magicdraw™

Architecture Made Simple

CODE ENGINEERING

version 18.1

user guide

No Magic, Inc.

2015

All material contained herein is considered proprietary information owned by No Magic, Inc. and is not to be shared, copied, or reproduced by any means. All information copyright 1998-2015 by No Magic, Inc. All Rights Reserved.

CONTENTS

Copyright © 1998-2015 No Magic, Inc.

Copyright © 1998-2015 No Magic, Inc.

CONTENTS

Copyright © 1998-2015 No Magic, Inc.

CONTENTS

Copyright © 1998-2015 No Magic, Inc.

INTRODUCTION

Overview

View Online Demos	Code Generation Code Reverse

NOTE: Code Engineering is available in Professional, Architect and Enterprise editions only.

MagicDraw code engineering provides a simple and intuitive graphical interface for merging code and UML models, as well as preparing both code skeletons out of UML models and models from code.

MagicDraw code engineering implements several cases where code engineering may by very useful:

- You already have code that needs to be reversed to a model.
- You wish to have the implementation of the created model.
- You need to merge your models and code.

The tool may generate code from models and create models out of code (reverse). Changes in the existing code can be reflected in the model, and model changes may also be seen in your code. Independent changes to a model and code can be merged without destroying data in the code or model.

MagicDraw code engineering supports Java, C++, CORBA IDL, DDL, XML Schema, WSDL, and C# languages. Your models can be converted to any of those languages, or UML models can be created from the source code written in those languages. Also reverse from Java Bytecode and CIL is supported.

The Code Engineering Sets tool is MagicDraw tool managing center for all code engineering matters.

Code engineering is available only in Professional or Enterprise editions. In the following table you'll find what languages are supported in different editions:

Language	Professional Edition	Enterprise Edition
Java	Java	+
Java Bytecode	Java	+
C++	C++	+
CORBA IDL	-	+
DDL/Database engineering	With Cameo Data Modeler plugin (separately purchaseable)	With Cameo Data Modeler plugin (free of charge)
CIL	C#	+
CIL Disassembler	C#	+
XML Schema	With Cameo Data Modeler plugin (separately purchaseable)	With Cameo Data Modeler plugin (free of charge)
WSDL	-	+
C#	C#	+

Copyright © 1998-2015 No Magic, Inc.

Code Engineering Sets

You may manage code engineering through the **Code Engineering Sets** in the Browser tree. The **Code Engineering Sets** tree contains the list of all sets created in the project and instruments for managing those sets.

To add a new set

1. From the **Code Engineering Sets** shortcut menu, choose **New**.
2. Choose the language you want (possible choices include: **Java**, **Java Bytecode**, **C++**, **C#, CIL**, **CIL Disassembler**, **CORBA IDL**, **DDL** (**Cloudscape**, **DB2**, **Microsoft Access**, **Microsoft SQL Server**, **MySQL**, **Oracle**, **Pervasive**, **Pointbase**, **PostgreSQL**, **Sybase**), **XML Schema**, and **WSDL**). The new set is created.

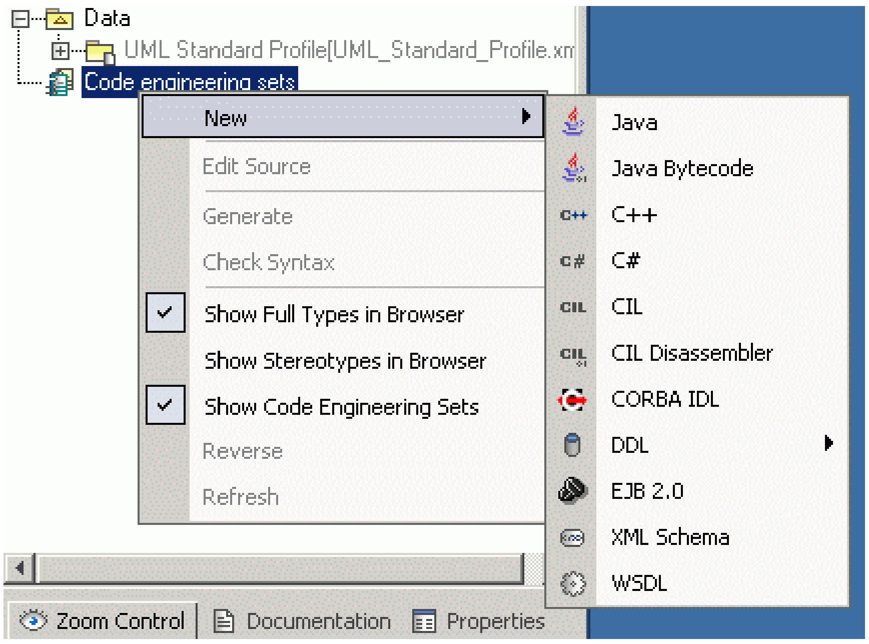
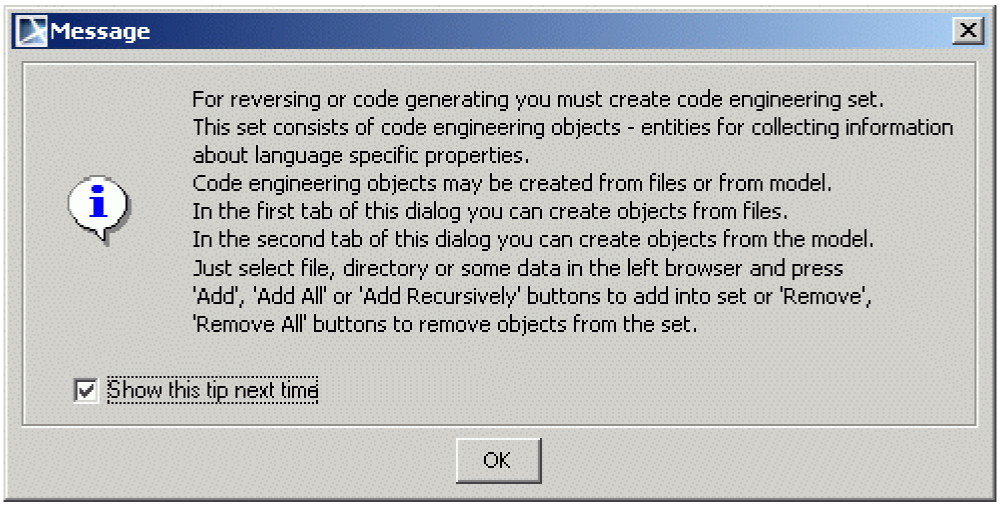


Figure 1 -- Code engineering language options

Edit sets in the **Round Trip Set** dialog box. To open this dialog box

- Choose **Edit** from the set shortcut menu.

If you are performing round trip for the first time, the tip message box appears.


Message

For reversing or code generating you must create code engineering set.
This set consists of code engineering objects - entities for collecting information about language specific properties.
Code engineering objects may be created from files or from model.
In the first tab of this dialog you can create objects from files.
In the second tab of this dialog you can create objects from the model.
Just select file, directory or some data in the left browser and press 'Add', 'Add All' or 'Add Recursively' buttons to add into set or 'Remove', 'Remove All' buttons to remove objects from the set.

☑ Show this tip next time

OK

Figure 2 -- Code Engineering Sets tip message box

Copyright © 1998-2015 No Magic, Inc.

Disable the tip message box by deselecting the **Show this tip next time** check box.

The **Round Trip Set** dialog box allows you to manage entities to be added/removed to your set.

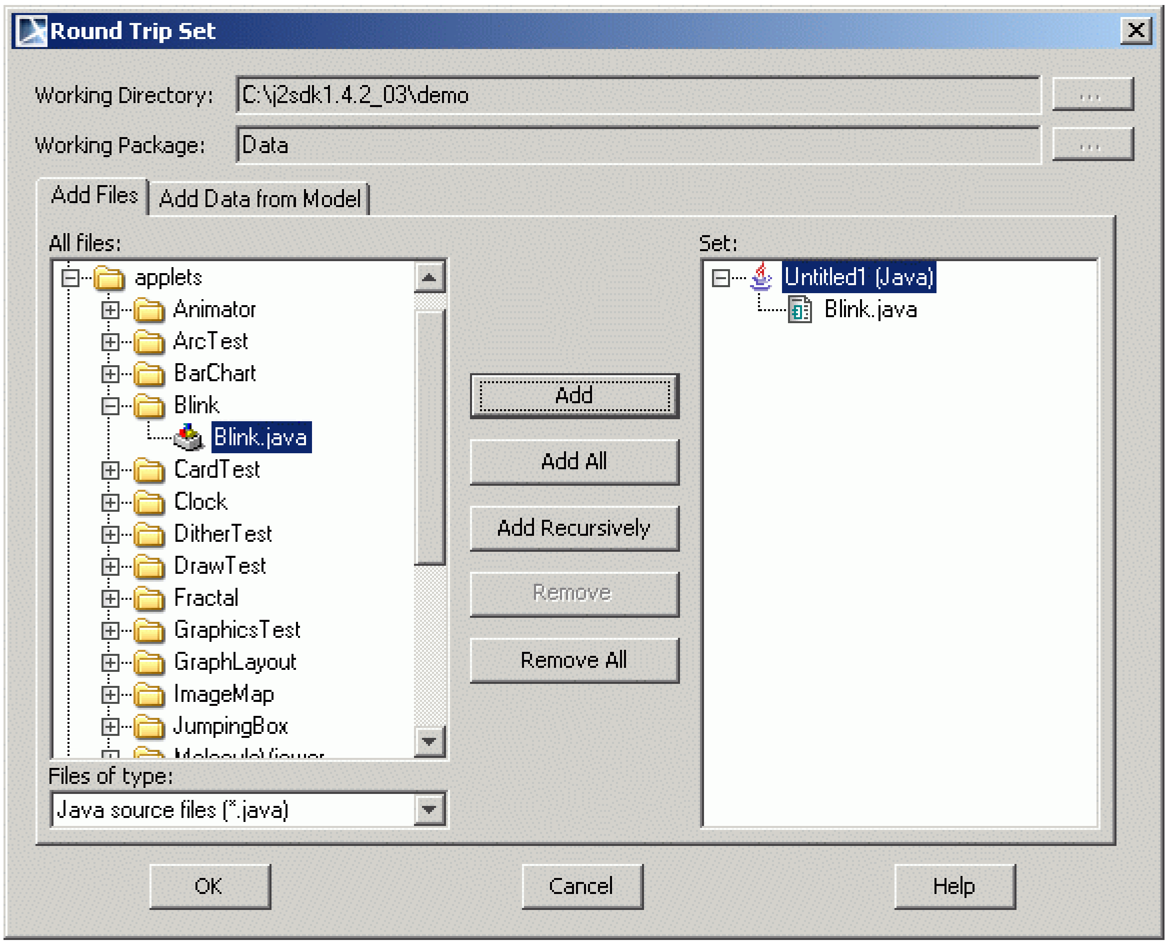


Figure 3 -- Round Trip Set dialog box. Add files tab

Specify **Working Directory** for displaying source files. This option indicates files and required sub-directories, where a code generation output goes. Type a path manually or by browsing in the directory tree, by clicking the '…' button.

The **Working Package** option allows to define any package for reverse output or code generation. Model will be reversed or code generated from this specified package.

NOTE	The working package may be selected or changed only prior to the addition of files from working directory to code engineering set.

The **Round Trip Set** dialog box has two tabs: **Add Files** and **Add Data from Model**.

The **Add Files** tab helps you manage the files of source code involved in your code engineering set.

Element name	Function
All files	Helps you find directories with the source files for the set.
Files of type	Contains possible file name extensions for the chosen language.

Copyright © 1998-2015 No Magic, Inc.

The **Add Data from Model** tab helps you manage elements located in the UML model.

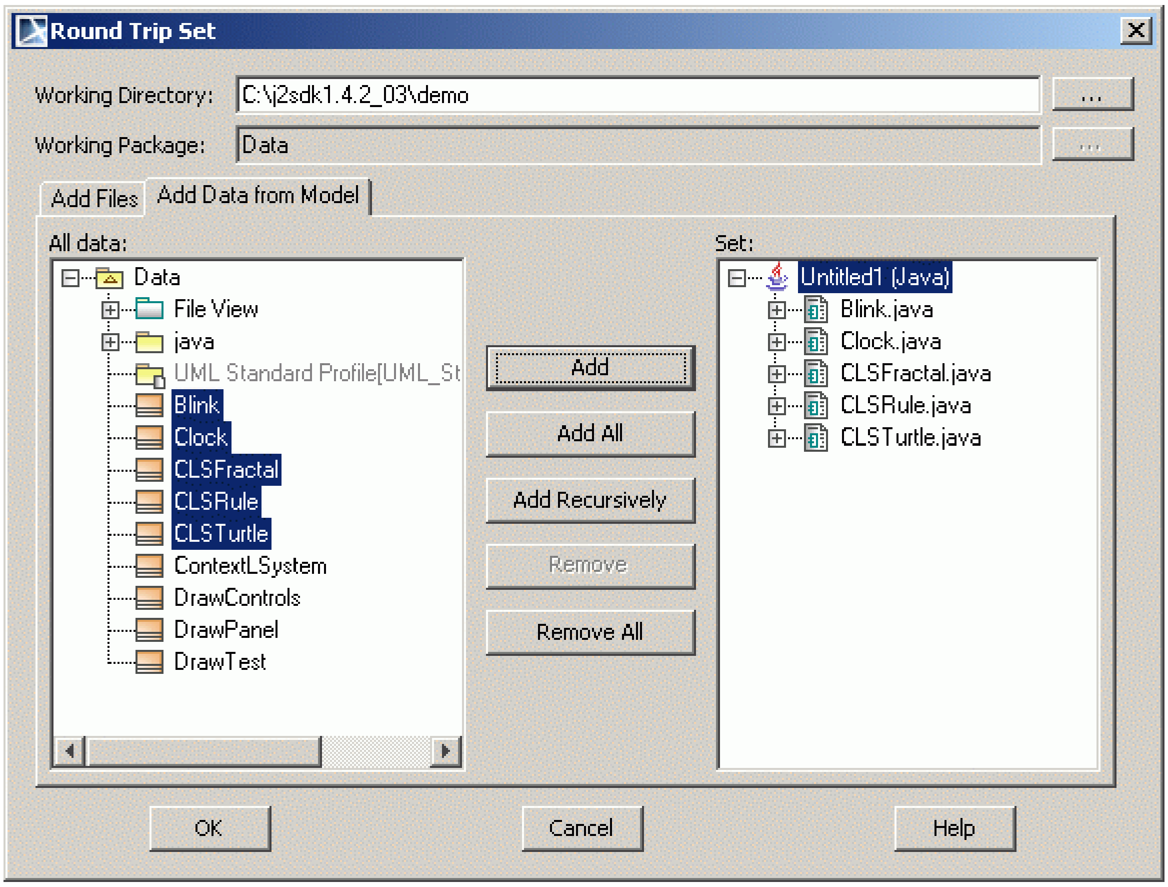


Figure 4 -- Round Trip Set dialog box. Add data from model tab

The **All Data** list contains the hierarchy of UML model packages with model elements (or other packages) inside of them. Your code engineering set can be combined out of model and code elements.

The following buttons are available in the **Round Trip Set** dialog box:

Add	The selected file in the **All Files** or **All Data** list is added to the set.
Add All	All files in the opened or selected directory are added to the set.
Add Recursively	All files in the selected directory and its subdirectories are added to the set.
Remove	Removes the selected entity from the set.
Remove All	Removes all entities from the set.

Generating Code

View Online Demo Code Generation

You may generate code for the selected and prepared set and directly for model elements.

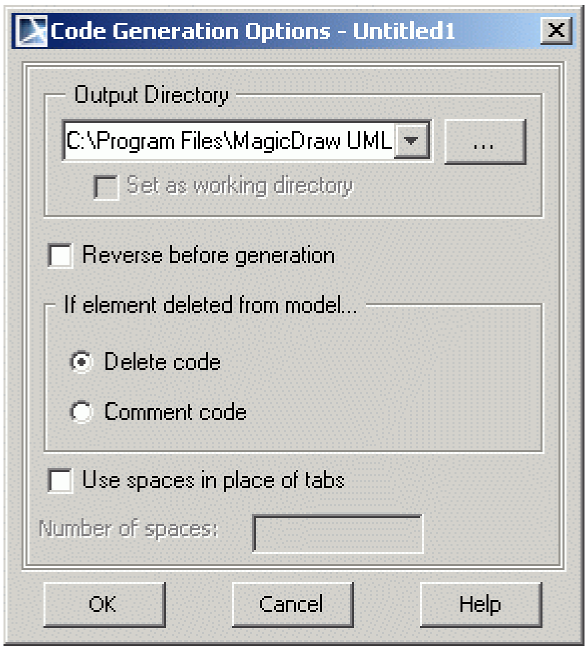
Copyright © 1998-2015 No Magic, Inc.

Code Generation for Set

Start code generation once the set or sets are prepared. For more details about creating and editing sets, see “Code Engineering Sets” on page 8.

- Choose **Generate** from the **Code Engineering Sets** item shortcut menu. It allows code generating for all created sets.
- Choose **Generate** from the selected set shortcut menu. It allows code generating only for the selected set.

The **Code Generation Options** dialog box appears.



Code Generation Options - Untitled1

Output Directory

C:\Program Files\MagicDraw UML

☐ Set as working directory

☐ Reverse before generation

If element deleted from model...

(●) Delete code

( ) Comment code

☐ Use spaces in place of tabs

Number of spaces:

Figure 5 -- Code Generation Options dialog box.

The **Code Generation Options** dialog box allows you to specify the way your code will be generated.

Once generating options are specified for the set, code can be generated.

Box name	Function
Output Directory	Type the directory where the generated files will be saved.
'...'	The **Set Output Directory** dialog box appears. Select the directory for the generated files.
Set as Working Directory	The output directory is set as a working directory and files are saved to the working directory.
Reverse before generation	Changes your model according to changes in the existing code. **WARNING:** Exercise caution when selecting the **Reverse before generation** check box. If the model differs from the code, all differences in the model will be discarded. In such cases, you will lose some your work.

Copyright © 1998-2015 No Magic, Inc.

Box name	Function
If element deleted from model	To influence the structure of generated code, click one of the following option buttons: • **Delete code**. The representation of deleted entities will be deleted from the code file. • **Comment code**. Deleted entities will be commented in the code file.
Use spaces in place of tabs	When selected, spaces (instead of tabs) will be written to the code file.
Number of spaces	Specify the number of spaces to be written.
OK	The Messages Window appears, displaying how code files are being generated. The Messages Window informs you of problems and errors (mainly file access and syntax errors) found in the code generation process and generation summary. You are also prompted to confirm that you wish to overwrite the file if the output directory already contains one with the same name.
Cancel	Closes the dialog box without saving changes.
Help	Displays MagicDraw Help

Code Generation for Model Element

All the classes contained in the component will be written to one file. However, code for the class can be generated in a different way. Select the class you wish to generate in the browser Data package and click Generate in the class shortcut menu. For packages and components, you may also select Generate, but you will not be able to specify the generation options. All the options related to that task will be set according to the default values.

If you have chosen framework generation for a single class or for packages, the **Code Generation Options** dialog box does not appear. The code is generated according to the default values.

If no errors occurred, you may view the results with your favorite file viewer or programming environment. Look for the files in the directory that you specified as your Working directory in the Round trip set dialog box or in the **Project Options** dialog box. Additional sub-directories could be created.

Reverse

View Online Demo	Code Reverse

A reverse is an opposite operation to the code generation. The existing code can be converted to UML models with the help of MagicDraw reverse mechanism.

Prepare the sets in the exact same way that you did for code generation (see “Code Engineering Sets” on page 8)

- Choose **Reverse** from the **Code engineering sets** item shortcut menu. It allows code reversing for all already created sets.
- Choose **Reverse** from the selected set shortcut menu.

Copyright © 1998-2015 No Magic, Inc.

The UML model for the component can be reversed in the same way. Just select the component you are interested in from the browser and click **Reverse** on it shortcut menu.

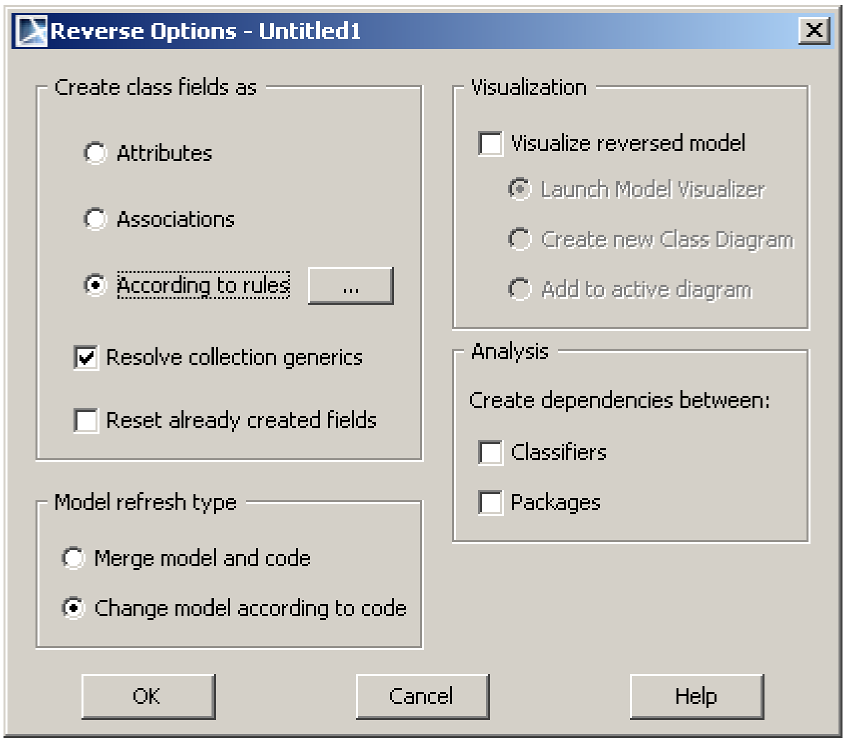
Models can be reversed without creating a set.

To reverse a model without creating a set

1. From the **Tools** menu, choose **Quick Reverse**. The **Round Trip Set** dialog box appears.

NOTE: Quick Reverse is available only in Professional and Enterprise editions.

2. Select the files from the **Round Trip Se**t dialog box, **Add Files** tab.
3. Click **OK**. The **Reverse Options** dialog box appears.


Reverse Options - Untitled1

Create class fields as
- ( ) Attributes
- ( ) Associations
- (●) According to rules ...
- [✓] Resolve collection generics
- [ ] Reset already created fields

Model refresh type
- ( ) Merge model and code
- (●) Change model according to code

Visualization
- [ ] Visualize reversed model
- (●) Launch Model Visualizer
- ( ) Create new Class Diagram
- ( ) Add to active diagram

Analysis

Create dependencies between:
- [ ] Classifiers
- [ ] Packages

Figure 6 -- Reverse options dialog box

Element name	Function
CREATE CLASS FIELDS AS	
Attributes	Class fields are represented in model as attributes.
Associations	Class fields are represented in model as association ends.
According to rules	Association or Attribute creation on reverse is ability to enter rules that help to decide if an association or attribute must be created on reverse. For more information, see “Rules of the association or attribute creation on reverse” on page 14.
Resolve collection generics	Reverse engineering is able to create associations when one class has collection of other classes and uses Java generics (for example, List<String>). If selected, types of collection will be resolved (property type will be not collection, but real type). Predefined container types in Java language properties will be appended by all the same containers in form: ·java.util.List<$$type$$> where $$type$$ replaced to value of "type" property when code is generated.

Copyright © 1998-2015 No Magic, Inc.

Element name	Function
Reset already created fields	Select this option if you want to keep already created UML representation (attribute or association) for class fields.
MODEL REFRESH TYPE	
Merge model and code	The model elements are updated by code. Elements that do not exist in the code will not be removed from the model.
Change model according to code	Model will be created strictly by code. Entities in the model that do not match entities in the code will be discarded.
VIZUALIZATION	
Visualize reversed model	Classes that are created while reversing can be added to a diagrams.
Launch Model Visualizer	After reversing, the **Model Visualizer** dialog box appears. It will assist you in creating a class diagram or sequence diagram (Java only) for newly created entities.
Create new class diagram	After reversing, the **Create Diagram** dialog box appears. Create a new diagram where the created entities will be added.
Add to active diagram	After reversing, all created entities will be added to the current opened diagram.
ANALYSIS - create dependencies between	
Classifiers	Dependencies between classes will be analyzed and created.
Packages	Dependencies only between packages will be created.
	NOTE: Method bodies are not parsed on dependency search. Only static information is used.

