AT018	IEC61850 FCDA attribute with constraint	High
AT019	Attribute with existing enum literal as init value	High
AT020	IEC61850 DO attribute with name abbreviation	High
AT021	CIM attribute name starting with lower case	High
AT022	CIM attribute name is singular	High
AT023	CIM attribute name does not start with class name	High
AT024	IEC61850 abbreviation literal name starting with upper case	High
AT025	IEC61850 DO attribute name starting with upper case	High
AT026	Attribute with type from allowed owner	High
AT027	IEC61850 DO abbreviation literal has unique name	High
AT028	IEC61850 DO abbreviation literal has unique description	High

AT029	IEC61850 DO abbreviation literal is used in DO name	High
AT030	IEC61850 DO attributes with same name and identical type	High

23.5.6.Rules for operations

Code	Rule	Severity
OP001	Operation with lower case name	Medium
OP002	Operation with allowed stereotype only	High
OP003	Operation parameters with allowed stereotype	High
OP004	Operation with valid return type null	High
OP005	Operation with valid arg type	High
OP006	Operations with allowed tag names	High
OP007	Operation parameters with allowed tag names	High
OP008	Operation with valid characters in name	High
OP009	Operation parameters with valid characters in name	High

23.5.7.Rules for associations

Code	Rule	Severity
AS001	Association with unspecified direction	High
AS002	Association roles with unspecified direction	High
AS003	Association with name	Medium
AS004	Association with allowed stereotype	High

AS005	Association ends with allowed stereotype	High
AS006	Informative association with informative stereotype	High
AS007	Associations with allowed tag names	High
AS008	Associations with multiplicity	High
AS009	Associations with correct source	High
AS010	IEC61850 association with be private ends	High
AS011	IEC61850 association with identical end visibility	High
AS012	Association ends with valid characters in name	High
AS013	CIM Association ends names starting with upper case	High
AS014	CIM Association ends names are plural according multiplicity	High
AS015	CIM Association ends names are singular according multiplicity	High

23.5.8.Rules for dependencies

Code	Rule	Severity
AS016	Dependency with allowed stereotype	High
AS017	Dependency with allowed direction	High
AS018	Dependency with allowed tag names	High

23.5.9.Rules for diagrams

Code	Rule	Severity
DI001	Diagram with correct orientation	Low

DI002 Diagram with allowed stereotype High

DI003 Diagram name with valid characters High

23.5.10. Excluded JCleanCIM rules

There are some JCleanCIM rules that are excluded due to the following reasons

- They are not relevant anymore due to new versions of the EA
 - “Attributes with type id mismatch”,
 - “Associations with role bad direction”
- Their meaning is not clear
 - “Enum literals with superfluous type”,
 - “Attributes with invalid type null” ,
 - “Attributes with invalid type string”,
 - “IEC61850 condition literals never used as constraints”,
 - “Classes with questionable attribute count”,
 - “Iec61850 classes with invalid constraints”,
 - “Iec61850LN classes with superfluous constraints”,
 - “Iec61850 classes with missing CondID text in constraints”
- Their applicability is very limited or is covered by other rules already
 - “Association ends with unallowed tag names”,
 - “Operations with invalid exc type null”,
 - “Package unexpected elements”,
 - “Package unexpected connectors”

23.6.Validation test package

To clarify validation rules and to support the definition of configuration files, ENTSO-E provides an EA project (ValidationTestProject.eap) containing test cases for validating the following entity types:

- Packages
- Classes
- Attributes
- Operations
- Associations (also Dependencies)
- Diagrams

Configuration files (properties files) are supplied for each entity type validation. There are one or more EA packages with the name of the validation rule for each validation rule, e.g. AT003, and additional packages (e.g. IEC61850_AT003Classes) containing the rule name.

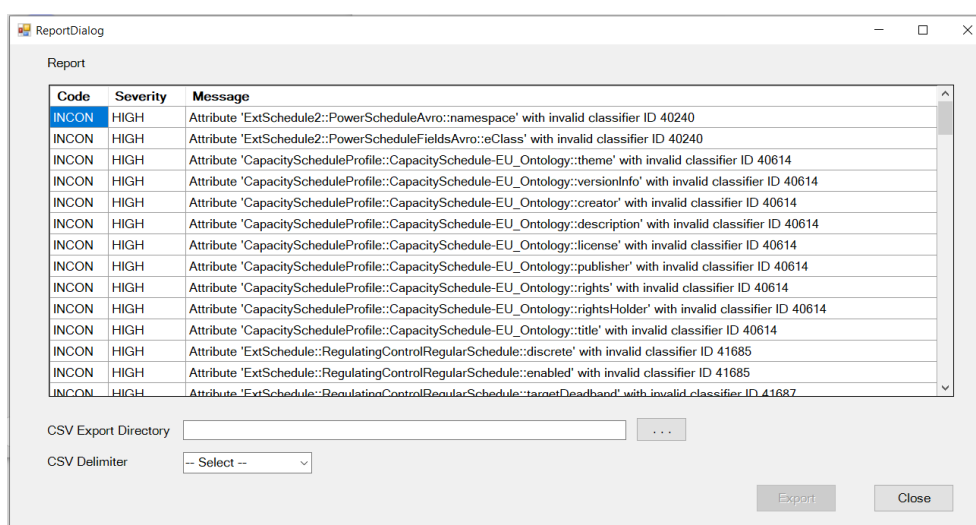
23.7. Known problems and how to solve them

23.7.1. Model Inconsistencies

In some situations, the EA model contains inconsistencies regarding references of one element to another, e.g. the reference of an attribute to the classifier which is the type of the attribute.

In case of this kind of inconsistency the referred element does not exist under the element ID used for the reference. E.g. in case of an attribute and its data type it might happen, that the classifier ID of the data type points to an element that does not exist in the model with this element ID.

Such inconsistencies are detected when the model loading mechanism is loading all modelling elements and tries to resolve references. To support fixing such inconsistencies, the validation has to be started (regardless of the validation configuration). In this case the validation mechanism is just used to report the inconsistencies which will list all inconsistencies marked by INCON. The following screenshot shows a sample output of inconsistencies:



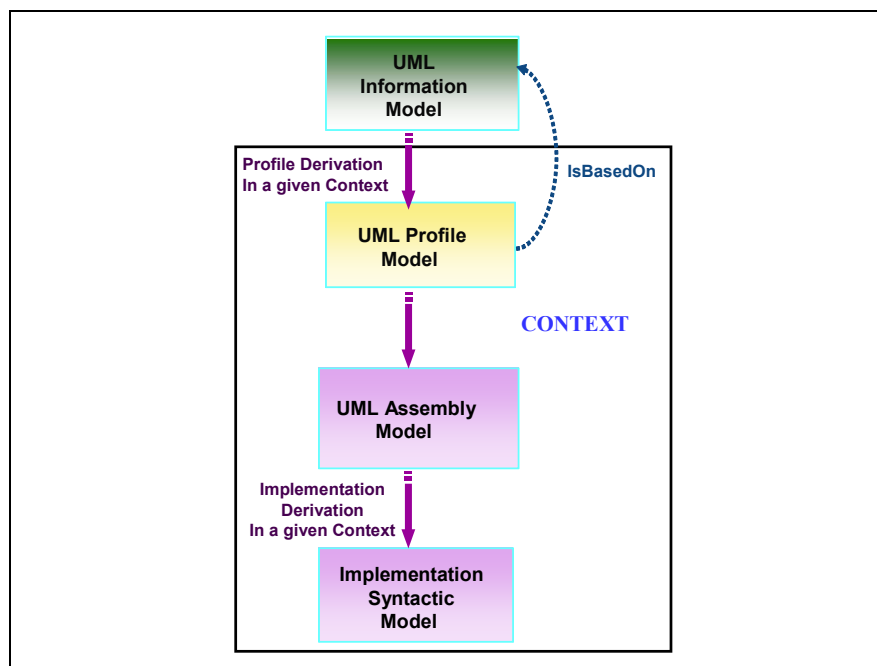
The report can be stored as CSV file and the inconsistencies have to be fixed.

NOTE: The validation will not provide a reliable result as long as there are such kind of inconsistencies in the model.

A. Annex A: Modelling methodology summary

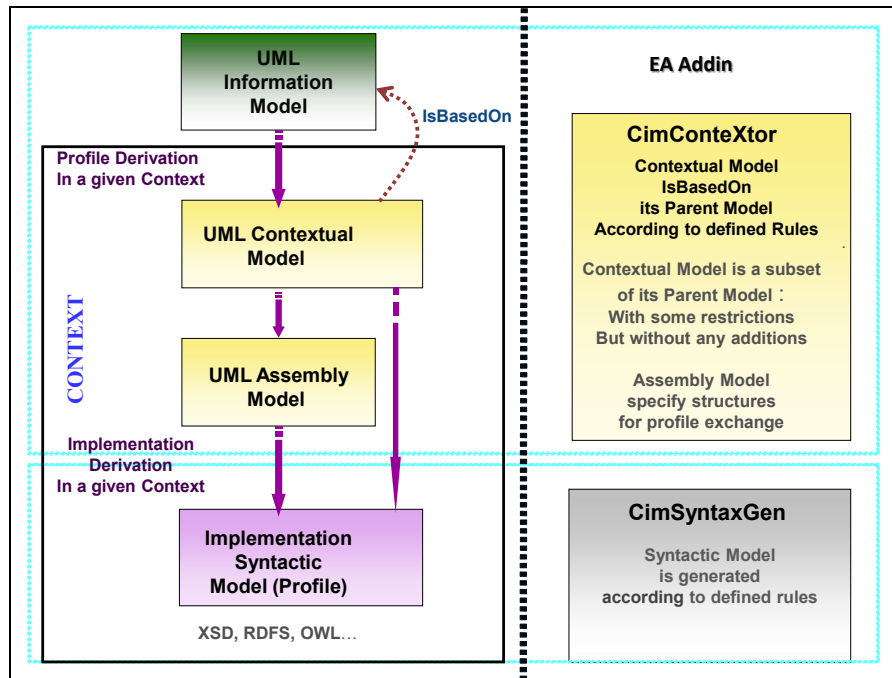
The modelling methodology that is behind *CimConteXtor* defines four modelling levels:

- the Information Model level that defines a domain model (for example the CIM for Electro technical Domain),
- the Profile Model Level, that 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 -- network exchange): this is the “*IsBasedOn*” concept,
- the Assembly level, that defines how the profile artefacts are assembled for exchange,
- 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.

CimConteXtor is a tool that builds an UML Profile and possibly an Assembly Model based on an UML Information Model. *CimConteXtor* is used with companion tools that generate syntactic schemas (XSD or RDFS) according to given Assembly rules and Syntax Naming and Design Rules (See *CimSyntaxGen* Add-In Tool user Guide for XSD and RDFS generation).



B. Annex B: Information Model UML modelling

B.1. UML Class Modelling

UML models artefacts (or objects) are classes that have attributes (their simple properties) and that have associations (their complex properties) between them. UML uses a graphical notation to represent all these elements (Class, attribute, association).

Attributes:

- have cardinality (optional or mandatory).
- have a type: either a Primitive, an enumeration, a Datatype, or a Compound,
- can be grouped (in CIM, grouping is defined by a named class stereotyped by <<Compound>>), to be able to reuse this group in different classes.

Associations:

- define relations between classes,
- could be of different types: bidirectional, aggregation or composition,

- have multiplicity and have end role names.

B.2. UML element attached information

In UML various information can be attached to classes, attributes, associations, datatypes and compounds. This information are Stereotype, Note, Tagged Value, Constraint...

Stereotypes:

Stereotype is a general mechanism that is used to categorize the various elements: classes, attributes, associations. In UML, they are shown between double brackets <<...>>.

Notes:

Notes are used to carry all textual information about an element (Class, attribute, association, datatype). A good practice is to use it just for giving the definition of the element (to make a good definition see ISO/IEC 11179).

Most of the time, people use it also to give usage rules of the element, but they should use another item for that: *TaggedValue*.

TaggedValue:

TaggedValue specifies additional information about an element. TaggedValue has a name, content (the value) and notes. So a TaggedValue could be used to specify a category of additional information like "Usage Rule", "Business Term," ...

Constraints:

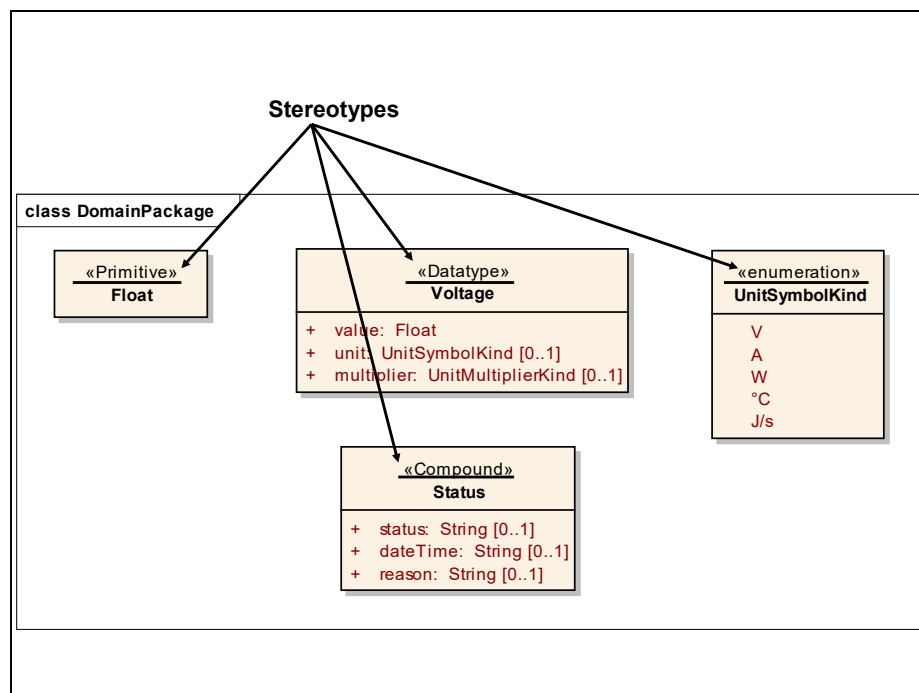
Constraints are conditions imposed on one or multiple elements.

- For a class it could be the conditions under which the element can exist or function,
- For an attribute, it could say something about the rules and conditions under which the attribute is used,
- for association, it could say something about the rules and conditions under which a relation operates.

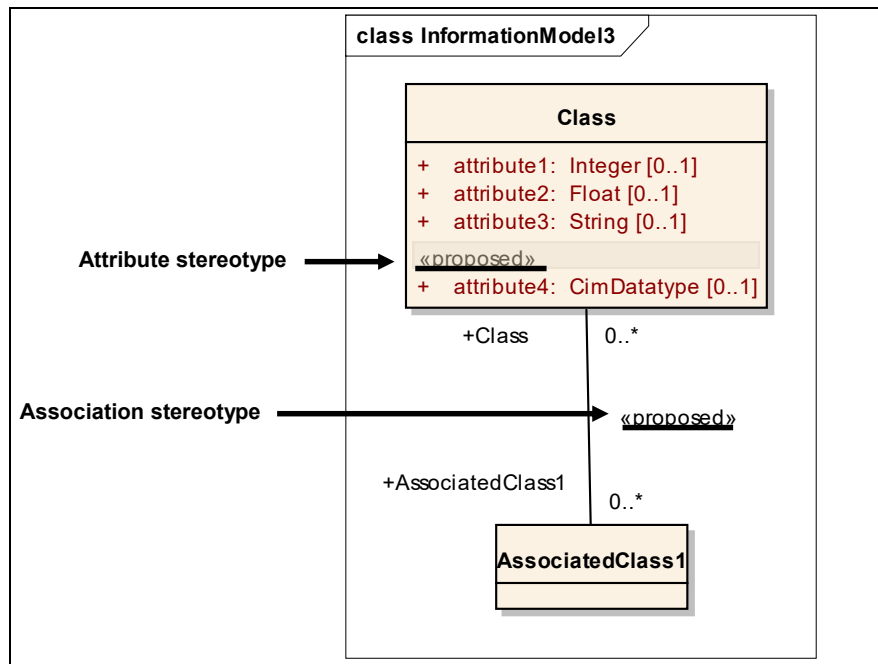
Constraints could be expressed in different formats or languages that could be specified too. UML proposes to use OCL (Object Constraint Language), but other languages could be used too. For example, *CimConteXtor* is using OCL to express facets for Datatypes.

B.3. UML stereotype examples

All elements can be stereotyped. A stereotype is information that is attached to an element to classify it, and to group elements that have common characteristics. For example, there are different kinds of attribute types: Primitive, enumeration, Datatype and Compound. Each of these types is stereotyped: in UML, stereotypes are represented by a name enclosed in double brackets <<Primitive>>, <<enumeration>>, <<Datatype>>, <<Compound>>...



Associations and attributes can also have stereotypes. Example: stereotype << CIMExtension>> can be used to express an Information Model proposed extension.



B.4. Information Model Naming Rules

At the information model level, all the naming rules should be in conformity with TC 57 naming rules (like UN/CEFACT CCTS (Core Component Technical Specification) ones):

- Use of UpperCamelCase for Class and Association,
- Use of lowerCamelCase for Attributes,
- Names should be in their singular form, except if the concept is plural,
- Names can include several terms, but use of articles and prepositions are prohibited,
- Use of Acronyms and Abbreviations should be explained in definition, and a good practice is to control the list of allowed acronyms and abbreviations.

Enumeration names always ended up with the “Kind” term: “EnumerationKind”.

NOTE: There are several standard CIM classes that do not comply with this naming convention. This is an instance where backwards compatibility supersedes a modelling rule.

B.5. Information Model modelling rules

At the information model level:

- All attributes and associations are optional,
- Allowed attribute types are Primitive, enumeration, Datatype and compound,
- Attribute types are the least restricted ones,
- All associations are bidirectional ones,
- All classes, attributes and association end role names follow naming rules.

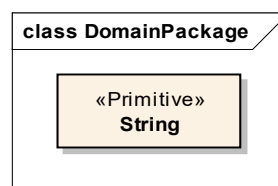
B.6. Datatype Types and UML modelling rules

There are different types of datatypes: basic and domain datatypes.

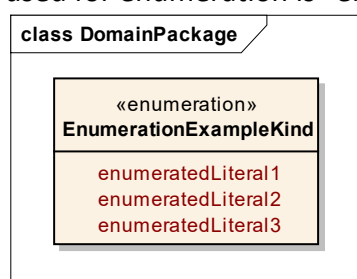
Basic Datatypes:

By definition, a datatype is a set of distinct values, characterized by properties of those values and by operations on those values (see ISO/IEC 11404 standard). According to this standard, there are three kinds of basic datatypes:

- Primitive: a primitive defines a value space axiomatically, without reference to any other value spaces. In UML, primitives are represented as a classifier with no attributes. Here, the stereotype used for a “*user primitive*” is “Primitive” with an upper case to distinguish it from UML own primitives (that are stereotyped with a lower case “primitive”) or even from UML tool primitives:



- Enumeration: an enumeration is a subset of a primitive value space and defines a list of allowed values taken in the primitive value space. In UML, enumerations are represented as a classifier with “*enumeratedLiterals*” not with attributes. Stereotype used for enumeration is “enumeration”:



NOTE: It is strongly recommended to use the Enterprise Architect built-in data type “Enumeration” instead of defining a stereotype. The built-in data type is already stereotyped by ‘enumeration’.

- **Simple Datatype:** a simple datatype defines a subset of a primitive value space by some kind of restrictions. Simple Datatypes are represented as a classifier with one attribute named “value”, whose type is a primitive, and whose primitive value space restrictions are expressed as value attribute constraints. Here, stereotype used for simple datatype is “Datatype” to distinguish it from usual UML datatypes (that are stereotyped with “datatype”). There is a relationship between the primitive and the datatype, and this relation is an “*IsBasedOn*” one (see sections on profile and on Datatype IsBasedOn process)

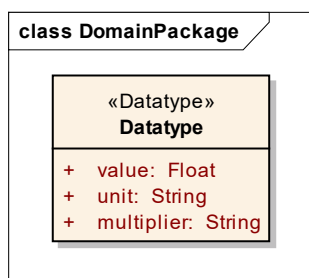
Domain Datatypes

Domain datatypes are special datatypes that, alongside with the value space definition, gives the domain where the value space is meaningful, by adding some extra information to express this domain. CimDatatypes, Core and Business UN/CEFACT datatypes are a good example of these datatypes.

Domain datatype defines:

- a primitive value space or a subset of it by some kind of restrictions, as does simple datatype,
- the value domain for this value space with some extra information expressed as supplementary attributes alongside with the “*value*” attribute (in case of CimDatatype) or “*Content*” attribute (in case of UN/CEFACT datatypes).

Domain Datatypes are represented as a classifier with one attribute named “value”, and supplementary attributes. In CIM, those attributes are: unit, multiplier, denominatorUnit and denominatorMultiplier. Stereotype used for “*Domain datatypes*” is “*Datatype*” (or user defined with some prefix added to Datatype):



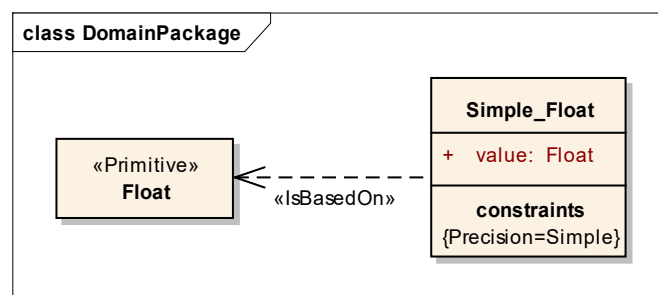
B.7. Primitive value space restrictions

The value space of a <<Primitive>> can be restricted. Each <<Primitive>> value space has its own kind of allowed restrictions (or facets). The list of allowed facets has been defined: most of them are like the ones defined for XML datatypes, but there are some more.

Primitive Type	Constraint/facet
Boolean	no facet
String	String_kind (normalizedString, token, NMTOKEN, Name, NCName, anyURI, ID, IDREF)
	length
	minLength
	maxLength
	pattern
	whiteSpace
	enumeration
Integer	Integer_kind (long, int, short, byte, unsignedLong, unsignedInt, unsignedShort, unsignedByte)
	totalDigits
	minInclusive
	maxInclusive
	minExclusive
	maxExclusive
	enumeration
Float	precision (simple, double)
	minInclusive
	maxInclusive
	minExclusive
	maxExclusive
	enumeration
	pattern

Decimal	totalDigits
	fractionalDigits
	minInclusive
	maxInclusive
	minExclusive
	maxExclusive
	enumeration
TimePoint <i>note this Primitive type is missing in CIM, here it is used for defining time-related facets</i>	timePoint (dateTime, date, time)
	minInclusive
	maxInclusive
	minExclusive
	maxExclusive
	enumeration
	pattern

When the value space of a <<Primitive>> is restricted, it specifies a new value space that is represented by a <<Datatype>> that has just a value attribute. This <<Datatype>> has an “*IsBasedOn*” relationship with the <<Primitive>> (we say it is “*IsBasedOn*” the <<Primitive>>).

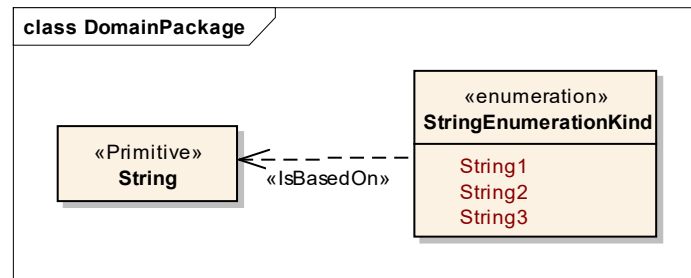


B.8. Enumeration value space

The list of the enumerated literals of an <<enumeration>> defines the allowed values in a value space that is defined by another datatype (primitive, other enumeration or simple datatype): so an

<<enumeration>> has always an “*IsBasedOn*” relationship with a datatype (and at first with a primitive).

Example: an enumeration whose “*enumeratedLiterals*” are String:

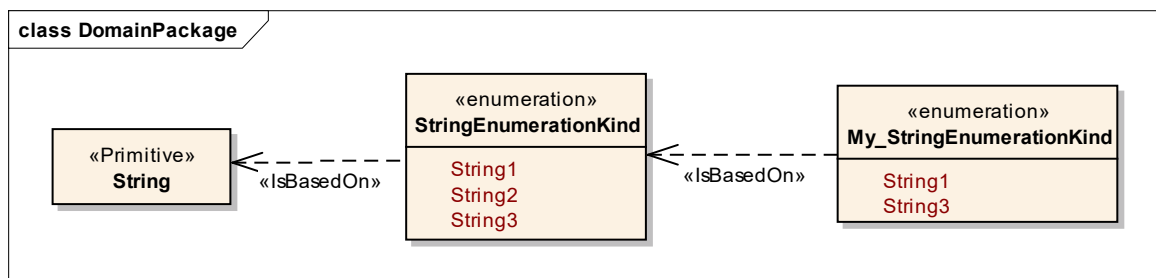


B.9. CIMDatatype “IsBasedOn” process

As stated above, a simple <<CIMDatatype>>, whose value space is a subset of a <<Primitive>> has an “*IsBasedOn*” relationship with this primitive.

As stated above, all <<enumerations>> have an “*IsBasedOn*” relation with a <<Primitive>> (or a <<CIMDatatype>> or an enumeration that have themselves an “*IsBasedOn*” relationship to a <<Primitive>>).

An <<enumeration>> could be “*IsBasedOn*” on another one: in this case its enumerated literals are a subset of the upper-level ones,



A <<CIMDatatype>> can be “*IsBasedOn*” on another one. All the rules that apply on the “*IsBasedOn*” class process are applying to “*IsBasedOn*” datatype process.

All the restrictions (or “*IsBasedOn*” process) could be done both at Information Model Level or Profile Level. Good practices are:

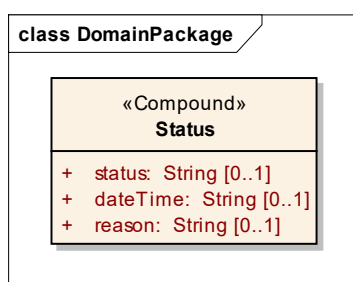
- To group all datatypes (Primitive, enumerations and CIMDatatypes) in a “Domain” Package both at Information Model level and Profile level,
- To put all <<Primitives>> at the Information Model level.

All this defines a hierarchy of “*IsBasedOn*” datatypes (Primitives, enumerations and CIMDatatypes).

Important: Primitive, enumeration, CIMDatatype (and Compound) names MUST be unique in the Information Model and Profile Model name space if there is a need to share all these elements.

B.10.Compound

Compounds are a special kind of attribute typing: they define a named group of attributes. They are stereotyped by the term “*Compound*”. A Compound follows the same rules for naming as classes and follow the same rules as classes for the “*IsBasedOn*” process (see section 24 Annex C: profiling rules). The only difference with classes is that they cannot be instantiated as objects and that they do not have associations.