If you have a code set combined from several files, you may see changes you wish to model without reversing all the code. Only changed files should be reversed. This type of reversing can be done by clicking the Refresh button on the set shortcut menu, or by performing model refresh from the Code Engineering Sets dialog box.

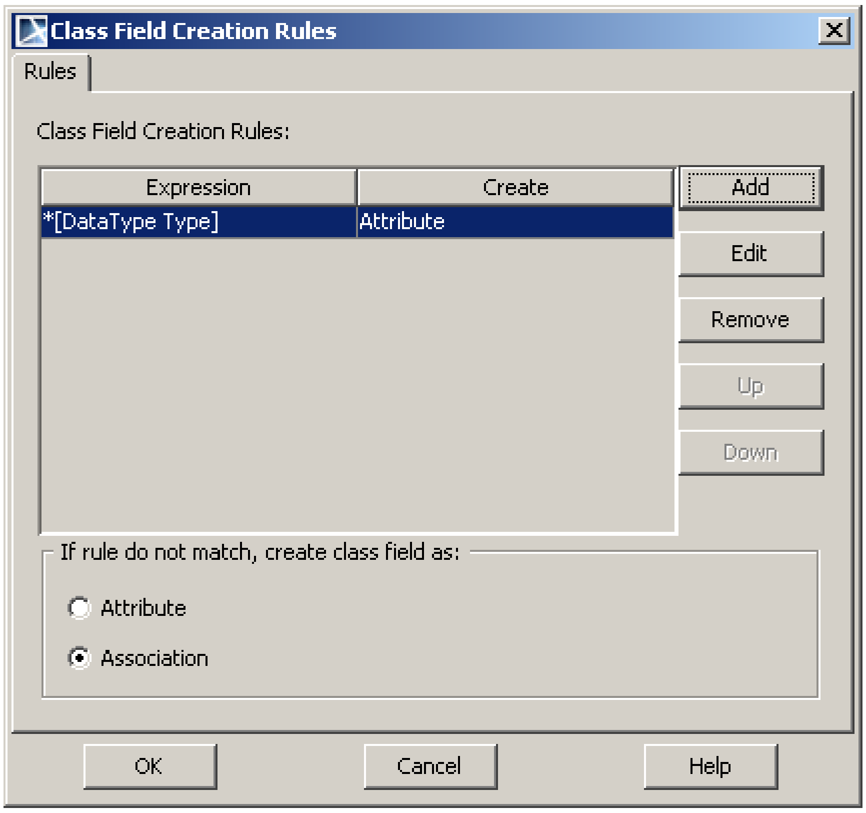
Rules of the association or attribute creation on reverse

Association or Attribute creation on reverse is ability to enter rules that help to decide if an association or attribute must be created on reverse.

Creating association or attribute creation rules on reverse

1. In the MagicDraw Browser, select the code engineering set for reverse.
2. From the set shortcut menu, choose the **Reverse** command. The **Reverse Options** dialog box is opened.
3. In the **Create class fields as** group, select the **According to rules** radio button. Then press the "…" button. The **Class Field Creation Rules** dialog box opens (see Figure 7 on page 15).

Copyright © 1998-2015 No Magic, Inc.



Expression	Create
*[DataType Type]	Attribute

If rule do not match, create class field as:

( ) Attribute
[✓ Association]

Figure 7 -- The Class Field Creation Rules dialog box

Managing association or attribute creation on reverse rules

The **Class Field Creation Rules** dialog box lists the described rules. If type qualified name matches any of the rule, then specified type is created. Rules described in this dialog are executed in order from the top to bottom. If one rule matches, no others are executed. Dialog allows to add a new rule, edit or remove an old one, order rules.

If no rules are matched, attribute or association according the radio button choice at the bottom of dialog is created.

Press the **Add** button for entering expression. On editing success new rule will be created. The **Rule** dialog box opens.

Press the **Edit** button for editing selected rule. Button is disabled, if no rule is selected. The **Rule** dialog box opens.

Defining rules

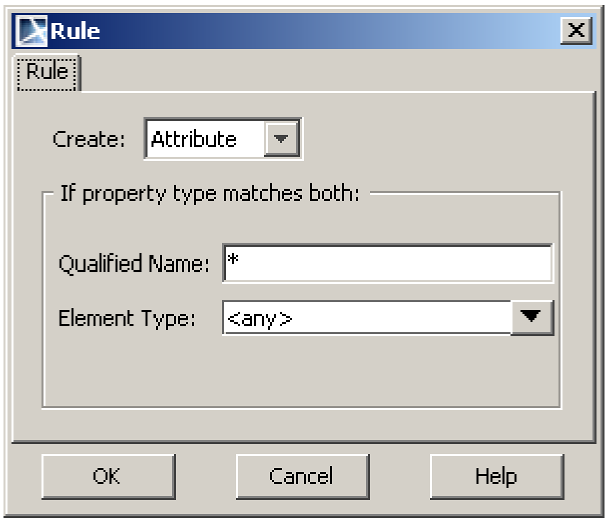
You can define rules in the **Rules** dialog box (see Figure 8 on page 16). To invoke this dialog box, in the **Class Field Creation** dialog box, press the **Add** or **Edit** button.

The **Rules** dialog box description:

- In the **Create** drop down list select to create **Attribute** or **Association** on reverse if property matches defined critters.
- The **Qualified Name** expression field supports simple search patterns - supports '*' and '?'

Copyright © 1998-2015 No Magic, Inc.

- The Qualified Name expression pattern can contain qualified name of the target element, which contains separator of UML separator style (*::*) and qualified name is counted to the first package with <<*modelLibrary*>> stereotype
- The **Element Type** drop down list contains subset of UML Type names: Any, Class, Interface, DataType, Primitive, Enumeration, Stereotype values are displayed.
- If in the **Element Type** drop down list **Any** value is selected, the element type will be matched by qualified name expression.



Create: Attribute
If property type matches both:
Qualified Name: *
Element Type: <any>

Figure 8 -- The Rule dialog box

Sample of the association creation on reverse

In these two samples it will be presented how to create associations among my reversed classes, but avoid creating associations to the used libraries (java).

Suppose you have your classes placed inside the package "*com::myProduct*". Creation rules must be following:

- Create "Association" if property type qualified name match "*com::myProduct::**" with "*all*" element type
- If rules do not match, create class fields as attributes.

Suppose that your classes are placed inside two packages "*myClient*" and "*myServer*". Creation rules should be:

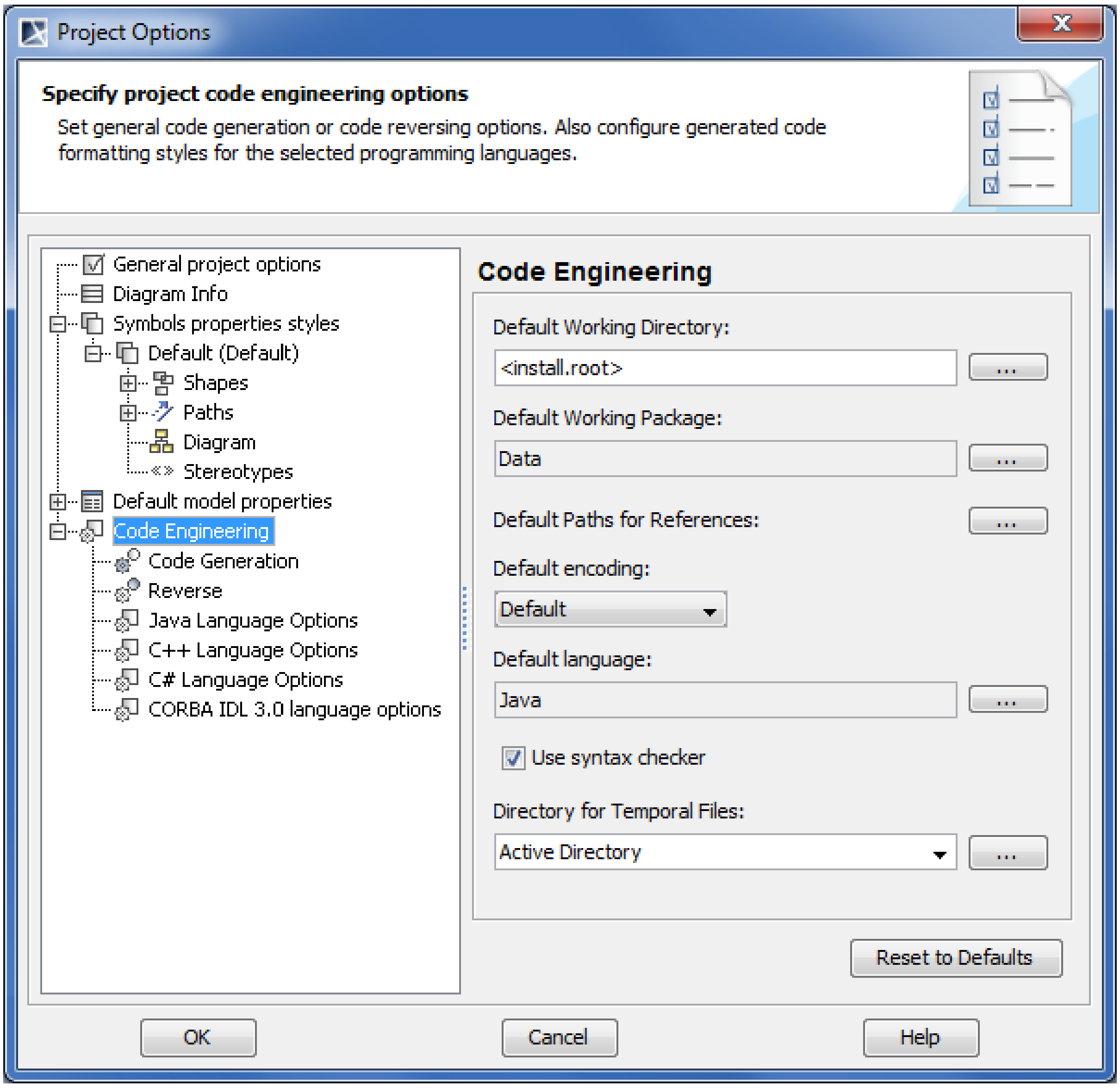
- Create "*Association*", if type name "match" path "*myClient::**" with "*all*" meta type
- Create "*Association*", if type name "match" path "*myServer::**" with "*all*" meta type
- If rules do not match, create class fields as attributes.

Copyright © 1998-2015 No Magic, Inc.

Global options for Code Engineering

Code engineering options for all sets in your project

From the **Options** menu, choose **Project**. The **Project Options** dialog box appears.


Project Options

Specify project code engineering options

Set general code generation or code reversing options. Also configure generated code formatting styles for the selected programming languages.

- General project options
- Diagram Info
- Symbols properties styles
  - Default (Default)
    - Shapes
    - Paths
    - Diagram
    - Stereotypes
- Default model properties
- Code Engineering
  - Code Generation
  - Reverse
  - Java Language Options
  - C++ Language Options
  - C# Language Options
  - CORBA IDL 3.0 language options

Code Engineering

Default Working Directory: <install.root>

Default Working Package: Data

Default Paths for References:

Default encoding: Default

Default language: Java

[✓] Use syntax checker

Directory for Temporal Files: Active Directory

Figure 9 -- Project Options dialog

The **Project Options** dialog box has two main collections of customizable options, which are represented by the hierarchy tree on the left side of the dialog box:

- **Styles** – expands the tree hierarchy of all the styles defined within the project. You may use as many of these styles as you wish. See MagicDraw main User's Manual, working with Projects Section.
- **Code engineering** – these options are found on the right side of the Project options dialog box:

    - **Default Working Directory** field - type the name or browse by clicking the button in the working directory.

Copyright © 1998-2015 No Magic, Inc.

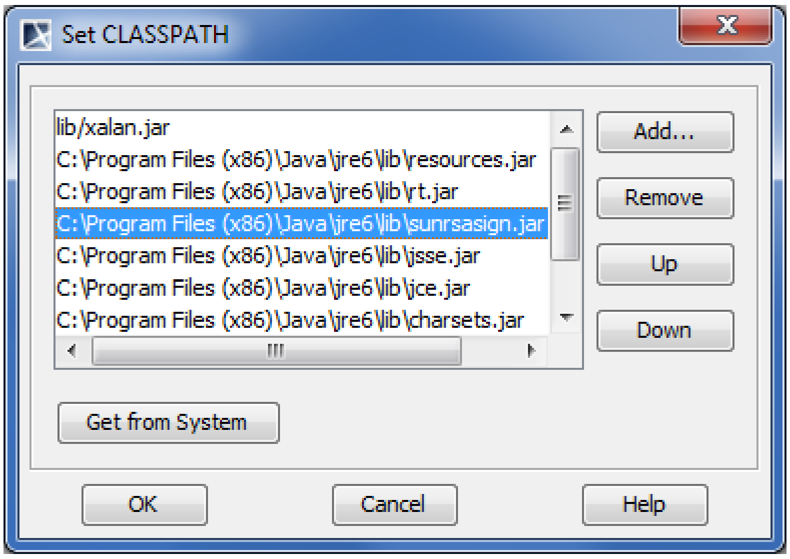
- **Default Working Package** - allows to define any package for reverse output or code generation. Model will be reversed or code generated from this specified package.
- **Default Paths for References** - add specific profiles, modules[1], libraries to define where to search paths for references during reverse/code generation.
- **Default Encoding** - a list of available encodings appears.
- **Default language** drop-down box – select the default generation language.
- **Use Syntax Checker** check box – when selected, the syntax checker runs while Code Engineering is executed
- **Directory for Temporal Files -** it can be **Active Directory**, **System** or define other by clicking “...” button.

Tab name	Description
Code generation	Set code generation options using the fields listed in the right side of the Project options dialog box. The Code generation area contains boxes that have the same functionality as in the Code generations options dialog box (see “Generating Code” on page 10).
Reverse	Set reverse options for all reverse actions of the project using the options listed on the right side of the Project options dialog box. The Reverse area contains boxes that have the same functionality as in the Reverse options dialog box (see “Reverse” on page 12).

Tab name	Element name	Function
Java Language Options Set the generated code style for Java programming languages in the **Default** language field found on the right side of the **Project Options** dialog box.	**Generate opening bracket in new line**	Opens a bracket in the new line that is being generated.
	Generate spaces	Generates spaces inside an assignment and other operators.
	Generate empty documentation	Comment brackets are placed in your code, unless class in the model has no documentation.
	Automatic “import” generation	Automatic generation of "import" sentences according to classes that are referenced in the generated class.
	Class count to generate import on demand	Specify number of classes imported from a single package until all statements importing a single class are substituted with a statement importing an entire package.
	Documentation Processor	After selecting **Java Doc** processor, click the “...” button to open the **Documentation Properties** dialog box.
	Style	Two styles are available for documentation.

1. Starting from version 18.1, referred as “used project” in MagicDraw UI. This user guide mentions the old keyword, which will be replaced in the documentation of the next MagicDraw version.

Copyright © 1998-2015 No Magic, Inc.

Tab name	Element name	Function
	Use CLASSPATH	The '...' button is activated. Search a classpath for importing sentences generation in the Set classpath dialog box.  lib/xalan.jar C:\Program Files (x86)\Java\jre6\lib\resources.jar C:\Program Files (x86)\Java\jre6\lib\rt.jar C:\Program Files (x86)\Java\jre6\lib\sunrsasign.jar C:\Program Files (x86)\Java\jre6\lib\jsse.jar C:\Program Files (x86)\Java\jre6\lib\jce.jar C:\Program Files (x86)\Java\jre6\lib\charsets.jar *Figure 10 -- Set classpath dialog* Click the **Get from System** button to get CLASSPATH variable defined by operating system or click the Add button and select the classpath directory in the **Add Classpath** dialog box.
	Java Source	Available choices **1.4** , **5.0**, or **7.0**.
	Header	Add the specific header to all your code files. Click the button and enter header text in the **Header** screen. You may also define $DATE, $AUTHOR, and $TIME in the header.
C++ Language Options Set the generated code style for C++ programming languages.	**Generate opening bracket in new line**	Opens a bracket in the newly generated line.
	Generate spaces	Spaces inside an assignment and other operators are generated.
	Generate empty documentation	Comment brackets are placed in your code, unless class in the model has no documentation.
	Generate methods body into class	Select check box to generate methods body into class.
	Documentation Style	Two styles are available for documentation.
	Use include path	Click the '...' button and then specify the path for the includes in the **Set Include Path** dialog box.
	Use explicit macros	Select check box. The '...' button is activated, click it and in the **C++ Macros** dialog box use a set of predefined macros.
	Header	Add the specific header to all your code files. Click the "…" button and enter header text in the **Header** screen. You may also define $DATE, $AUTHOR, and $TIME in the header.
CORBA IDL 3.0 Language Options	**Generate documentation**	Includes the documentation of an element in the comment.

Copyright © 1998-2015 No Magic, Inc.

Tab name	Element name	Function
	Generate opening bracket in new line	Opens a bracket in the new line generating.
	Generate spaces	Spaces inside an assignment and other operators are generated.
	Generate empty documentation	Comment brackets are placed in your code, unless class in the model has no documentation.
	Generate imports	Generation of "import" statements for classes that are referenced in the generated class.
	Generate preprocessor directives	Generates pre-processors directives.
	Documentation Style	Three styles are available for documentation.
	Header "..."	Add the specific header to all your code files. Click the "..." button and enter header text in the **Header** dialog. You may also define $DATE, $AUTHOR, and $TIME in the header. Since MagicDraw version 17.0, the variable $DOCUMENTATION can be used for exporting the documentation of the file component (from the File View package) as a header of the IDL file. **IMPORTANT!** The variable $DOCUMENTATION is available with MagicDraw 17.0 Service Pack 4 and later versions.
	Set Include Path	Specify the path for the "includes". Click the "..." button to open the **Select Folder** dialog box.
DDL Language Options	**Generate opening bracket in new line**	Opens a bracket in the new line generating.
	Generate spaces	Spaces inside an assignment and other operators are generated.
	Generate documentation	Comment brackets are placed in your code, unless class in the model has no documentation.
	Header	Add the specific header to all your code files. Click the button and enter header text in the Header screen. You may also define $DATE, $AUTHOR, and $TIME in the header.
C# Language Options Set the generated code style for C# programming languages.	**Generate opening bracket in new line**	Opens a bracket in the newly generated line.
	Generate spaces	Generates spaces inside an assignment and other operators.
	Generate empty documentation	Comment brackets are placed in your code, unless class in the model has no documentation.
	Generate required "using" directives	Automatic generation of "using" directives. This option facilitates the usage of namespaces and types defined in other namespaces.

	Concatenate namespace names	If not selected namespace names are separated into several lines.
		e.g.
		`namespace A`
		`{`
		`namespace B`
		`{`

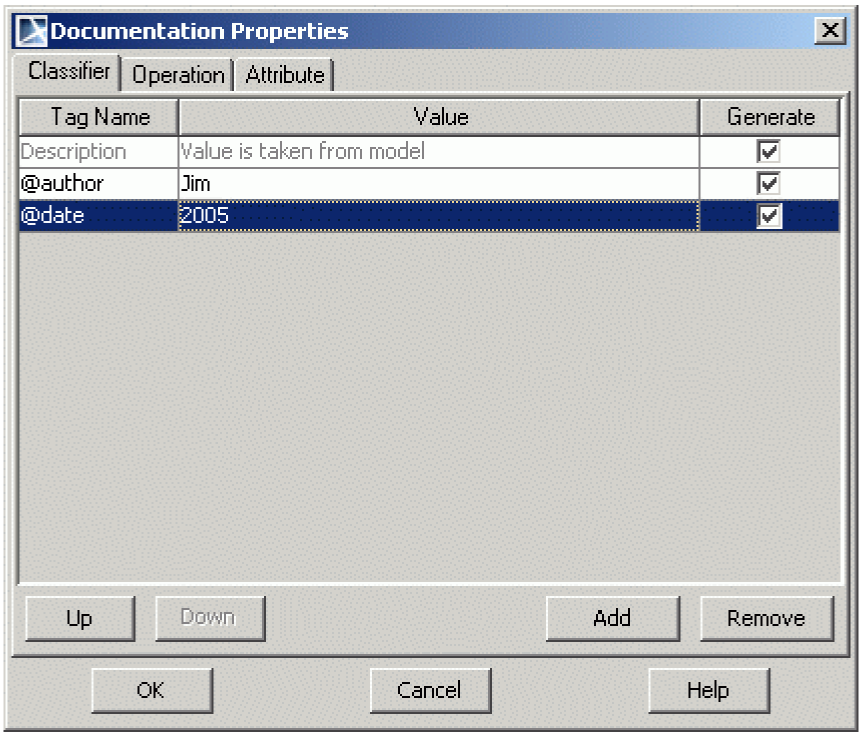
Copyright © 1998-2015 No Magic, Inc.

Tab name	Element name	Function
	Documentation: • Processor • Style	• Use C# XMI processor then generates c# xmi documentation for commenting the code. • Select one of the listed comment styles.
	Header	Adds the specific header to all your code files. Click the **'...'** button and type header text in the **Header** dialog box. You may also define $DATE, $AUTHOR, and $TIME in the header.
	Conditional Symbols	Add the conditional symbols, which cannot be recognized and should be skipped during reverse. Click the **'...'** button and add conditional symbols in the **Define Conditional Symbols** dialog box.

Java Documentation Properties dialog box

To open the **Java Documentation Properties** dialog box

In the **Project Options** dialog box, **Java Language Options** group, select the **Java Doc** processor in the **Documentation** field and click the “...” button to open the **Documentation Properties** dialog box.



Tag Name	Value	Generate
Description	Value is taken from model	✓
@author	Jim	✓
@date	2005	✓

Figure 11 -- Documentation Properties dialog box

Box Name	Function
Tag Name	Type a tag name.
Value	Type the value of the tag.
Generate	The selected tag will be placed in the generated code as a comment before classifier (class or interface), operation or attribute.
Up	Moves the selected item up the list.

Copyright © 1998-2015 No Magic, Inc.

Box Name	Function
Down	Moves the selected item down the list.
Add	Adds a new item in the list.
Remove	Removes the selected item from the list.
OK	Saves changes and closes the dialog box.
Cancel	Closes the dialog box without saving changes.
Help	Displays MagicDraw Help.

Round Trip

MagicDraw round trip keeps your code and model synchronized, and because Round trip traces all the model and code changes, you may freely change entity specifications without discarding code changes made outside the tool.

For example, Round Trip prevents a job from being damaged by code additions or changes when these steps are followed:

Within the tool, class Base is created.

1. Operation getInstance is added to class.
2. Code is generated
3. With external tool, programmer adds code to that operation.
4. With MagicDraw UML, operations name is changed to Instance.
5. Code is generated.

If the tool rewrites the whole code, these changes are made without corrupting the programmer’s job. The name of the operation is changed, but the internals remain the same.

Round trip catches all changes in your project and controls the following actions:

- If the source code is not changed, it is not allowed to refresh UML model. The **Refresh** command from the set shortcut menu is unavailable.
- If the model is changed but the code remains the same (new members were added or their headers were changed), refresh is not allowed, and the **Refresh** command from the set shortcut menu is unavailable. When generating code according to changes, all changes in the model are written to the signatures of class members, leaving the old implementation in place.
- If the code is changed but the model remains the same, refresh can be executed: code will be reversed to the UML models. If the **Code Generation Options** dialog box appears when you are attempting to generate code, you may select a code action that differs from the UML model.
- If the code and model are changed while refreshing, all changes in the code are treated as new items and added to the model.
- If data in the model file is deleted, it will be restored while refreshing, even when the code has not been changed or the data itself is unimportant.

Type Mapping Table

Languages supported by MagicDraw UML have their own built-in types. One language’s type might have no matches in another language, or it might have multiple matches. Additionally, some names are interpreted differently in different languages. When performing code generation, therefore, problems may occur when switching between different languages. To avoid this, MagicDraw UML uses type-mapping tables to manage

Copyright © 1998-2015 No Magic, Inc.

mapping between languages. It describes the rules of how one language’s built-in types are converted to those of another language

Files of Properties

The code can be generated out of prepared UML models. The mapping between the identifiers, used in the UML model and the language to which the model is being generated, should be implemented. This mapping includes the following sections:

- Build-in types (their default values)
- Generalization types
- Possible class declarations. Attributes and operations declaration and visibility modifiers
- Code generation options.

The separate prop file is created for every language that is supported by MagicDraw. Files are located in the <MagicDraw installation directory>/data folder. The file name pattern is lang.prop, where lang stands for the name of the programming language.