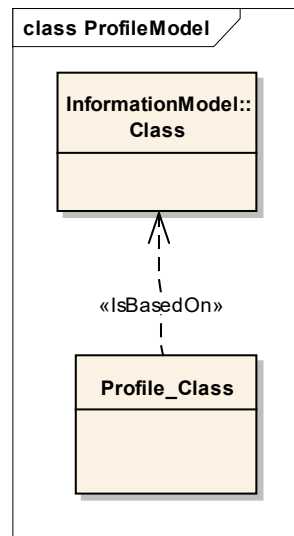
C. Annex C: Profile UML modelling rules

C.1. Profiling rules:

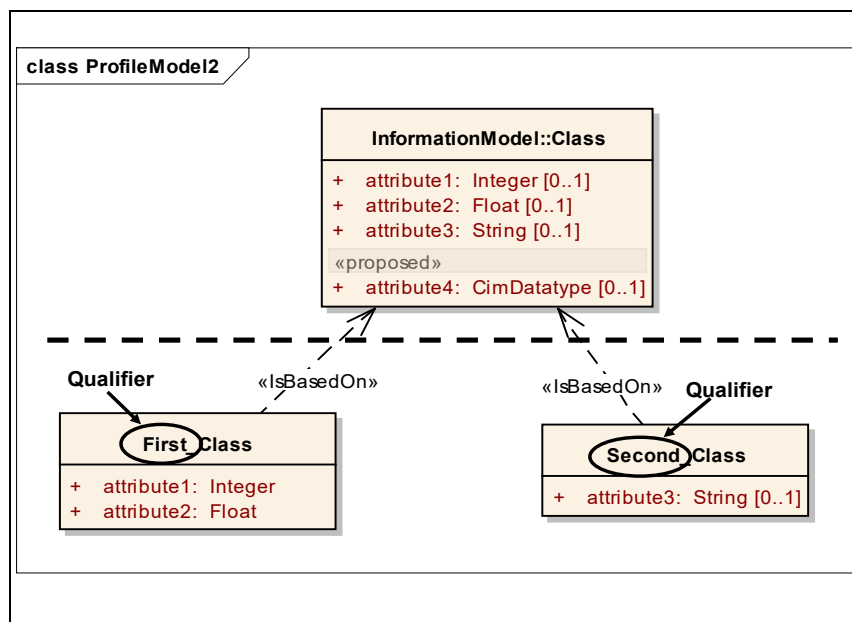
Profile is a subset of another UML Model. The upper model level is the Information Model. Nothing can be added at the profile level that is not already expressed at the upper level (usually the Information Model Level): no new classes, no new attributes and no new associations.

C.2. “IsBasedOn” for Profile Classes

A UML Profile Class is derived from the UML Information Model Class: the profile class is “*IsBasedOn*” (“IBO”) on the upper level one. In *CimConteXtor*, “*IsBasedOn*” is marked as a “*Dependency*” association stereotyped with <<IsBasedOn>>.



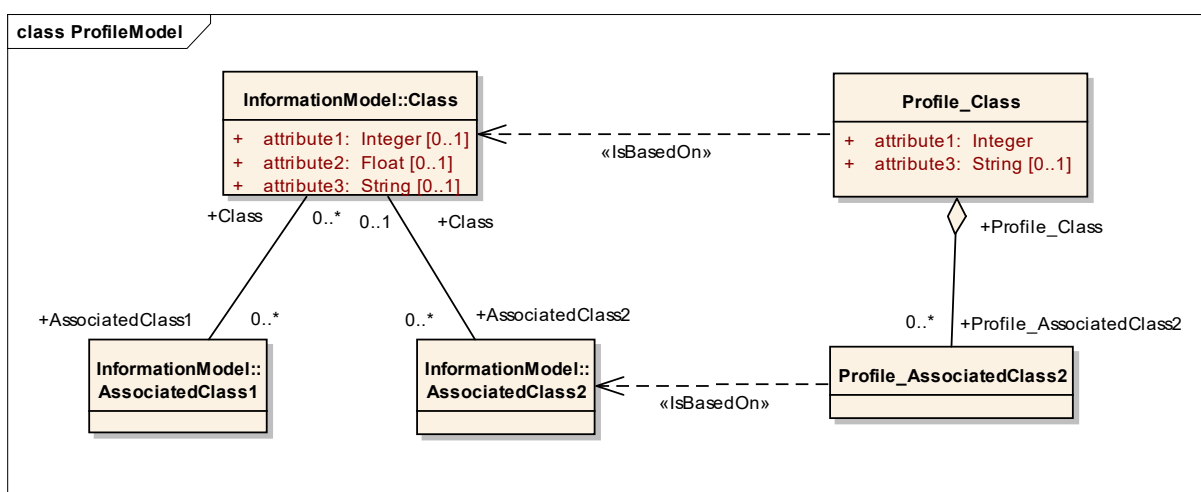
Several Profile classes can be based on the same information model class: in this case the names of these profile classes should be differentiated by the use of qualifiers (see naming rules):



A Profile UML class (*“IsBasedOn”* class) can:

- have as attributes only the attributes that are defined in the class on which it is based,

- have as attributes all or a subset of the attributes that are defined in the class on which it is based,
- have as associations only the associations that are defined in the class on which it is based,
- have as associations all or a subset of the associations that are defined in the class on which it is based.



C.3. “IsBasedOn” for Profile class attributes

A Profile class attribute can:

- have a cardinality that is the same or a more restricted one than that of the corresponding attribute in the information model class,
- have a type that is the same or a more restricted one than that of the corresponding attribute in the information model class (see Datatypes and Compounds section).

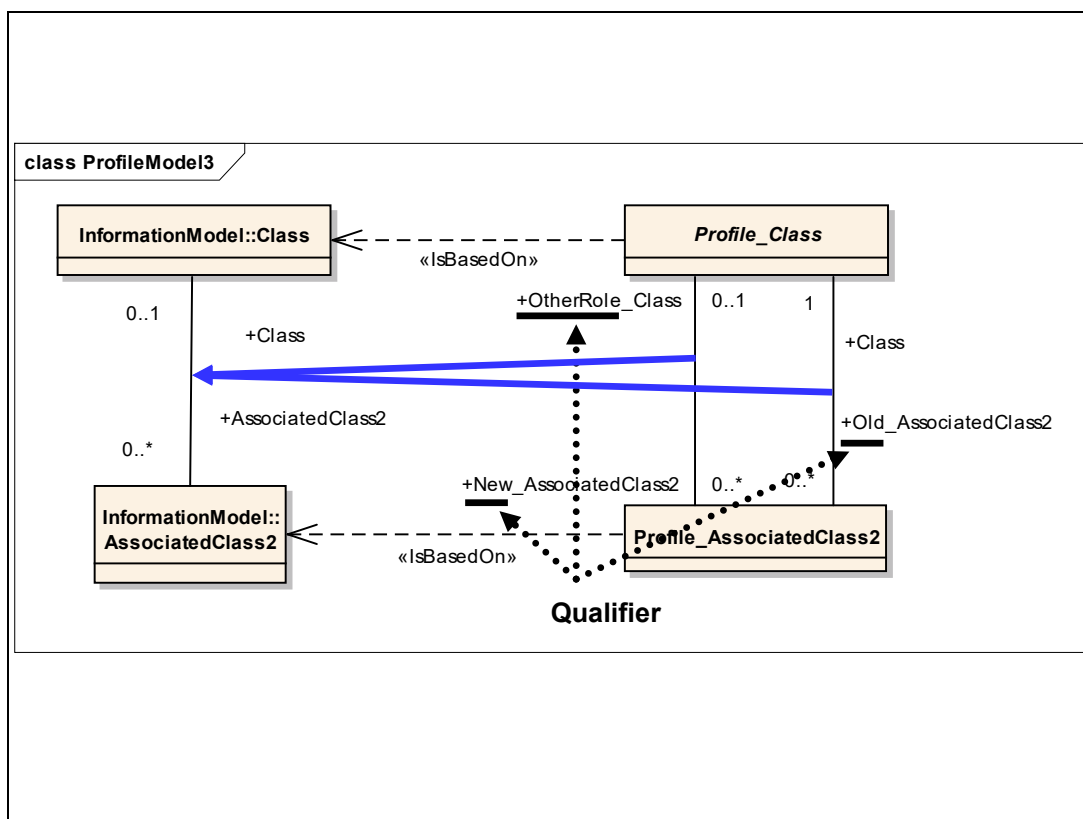
C.4. “IsBasedOn” for Profile Class associations

A Profile class association could:

- have a cardinality that is the same or a more restricted one than that of the corresponding association in the information model class,

- have a type that is the same or a more restricted one than that of the corresponding association in the information model class, for example the association could become an aggregation,
- have role end names that are more refined or specialized than that of the corresponding association in the information model class,

Several Profile class associations can be based on the same information model class association: in this case the end role names of these profile associations should be differentiated by the use of qualifiers (see naming rules).



C.5. Profile naming rules

At the Profile level, the names of Classes, Attributes and Associations are the same as their corresponding counterpart in the information model level: but all classes and association names can be “qualified” to differentiate them when there are several classes or associations that are based on the same class or association of the information model.

Qualifier:

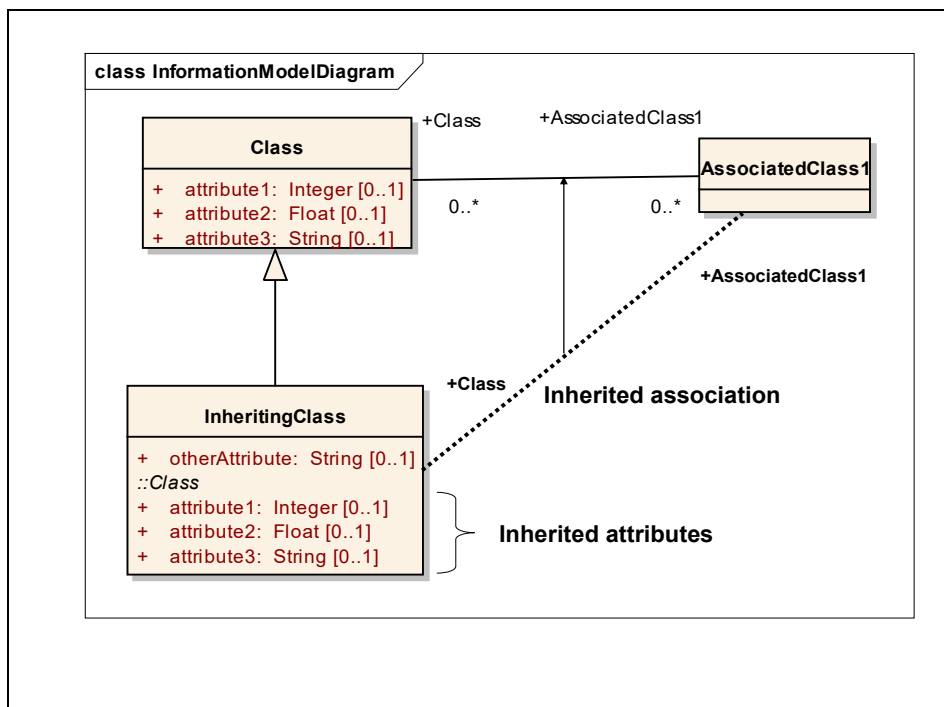
- Qualifiers follow the same naming rules as class and association names,
- Qualifiers precede the class or association names and are separated from them

by an “underscore” (_).

D. Annex D: Profile rules for inheritance

D.1. Inheritance principles

A Class could inherit from another class (its “*super Class*”): it means that it inherits the super Class properties (attributes and associations). At the Information level all classes are concrete classes.



“*InheritingClass*”

- has its own attribute named “*otherAttribute*”,
- inherits from “*Class*”, so :
 - it inherits attributes named “*attribute1*”, “*attribute2*” and “*attribute3*” from “*Class*”,
 - and it inherits association with “*AssociatedClass1*”, association whose end role names are “*Class*” and “*AssociatedClass1*”.

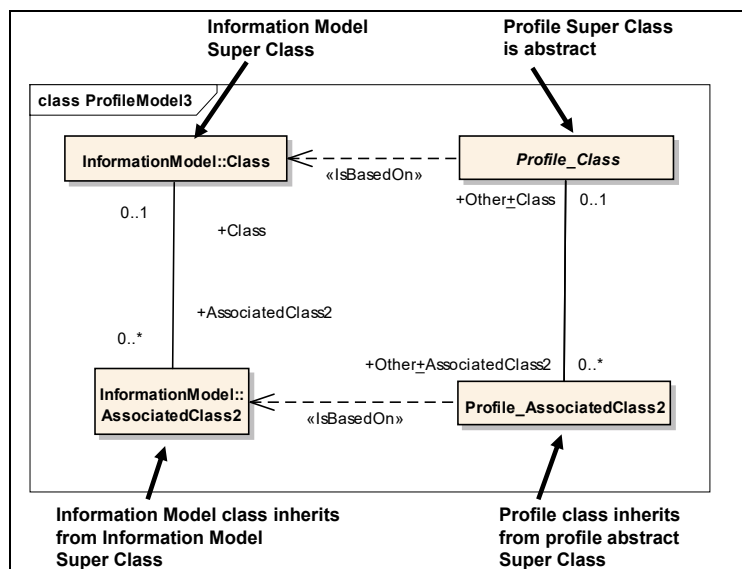
D.2. Profile level class inheritance rules

At Profile level, a profile class can inherit from a profile superclass only if this profile superclass is abstract (in UML abstract classes have a name in italics).

Profile classes are still “*IsBasedOn*” on an Information Model level class, whether they are profile simple or super classes.

Profile classes can inherit from another Profile Class (its superclass) only if the two Information Level Classes that serve for the “*IsBasedOn*” process have themselves an inheritance relationship.