Supported language	File of Properties
JAVA	java.prop
C++	C++.prop
CORBA IDL	idl.prop
JAVABytecode	javabytecode.prop
DDL	ddl.prop
CIL	cil.prop
CIL Disassembler	cil disassembler.prop
C#	c#.prop
IDL	idl.prop
XML Schema	xmlschema.prop
WSDL	wsdl.prop

Files of language properties are separated into sections where all related elements are grouped. You may edit existing entities, add new ones, and change the default values.

We strongly recommend that you edit default values only. In general, all the sections have the list of possible and default values for the element.

Copyright © 1998-2015 No Magic, Inc.

JAVA CODE ENGINEERING

Introduction

Java Code Engineering chapter describes how Java language elements are mapped to UML by MagicDraw, what profiles to use and describes Java code engineering properties.

Chapter "Java Mapping to UML", on page 25 describes general rules how each Java element is mapped to UML by MagicDraw and what profile is used. You will find an example and corresponding model in MagicDraw with marked properties used in Java to describe mapping rules.

Chapter "Java CE Properties", on page 52 introduces specific Java options.

Abbreviations

UML	Unified Modeling Language
JLS	Java Language Specification
CE	Code Engineering
CES	Code Engineering Set
JVM	Java Virtual Machine

References

Gosling, J., Joy, B., Steele, G., Bracha, G., & Buckley, A. (2012, February 6). *The Java™ Language Specification. Java SE 7 Edition.* Retrieved March 29, 2012, from http://docs.oracle.com/javase/specs/jls/se7/jls7.pdf

Java support in MagicDraw

You may perform the following actions with MagicDraw:

- Import Java source code into model (reverse engineering).
- Generate Java code from the model (code generation).
- Apply changes to the source code from the model (round-trip). You may change Java declaration headers and apply them to already existing source code, however you cannot change method implementation.
- Create sequence diagram from the selected method body.
- Create model from the Java byte code.

Copyright © 1998-2015 No Magic, Inc.

Java Mapping to UML

Java Profile

UML specification does not provide elements to cover fully JLS, therefore MagicDraw is using UML stereotypes to mark UML class or interface to be some specific Java element.

Java stereotypes are provided in Java_Profile.xml file in MagicDraw profiles directory. Some stereotypes have tagged values used for mapping special Java language elements or keywords, which are not mapped to the standard UML properties. All Java stereotypes are derived from the «JavaElement». Another abstract stereotype «JavaTypeElement» is used to group all Java type elements. These two stereotypes are abstract and are not used directly.

Each other stereotype is used to represent appropriate Java element: «JavaImport» represents Java import, «JavaOperation» - Java operation, «JavaClass» - Java class type, etc.

Copyright © 1998-2015 No Magic, Inc.

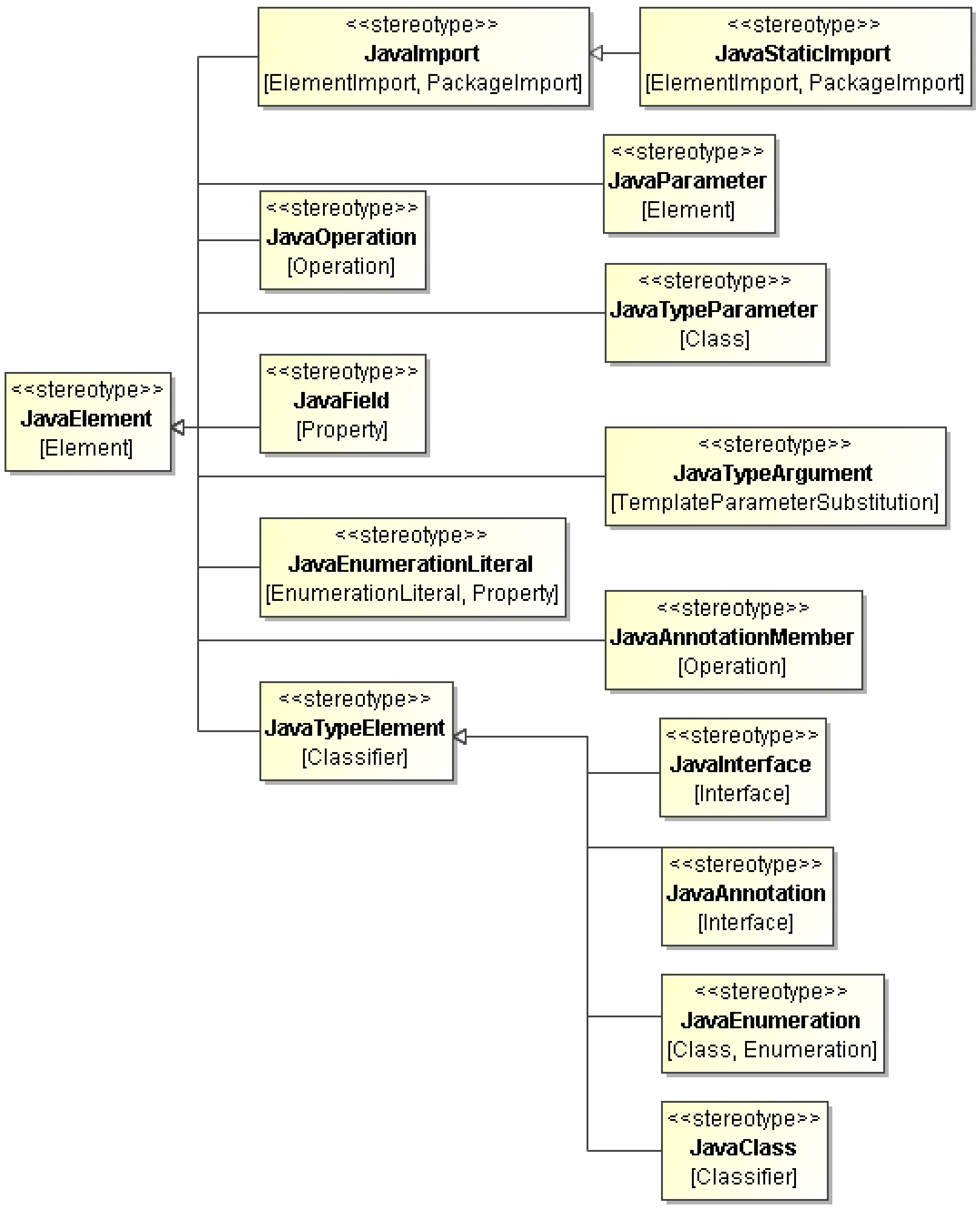


Figure 12 -- Stereotypes

Java referenced types

Java built-in types are used from the “UML Standard Profile”, which is automatically loaded with every new project.

NOTE	UML Standard Profile by default is hidden. If you want see it, click the **Show Auxiliary Resources** button in the Browser.

Every referenced class from the other libraries (including JDK libraries) should be imported/created into project and referenced in CES reference path (by default reference path is “Data” package in the model).

MagicDraw resolves referenced classes from the specified class path (by default class path is boot class path taken from the JVM on which MagicDraw is started) and creates in appropriate package structure (for more

Copyright © 1998-2015 No Magic, Inc.

information see "Sorting reversed classes according to the classpath", on page 27).

If referenced class is not found nor in model neither in class path, reference is created in the “Default“ package.

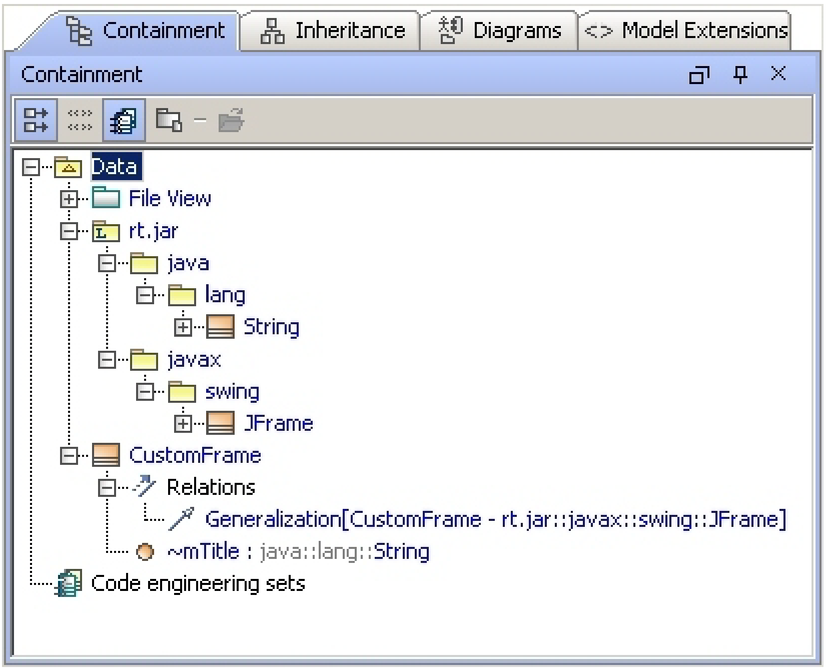
Java Profile defines UML Class with a name “?“. It is used for mapping parameterized types.

Sorting reversed classes according to the classpath

References are sorted to packages according to the classpath entry.

See a sample in Figure 13 on page 27, there reverse result is represented of the following java code:

```
import javax.swing.*;
public class CustomFrame extends JFrame
{
    String mTitle;
}
```



```
Data
  File View
  rt.jar
    java
      lang
        String
    javax
      swing
        JFrame
  CustomFrame
    Relations
      Generalization[CustomFrame - rt.jar::javax::swing::JFrame]
    ~mTitle : java::lang::String
  Code engineering sets
```

Figure 13 -- Sample: The String, JFrame and CustomFrame classes location after reverse

Mapping to UML rules

Package

Java package is mapped to the general UML package. It does not have any specific stereotypes and properties. However if UML package represents Java package, it must not have «modelLibrary» stereotype from standard UML profile. «modelLibrary» stereotype is used to show root package, where Java package tree ends and all parent packages, including «modelLibrary» are not part of Java package structure. In picture

Copyright © 1998-2015 No Magic, Inc.

below is “java.lang.String” added into the “working package“. Packages “java” and “lang“ represents general Java packages, but “working package“ with “L“ is stereotyped with «modelLibrary» and is not part of the Java.

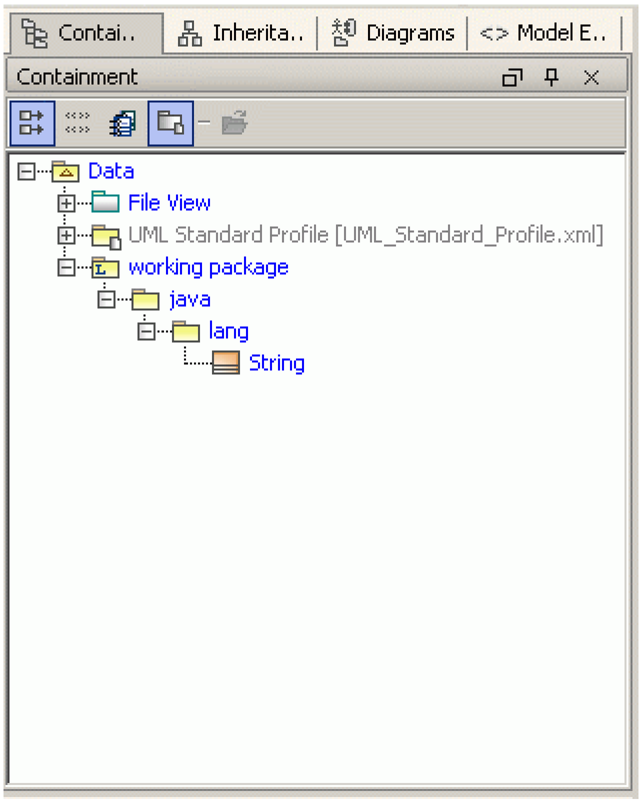


Figure 14 -- Package Structure

Class

Java class is mapped directly to the UML Class with stereotype «JavaClass». This stereotype is optional and if UML class doesn’t have any stereotype, Java CE treats it as Java class. Class modifiers are mapped into UML Class properties or to the Java language properties for class, if no appropriate property is found in UML.

Java class fields, operations and inner classes are mapped to the appropriate UML Properties, UML Operations and UML Classes

Class mapping table

Java element	**MagicDraw-UML element**
Class declaration	UML Class with stereotype «JavaClass» (optional)
Class name	UML Class name
Class documen-tation	UML Class Documentation
Class e	Is mapped to the UML Generalization relationship, where supplier is extended class and client is mapped class
Class imple-ments clause	Is mapped to the UML Interface Realization relation-ship, where supplier is extended class and client is mapped class.
Visibility modifier	UML Class “Visibility” property
Abstract modifier	UML Class “Is Abstract” property

Copyright © 1998-2015 No Magic, Inc.

Java element	MagicDraw-UML element
Final modifier	UML Class “Is Final Specialization“ property
Static modifier	Java Language property “Static modifier“
Strictfp modifier	Java Language property “Strictfp modifier“

Example

Java Source Code

```
 /**
  * Comment of the class MyList
  */
public final class MyList extends ArrayList implements Cloneable
 {
 }
```

MagicDraw UML Model

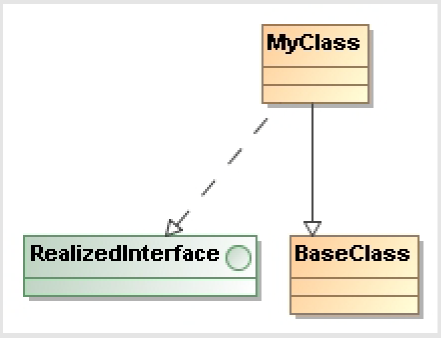
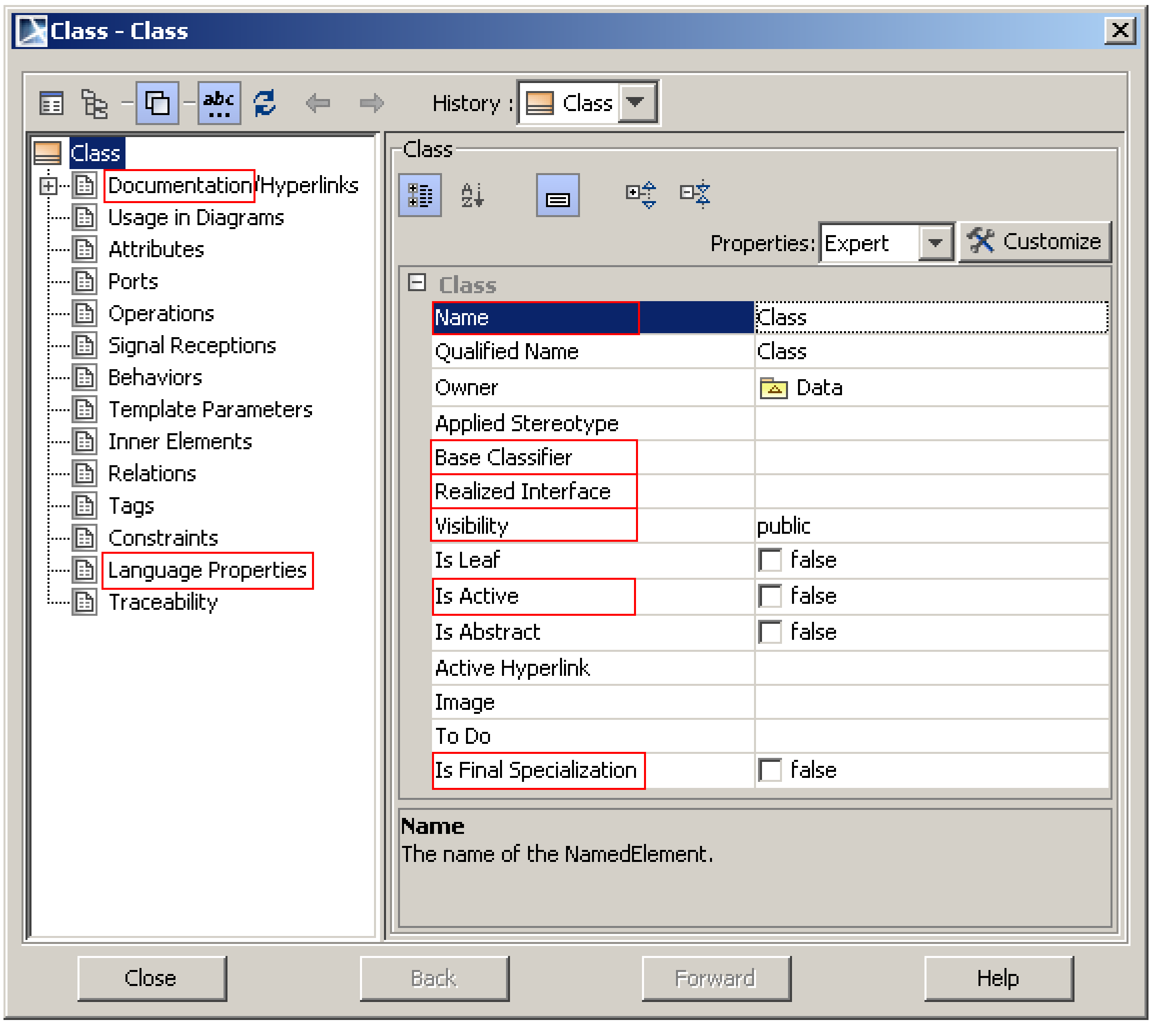


Figure 15 -- Class diagram

Copyright © 1998-2015 No Magic, Inc.

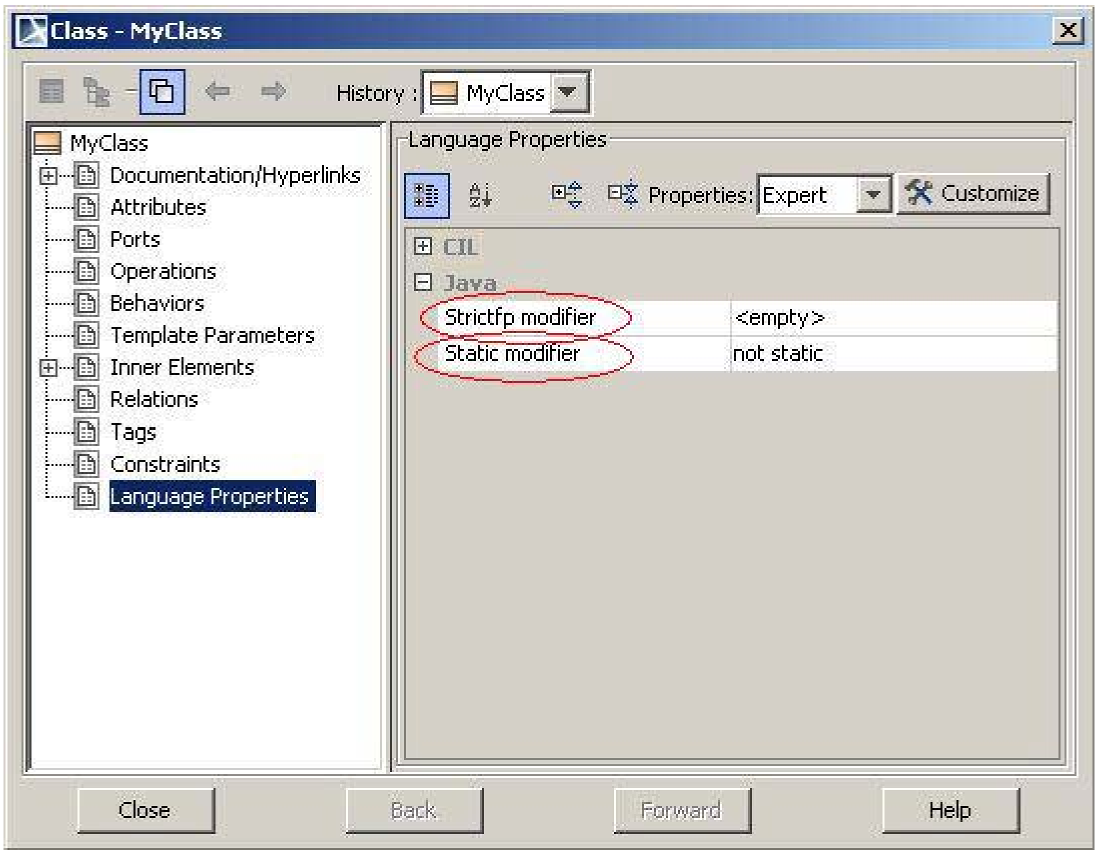


Class	
Name	Class
Qualified Name	Class
Owner	Data
Applied Stereotype	
Base Classifier	
Realized Interface	
Visibility	public
Is Leaf	false
Is Active	false
Is Abstract	false
Active Hyperlink	
Image	
To Do	
Is Final Specialization	false

Name
The name of the NamedElement.

Figure 16 -- UML Class specification dialog

Copyright © 1998-2015 No Magic, Inc.



Language Properties

Strictfp modifier	<empty>
Static modifier	not static

Figure 17 -- UML Class language properties

Field

Java field is mapped directly to the UML Property with stereotype «JavaField». This stereotype is optional and if UML class doesn't have any stereotype, Java CE treats it as Java field. Field modifiers are mapped into UML Property properties or to the Java language properties, if no appropriate property is found in UML.

Worth to know, that Java field type modifiers is mapped to the MagicDraw specific property "Type Modifier", but not to the UML Multiplicity.

Field mapping table

Java element	**MagicDraw-UML element**
Field declaration	UML Property, owned by UML Class, with stereotype «JavaProperty» (optional)
Field name	UML Property Name
Field documentation	UML Field Documentation
Field type	Is mapped to the UML Type property. It is reference to the UML Classifier, which by its package structure and name represents referenced Java class
Field type modifiers	Is mapped to the MagicDraw specified property "Type Modifier"
Visibility modifier	UML Property "Visibility" property
Final modifier	UML Property "Is Read Only" property

Copyright © 1998-2015 No Magic, Inc.

Static modifier	UML Property “Is Static“ property
Transient modifier	Java Language property “Transient modifier”
Volatile modifier	Java Language property “Volatile modifier”

Example

Java Source Code

```
public final class MyClass
{
     /**
      * myList comment
      */
     public static java.util.List myList;
}
```

MagicDraw UML Model

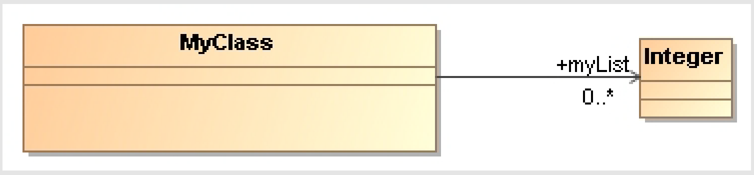
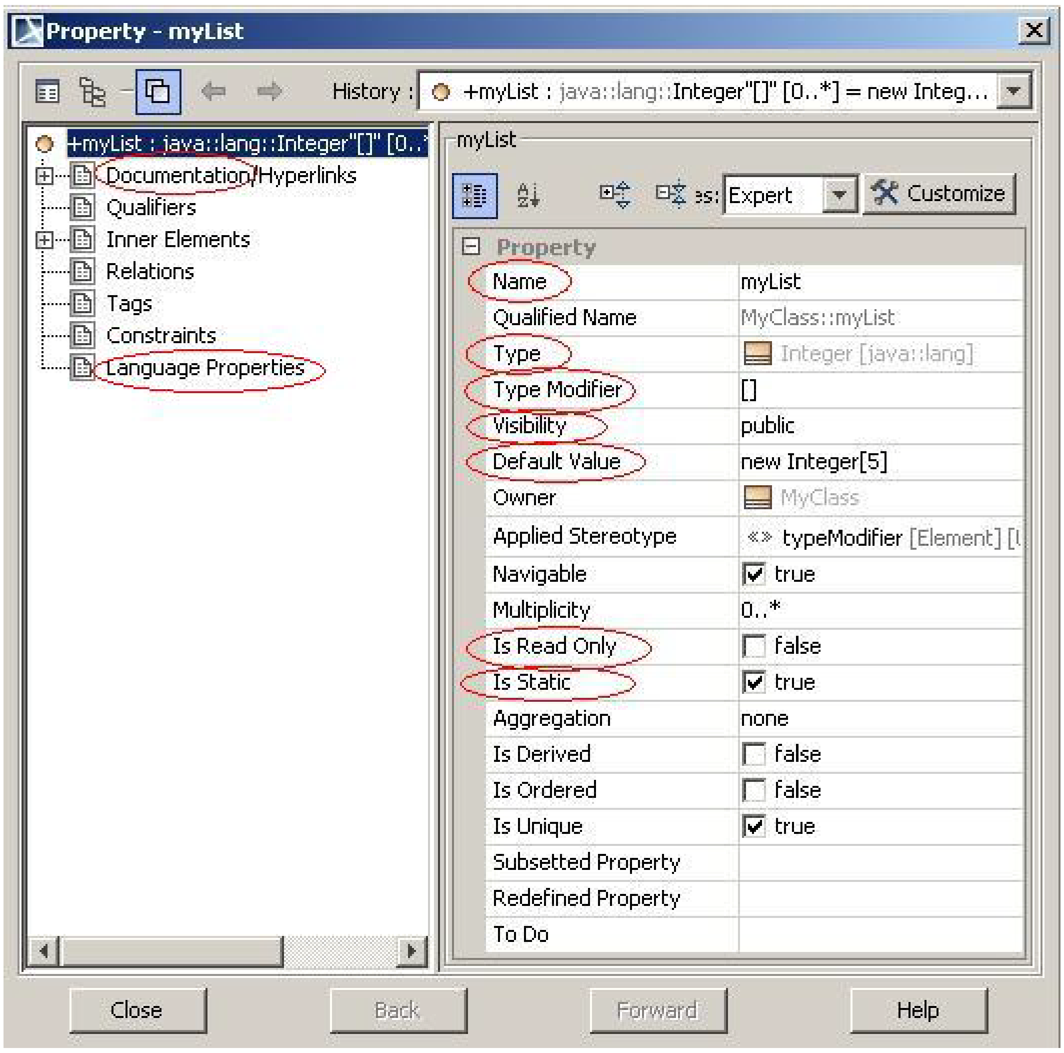


Figure 18 -- Class with property

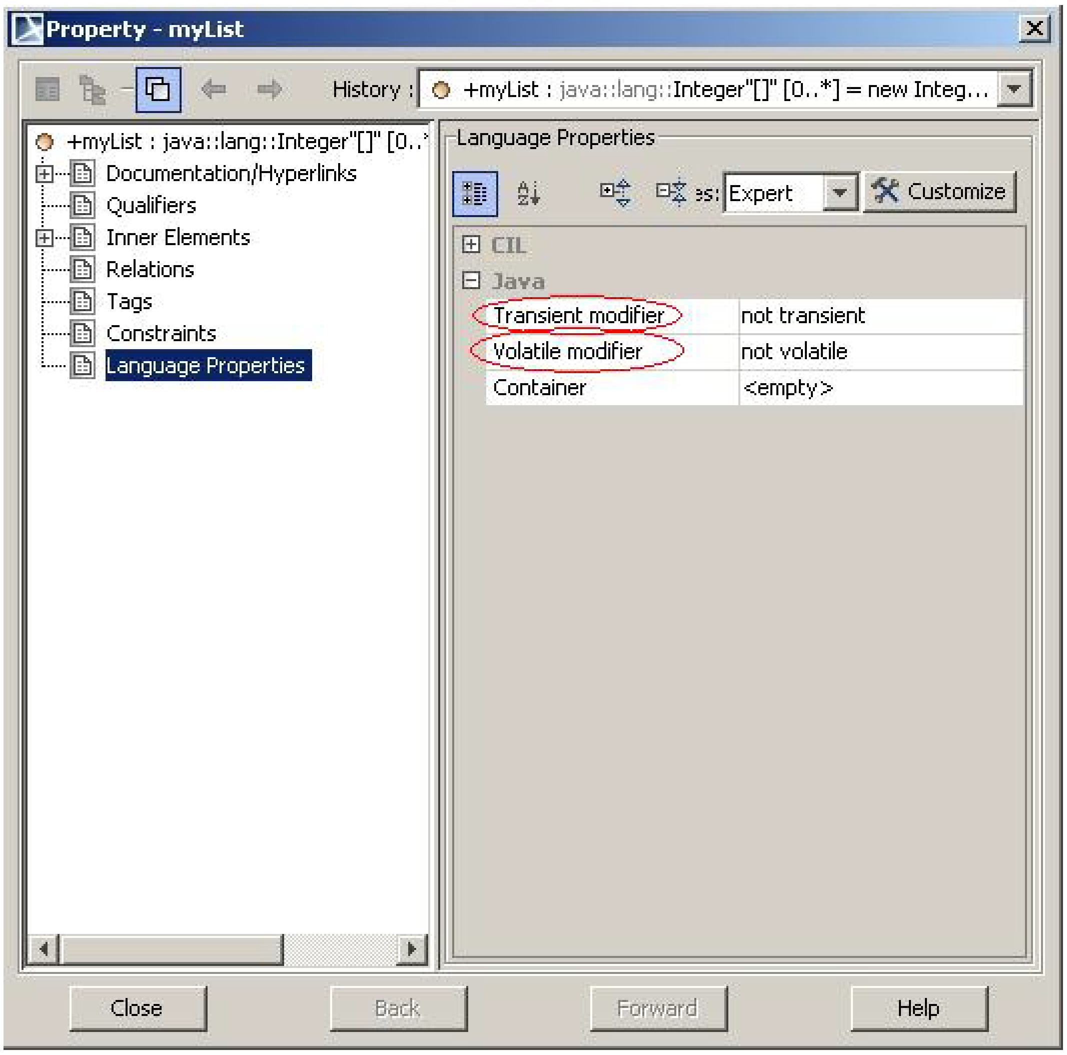
Copyright © 1998-2015 No Magic, Inc.



Property	
Name	myList
Qualified Name	MyClass::myList
Type	Integer [java::lang]
Type Modifier	[]
Visibility	public
Default Value	new Integer[5]
Owner	MyClass
Applied Stereotype	«» typeModifier [Element] [
Navigable	✓ true
Multiplicity	0..*
Is Read Only	false
Is Static	✓ true
Aggregation	none
Is Derived	false
Is Ordered	false
Is Unique	✓ true
Subsetted Property	
Redefined Property	
To Do	

Figure 19 -- UML Property specification dialog

Copyright © 1998-2015 No Magic, Inc.



Transient modifier	not transient
Volatile modifier	not volatile
Container	<empty>

Figure 20 -- UML Property language properties

Operation

Java operation is mapped directly to the UML Operation with stereotype «JavaOperation». This stereotype is optional and if UML class doesn't have any stereotype, Java CE treats it as Java operation. Operation modifiers are mapped into UML Property properties or to the Java language properties, if no appropriate property is found in UML.

Java operation return type is mapped to the UML Type property of UML Parameter with "Return" direction kind. Java Operation parameters are mapped to the UML Parameters. Direction kind is set "In" for "final" parameters.

Worth to know, that Java parameter type modifiers is mapped to the MagicDraw specific property "Type Modifier", but not to the UML Multiplicity.

Operation mapping table

Java element	**MagicDraw-UML element**
Operation declaration	UML Operation, owned by UML Class, with stereotype «JavaOperation» (optional)
Operation name	UML Operation Name

Copyright © 1998-2015 No Magic, Inc.

Operation documentation	UML Operation Documentation
Parameters list	UML Operation “Parameters” list
Return type	Is mapped to the UML Parameter type with “return“ direction kind (resides in UML Operation parameters list). It is reference to the UML Classifier, which by its package structure and name represents referenced Java class
Return type modifiers	Is mapped to the MagicDraw specified property “Type Modifier“
Visibility modifier	UML Operation “Visibility” property
Final modifier	UML Operation “Is Leaf“ property
Abstract modifier	UML Operation “Is Abstract“ property
Static modifier	UML Operation “Is Static“ property
Synchronized modifier	UML Operation “Concurrency“ kind “guarded”
Throws list	UML Operation “Raised Exception“ list.It is list of references to the UML Classes, which by its package structure and name represents referenced Java exception class
Native modifier	Java Language property “Native modifier”
Strictfp modifier	Java Language property “Strictfp modifier”

Parameter mapping table

Java element	**MagicDraw-UML element**
Parameter declaration	UML Parameter, owned by UML Operation, with stereotype «JavaParameter» (optional)
Parameter name	UML Parameter Name
Parameter documentation	When is used JavaDoc preprocessing, it is mapped to UML Parameter Documentation, else it is part of UML Operation Documentation.
Return type	Is mapped to the UML Type property. It is reference to the UML Classifier, which by its package structure and name represents referenced Java class.
Return type modifiers	Is mapped to the MagicDraw specified property “Type Modifier“
Final modifier	Direction kind “in“ of UML Parameter

Copyright © 1998-2015 No Magic, Inc.

Example

Java Source Code
```
public class MyList
{
     /**
      * Operation Comment
      */
     public abstract void foo(final List list) throws
IllegalArgumentException;
}
```

MagicDraw UML Model

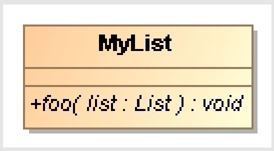
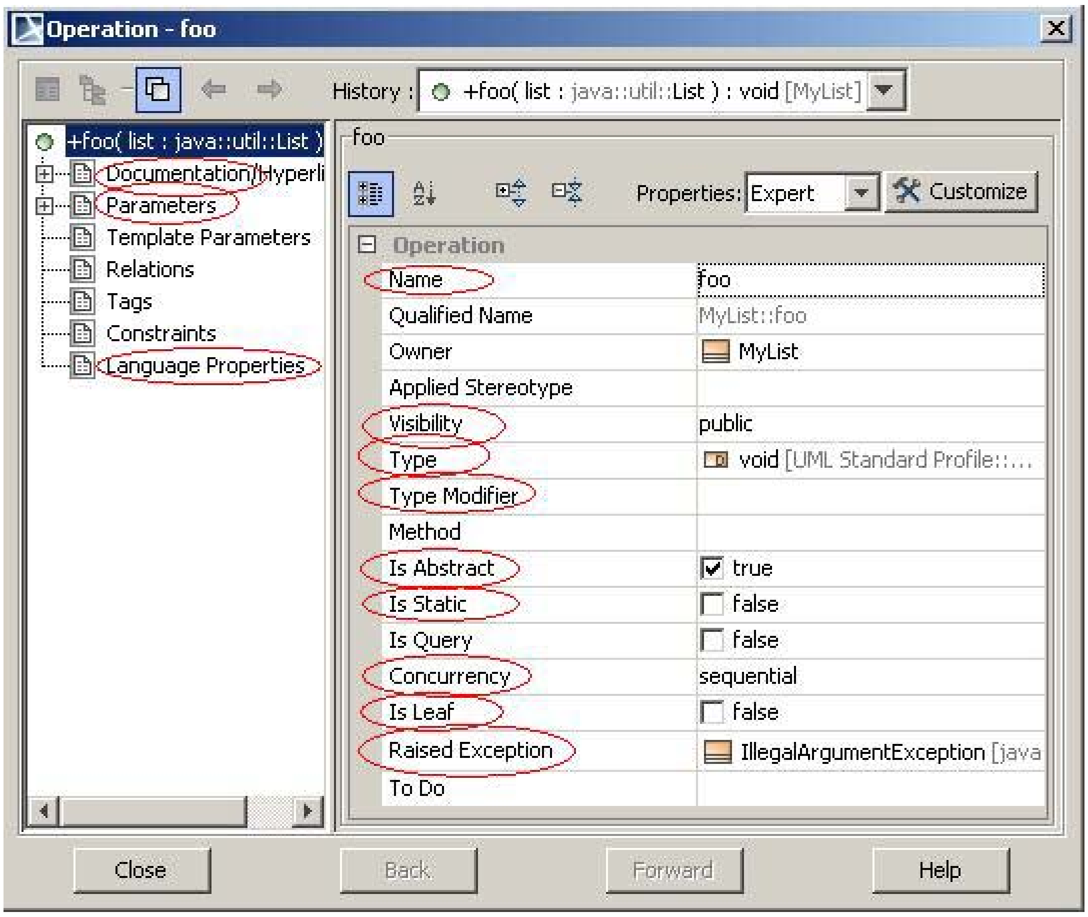


Figure 21 -- Class with operation



History : +foo(list : java::util::List) : void [MyList]

+foo(list : java::util::List)
- Documentation/Hyperli
- Parameters
- Template Parameters
- Relations
- Tags
- Constraints
- Language Properties

foo

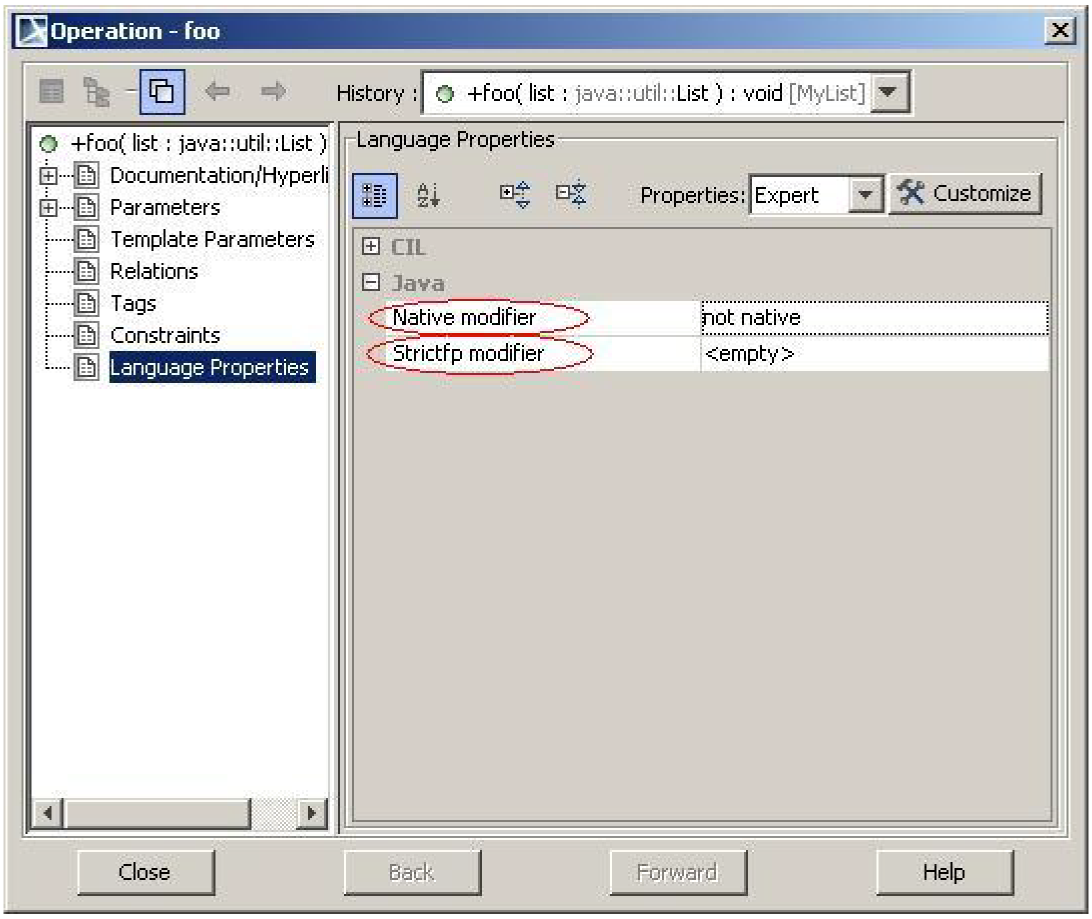
Properties: Expert

Operation

Name	foo
Qualified Name	MyList::foo
Owner	MyList
Applied Stereotype	
Visibility	public
Type	void [UML Standard Profile::...
Type Modifier	
Method	
Is Abstract	☑ true
Is Static	☐ false
Is Query	☐ false
Concurrency	sequential
Is Leaf	☐ false
Raised Exception	IllegalArgumentException [java
To Do	

Figure 22 -- UML Operation specification dialog

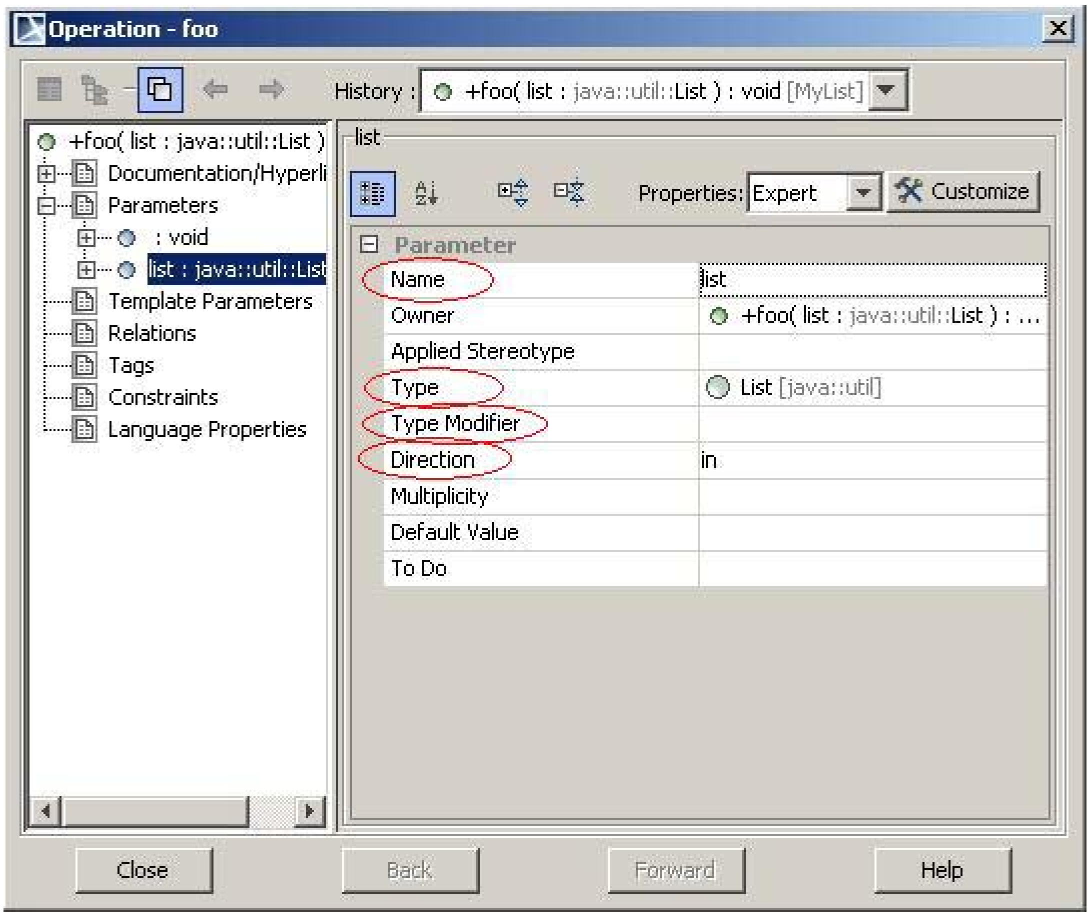
Copyright © 1998-2015 No Magic, Inc.



Native modifier	not native
Strictfp modifier	<empty>

Figure 23 -- UML Operation language properties

Copyright © 1998-2015 No Magic, Inc.



Parameter	
Name	list
Owner	+foo(list : java::util::List) : ...
Applied Stereotype	
Type	List [java::util]
Type Modifier	
Direction	in
Multiplicity	
Default Value	
To Do	

Figure 24 -- UML Parameter specification dialog

Interface

Java interface is mapped directly to the UML Interface with stereotype «JavaInterface». This stereotype is optional and if UML Interface doesn't have any stereotype, Java CE treats it as Java interface. Interface modifiers are mapped into UML Interface properties or to the Java language properties for interface, if no appropriate property is found in UML.

All mapping rules used in Java class mapping is applicable to the Java interface. See “Class” on page 28.

Enumeration

Java enumeration is mapped directly to the UML Class with stereotype «JavaEnumeration». Enumeration modifiers are mapped into UML Class properties or to the Java language properties for interface, if no appropriate property is found in UML.

Java enumeration literals are mapped to the UML Property, but with stereotype «JavaEnumerationLiteral». All contained fields, operations and inner classes are mapped to appropriate UML Properties, UML Operations, UML Classes.

All mapping rules used in Java class mapping is applicable to the Java enumeration. See “Class” on page 28.

Copyright © 1998-2015 No Magic, Inc.

Example

Java Source Code

```
enum MyInterface
{
}
```

MagicDraw UML Model

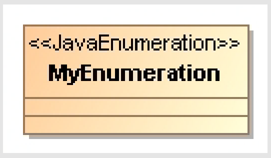


Figure 25 -- UML Enumeration

Enumeration Literal

Java enumeration literal is mapped directly to the UML Property with stereotype «JavaEnumerationLiteral». It is not required to specify any specific modifiers for enumeration literal.

Enumeration literal mapping table

Java element	**MagicDraw-UML element**
Literal declaration	UML Parameter with stereotype «JavaEnumerationLiteral», owned by UML Class with stereotype «JavaEnumeration»
Literal name	UML Property Name
Literal documentation	When is used JavaDoc preprocessing, it is mapped to UML Parameter Documentation, else it is part of UML Operation Documentation.

Example

Java Source Code

```
enum MyEnumeration
{
     ONE, TWO, THREE;

     int attribute1;
     String attribute2;
}
```

MagicDraw UML Model

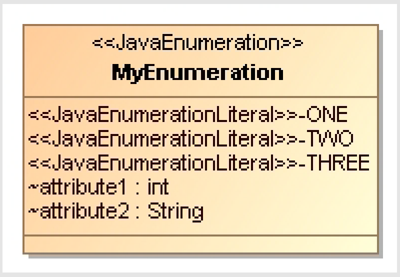

<<JavaEnumeration>>
MyEnumeration
<<JavaEnumerationLiteral>>-ONE
<<JavaEnumerationLiteral>>-TWO
<<JavaEnumerationLiteral>>-THREE
~attribute1 : int
~attribute2 : String

Figure 26 -- UML Class representing Java enumeration with enumeration literals

Copyright © 1998-2015 No Magic, Inc.

Annotation type

Java annotation declaration is mapped directly to the UML Interface with stereotype «JavaAnnotation». Annotation modifiers are mapped into UML Interface properties. Annotation members are mapped to the UML Interface operations with stereotype «JavaAnnotationMember».

Annotation mapping table

Java element	**MagicDraw-UML element**
Annotation declaration	UML Interface with stereotype «JavaAnnotation» (optional)
Annotation name	UML Interface name
Annotation documentation	UML Interface Documentation
Visibility modifier	UML Interface “Visibility” property

Example

Java Source Code

```
/**
 * Comment of annotation
 */
public @interface Annotation
{
}
```

MagicDraw UML Model

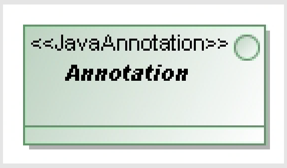


Figure 27 -- UML Interface representing Java annotation type

Annotation Member

Java annotation member is mapped directly to the UML Operation, owned by the interface stereotyped as «JavaAnnotation». Operation, by itself, can have stereotype «JavaAnnotationMember», but it is optional, unless you are going to specify default value for it.

Java annotation member type is mapped to UML Type property of UML Parameter with “Return“ direction kind.

Annotation member mapping table

Java element	**MagicDraw-UML element**
Annotation member declaration	UML Operation with stereotype «JavaAnnotationMember» (Optional), owned by UML Interface with Stereotype «JavaAnnotation».
Annotation member name	UML Operation Name

Copyright © 1998-2015 No Magic, Inc.