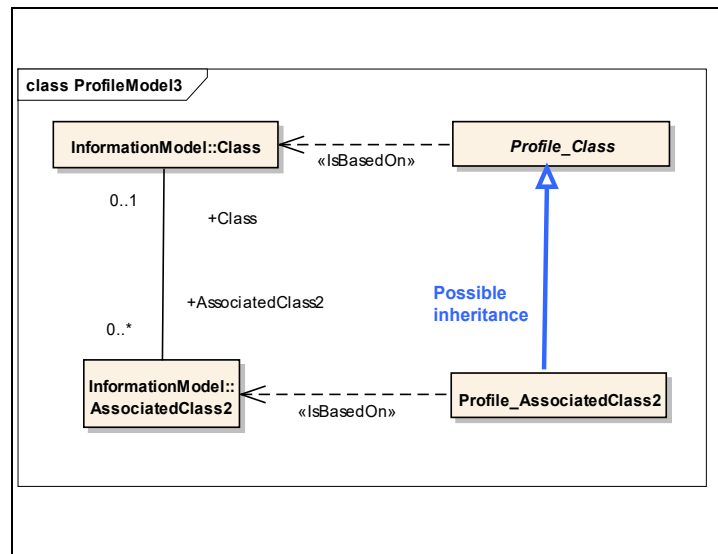
The inheritance rules between profile classes are the usual inheritance rules.



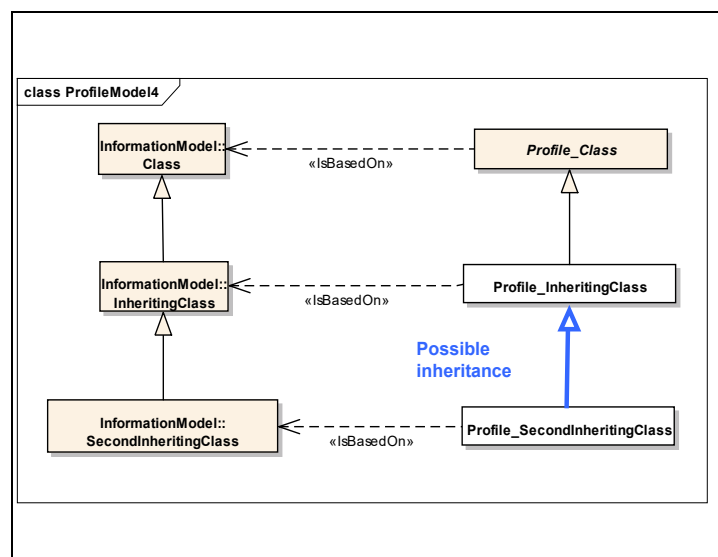
D.3. Profile level “IsBasedOn” class inheritance rules

At Profile level, there are several inheritance possibilities for a profile class. The profile class inherits from a profile superclass that:

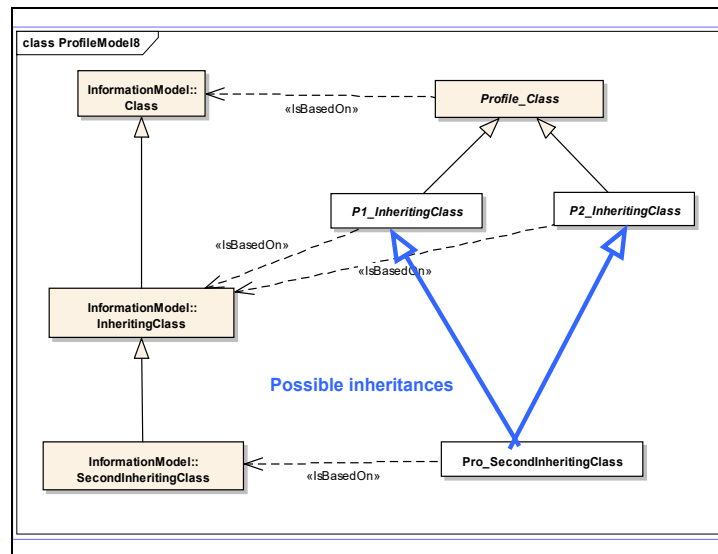
- stands by itself,



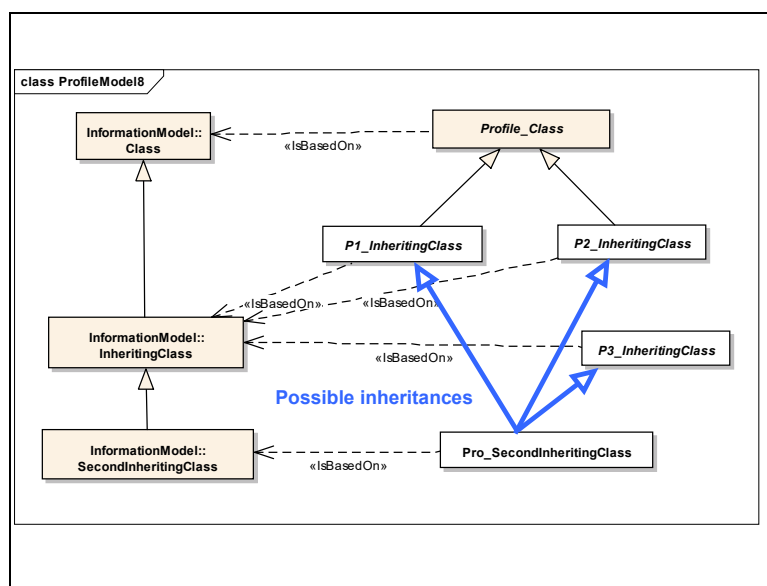
- is the last one of a profile inheritance hierarchy that mimics the information model level inheritance hierarchy,



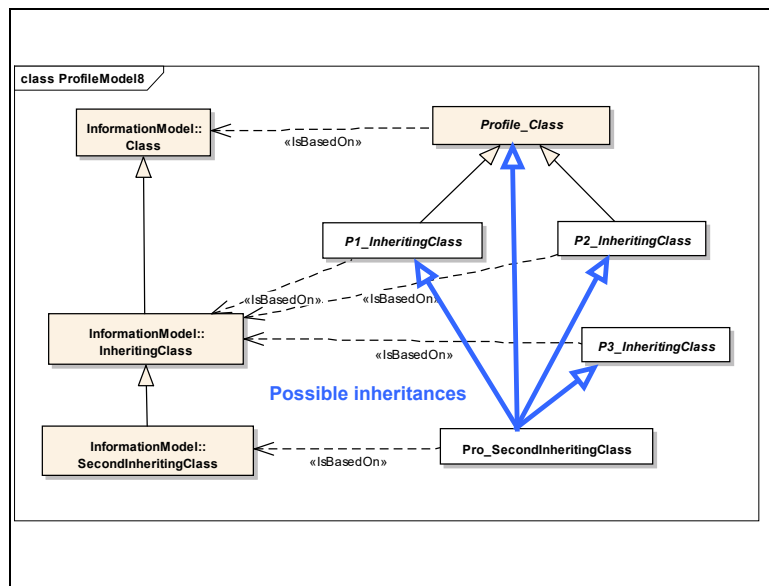
- is one of the profile super classes that are based on the same information model superclass,
- first example



- second example

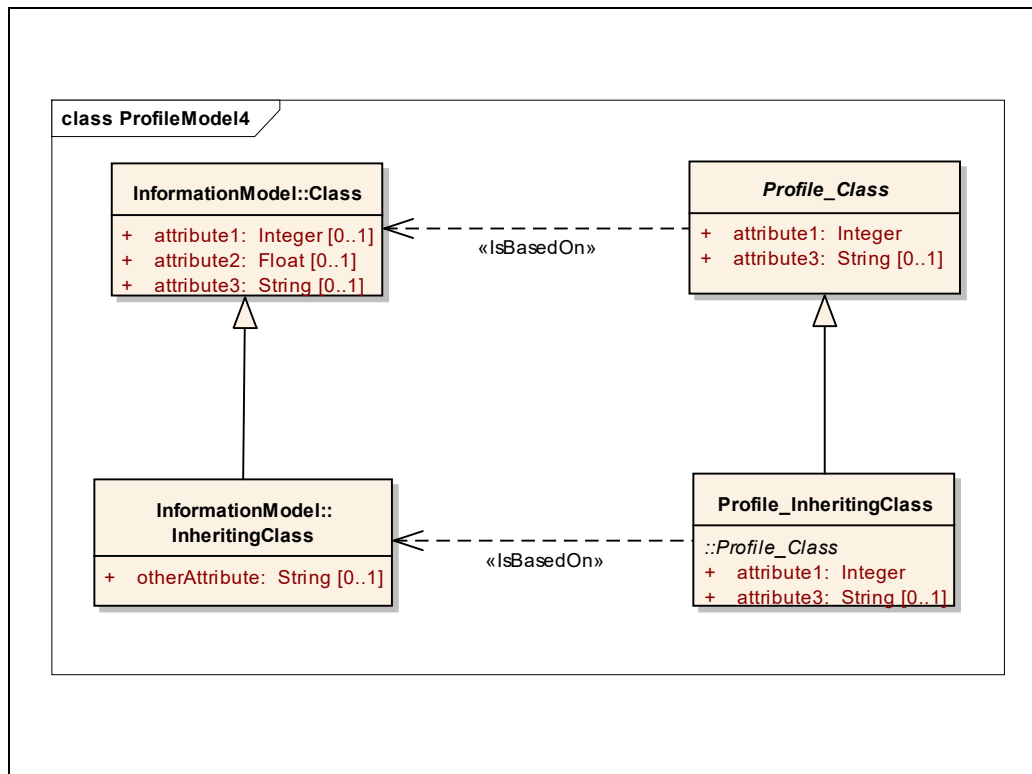


- Finally, it could be any of the profile super classes that are based on one of the information model level hierarchies :



D.4.Profile level attributes inheritance rules

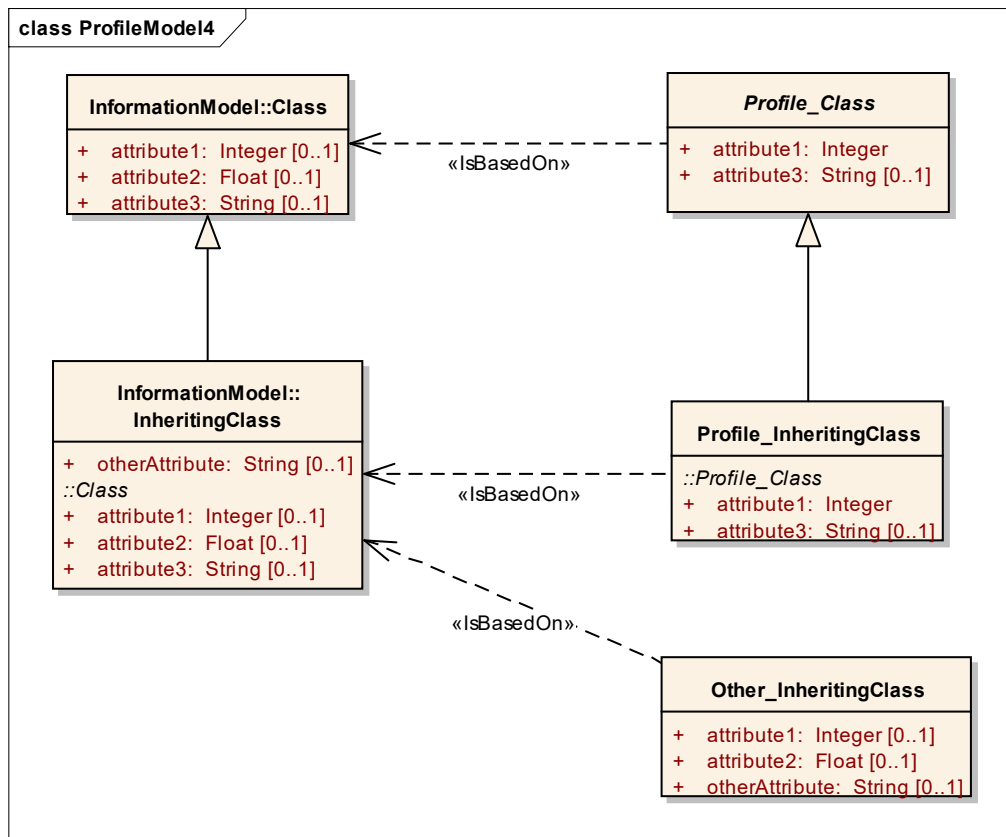
At profile level, the profile class attributes could be inherited from the profile superclass and could come from the Information Model Class on which it is “*IsBasedOn*”.



"ProfileClass" is an Abstract Class : it is "IsBasedOn" on "Class" (from Information Model). "Profile_InheritingClass" is "IsBasedOn" on "InheritingClass" (from the Information Model) and inherits from "Profile_Class".

So, at Profile level, there could be different cases of "IsBasedOn" classes, see the following example:

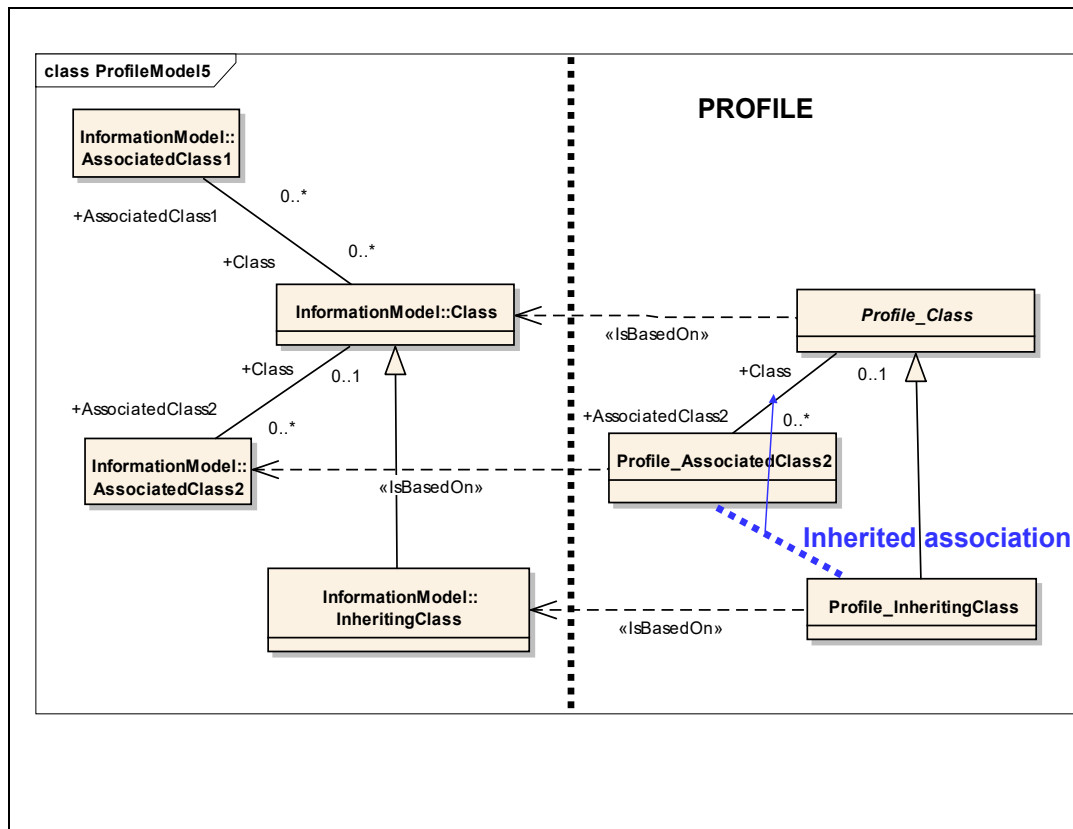
- A Profile Class named "Profile_InheritingClass" that is "IsBasedOn" the Information Model "InheritingClass" and inherits from the abstract Profile_Class named "Profile_Class",
- and a Profile Class named "Other_InheritingClass" that is "IsBasedOn" the same Information Model "InheritingClass".



D.5.Profile level association inheritance rules

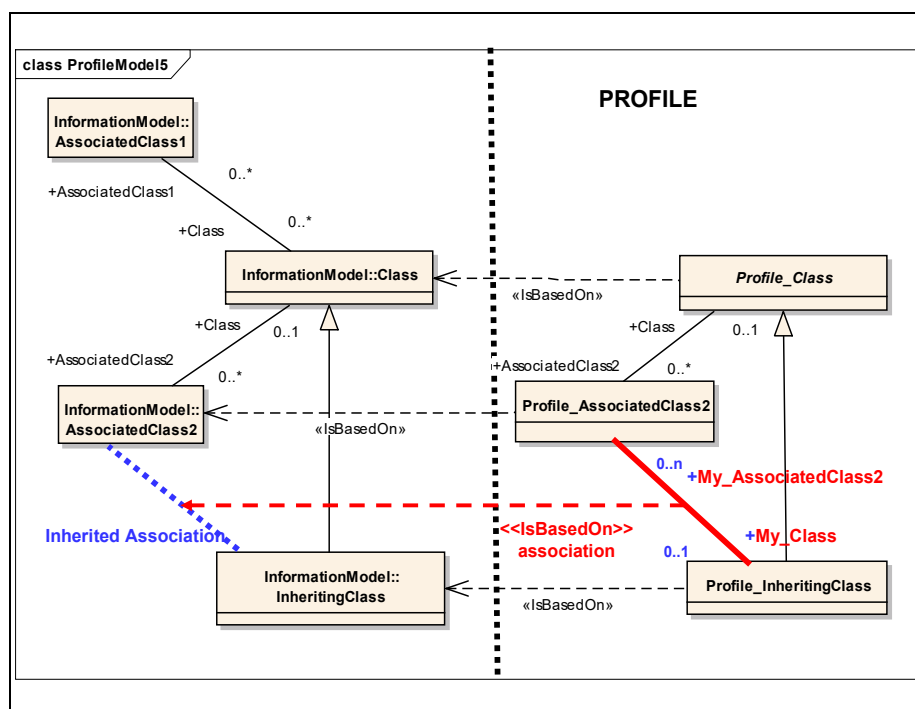
At profile level, the profile class associations could be inherited from the profile superclass and could come from the Information Model Class on which it is “*IsBasedOn*”.

Example of inherited profile associations:



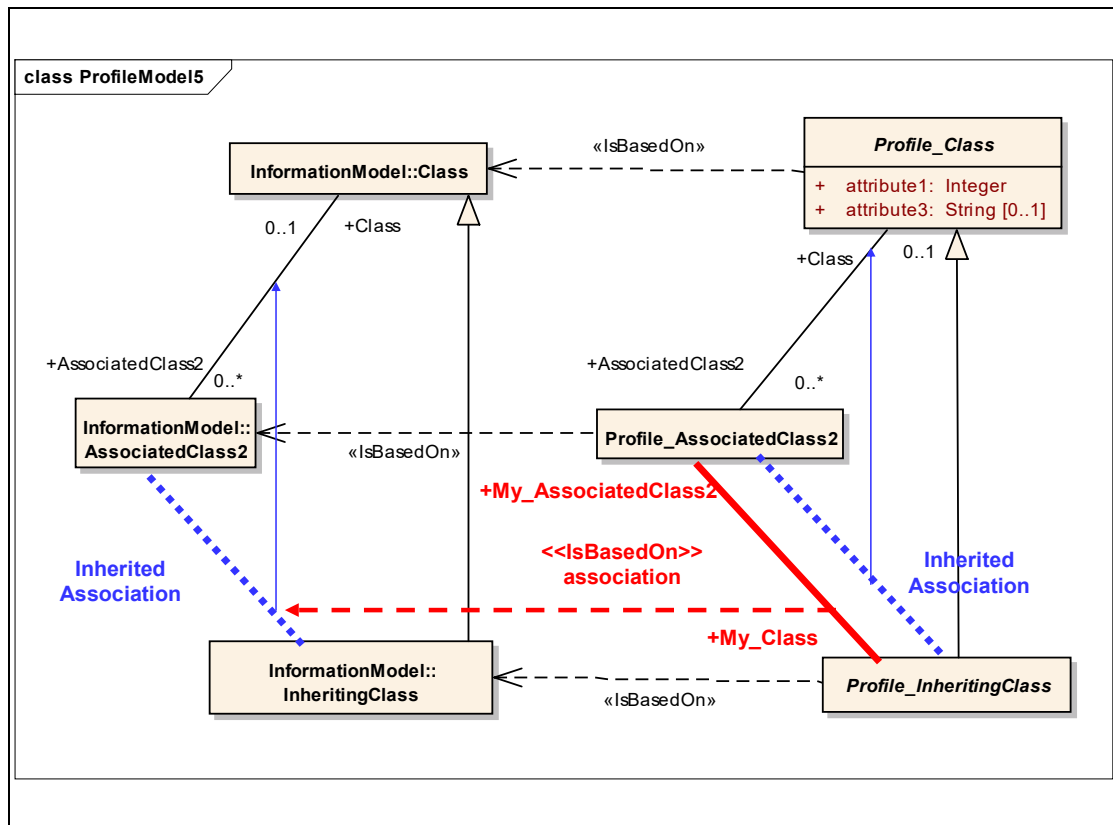
"Profile_InheritingClass" inherits from "Profile_Class" and thus inherits the association with "Profile_AssociatedClass2".

Example of an "IsBasedOn" association for the profile class that has a profile superclass:



"Profile_InheritingClass" is "IsBasedOn" on Information Model Class named "InheritingClass", thus it could have an association with "Profile_AssociatedClass2" because at Information Model level an association exists between "InheritingClass" and "AssociatedClass2".

So, a profile class could have both "Inherited" and "IsBasedOn" associations. But one has to remember that all these associations should be differentiated by the use of appropriate "Qualifiers".



"Profile_InheritingClass"

- inherits an association with "Profile_AssociatedClass2". The end role names are "Class" and "AssociatedClass".
- has an association of its own with "Profile_AssociatedClass2". The end role names are "My_Class" and "My_AssociatedClass".

Note: the use of qualifiers like "My" to distinguish the associations by their end role names.

D.6. Use of Association CHOICE Constraint with inheritance rules

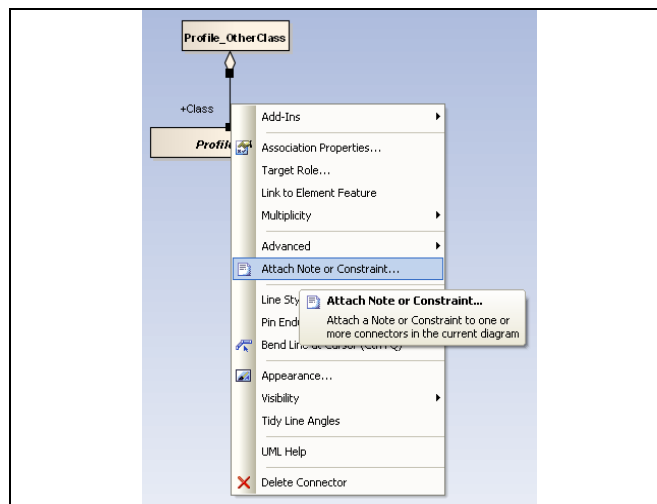
When a profile class has an association with a profile abstract class that has subclasses (inheriting classes): this means that this profile class has an association with each profile subclass. If we want

to express that these associations are XORed or that at instance level a profile class can be associated with only one profile subclass, we must put a constraint somewhere.

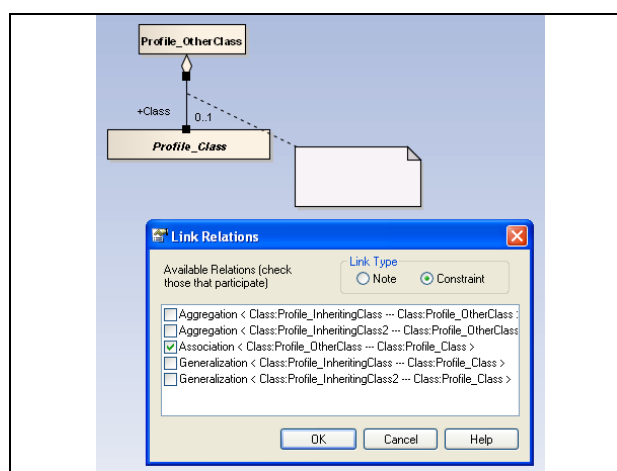
There are two ways to do that:

- Express the constraint on the association that will be inherited (i.e. between profile class and profile abstract class),
- Or express it on the inherited associations between profile class and profile subclasses.

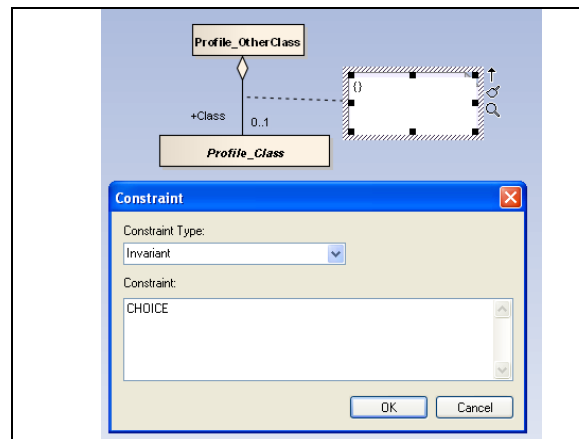
In the first case, a “CHOICE” constraint could be put on an association between the profile class and the profile abstract class, using the EA association “Attach Note or Constraint” Menu:



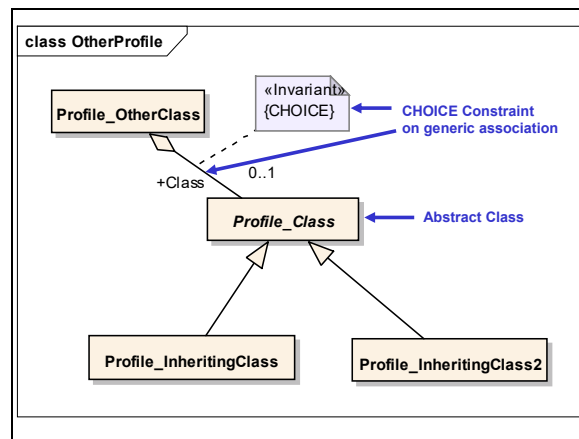
Select Constraint and association:



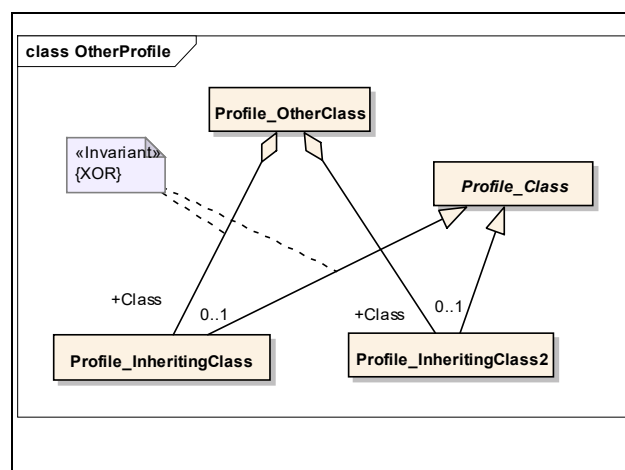
Enter “invariant” as constraint Type and “CHOICE” as constraint:



This will attach the “CHOICE” constraint to the association:



This is the same as putting a XOR constraint on the inherited association for the inheriting class:



WARNING: the attached XOR or CHOICE constraint note must be recreated each time the “Edit connectors” is reused.

E. Annex E: Information Model Extension

E.1. General

When extending Information Model with new packages:

- Use packages to create manageable and logical model pieces.
- Before creating a new package, do a search for the intended package name; if such a package already exists in the model, consider using another name or adding a prefix to the name of the new package.