Annotation member documentation	UML Operation Documentation
Annotation member type	Is mapped to the UML Parameter type with “return“ direction kind (resides in UML Operation parameters list)
Annotation member type modifiers	Is mapped to the MagicDraw specified property “Type Modifier“
Default value	{JavaAnnotationMemberDefaultValue} tagged value of «JavaAnnotationMember» stereotype. Stereotype is set on UML Operation

Example

Java Source Code

```
/**
 * Comment of annotation
 */
public @interface Annotation
{
    int    id();
    String name() default "[unassigned]";
}
```

MagicDraw UML Model

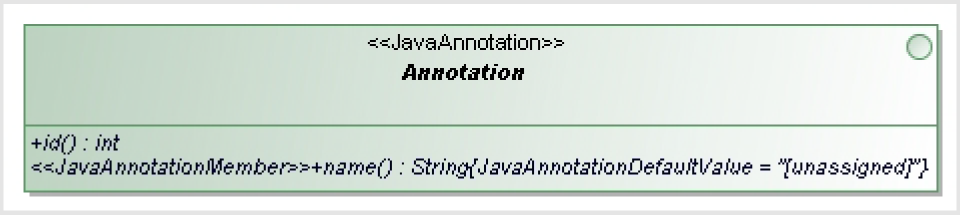
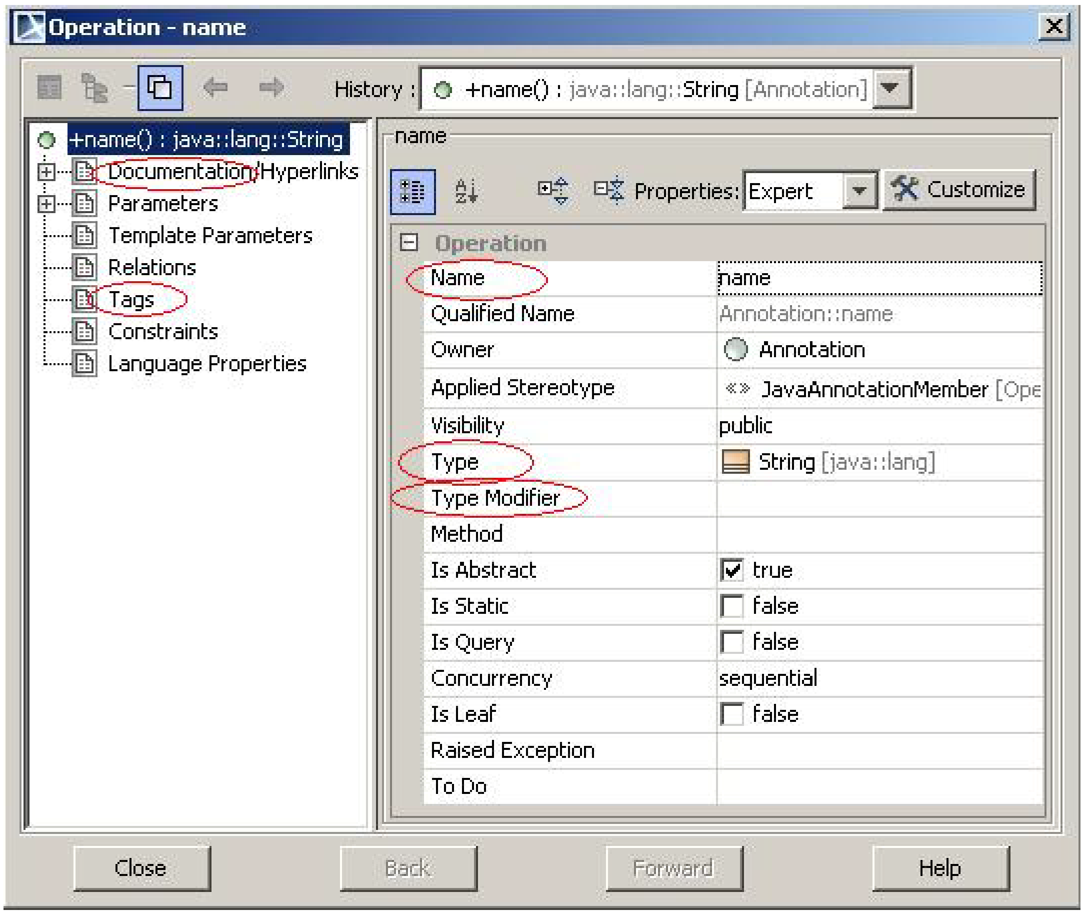

<<JavaAnnotation>>
Annotation
+id() : int
<<JavaAnnotationMember>>+name() : String{JavaAnnotationDefaultValue = "[unassigned]"}

Figure 28 -- UML Interface representing Java annotation type with Java annotation members

Copyright © 1998-2015 No Magic, Inc.



Operation	
Name	name
Qualified Name	Annotation::name
Owner	Annotation
Applied Stereotype	«» JavaAnnotationMember [Ope
Visibility	public
Type	String [java::lang]
Type Modifier	
Method	
Is Abstract	☑ true
Is Static	☐ false
Is Query	☐ false
Concurrency	sequential
Is Leaf	☐ false
Raised Exception	
To Do	

Figure 29 -- UML Operation, representing Java annotation member, specification dialog

Copyright © 1998-2015 No Magic, Inc.

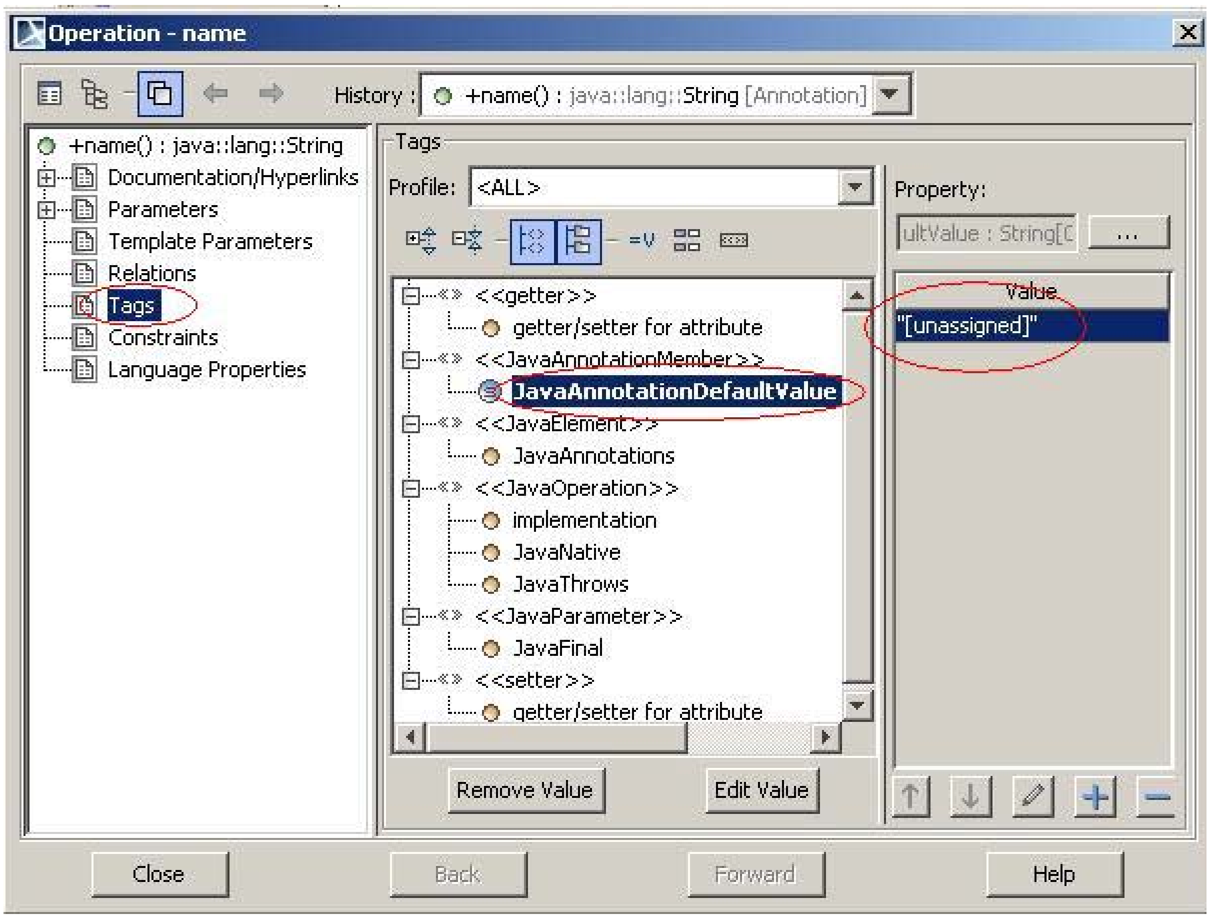


Figure 30 -- Default value set for UML Operation, representing Java annotation member

Annotations Usage

Java element can be annotated. Such annotation is mapped top the {JavaAnnotations} tagged value of the stereotype «JavaElement». «JavaElement» is base stereotype for all stereotypes, used in Java mapping, and it can be used directly or any other stereotype derived from it. Annotation is mapped as simple string value.

Copyright © 1998-2015 No Magic, Inc.

Example

Java Source Code

```
public class Test
{

    @Annotation
    (
        id      = 2,
        name = "Rick"
    )
    public void foo(){}
}
```

MagicDraw UML Model

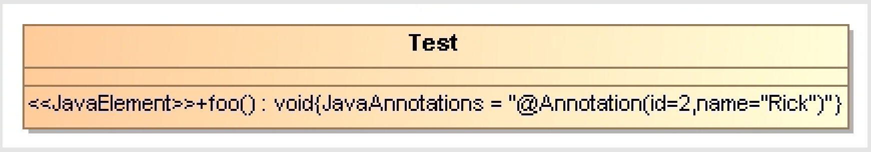

Test
<<JavaElement>>+foo() : void{JavaAnnotations = "@Annotation(id=2,name="Rick")"}

Figure 31 -- Java annotation usage

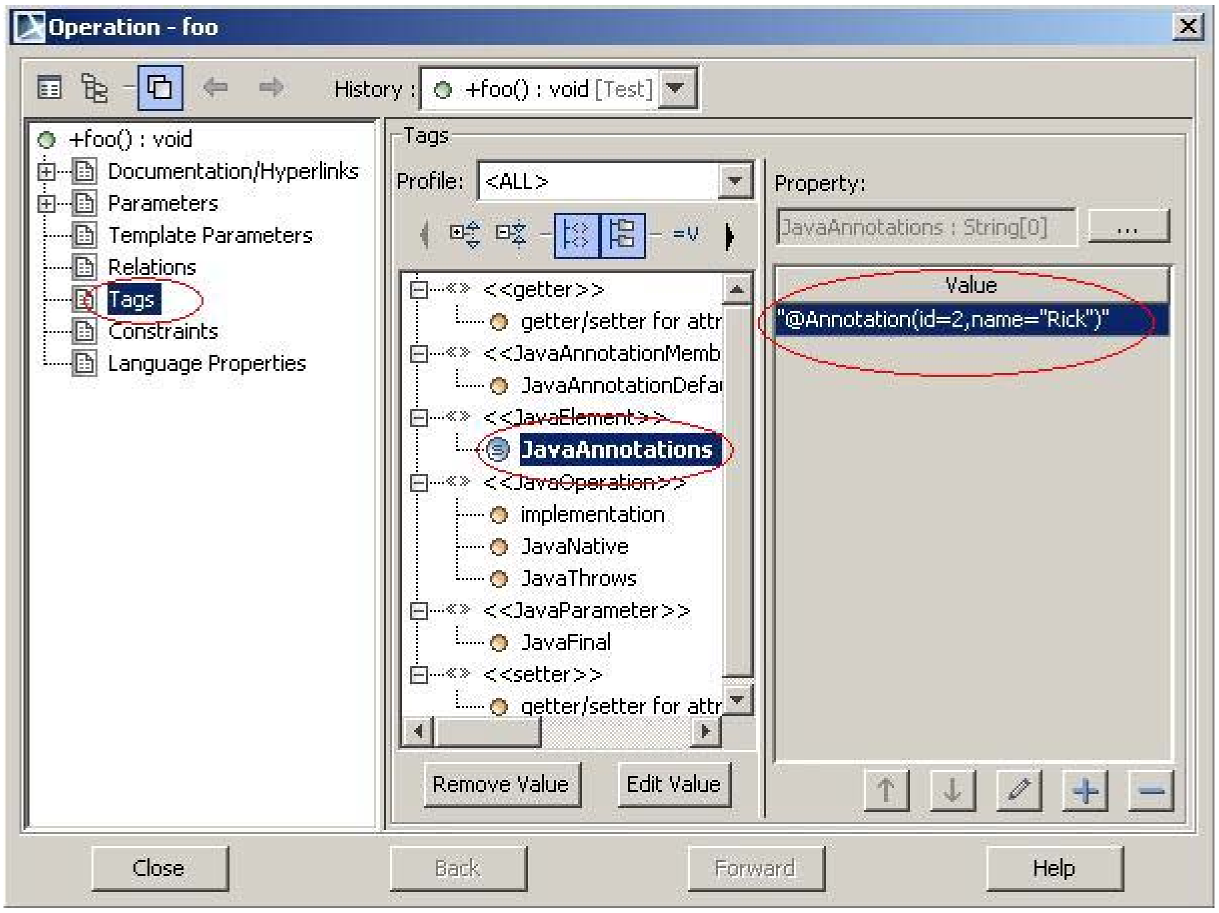


Figure 32 -- Annotation used on operation “foo”

Copyright © 1998-2015 No Magic, Inc.

Type Variables

Type variables are mapped to the UML Template Parameter of UML Class, Interface or UML Operation, regarding to what Java element has type variables. If bound type are present, they are mapped into the UML Class or UML Interface connected with UML Generalization or UML Interface Realization as a supplier and client is UML Class, which is “Parametered Element” of the UML Template Parameter.

Type variable mapping table

Java element	MagicDraw-UML element
Type variable declaration	Is mapped to the UML Template Parameter. This UML Template Parameter is of the Class type from the UML Metamodel. UML Template Parameter has property “Parametered Element“ of the UML Template Parameter. Metamodel type “Class“ is taken from the “UML Standard profile/UML 2 Metamodel”
Type variable name	Is mapped to the UML Class name. This UML Class is “Parametered Element“
Type bounds	If bound type is a Java class, it is mapped to the General class of the “Parametered Element“. If bound type is a Java interface, than it is mapped to the Realized Interface.

Example

Java Source Code

```
public class Test <E extends Cloneable>
{
     E attribute;
}
```

MagicDraw UML Model

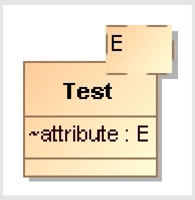


Figure 33 -- UML Class with template parameter E, representing Java type variable

Copyright © 1998-2015 No Magic, Inc.

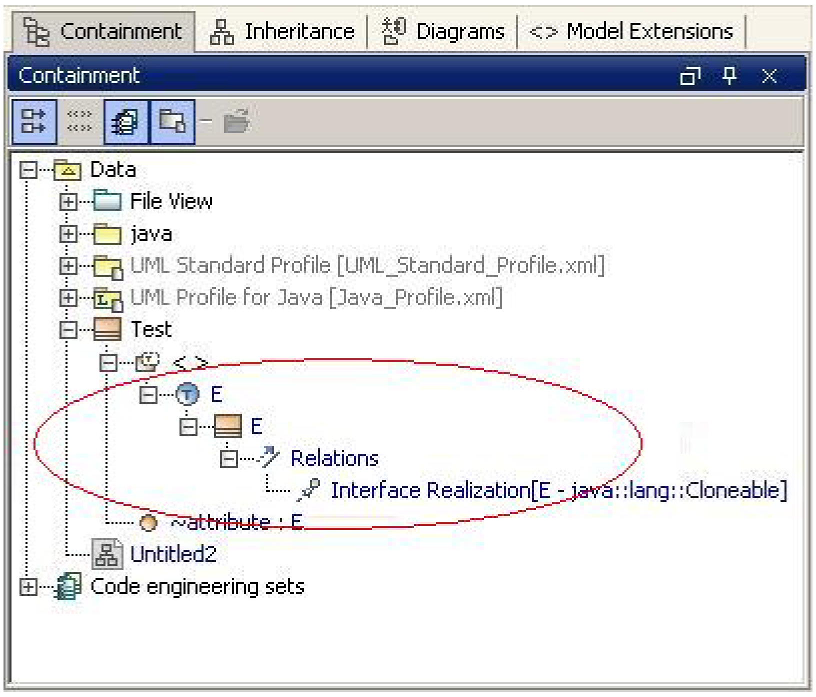
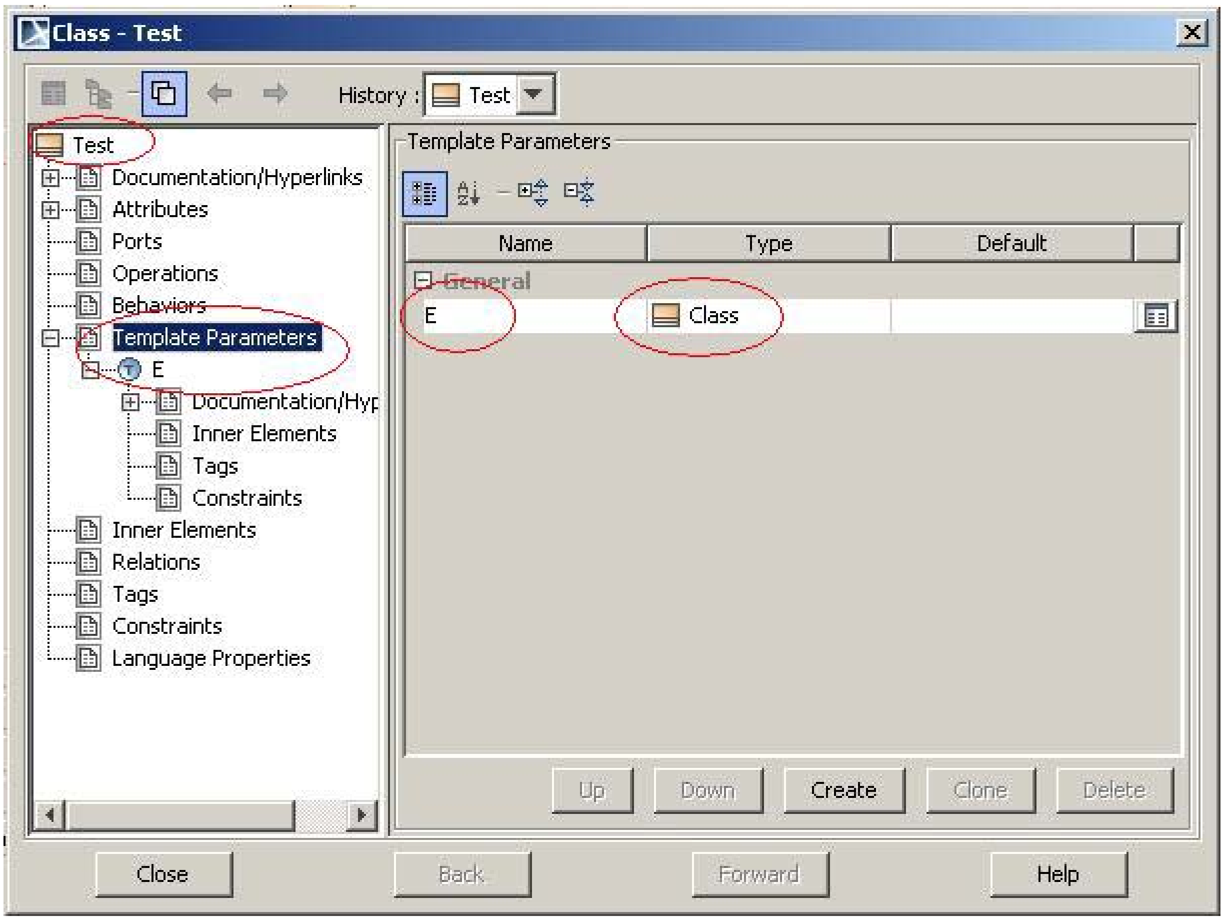


Figure 34 -- Marked UML Template Parameter, with residing UML Class named E (Parametered Element)

Note, that residing UML Class E in UML Template Parameter is realizing interface “java.util.Cloneable“ and this class is used as type for UML Class attribute.



Name	Type	Default
General		
E	Class	

Figure 35 -- Template parameters in UML Class specification dialog

Copyright © 1998-2015 No Magic, Inc.

Parameterized Type

Parameterized types are mapped to the general UML Classifier connected with UML Template Binding to the UML Classifier. Supplier of this binding link is UML Classifier with UML Template Parameters and represents Java generic type with type parameters. Client of UML Template Binding is UML Classifier of the same UML type as supplier is. Java type parameters are mapped directly to the UML Template Parameter Substitution of the UML Template Binding.

Parameterized type mapping table

Java element	**MagicDraw-UML element**
Parameterized type	In UML it is called bounded elements, which is connected to the UML Classifier by the UML Template Binding as a client. Client must be of the same type as is supplier. Supplier must have at least one UML Template Parameter.
Parameter for type	Is mapped to the UML Template Parameter Substitution of the UML Template Binding. Each UML Template Parameter from the supplier must be substituted by the UML Template Parameter Substitution. Type of type parameter is any reference to the UML Classifier from the model which is set as “Actual“ value of the UML Template Parameter Substitution
Type modifiers of the parameter	It is mapped to the MagicDraw property “Type modifiers” of the UML Template Parameter Substitution
Wildcard	UML Class with a name “?“ from the Java Profile is used as “Actual” value in UML Template Parameter Substitution.
Wildcard with bounds	Java bounding type is mapped to the UML Classifier and it is used as “Actual” value in UML Template Parameter Substitution. “? extends“ or “? super“ bounding is mapped to the appropriate tag {JavaArgumentBount} value “extends“ or “super“ of the «JavaTypeArgument» stereotype applied to the UML Template Parameter Substitution.

Copyright © 1998-2015 No Magic, Inc.

Example

When you have type with type variables represented in the model, you can create parameterized type for Java. For this you need to create empty UML Classifier of the type the template classifier (type with type variables) is. Then create a UML Template Binding and create UML Template Parameter Substitution for UML Template Parameters.

In order to create “java.util.List<String>” type, we need to create UML Interface first, with UML Template Parameter representing “java.util.List”.

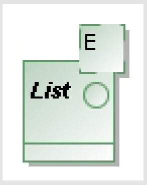


Figure 36 -- UML Interface representing “java.util.List” with template parameter

Then create another UML Interface and connect with List interface with UML Template Binding.

Note, that in order to draw UML Template Binding, client element must have at least one UML Template Parameter created

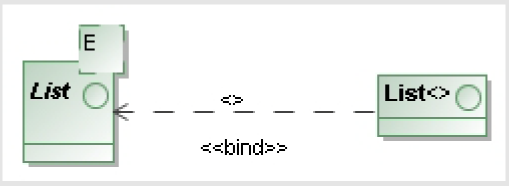


Figure 37 -- UML Template Binding between template class and bounded element

You need to open UML Template Binding specification dialog and create UML Template Parameter Substitution for appropriate UML Template Parameter.

Copyright © 1998-2015 No Magic, Inc.

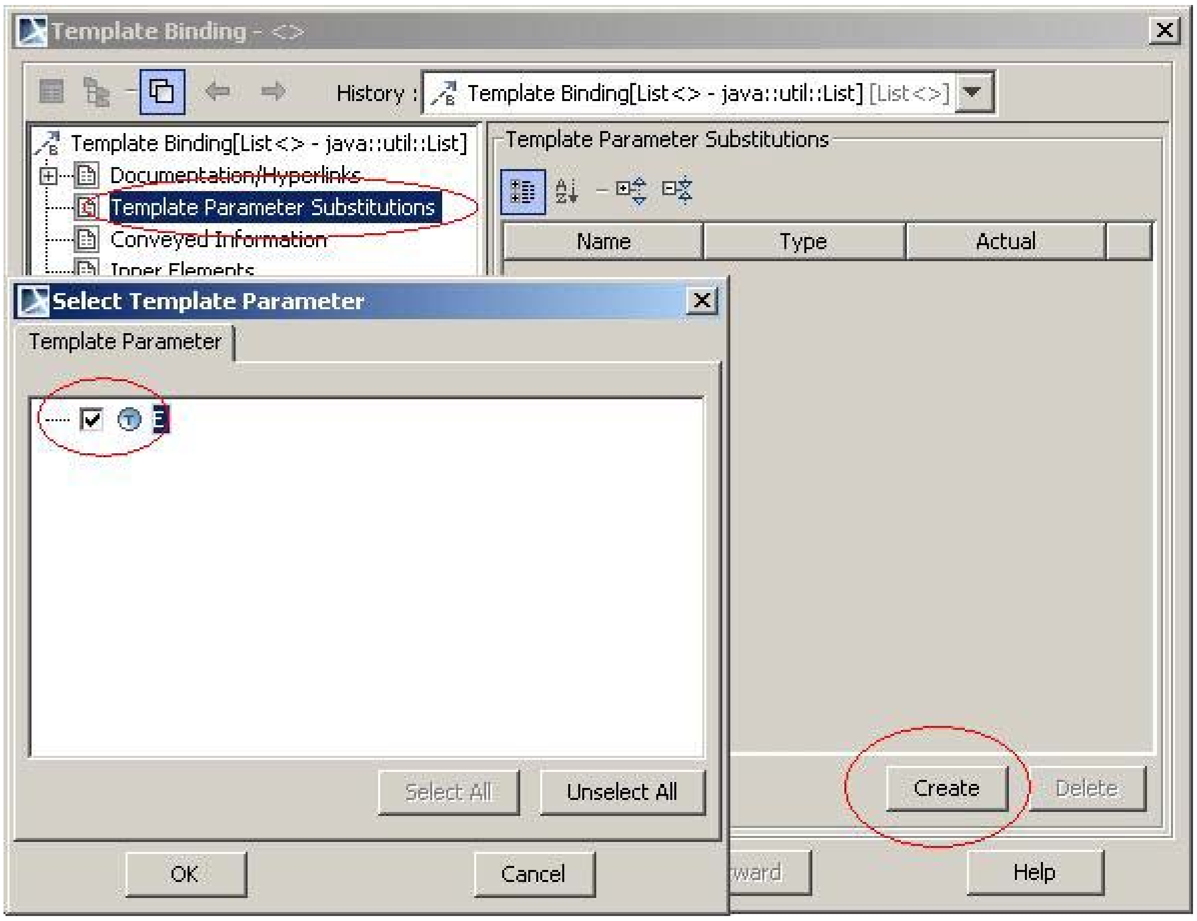

Template Binding - <>
History : Template Binding[List<> - java::util::List] [List<>]
Template Binding[List<> - java::util::List]
Documentation/Hyperlinks
Template Parameter Substitutions
Conveyed Information
Inner Elements
Template Parameter Substitutions
Name
Type
Actual
Select Template Parameter
Template Parameter
E
Select All
Unselect All
Create
Delete
OK
Cancel
Help

Figure 38 -- Creating UML Template Parameter Substitution

For created substitution element, you need to select actual value - at current situation it is UML class representing "java.lang.String".