When extending the information with new classes or associations:

- Avoid creating new primitive data types. Reason is that Information Model (CIM) is a semantic model that stays away from the extensive data typing available in many implementation technologies.
- Avoid creating synthetic domain objects that do not represent objects in the domain. Reason is that naming and identification of synthetic objects do not align with the problem domain and create extra administration.
- Before creating a new class, do a search for intended class name; if such a class already exists in the model, consider using another name.
- Create new domain object classes by sub classing an existing class, e.g. IdentifiedObject, Device, Schedule, etc. If it is not possible to subclass the object, an association between the two objects may be created. When adding an association, the information from the use case that motivated the addition is typically lost. A way to avoid this loss is to include the information in the documentation of the attribute or association end.
- When defining a new subclass, display on a diagram the superclass and its related classes with all their attributes to avoid inadvertent duplication: (a) either with the same attribute/association end names that get inherited, or (b) by duplicating an existing concept but with a different name.

E.2. Custom Information Model extensions

E.2.1. General

A custom extension is adding an element to the standard IM with the goal of keeping this element distinct from the IM elements, using another namespace.

The objective of these recommendations is to allow for clear modularization of extensions such that they can be readily identified as extensions and migrated more easily to updated versions of the standard IM, if needed,

Extensions to the IM model should be included in the extension package just as any other IM modelling except for the differences noted in this clause. The extension package should have a stereotype to identify all its contents as extensions to the IM information model, i.e. «CIMExtension». The extension package should also introduce a new namespace and have a CIMVersion Class.

The sub-package structure of the extensions may pattern after that of standard IM packages, but the package structure should reflect the needs for individually managing and organizing parts of the extensions.

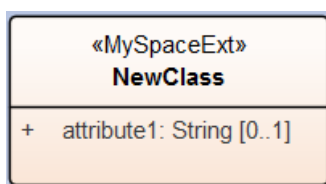
It is recommended to utilize existing data types and IM classes where possible. New classes can be created where new business objects concepts are introduced with their own lifecycle.

There are several cases for making an extension:

- Adding an extension class,
- Adding an extension attribute to an IM class,
- Adding an extension enumerated literal to an IM enumeration,
- Adding an extension association between an extension class and an IM class,
- Adding an extension association between two IM Classes.

E.2.2. Adding an extension class

Name the new extension class. Give the class the stereotype used for the specific extension (Example MySpaceExt). Add the class attributes.



Do not forget to give definition to the class and attributes.

E.2.3. Adding an extension attribute

In case of adding an extension attribute to an IM existing class, a method of using multiple inheritance is allowed.

Name the new extension class with the same name as the standard IM class with the suffix "Ext". Stereotype it with the stereotype used for extension (Example MySpaceExt). Add the needed attribute with the stereotype used for extension (Example MySpaceExt).

Give the new class a generalization relationship from the standard IM class such that the standard IM class is a specialisation of the extension class. Generally, the extension class will have no base class, and the standard IM class will now have multiple inheritance. The generalization relationship should have the stereotype with the stereotype used for extension (Example MySpaceExt). Next figure shows an example of adding an extension attribute to an existing IM class:

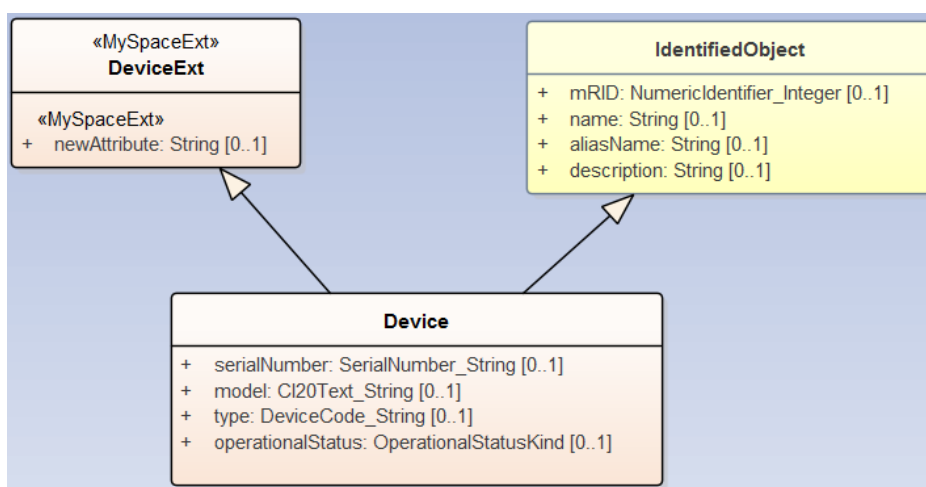


Figure 1: adding an extension attribute

E.2.4. Adding an extension enumerated literal

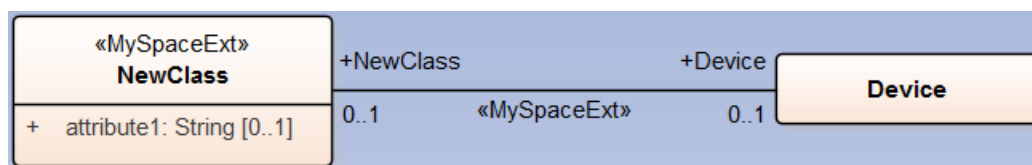
In case of adding an extension enumerated to an IM existing enumeration, the process is the same that one used for adding an attribute, except there will be no multiple inheritance because Enumerations do not inherit from another class.

Name the new extension enumeration with the same name as the standard IM enumeration with the suffix "Ext". Stereotype it with the stereotype used for extension (Example MySpaceExt). Add the needed enumerated literal with the stereotype used for extension (Example MySpaceExt).

Give the new enumeration a generalization relationship from the standard IM enumeration such that the standard IM enumeration is a specialization of the extension enumeration. The generalization relationship should have the stereotype used for extension (Example MySpaceExt).

E.2.5. Adding an extension association between extension and IM classes

The typical case of adding a new extension class and associating with existing IM classes can be modelled in a diagram within the extension package as shown in next figure. This situation does not require multiple inheritance.



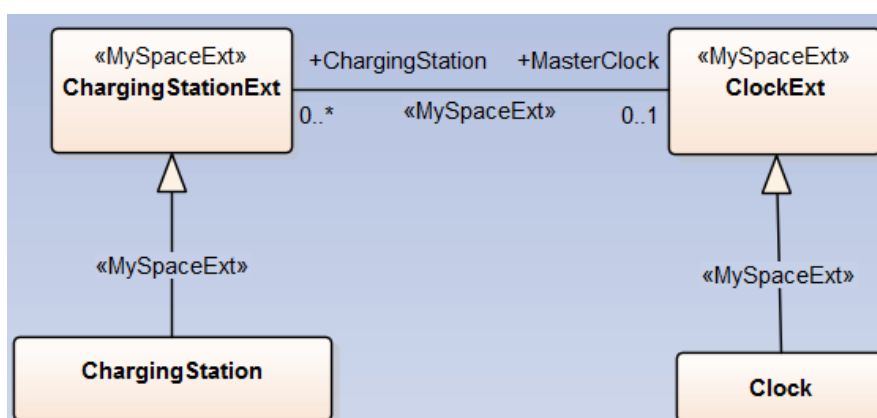
Note that the introduction of an association from a new extension class to an existing standard IM class requires nothing special, just make the association as it is implicitly owned by the depending extension class's package, but start to draw the association from the extension class. It is clearer to put both ends of the association into the extension package as the namespace for both association ends will follow IM package containment rules which rely upon package dependency.

E.2.6. Adding an extension association between two IM classes

In the case of a new association with both ends referencing existing classes, a method of using multiple inheritance is allowed.

Make two extension classes corresponding to the IM classes that should get a new association. Name the new extension classes with the name of the standard IM classes suffixed by Ext. Give the classes the stereotype used for the specific extension (Example MySpaceExt). Give the new classes a generalization relationship from the standard IM classes such that the standard IM classes are a specialization of the extension classes. The generalization relationships should have stereotype used for the specific extension (Example MySpaceExt).

The new association can be added to the new extension classes and they will be inherited by the standard M classes and in effect appearing as new properties of the standard classes. Next figure shows an example of adding an extension association between two IM classes:



An extension class should be used only once within a namespace. The extension class should have the extension generalization pointing to a standard IM class of the same name. A particular extension is an extension to the one and only standard IM class.

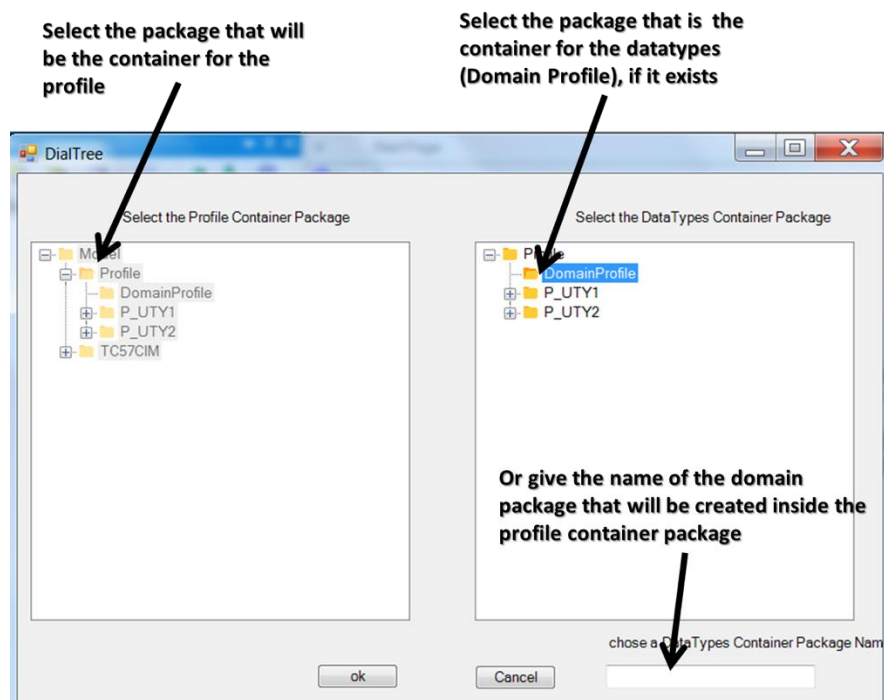
F. Annex F: CreateGlobalProfile description

"*CreateGlobalProfile*" is a feature to speed up profile building, like selecting an information model package and making a profile with all the classes of the package, plus the associated super classes. Same is done when starting from a diagram. These rules are profiling rules for profiles where you want to keep the inheritance path (example is WG13 or CGMES profiling style).

F.1. Starting CreateGlobalProfile

When selecting a package or diagram and launching *CreateGlobalProfile*, a pop up window will open:

- Asking to select the Package that will be the container for the intended profile,
- Asking to select the package that contains the datatypes, if it exists. If it does not exist, give the name of the package that will be created inside the Profile Container and will hold the profile datatypes.



Note: in this version there is only one package structure:

- *Profiles and Domain container package,*

- *Domain profile package (this means that the Domain profile is a global one and will be updated with new datatypes if a new profile is added with CreateGlobalProfile)*
- *Profiles packages.*

Note: in this version, by default the created profile package will be based on the Information Model packages: so check the right dependencies.

F.2. CreateGlobalProfile for a package

F.2.1. Package CreateGlobal Profile rules

The rules for package profiling are:

All selected package classes are copied in a profile package (including attributes), and the hierarchy of each class is replicated also in the profile. The name of the profile package is the name of the package prefixed by "P_" (Example package name is "TestUTY11", the profile package name will be "P_TestUTY11").

And:

- The profile class attribute keep the optional cardinality,
- The profile super classes are concrete,
- The profile super classes:
 - do not get their native attributes, if their associated CIM classes are not in the package that was selected for the CreateGlobalProfile,
 - get their native attributes, if their associated CIM classes are in the same package than the one that was selected for the CreateGlobalProfile.

For associations:

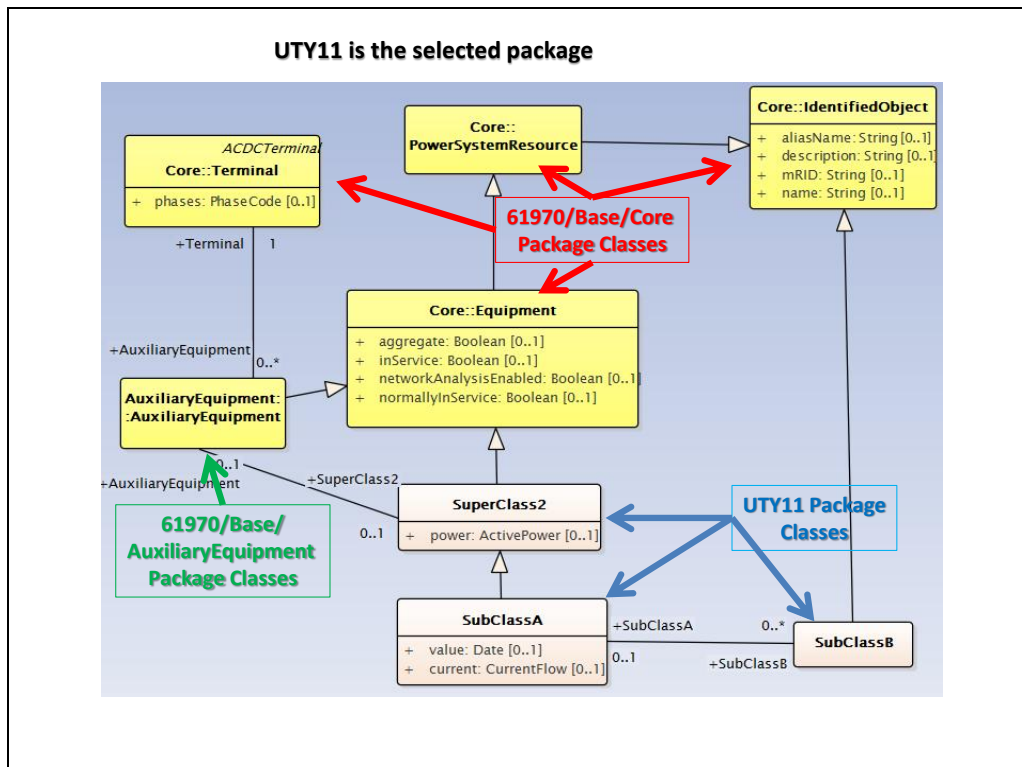
- associations between selected package classes are drawn without restrictions,
- association between a selected package class and a class outside the selected package, this outside class is put in a package of the profile, and the association is drawn without restrictions,
- all other associations are discarded,
- to refine association, you must use one of the Edit connector menus.