Now we have created parameterized type, which can be used in the model to represent type java.util.List<String>

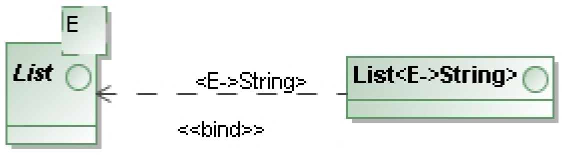


Figure 39 -- Created parameterized type for "java.util.List<String>"

Imports

parameters

Java element	MagicDraw-UML element
Type import	Mapped to the UML Element Import with stereotype «JavaImport» (optional). Supplier is imported UML Classifier, which represents imported Java type, and client is UML Classifier which requires imported element.

Copyright © 1998-2015 No Magic, Inc.

Package import	Mapped to the UML Package Import with stereotype «JavaImport» (optional). Supplier is imported UML Package, which represents imported Java package, and client is UML Classifier which requires imported elements.
Static import for all static members	Mapped to the UML Element Import with stereotype «JavaStaticImport». {JavaImportAll} tag of «JavaStaticImport» must have “true“ value. Supplier is imported UML Classifier, owner of static members, which are imported, and client is UML Classifier which requires imported elements.
Static import for single static members	Mapped to the UML Element Import with stereotype «JavaStaticImport». {JavaImportAll} tag of «JavaStaticImport» must have “false“ value and tag {JavaImportedMember} must have reference to the imported member. Supplier is UML Classifier, owner of static member, which is imported, and client is UML Classifier which requires imported elements. If there are several static members imported, {JavaImportedMember} can have listed all of them.

Copyright © 1998-2015 No Magic, Inc.

Example

Java Source Code

```
import java.util.List;
import java.util.*;
import static java.lang.Math.*;

public class Test
{
}
```

MagicDraw UML Model

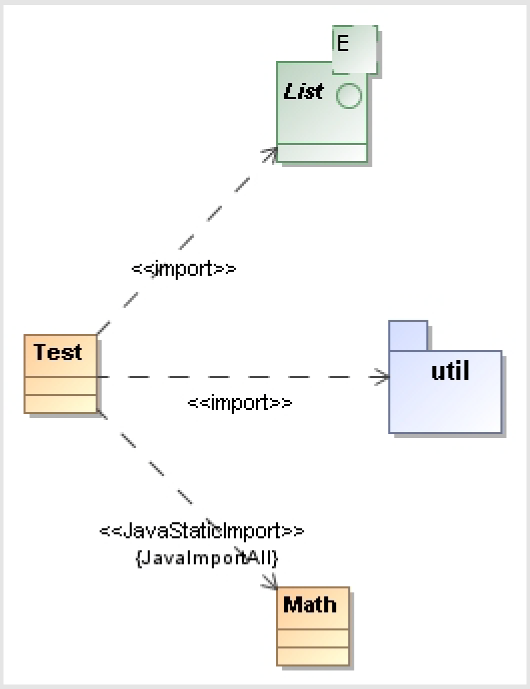


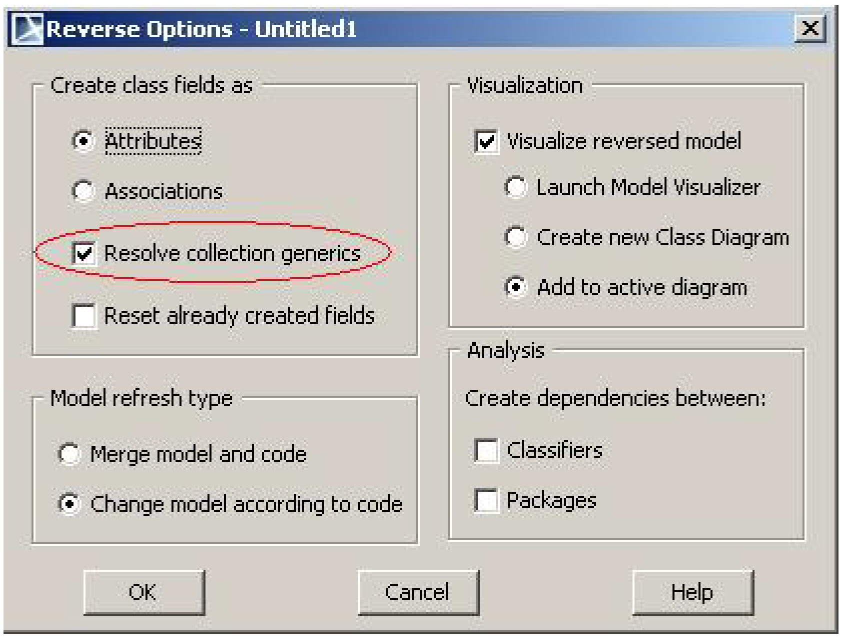
Figure 40 -- Java imports in diagram

Copyright © 1998-2015 No Magic, Inc.

Java CE Properties

Java Reverse Properties

These options are visible every time you attempt to reverse source code. Here we will describe marked options in Figure 41 on page 52. Other options are common for all languages and are described in "Reverse", on page 12.



Reverse Options - Untitled1

Create class fields as
- [✓ Attributes]
- Associations
- [✓] Resolve collection generics
- [ ] Reset already created fields

Model refresh type
- Merge model and code
- [✓ Change model according to code]

Visualization
- [✓] Visualize reversed model
  - Launch Model Visualizer
  - Create new Class Diagram
  - [✓ Add to active diagram]

Analysis

Create dependencies between:
- [ ] Classifiers
- [ ] Packages

Figure 41 -- Java reverse properties

Resolve Collection Generics Option

By default this option is turned on. Since the JLS 3, in Java was introduced parameterized types and to all Java collections were added type variables. Now, on reverse engineering now it is possible to find out what type is in Java collection and make association directly to the contained type instead of the Java collection.

On reverse engineering, it finds out Java parameterized collection and retrieves Java type which is used in container. This type is set as a “Type” to the UML Property. Container type is set to the UML Property Java language property “Container“ as simple string.

Example

Source Code Sample

```
public class Test
{
     java.util.List<String> attribute;
}
```

MagicDraw UML Model

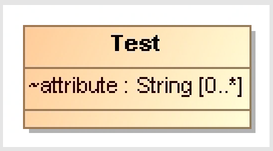
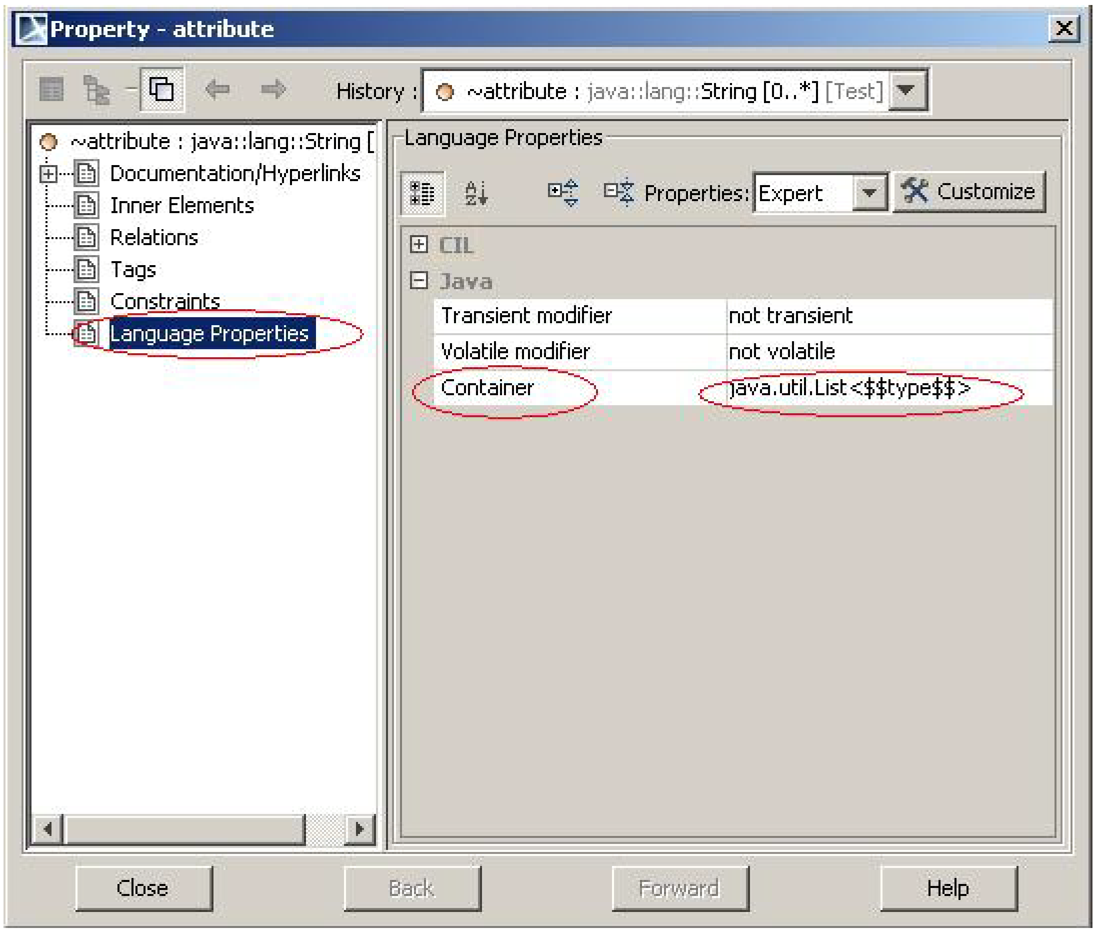


Figure 42 -- Attribute type, retrieved from collection

Copyright © 1998-2015 No Magic, Inc.



Transient modifier	not transient
Volatile modifier	not volatile
Container	java.util.List<$$type$$>

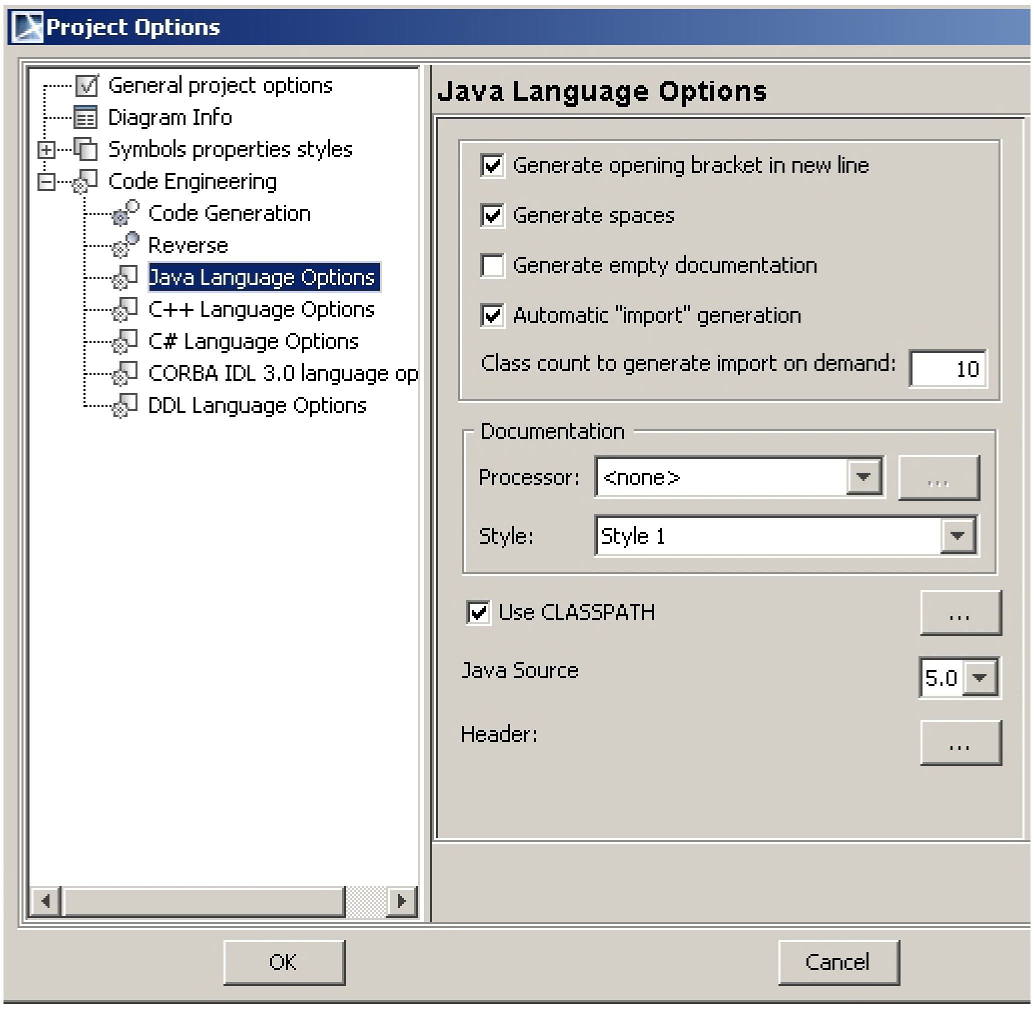
Figure 43 -- Collection type in UML Property specification

Note, that $$type$$ shows where should be in lined UML Property type on code generation.

Java Language Options

You can find these options at the Options-> Project-> Code Engineering-> Java Language Options.

Copyright © 1998-2015 No Magic, Inc.



Java Language Options

- [✓] Generate opening bracket in new line
- [✓] Generate spaces
- [ ] Generate empty documentation
- [✓] Automatic "import" generation

Class count to generate import on demand: 10

Documentation
Processor: <none>
Style: Style 1

- [✓] Use CLASSPATH

Java Source 5.0
Header:

Figure 44 -- Java language options

Generate Opening Bracket In New Line

By default is turned on. If element (Java type or operation) is generated for the first time into source, curly bracket is generated from the new line, if options is on, or in the same line as declaration header ends, if option is off.

Generate Spaces

By default it is turned on. If option is on - adds additional space after open bracket and before close bracket in parameter declaration list.

Generate Empty Documentation

By default is turned off. If option is turned on, MagicDraw is generating documentation to the source even, if there are no documentation in model. Just adds Java documentation start and ending symbols.

Here is generated class header with options turned on. UML Class A doesn’t have documentation, but still in code is added documentation elements.

```
/**
 *
 */
```

Copyright © 1998-2015 No Magic, Inc.

```
class A
```

Automatic “import“ Generation

By default is turned on. If option is turned on, all required imports are added automatically by used references on code generation. If this option is off, than imports are managed by user using specific mapping (see “Imports” on page 49).

Note, that if option is turned off, than no imports are created automatically and all imports, retrieved from source code on reverse engineering, are mapped into UML relationships as described in “Imports” on page 49.

Class Count To Generate Import On Demand

By default value is 10. It means, that if there will be 9 references from the same package to different types, then imports will be generated explicitly to these classes, but if there will be 10 and more references, than it will be generated one import to the package.

This option is valid only, when “Automatic import generation“ is turned on.

Documentation

Java documentation has two options by itself. It has **Processor** and **Style**. **Style** is used to define how to format documentation by adding some comments. There are predefined two styles:

Style 1

```
/**
 *
 */
```

Style 2

```
/**
*
*/
```

Processor is responsible for analyzing documentation context. There are two types <none> and Java Doc.

<none> options does nothing with documentation and just set it as is on element.

Java Doc is processing documentation by resolving parameter tags or on code generation building documentation by collecting comments from the UML Parameters and adding missed tags for thrown exceptions or return.

There are additional Java Doc options (button “...”). These options can be used to declare what tags would not be generated or what order to use on code generation or perhaps to add some additional tag to documentation for all elements.

Note, that **Java Doc** processor splits operation documentation for UML Parameter and UML Operation on source code reverse and on code generation UML Parameter documentation is used to build Java documentation for operation.

Classpath

You can define classpath here by referencing **jar** files or **class** files directories.

This options is used by Java reverse engineering. If referenced element is not found in model for some reasons, that it is searched in this defined path. And if class is matched by name, this class is added into model.

Copyright © 1998-2015 No Magic, Inc.

By default, MagicDraw imports boot classpath of the JVM, on which is running.

Java Source

By default is Java 5.0. There are options **1.4** and **5.0**. If you are reversing older specification source code, where, for example "**enum**" is not a keyword and can be a variable name, then you will need to choose **1.4** Java source, else MagicDraw parser can emit error.

Header

It is a header for a newly generated Java files. There can be added some template string which will be preprocessed on writing to source code.

Template strings

$FILE_NAME	File name, without a path
$DATE	System date
$TIME	System time
$AUTHOR	User name on the system

Method Implementation Reverse

Java reverse to Sequence diagram functionality allows visualizing Java method implementation with UML Sequence diagram. Created from method Sequence diagram cannot be updated, every time new diagram should be generated.

To launch **Sequence Diagram from Java Source Wizard** and specify options needed for the reverse

- You are able to reverse any operation from the Browser: right click an operation, choose **Reverse Implementation** and launch **Sequence diagram from Java Source Wizard**.
- From the **Tools** menu, choose **Model VIsualizer**, and then choose **Sequence Diagram from Java Source WIzard**.
- When reversing, in the **Reverse Options** dialog box, choose Launch Model Visualizer and then choose **Sequence Diagram from Java Source Wizard.**

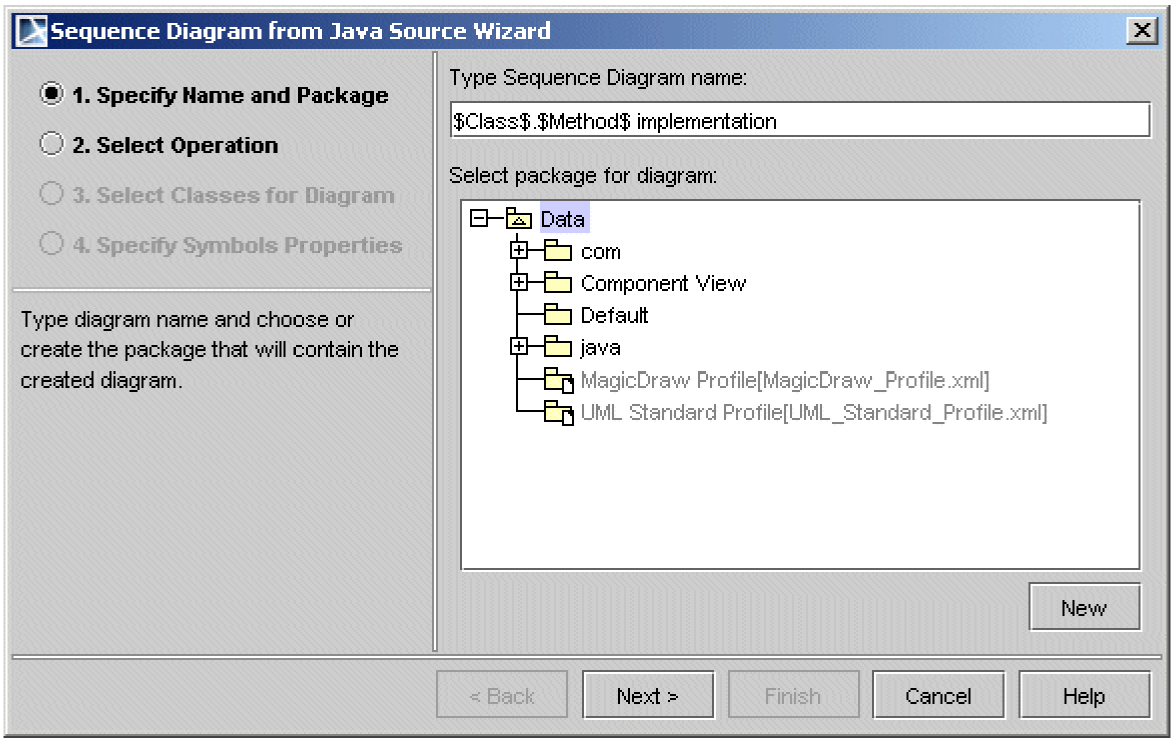
The more detailed example of how this functionality works, see MagicDraw Tutorials.pdf, which is locate in <MagicDraw installation directory>, manual folder.

Sequence Diagram from Java Source Wizard

Sequence Diagram from Java Source Wizard is the primary tool for reversing s sequence diagram from Java method. It contains four steps that are described below.

Copyright © 1998-2015 No Magic, Inc.

STEP 1 Specify Name and Package.


Sequence Diagram from Java Source Wizard

[✓ 1. Specify Name and Package]
2. Select Operation
3. Select Classes for Diagram
4. Specify Symbols Properties

Type diagram name and choose or create the package that will contain the created diagram.

Type Sequence Diagram name:
$Class$.$Method$ implementation

Select package for diagram:
- Data
  - com
  - Component View
  - Default
  - java
  - MagicDraw Profile[MagicDraw_Profile.xml]
  - UML Standard Profile[UML_Standard_Profile.xml]

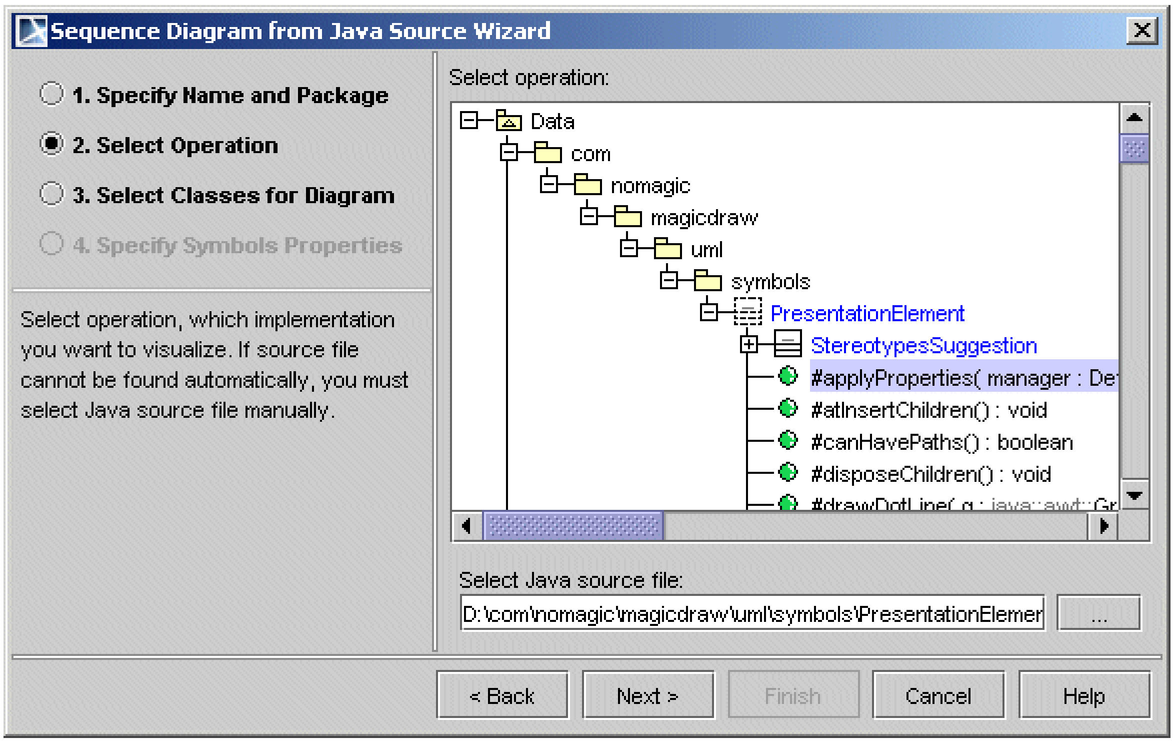
Figure 45 -- Sequence Diagram from Java Source Wizard

In this step, type the name of the newly created sequence diagram. Be default class name and selected operation name with a word “implementation” will be included in the sequence diagram name.

Also choose the package that will contain created sequence diagram. If you want to create a new package and place there a sequence diagram, click the **New** button and define package parameters in the **Package Specification** dialog box.

Copyright © 1998-2015 No Magic, Inc.

STEP 2 Select Operation


Sequence Diagram from Java Source Wizard
1. Specify Name and Package
2. Select Operation
3. Select Classes for Diagram
4. Specify Symbols Properties
< Back
Next >
Finish
Cancel
Help

Select operation, which implementation you want to visualize. If source file cannot be found automatically, you must select Java source file manually.

Select operation:

```
Data
  com
    nomagic
      magicdraw
        uml
          symbols
            PresentationElement
              StereotypesSuggestion
              #applyProperties( manager : De
              #atInsertChildren() : void
              #canHavePaths() : boolean
              #disposeChildren() : void
```

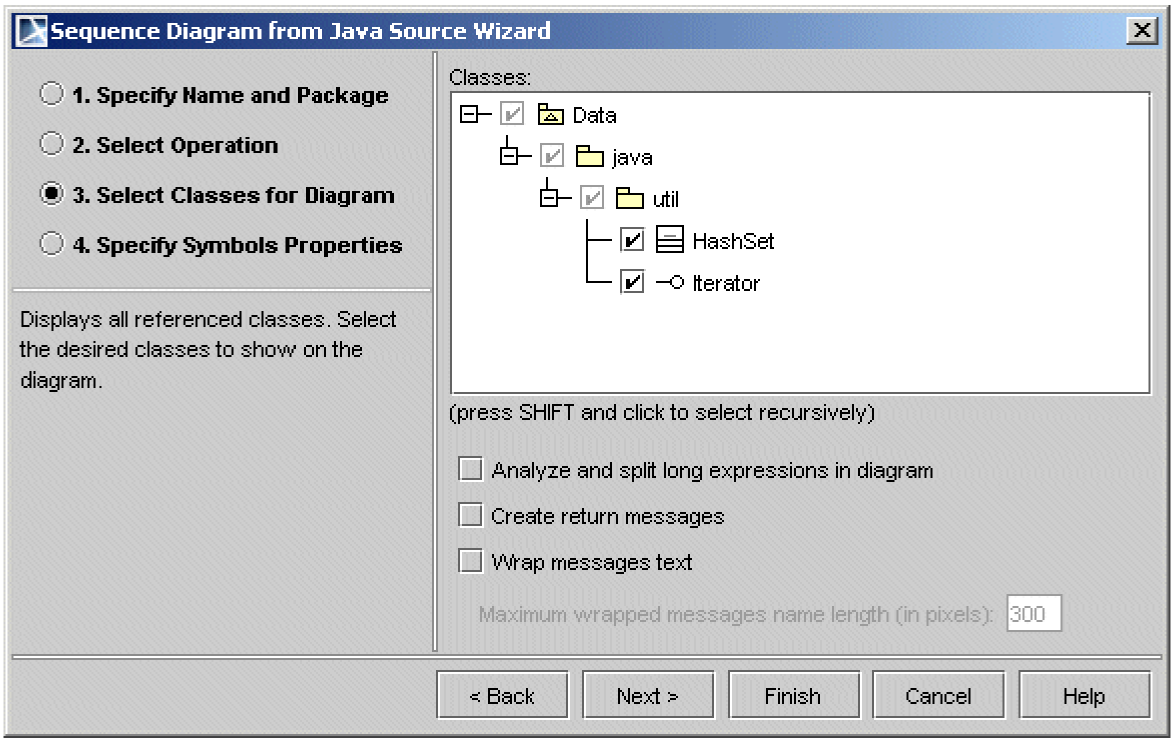
Select Java source file:
D:\com\nomagic\magicdraw\uml\symbols\PresentationElemer

In this step, select an operation for which you want to create a sequence diagram. If the Java source file is not shown you must select it manually.

IMPORTANT	To specify implementation files, we suggest, before reversing, to specify Java **Default working directory** in the **Project Options** dialog box (specify root folder where all source files can be found).

Copyright © 1998-2015 No Magic, Inc.

STEP 3 Select Classes for Diagram


Sequence Diagram from Java Source Wizard

( ) 1. Specify Name and Package
( ) 2. Select Operation
(●) 3. Select Classes for Diagram
( ) 4. Specify Symbols Properties

Displays all referenced classes. Select the desired classes to show on the diagram.

Classes:

- ✓ Data
  - ✓ java
    - ✓ util
      - ✓ HashSet
      - ✓ Iterator

(press SHIFT and click to select recursively)

☐ Analyze and split long expressions in diagram
☐ Create return messages
☐ Wrap messages text

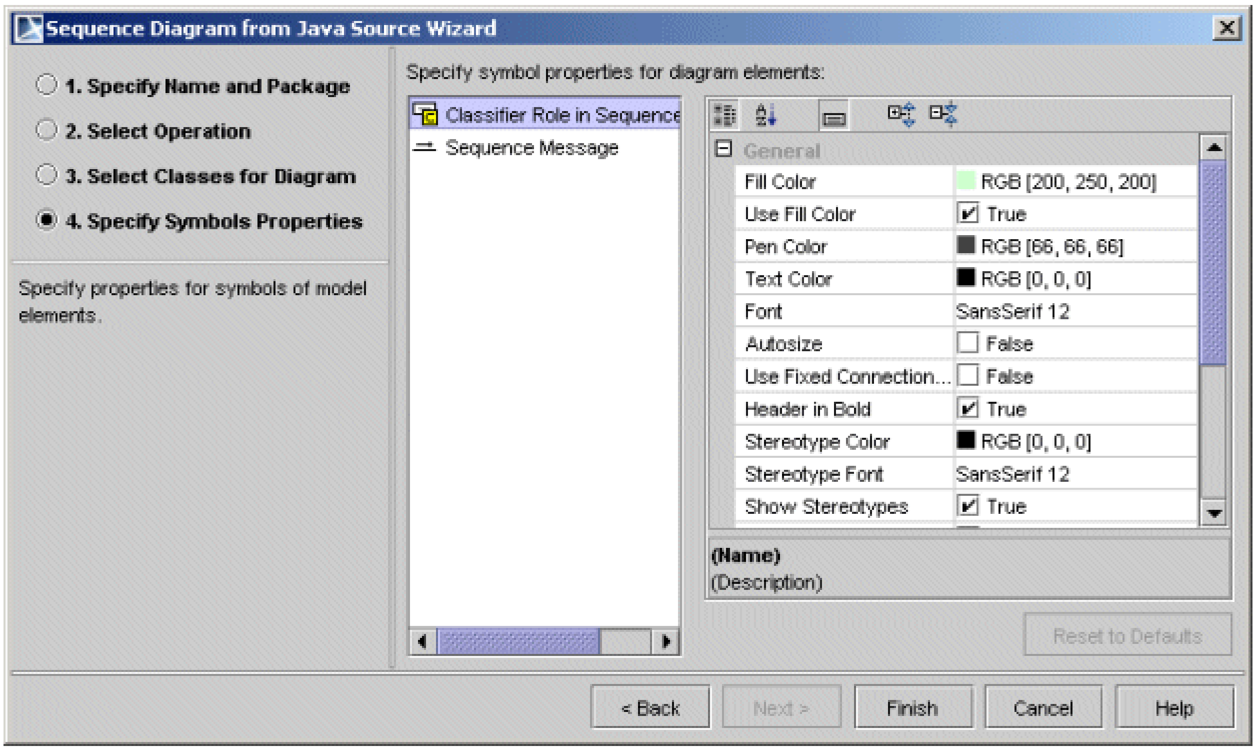
Maximum wrapped messages name length (in pixels): 300

In the Select Classes for Diagram step, all referenced classes are displayed. Select the desired classes and instances of those classes will be added into diagram with call messages to them.

- Select the **Analyze and split long expressions in diagram** check box if expression contains calls and cannot be displayed as call message. Then every call will be shown as separate call message with temporary variable initialization.
- Select the **Create return message** check box, if you want to display return message for every call message.
- Select the **Wrap message text** check box and specify the maximum message text length in pixels, to wrap longer message.

Copyright © 1998-2015 No Magic, Inc.

STEP 4 Specify Symbols Properties


Sequence Diagram from Java Source Wizard
1. Specify Name and Package
2. Select Operation
3. Select Classes for Diagram
4. Specify Symbols Properties
Classifier Role in Sequence
Sequence Message
General
Reset to Defaults
< Back
Next >
Finish
Cancel
Help

( ) 1. Specify Name and Package
( ) 2. Select Operation
( ) 3. Select Classes for Diagram
(●) 4. Specify Symbols Properties

Specify properties for symbols of model elements.

Specify symbol properties for diagram elements:

Fill Color	RGB [200, 250, 200]
Use Fill Color	✓ True
Pen Color	RGB [66, 66, 66]
Text Color	RGB [0, 0, 0]
Font	SansSerif 12
Autosize	False
Use Fixed Connection...	False
Header in Bold	✓ True
Stereotype Color	RGB [0, 0, 0]
Stereotype Font	SansSerif 12
Show Stereotypes	✓ True

(Name)
(Description)

Figure 46 -- Sequence Diagram from Sequence Wizard. Specify Symbols Properties

In this step, define symbols properties for lifelines and messages.

Copyright © 1998-2015 No Magic, Inc.

C++ CODE ENGINEERING

Abbreviations

GUI	Graphical User Interface
CE	Code Engineering
CES	Code Engineering Set
AST	Abstract Syntax Tree
RT	Round-Trip forward and backward code engineering without code loss
CLR	Common Language Runtime
DSL	Domain Specific Language

References

ISO/IEC 14882 C++ ANSI spec	ANSI_C++_Spec_2003.pdf
MSDN Library - Visual Studio 2005	ECMA-372 C++/CLI Language Specification

C++ ANSI Profile

The ANSI C++ profile is the base for all other C++ profiles. Other specialized C++ profiles need to inherit from this profile (by creating a generalization in MD).

Data Types

Fundamental types defined by ANSI (ANSI spec 3.9.1) are mapped to UML data type.

Each fundamental type with a modifier (signed, unsigned, short, long) can be declared in different order, but it is the same type. For example, short int or int short. Only versions defined as “type” by ANSI simple-type-specifiers are created as datatype in the profile. (See ANSI spec 7.1.5.2) If an UML synonym dataType is found during the reverse process, then a class is created with this name.

Each fundamental type has 3 corresponding cv-qualified versions: const, volatile, and const volatile (or volatile const). See *Const Volatile qualified type* for mapping.

UML data types defined in the MD UML profile contain char, int, double, float, or void. We use these datatypes for mapping C++ type.

These datatypes are MD specific. In UML 2 only Integer, Boolean, String, and UnlimitedNatural are defined and they are not defined as datatypes, but as primitives. In the current version of MD UnlimitedNatural is not defined in any profile and Integer and boolean (b is lowercase) are datatypes.

Copyright © 1998-2015 No Magic, Inc.

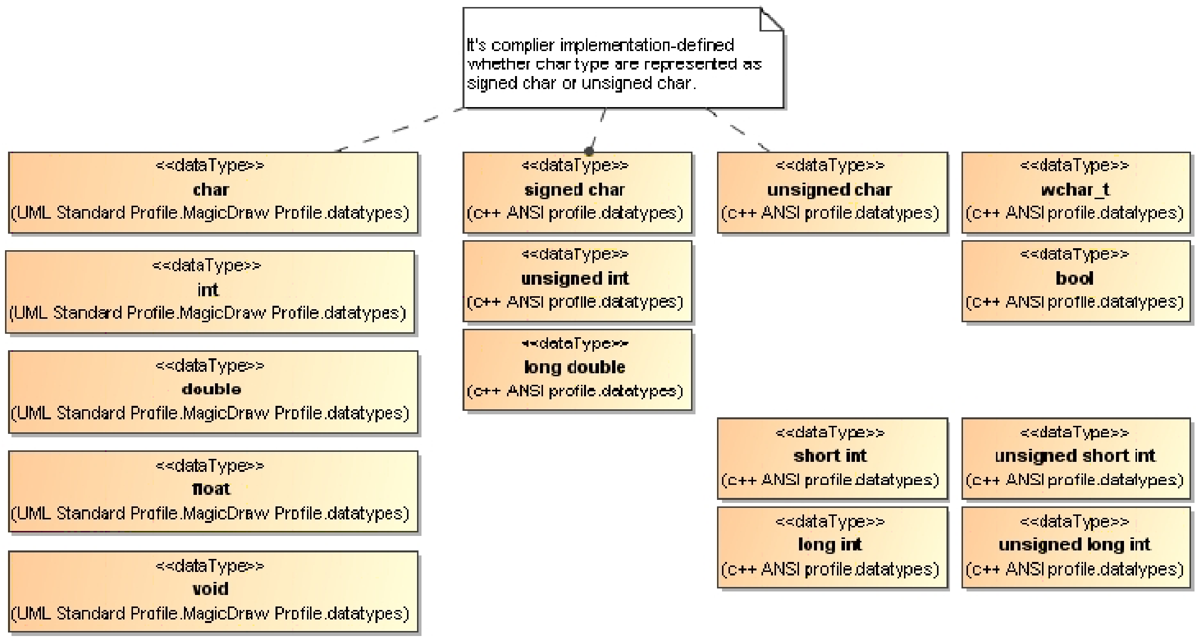

It's complier implementation-defined whether char type are represented as signed char or unsigned char.
<<dataType>> char (UML Standard Profile.MagicDraw Profile.datatypes)
<<dataType>> int (UML Standard Profile.MagicDraw Profile.datatypes)
<<dataType>> double (UML Standard Profile.MagicDraw Profile.datatypes)
<<dataType>> float (UML Standard Profile.MagicDraw Profile.datatypes)
<<dataType>> void (UML Standard Profile.MagicDraw Profile.datatypes)
<<dataType>> signed char (c++ ANSI profile.datatypes)
<<dataType>> unsigned int (c++ ANSI profile.datatypes)
<<dataType>> long double (c++ ANSI profile.datatypes)
<<dataType>> unsigned char (c++ ANSI profile.datatypes)
<<dataType>> wchar_t (c++ ANSI profile.datatypes)
<<dataType>> bool (c++ ANSI profile.datatypes)
<<dataType>> short int (c++ ANSI profile.datatypes)
<<dataType>> unsigned short int (c++ ANSI profile.datatypes)
<<dataType>> long int (c++ ANSI profile.datatypes)
<<dataType>> unsigned long int (c++ ANSI profile.datatypes)

Figure 47 -- C++ datatype

char

char types are represented as signed char and unsigned char according to the compiler implementation. char does not have a synonym.

signed char

signed char does not have a synonym.

unsigned char

unsigned char does not have a synonym.

int

Synonyms for int are

- signed
- signed int
- int signed

unsigned int

Synonyms for unsigned int are

- unsigned
- int unsigned

Copyright © 1998-2015 No Magic, Inc.

unsigned short int

Synonyms for unsigned short int are

- unsigned short
- short unsigned
- unsigned int short
- short unsigned int
- short int unsigned
- int short unsigned
- int unsigned short

unsigned long int

Synonyms for unsigned long int are

- unsigned long
- long unsigned
- unsigned int long
- long unsigned int
- long int unsigned
- int long unsigned
- int unsigned long

long int

Synonyms for long int are

- long
- int long
- signed long
- long signed
- signed long int
- signed int long
- long signed int
- long int signed
- int signed long
- int long signed

short int

Synonyms for short int are

- short
- int short
- signed short
- short signed
- signed short int

Copyright © 1998-2015 No Magic, Inc.

- signed int short
- short signed int
- short int signed
- int signed short
- int short signed

double

double does not have asynonym.

long double

long double does not have a synonym

float

float does not have a synonym.

void

void does not have a synonym.

bool

bool does not have a synonym.

wchar_t

wchar_t does not have asynonym.

Stereotypes

All C++ stereotypes are based on «C++Element» stereotypes.

Constraints described in this chapter are for information only, syntax of these constraints need to be checked with the future OCL interpreter.

«C++Class», «C++Operation», «C++Parameter», «C++Attribute», «C++LiteralValue», «C++Include», «C++Generalization», and «C++TemplateParameter» are invisible stereotypes and are used only to store C++ language properties. These stereotypes and their tag definitions are used by the DSL framework.

Copyright © 1998-2015 No Magic, Inc.

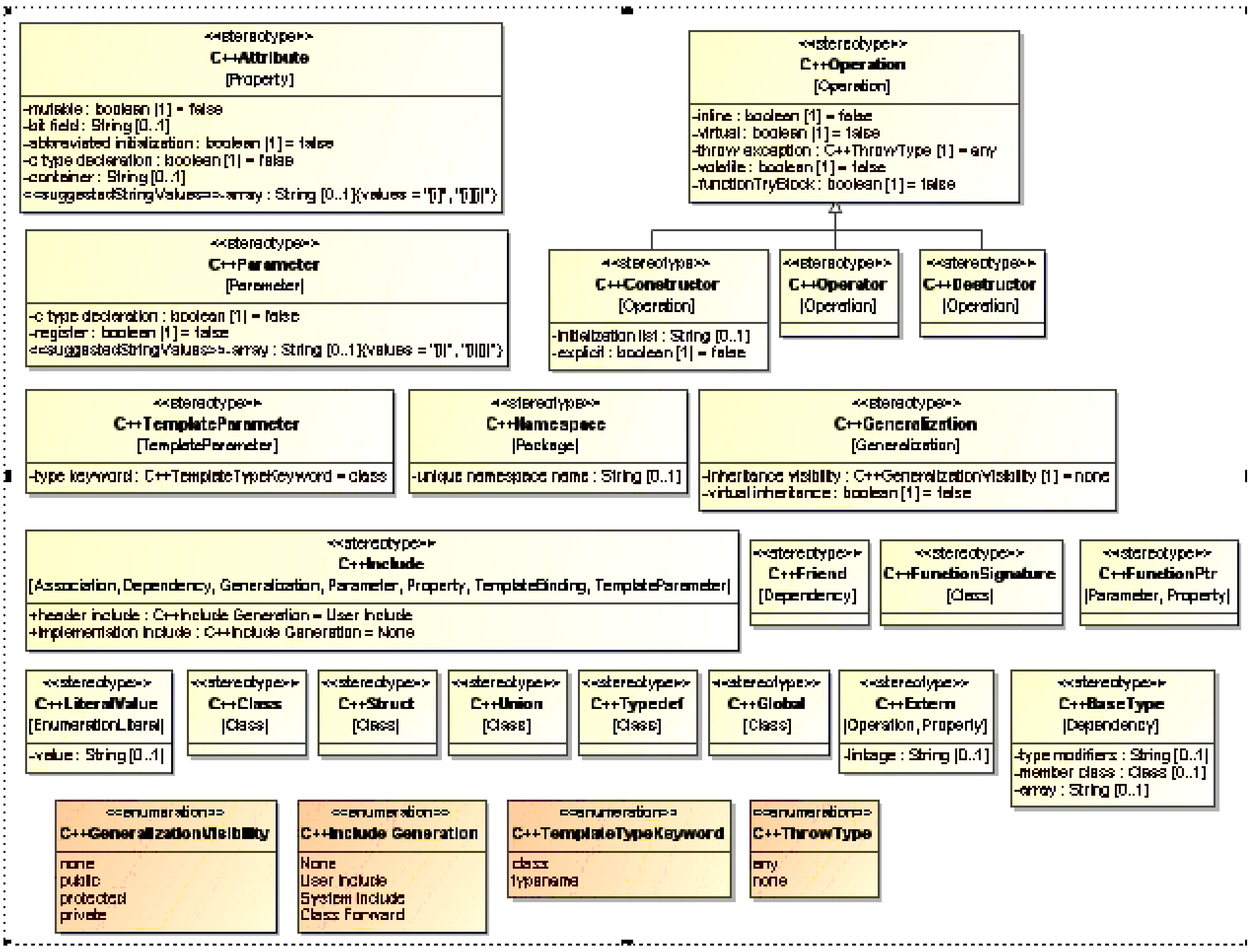

«stereotype»
C++Attribute
[Property]
-mutable : boolean [1] = false
-bit field : String [0..1]
-abbreviated initialization : boolean [1] = false
-c type declaration : boolean [1] = false
-container : String [0..1]
<<suggestedStringValues>>-array : String [0..1]{values = "[]", "[][]"}
«stereotype»
C++Operation
[Operation]
-inline : boolean [1] = false
-virtual : boolean [1] = false
-throw exception : C++ThrowType [1] = any
-volatile : boolean [1] = false
-functionTryBlock : boolean [1] = false
«stereotype»
C++Parameter
[Parameter]
-c type declaration : boolean [1] = false
-register : boolean [1] = false
<<suggestedStringValues>>-array : String [0..1]{values = "[]", "[][]"}
«stereotype»
C++Constructor
[Operation]
-initialization list : String [0..1]
-explicit : boolean [1] = false
«stereotype»
C++Operator
[Operation]
«stereotype»
C++Destructor
[Operation]
«stereotype»
C++TemplateParameter
[TemplateParameter]
-type keyword : C++TemplateTypeKeyword = class
«stereotype»
C++Namespace
[Package]
-unique namespace name : String [0..1]
«stereotype»
C++Generalization
[Generalization]
-inheritance visibility : C++GeneralizationVisibility [1] = none
-virtual inheritance : boolean [1] = false
«stereotype»
C++Include
[Association, Dependency, Generalization, Parameter, Property, TemplateBinding, TemplateParameter]
+header include : C++Include Generation = User Include
+implementation include : C++Include Generation = None
«stereotype»
C++Friend
[Dependency]
«stereotype»
C++FunctionSignature
[Class]
«stereotype»
C++FunctionPtr
[Parameter, Property]
«stereotype»
C++LiteralValue
[EnumerationLiteral]
-value : String [0..1]
«stereotype»
C++Class
[Class]
«stereotype»
C++Struct
[Class]
«stereotype»
C++Union
[Class]
«stereotype»
C++Typedef
[Class]
«stereotype»
C++Global
[Class]
«stereotype»
C++Extern
[Operation, Property]
-linkage : String [0..1]
«stereotype»
C++BaseType
[Dependency]
-type modifiers : String [0..1]
-member class : Class [0..1]
-array : String [0..1]
«enumeration»
C++GeneralizationVisibility
none
public
protected
private
«enumeration»
C++Include Generation
None
User Include
System Include
Class Forward
«enumeration»
C++TemplateTypeKeyword
class
typename
«enumeration»
C++ThrowType
any
none

Figure 48 -- C++ Stereotype tag definitions

Copyright © 1998-2015 No Magic, Inc.

C++Operation

«C++Operation» is an invisible stereotype used to include language properties for any C++ operation.

Name	Meta class	Constraints

C++Operation	Operation	Const function
		`void f() const;`
		Constraint: Only valid for member function
		`if isQuery then`
		`stereotype->select(name='C++Global')->isEmpty()`

Tag	**Type**	**Description**
inline	boolean[1]=false	Inline function `inline a();`

throw exception	C++ThrowType[1] =any	Exception specification.
		operation.raisedExpression is not empty, the throw expression is generated.
		`void f() throw(int);`
		If operation.raisedExpression is empty and throw expression is none, a throw expression without argument is generated.
		`void f() throw ()`
		If operation.raisedExpression is empty and throw expression is any, does not generate a throw keyword.
		`void f();`

virtual	boolean[1]=false	Virtual function
		`virtual a();`
		Constraint: Only valid for member function and non static
		`stereotype->select(name='C++Global')`
		`->isEmpty() and IsStatic = false`

volatile	boolean[1]=false	Volatile function `void f() volatile;`
funtionTryBlock	boolean[1]=false	Function try block void f() try{}

C++Operator

«C++Operator» stereotype is used to define a C++ operator function. This stereotype extends the «C++Operation» stereotype. See **Function operator** for more information.