For diagrams:

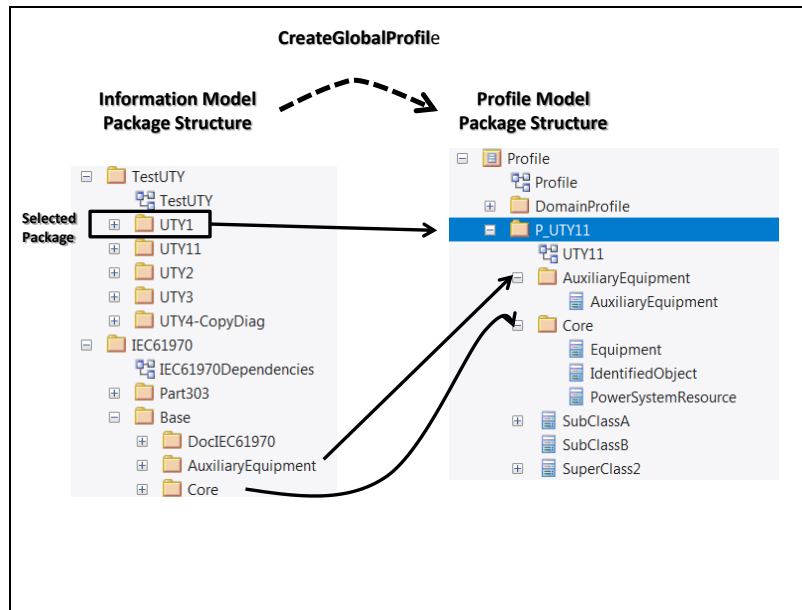
- Profile diagram is a duplicate of the selected package ones, with the above applied rules.

F.2.2. Example of rules for a package CreateGlobalProfile

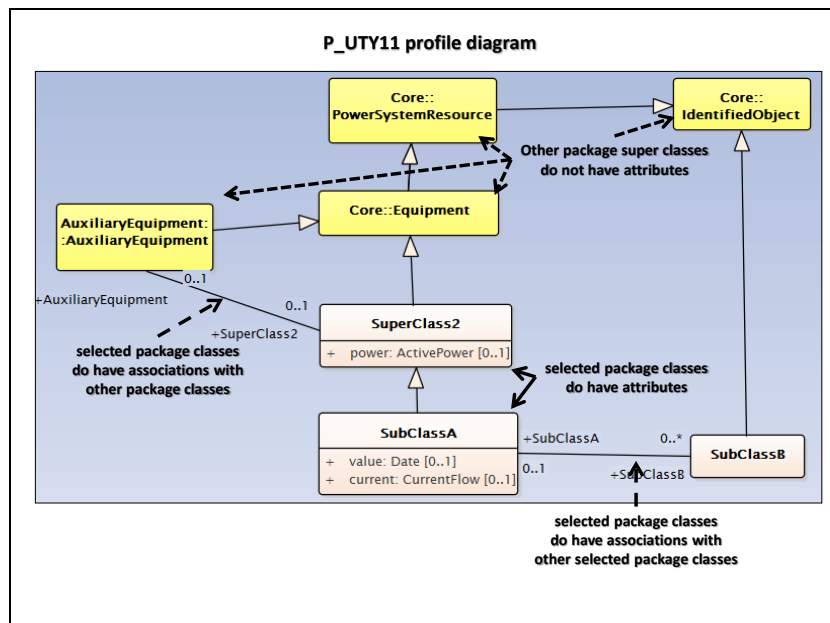
If we have a UTY11 package in the information model, that has some classes (SuperClass2, SubClassA and SubClassB): these are the selected package classes. These classes have relations with classes from other packages of the information model (Example: inheritance between SuperClass2 and Equipment or association between SubClass2 and AuxiliaryEquipment).



When launching the CreateGlobalProfile on the UTY11 package, a profile will be created as P_UTILITY package in the profile container with appropriate sub packages (in the example Core and AuxiliaryEquipment):



And a profile diagram will show some of the profile elements:



F.3. CreateGlobalProfile for a diagram

F.3.1. Diagram CreateGlobalProfile rules

The rules for diagram profiling are:

All selected diagram classes are copied in a profile package: the name of the profile package is the name of the diagram prefixed by "P_" (Example diagram name is "TestUTY", the profile package name will be "P_TestUTY"). Only classes from the selected diagram will be used in the profile.

And:

- The profile class (or super class) keep their attributes with the cardinality they have in the selected diagram,
- The profile classes keep the status they have in the selected diagram: concrete or abstract,

For associations:

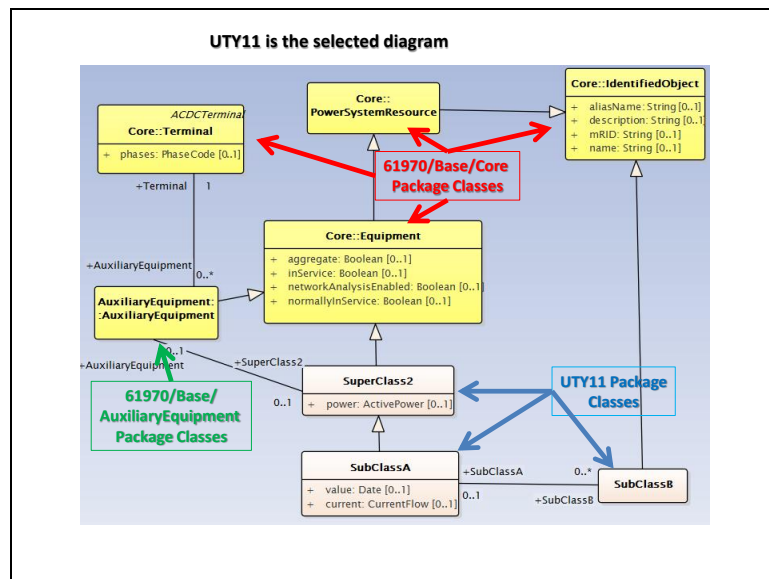
- associations between diagram classes are kept in the profile,
- association between a selected diagram class and a class outside the selected diagram, is discarded,
- to refine association, you must use one of the Edit connector menus.

For diagrams:

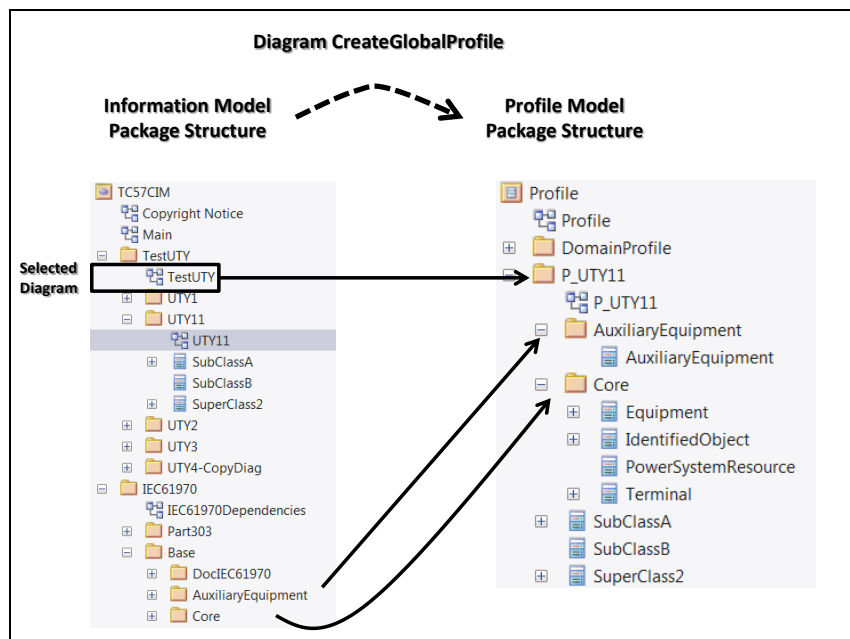
- Profile diagram classes are a duplicate of the selected diagram ones, with the above applied rules.

F.3.2. Example of results

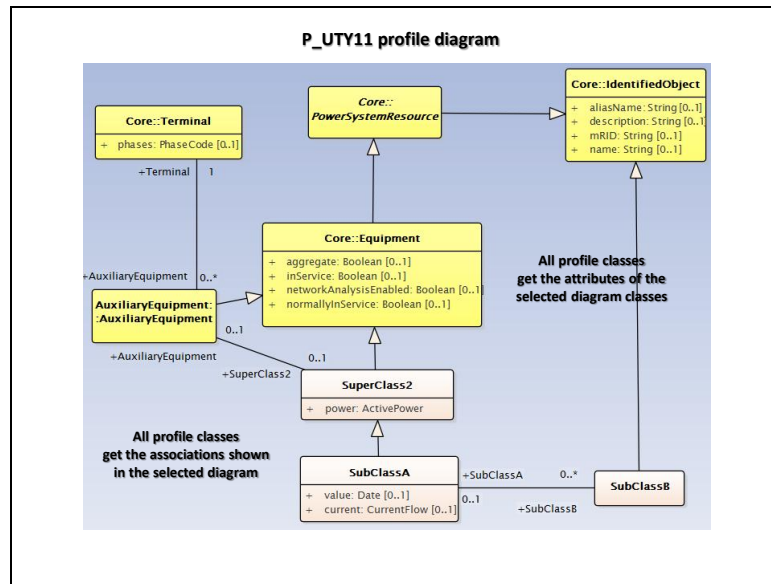
If we have a UTY11 diagram in the information model that displays some classes either from the package that contains the diagram (SuperClass2, SubClassA and SubClassB) or from another packages (IdentifiedObject, PowerSystemResource, Equipment, Terminal from a Core package or AuxiliaryEquipment from AuxiliaryEquipment package): these are the selected diagram classes. These classes have properties like attributes and relations shown in the diagram. All these properties will be kept in the profile.



When launching the CreateGlobalProfile on the UTY11 diagram, a profile will be created as P_UTILITY11 package in the profile container with appropriate sub packages (in the example Core and AuxiliaryEquipment):



And a profile diagram will show what is in the profile:



G. Annex G: Adapting profile to a new CIM version

When a new version of CIM is available, profiles based on a previous version might be impacted and aligned with some elements of the new version. Here are some proposals and procedure to handle changes:

- Export profiles package and extension package as XMI files,
- Import those files to the file with the new CIM version (because of the `IsBasedOn` features, all the relations between profiles and information model will be restored),
- Check with the revision information (e.g. an Excel file) that comes with the new CIM version what are the changes and if these changes impact the profiles (may be all changes will not be reported, a utility like Integrity check could also be used, but the current version need some upgrade to check everything). Changes could be:
 - Class name change: use EA class properties to change profile class name manually,
 - Attribute name change: use EA class attribute properties to change profile class attribute name manually,
 - Notes change: use `CopyAllNotes` for CGMES profile (For ENTSO-E ESMP, manually align when relevant, open question for other profiles),
 - Deleted class: delete profile class manually,
 - Deleted attribute: delete profile class attribute manually,
 - Deleted association: delete profile association manually, (or use `EditConnector`),
 - Added class, update profile if necessary, using Drag & Drop feature and `EditConnector`,

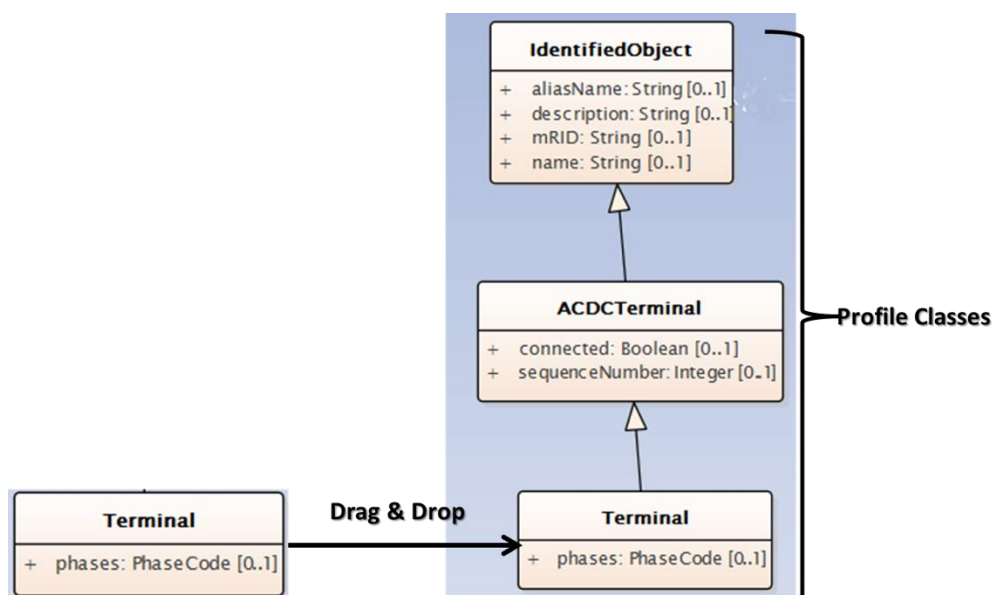
- Added Attribute, update profile if necessary, using EditIsBasedOn,
- Added Association, update profile if necessary, using EditConnector,
- Change or update of TaggedValues are done with EditIsBasedOn,
- Change or update of other items like Constraint, initial value, stereotype must be done manually for classes,
- Change or update of CIMDatatype supplementary attribute fixed initial value is done using EditIsBasedOn on the profile CIMDatatype. (Note: for default value, it must be done manually with EditIsBasedOn),
- Attribute type change (TBD),
- CIMDatatype change: (TBD).

A future version of CimConteXtor will offer some solutions for these issues.

H. Annex H: Enhanced Drag and Drop for IsBasedOn classes

This kind of feature is used for example by WG13 or CGMES profiling style. This feature is activated by checking the option "WG13AutomaticAncestorInProfile" in the configuration file or in the option menu.

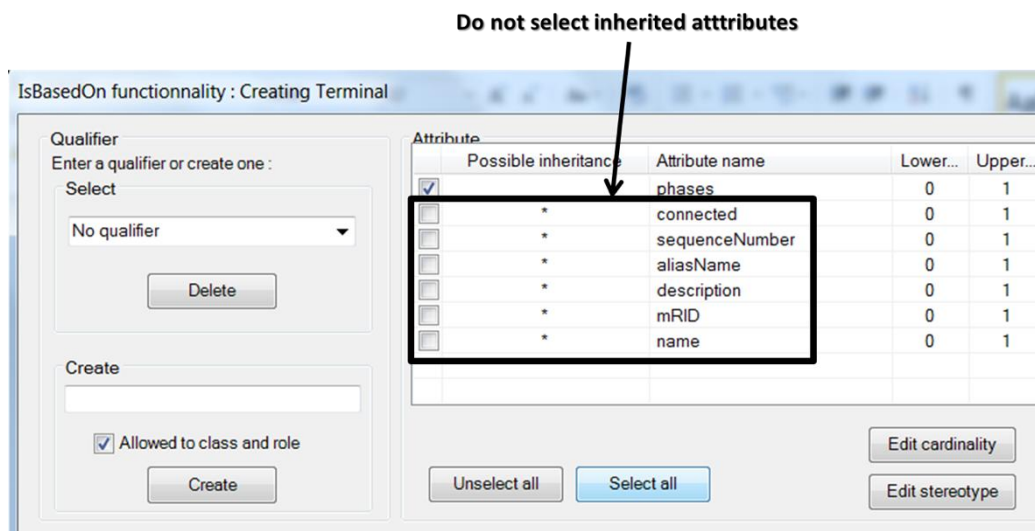
The main difference with the basic drag and drop function is that, at the end of drag and drop operation, the profile class is created with all its super classes. Example: drag and drop of Terminal Class will lead to have three profile classes: Terminal, inheriting from ACDCTerminal class, inheriting from IdentifiedObject.



Note: in this version of CimConteXtor:

- The super classes are not abstract, so with the EditIsBasedOn feature they should be made abstract,
- Each super class is coming with its attributes,
- Classes are created in the same package,
- The result is different from the one you get with CreateGlobalProfile.

Apart from the fact that at the end of the process the profile class is created with its super class(es), the procedure to specify the drag and drop profile is the same as the one described in section 10, except that you should not select inherited attributes (as shown here after):



Note: you will speed up your design if you always drag and drop the leaf classes first.

I. Annex I: IntegrityCheck Messages

This annex gives the list of the results of the different integrity check tests made.

I.1. CGMES Integrity Check List

Message	Description
NotIsBasedOn PackageName	The profile package must be IsBasedOn or in an IsBasedOn package, if it is not the case, a warning is issued
No associated message but test is done (abort)	All package names in a package profile must be unique, if it is not the case, a warning is issued

Message	Description
NotAGoodCimVersion	The CIM and the Profile must have a version element, if it is not the case, a warning is issued
Has constraint	Info about the presence on ocl constraint
Has note	Information on the presence of a note
NotGoodPackNames	The packages names in a profile package must reflect existing CimPackage name
Two elements have the same name	In a package an element name must be unique, if it is not the case, a warning is issued
HasNonAttachedNote	Info : the note is not attached to a Class or an association but belongs to the package (note in a diagram),
NotIsBasedOn	A profile element (class, attribute, association) must be based on a corresponding CIM element, if it is not the case, a warning is issued
NotSameNote	Info : the element has a different note than the one in CIM
HasMoreThanOnIsBasedOnLink	An element has more than one isBasedOn dependency link
HasAnUnkownLink	Link should be of kind notelink, association, aggregation, generalization, if it is not the case, a warning is issued
SomelInconsistency	The element attribute has no corresponding attribute in CIM: it may have been deleted in the CIM or his name has been changed in CIM
NotSameStereotypes	Profile attribute has different stereotypes than its CIM counterpart
NotProperlyTyped	The type of an attribute equal 0
NotInProfile	The type of a profile attribute does not refer to a datatype or an enumeration of the profile.
NotConformedType	The classifier of an attribute which should be a primitive has not the corresponding stereotype
NotSameType	The type of an attribute is different from the type of its CIM counterpart.
NotConsistantFeature	The attribute has not the same « isConst » or « isStatic » feature than its CIM corresponding one
NotConsistantFeature	the name of a profile enumeration member is not found in the CIM corresponding enumeration
NotConsistantEnumerationValues	The initial value of an attribute is not present in the enumeration of the CIM corresponding attribute
NotsameDefaultValue	The initial value of an attribute in a profile is different from the initial value of its CIM corresponding one
NotSameCardinality	The cardinality of the profile attribute is different from its CIM corresponding one
NotSameName	The name of an attribute is different from its CIM corresponding one (given by the GuidBasedOn TaggedValue)

Message	Description
NotSameStereotypes	A profile attribute has not the same stereotypes as its CIM counterpart
NotIsBasedOn	Profile element has no GUIDBasedOn TaggedValue or this TaggedValue is empty
NotAccessibleElement	The given GUIDBasedOn TaggedValue of a profile element does not correspond to a CIM element
NotIsBasedOnLinked	The profile element (Class, datatype) has no IsBasedOn dependency link with its CIM counterpart
NotProperlyLinked	The profile element has an isBasedOn dependency link to a CIM element that is not its CIM counterpart
NotSameStereotypes	The profile element and its CIM counterpart have different stereotypes
NotProperlyLinked	For a profile association BasedOn a parent CIM association if the sources are related (same hierarchy) the targets must also be related
NotSameRole	BasedOn association endRole Names and CIM counterpart association end role names must be the same.
NotSameRoleNote	BasedOn association endRole Names and CIM counterpart association end role name notes are in principle the same
NotSameCardinality	The multiplicity of the end role of the profile association and of its CIM counterpart are in principle the same
NotNaturalDirection	<p>if the association is oriented from cardinality (n,*) towards 0,1 or 1 it is normal,</p> <p>if the association is oriented from cardinality 0,1 towards 0,1 it has to be reported</p> <p>if the association is orientated from cardinality 1,0 towards n,* it has to be reported</p>
INFO association description	This is just informative about association

CGMES Integrity Check List

I.2. Integrity Check Message list

Message	Description
WARNING: the package xxxx is not IsBasedOn and none of his parents is	The profile package must be BasedOn or in a BasedOn package
Info: note beginning with yyyy ..."	Information of the presence of a note in a diagram and what is its content
ISSUE: a CimDataType must contain at least an attribute named value	Presence of value attribute in a CIMDatatype

Message

ISSUE: the default value of an element attribute whose type is an enumeration " + el.Name + "." + at.Name + " is different from his parent"); errors++;

ISSUE: IsBasedOn attribute atxxx of element elyyyy is not found

ISSUE: for element " + elxxxx" attribute name "atyyyy" is different from its CIM parent attribute " atzzzz"

info: the classifier of attribute "elxxxx.atyyyy" is different from the classifier of its parent attribute

info: the type of attribute "elxxx.atyyyy" is "el.zzzz"

ISSUE: problem in accessing an attribute's classifier for: " + el.Name + "." + at.Name"

WARNING: the type of attribute elxxxx.atyyyy is elzzzz is not included in the profile or in a profile Datatype Domain

ISSUE: the type of attribute "elxxxx .atyyyy" is "elzzzz" is not of the same type than the CIM one or is not an Enumeration

ISSUE: the type of attribute "elxxxx .atyyyy" has not a compliant value "atyyyy.Default "

ISSUE: the element "elxxxx" is not IsBasedOn on a CIM element (CIM element canceled)?"

ISSUE: there is no (or too many) IsBasedOn dependency link for the element "+ el.Name"

ISSUE: for the element "elxxxx" the parent element from GUIDBasedOn TaggedValue "elyyyy" is different from the supplier element of the IsbasedOn dependency link

ISSUE: non consistency between element elxxxx (Cimdatatype) and non primitive parent element parentelyyy

ISSUE: attribute value of a Cimdatatype is not properly typed

info: attribute value of element elxxxx (Cimdatatype) is typed with

Packageyyyyy ::elzzzz "

Description

A profile attribute default value belonging to an enumeration is compare to the default value of its CIM counterpart attribute.

A profile attribute has no counterpart in CIM: may-be it has been deleted in CIM

A profile attribute has not the same name as its CIM counterpart: may-be the attribute has been renamed in CIM

Usually the profile attribute points to a local profile classifier and not to the CIM's one.

Information about the type of the attributer

The attribute points to a classifier which does not refer to a proper attribute

Usually the attribute points to a local profile identifier, if it is not the case a warning is issued.

In a profile the restriction of a primitive value should be of the same primitive type or of an enumeration.

The initial value of a profile attribute should be compliant with one value of its parent's type when it is an enumeration.

An element must be IsBasedOn a CIM element.

A profile element should have only one IsBasedOn dependency association.

An element (class, package) should have only one dependency association to a CIM parent element and should have a GUIDBasedOn TaggedValue that is the GUID of the parent element. If the the two differ then a warning is issued.

The parent of a CIMDatatype element must be a Primitive or a CIMDatatype

A profile CIMDatatype value attribute type must point to a valid element

Information on attribute type

Message	Description
ISSUE: non consistency between name of element "elxxxx" and parent element "parentelyyyy"	A profile element has the same name of its CIM parent
WARNING: non consistency in list of stereotypes between profile element and its parent element	The list of stereotypes must be identical for a profile element and its parent
issue: get IBO element for " + elxxxx » + errorMessage	A system access interruption has been issued while trying to access an element in the EA repository (not finding the element in EA)
ISSUE: the connector of elxxxx con.ClientEnd.Role ---- con.SupplierEnd.Role is not properly BasedOn	A profile association is not BasedOn a parent association
ISSUE: the source "sourcexxxx" of one connector "elyyyy" is not properly BasedOn on the source of the IBO connector	The sources of a profile association and its parent association are not BasedOn
ISSUE: the target "targetxxxx" of one connector for "elyyyy" is not properly BasedOn on the target of the IBO connector	The targets of a profile association and its parent are not BasedOn
ISSUE: the connector "elxxxx + " is not properly BasedOn	A profile association must be IsBasedOn
ISSUE: in the association " Con.clientEndxxxx > con.Supplieryyyy" the profile association end role multiplicity is not compatible with the IBO multiplicity	The multiplicity of IsBasedOn association end role must be compatible with the range defined by the multiplicity of the CIM association counterpart end role.

Integrity Check Message List

I.3. ESMP Integrity Check Message List

The ESMP Integrity check does the same test as Integrity check, plus some specific tests. So, the list of messages is the same as the above list plus the following ones.

Message	Description
ISSUE: non consistency between element "elxxxx" and parent element "parentelyyyy "	A profile element has either the same name of its CIM parent or the same core name (name after the qualifier)
WARNING: non consistency in list of stereotypes between element and its parent element	A part from the use of ACC, ABIE or MBIE stereotypes, the list of stereotypes must be identical for a profile element and its parent
WARNING: the "ESMPG" TaggedValue is missing for elxxxx.atyyyy "	For all profile attribute element an order (ESMPG) TaggedValue must be present.
ISSUE: the "ESMPG" tagged value" rgxxxx" already exists for "elyyyy.at.zzzz"	For profile attribute, one cannot create an additional order (ESMPG) TaggedValue

Message	Description
WARNING: the "ESMPG" TaggedValue is missing for elxxxx con (con.Client.EndRole -> con.Supplier.EndRole)"	For all profile association element an order TaggedValue must be present
ISSUE: the "ESMPG" TaggedValue" already exists for elxxxx con (con.Client.EndRole -> con.Supplier.EndRole)"	For profile association, one cannot create an additional order (ESMPG) TaggedValue
WARNING: the test about hierarchy has been excluded for the association between "sourcexxxx" and "targetyyyy" because based on a recursive association: check if necessary	The program can't test the hierarchical ordering in case of a recursive association
ISSUE: for the association of element "elxxxx sourceyyyy.sourcerole " towards "targetzzzz + "." + targetrole" there is an inconsistency in end role name with the Is Based On association	The Association EndRole core names of the profile and CIM must be identical (core means after removing the qualifier of the end role name if there is one)
ISSUE: the hierarchy of one association (" con.ClientEnd.Role +," + con.SupplierEnd.Role + ")" + "from " + el.Name + " is not valid"	The End Role Name of the upper end of the hierarchy (aggregation side) is empty and the one of the lower end is the one pf the profile
ISSUE: the hierarchy of one association (" con.ClientEnd.Role , " con.SupplierEnd.Role)" + "from elxxxx should exist or reflects upperlevel hierarchy"	In a hierarchical profile, associations are of aggregation type. The upper end of a profile association is the aggregation side (drawn as a diamond). The lower end side could be the navigability side (drawn with an arrow)

ESMP Integrity Check Message List