Name	Meta class	Constraints
C++Operator	Operation	operator function `T& operator+(T& a);` Constraint: name start with “operator”

Copyright © 1998-2015 No Magic, Inc.

C++Parameter

«C++Parameter» is an invisible stereotype used to include language properties for any C++ function parameter.

name	Meta class	Constraints
C++Parameter	Parameter	
Tag	Type	Description

c type declaration	boolean[1]=false	Declare parameter's type in C style
		C style:
		`void a(enum Day x);`
		C++ style:
		`void a(Day x);`

register	boolean[1]=false	Register parameter `void a(register int x);`
array	String[0..1]	C++ array definition `void a(int x[2][2]);`

C++Attribute

«C++Attribute» is an invisible stereotype used to include language properties for any C++ variable.

name	Meta class	Constraints

C++Attribute	Property	Constraint for code generation. It is valid to have a default value for any kind of attribute, but it is illegal to initialize a member variable within its definition. (Eg. `class A { int x = 1; };`)
		`if defaultValue.size() > 0 then`
		`owner.stereotype->exists(name='C++Global') or (isStatic = true and typeModifiers.contains("const"))`

Tag	**Type**	**Description**

abbreviated initialization	boolean[1]=false	Initialize the attribute with the abbreviate form.
		`int x(5);`
		Constraint
		`owner.stereotype->exists(name='C++Global')`

bit field	String[0..1]	Bit field declaration
		`int x:2;`
		Constraint: Only valid for member function
		`stereotype->select(name='C++Global')`
		`->isEmpty()`

c type declaration	boolean[1]=false	Declare attribute's type in C style
		C style:
		`enum Day x;`
		C++ style:
		`Day x;`

container	String[0..1]	container of the attribute. $ character is replaced by the attribute type. `vector<$> x;`

Copyright © 1998-2015 No Magic, Inc.

mutable	boolean[1]=false	Attribute mutable modifier. `mutable int x;` Constraint: Only valid for member function `stereotype->select(name='C++Global') ->isEmpty()`
array	String[0..1]	C++ array definition `int x[2][2];`

C++LiteralValue

«C++LiteralValue» is an invisible stereotype used to include language properties for any C++ enum field. See *Enumeration* for more info.

name	**Meta class**	**Constraints**
C++LiteralValue	EnumerationLiteral	
Tag	**Type**	**Description**
value	String[0..1]	Value definition of an enum field. (A valid C++ expression) `enum Day {Mon = 2};`

C++Friend

«C++Friend» stereotype is used to define C++ friend relationship. See *Friend declaration* for more info.

Name	**Meta class**	**Constraints**
C++Friend	Dependency	Client is Class or Operation and supplier is Class `(client.oclIsTypeOf(Class) or client.oclIsTypeOf(Operation)) and supplier.oclIsTypeOf(Class)`

C++Struct

«C++Struct» stereotype is used to define C++ struct. See *Struct* for more info.

name	**Meta class**	**Constraints**
C++Struct	Class	

C++Typedef

«C++Typedef» stereotype is used to define C++ typedef. See *Typedef* for more info.

name	**Meta class**	**Constraints**
C++Typedef	Class	A typedef does not contain operation and attribute `feature->isEmpty()` A «C++BaseType» dependency is defined

Copyright © 1998-2015 No Magic, Inc.

C++Union

«C++Union» stereotype is used to define C++ union. See *Union* for more info.

name	Meta class	Constraints
C++Union	Class	

C++Global

«C++Global» stereotype is used to define global functions and variables. (functions and variables outside a class/struct/union declaration). See *"Global functions and variables", on page 88* for more info.

name	Meta class	Constraints

C++Global	Class	Only 1 «C++Global» class into a package
		`owner.ownedElement->select(`
		`    stereotype->select(name='C++Global')).size()=1`
		All operations and attributes are public
		`feature->forAll(visibility = #public)`

C++Namespace

«C++Namespace» stereotype is used to define C++ namespace. See *Namespace* for more info.

name	Meta class	Constraints
C++Namespace	Package	
Tag	**Type**	**Description**
unique namespace name	String[0..1]	Unnamed namespace `namespace {}`

C++Constructor

«C++Constructor» stereotype is used to define C++ constructor. This stereotype extends «C++Operation» stereotype. See *Class Constructor/Destructor* for more info.

Name	Meta class	Constraints
C++Constructor	Operation	name = owner.name
Tag	**Type**	**Description**
explicit	boolean[1]=false	Explicit constructor `explicit a();`
initialization list	String[0..1]	Constructor initialization `a() : x(1) {}`

C++Destructor

«C++Destructor» stereotype is used to define C++ destructor. This stereotype extends «C++Operation» stereotype. See *Class Constructor/Destructor* for more info.

Name	Meta class	Constraints

C++Destructor	Operation	name = “~”+owner.name

C++Extern

«C++Extern» stereotype is used to define C++ extern variable. See *Variable extern* for more info.

Name	Meta class	Constraints
C++Extern	Operation, Property	owner.stereotype->exists(name=’C++Global’)
Tag	**Type**	**Description**
linkage	String[0..1]	Linkage specification `extern “C”`

C++FunctionPtr

«C++FunctionPtr» stereotype is used to define C++ function pointer. See *Function pointer* for more info.

Name	Meta class	Constraints
C++FunctionPtr	Parameter, Property	
Tag	**Type**	**Description**
signature	Operation	The signature of the function (C++ function pointer definition without the operation name)
member class	Class	The class used for pointer to member function.

C++FunctionSignature

«C++FunctionSignature» stereotype is used as a container to model C++ function pointer. See *Function pointer* for more info.

Name	Meta class	Constraints
C++FunctionSignature	Class	The class cannot have properties. `properties->isEmpty()`

C++Class

«C++Class» stereotype is an invisible stereotype used to include language properties for any C++ variable.

Name	Meta class	Constraints
C++Class	Class	

C++BaseType

«C++BaseType» stereotype is used to link base type of a typedef.

Name	Meta class	Constraints
C++BaseType	Dependency	Client is type of Class with «C++Typedef» stereotype.
Tag	**Type**	**Description**

Copyright © 1998-2015 No Magic, Inc.

type modifiers	String[0..1]	Type modifiers of the typedef or function pointer.
member class	Class[0..1]	Memberclass of typedef or function pointer.
array	String[0..1]	Array definition.

C++Include

«C++Include» stereotype is used to keep the information about include type used for generating include and forward class declaration.

Name	**Meta class**	**Constraints**
C++Include	Association, Dependency, Generalization, Parameter, Property, TemplateBinding, TemplateParameter	Client is type of Component
Tag	**Type**	**Description**
header include	String	The value of tag is one of the following • None • User Include • System Include • Class Forward
implementation include	String	The value of tag is one of the following • None • User Include • System Include • Class Forward

The *header include* tag is used when the client component has header file extension, “*.h”, otherwise the *implementation include* tag will be used.

The *User Include* tag value is used for generating user include, such as #include “test.h”.

The *System Include* tag value is used for generating system include, such as #include <string.h>

The *Class Forward* tag value is used for generating forward class declaration.

C++TemplateParameter

«C++Template Parameter» is used to keep type keyword between class and typename for template parameter declaration.

Name	**Meta class**	**Constraints**
C++TemplateParameter	TemplateParameter	
Tag	**Type**	**Description**
type keyword	C++TemplateTypeKeyword=class	The value of tag is one of the following • class • typename

Copyright © 1998-2015 No Magic, Inc.

C++Generalization

«C++Generalization» is used for information related to generalization and interface realization.

Name	Meta class	Constraints
C++Generalization	Generalization, InterfaceRealization	
Tag	**Type**	**Description**
inheritance visibility	C++GeneralizationVisibility[1]=none	The value of tag is one of the following • None • public • protected • private Example: class A : private B {};
virtual inheritance	boolean[1]=false	Virtual inheritance Example: class A : virtual B{};

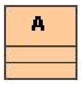
Mapping

This chapter describes the mapping between C++ and UML.

Class

C++ class map to a UML class

Code	MD-UML

`class A {`
`};`
	

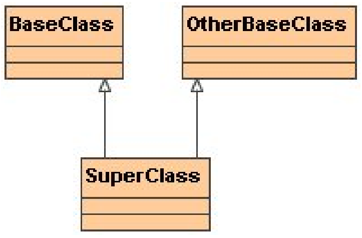
Base Class Definition

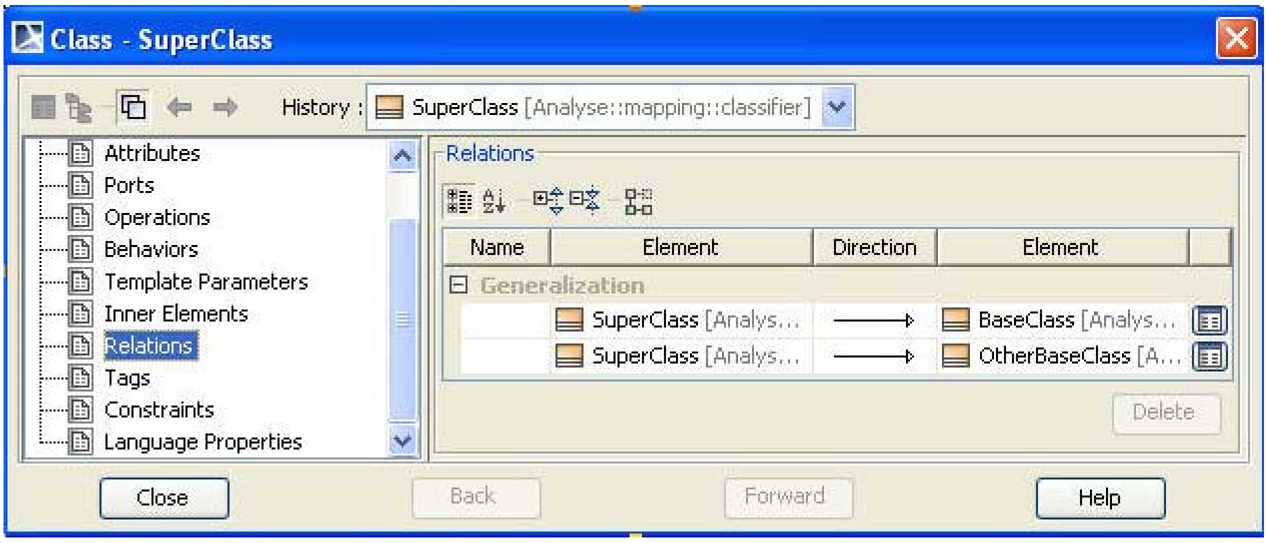
Base class definition is mapped to UML generalization, a generalization is created between the base class and the super class.

Access visibility (public, protected and private) and virtual properties of the base class are mapped to C++ language properties of the UML generalization.

Copyright © 1998-2015 No Magic, Inc.

Code	MD-UML

```
class BaseClass {};
class OtherBaseClass {};
class SuperClass :
     public BaseClass,
     protected virtual OtherBaseClass {
};
```
	



Class - SuperClass

History : SuperClass [Analyse::mapping::classifier]

Attributes
Ports
Operations
Behaviors
Template Parameters
Inner Elements
Relations
Tags
Constraints
Language Properties

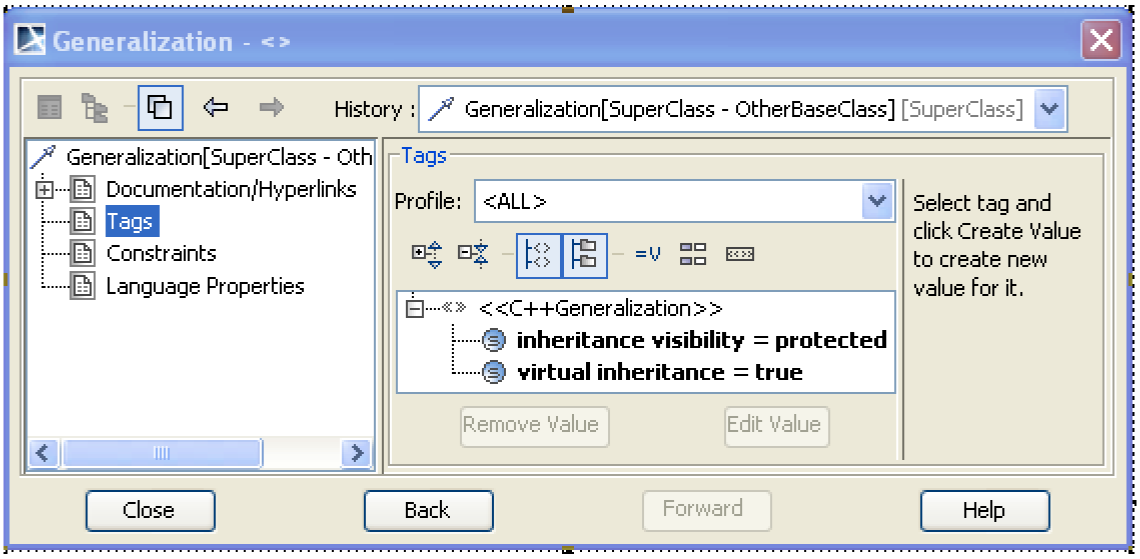
Relations

Name	Element	Direction	Element
Generalization			
	SuperClass [Analys...	⟶	BaseClass [Analys...
	SuperClass [Analys...	⟶	OtherBaseClass [A...

Delete

Close Back Forward Help

Figure 49 -- SuperClass Generalization relationship



Generalization - <>

History : Generalization[SuperClass - OtherBaseClass] [SuperClass]

Generalization[SuperClass - Oth
Documentation/Hyperlinks
Tags
Constraints
Language Properties

Tags

Profile: <ALL>

Select tag and click Create Value to create new value for it.

<<C++Generalization>>
inheritance visibility = protected
virtual inheritance = true

Remove Value Edit Value

Close Back Forward Help

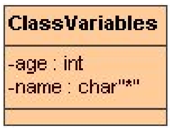
Figure 50 -- Generalization language properties

Class Member Variable

Class member variables are mapped to UML attributes. See *Variable* for more info.

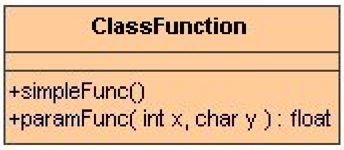
Code	MD-UML

Copyright © 1998-2015 No Magic, Inc.

`class ClassVariable {` `int age;` `char* name;` `};`	

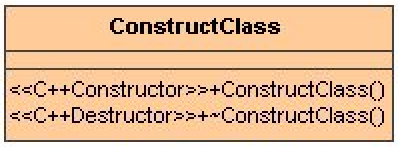
Class Member Function

Class member functions are mapped to UML operations. See *Function* for more info.

Code	MD-UML
`class ClassFunction {` `public:` `void simpleFunc();` `float paramFunc(int x,char y);` `};`	 ClassFunction +simpleFunc() +paramFunc(int x, char y) : float

Class Constructor/Destructor

C++ class constructor and destructor are mapped to UML operation with «C++Constructor» stereotype and «C++Destructor» stereotype. See *C++Constructor* and *C++Destructor* for more info.

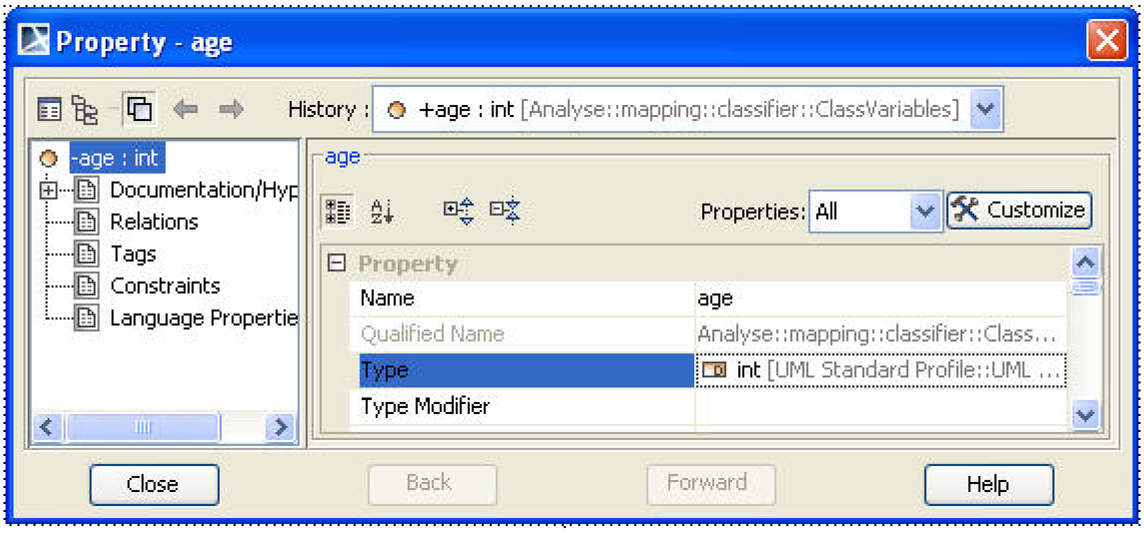
Code	MD-UML
`class ConstructClass {` `public:` `ConstructClass();` `~ConstructClass();` `}`	 ConstructClass <<C++Constructor>>+ConstructClass() <<C++Destructor>>+~ConstructClass()

Copyright © 1998-2015 No Magic, Inc.

Variable

C++ variable is mapped to UML attribute, the variable type is mapped to the attribute type.

Code	MD-UML
`int age;`	



Property - age

History : +age : int [Analyse::mapping::classifier::ClassVariables]

Properties: All

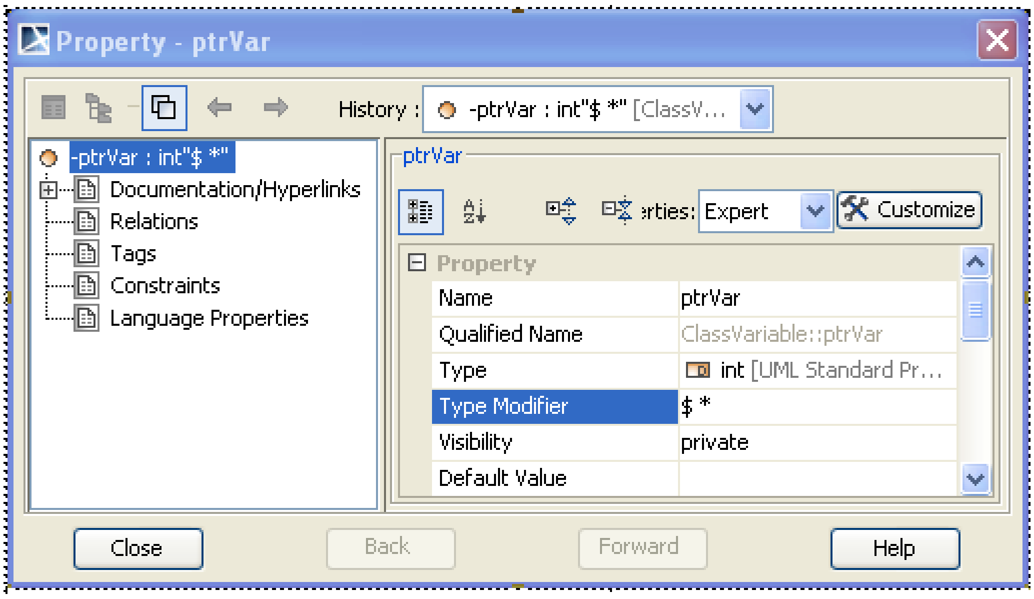
Property	
Name	age
Qualified Name	Analyse::mapping::classifier::Class...
Type	int [UML Standard Profile::UML ...
Type Modifier	

Figure 51 -- Variable property age

C++ type pointer/reference is mapped to Type Modifier property of the attribute. $ character is replaced by the type name.

Code	MD-UML

`int* ptrVar;`
`int& refVar`	



Property - ptrVar

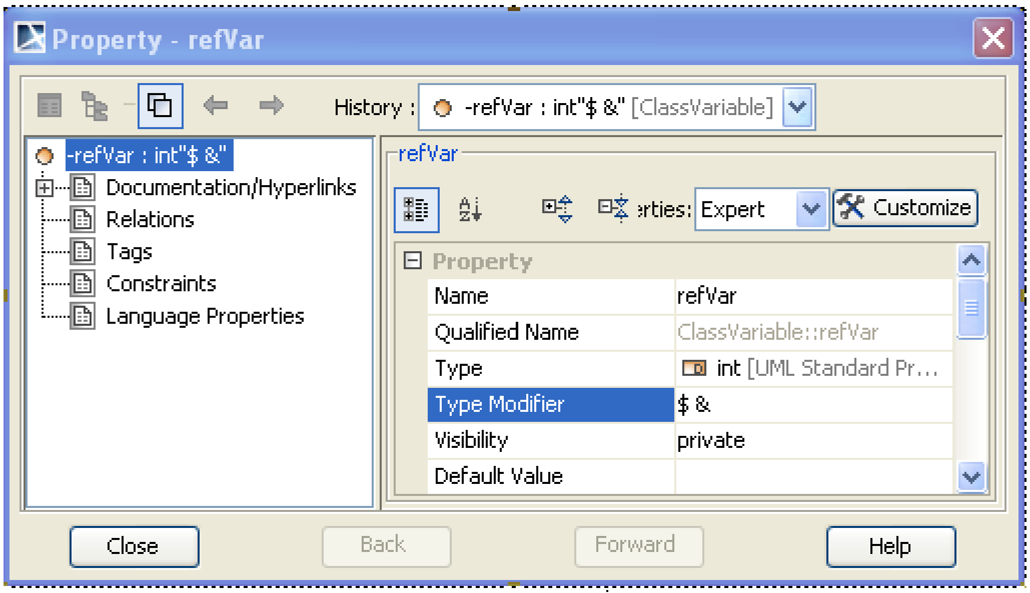
History : -ptrVar : int"$ *" [ClassV...

Properties: Expert

Property	
Name	ptrVar
Qualified Name	ClassVariable::ptrVar
Type	int [UML Standard Pr...
Type Modifier	$ *
Visibility	private
Default Value	

Figure 52 -- Variable property ptrVar

Copyright © 1998-2015 No Magic, Inc.

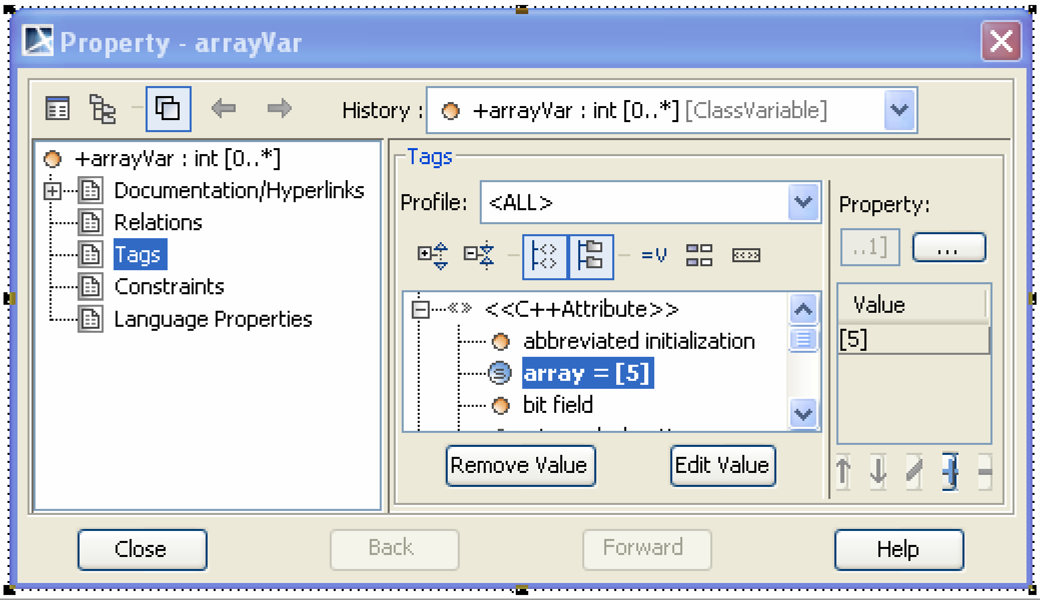


Property	
Name	refVar
Qualified Name	ClassVariable::refVar
Type	int [UML Standard Pr...
Type Modifier	$ &
Visibility	private
Default Value	

Figure 53 -- Variable property refVar

C++ array type is mapped to *array* tag value of the attribute. If *array* is set, then multiplicity property of UML attribute is set to “[0..*]”

Code	MD-UML
`int arrayVar[5];`	



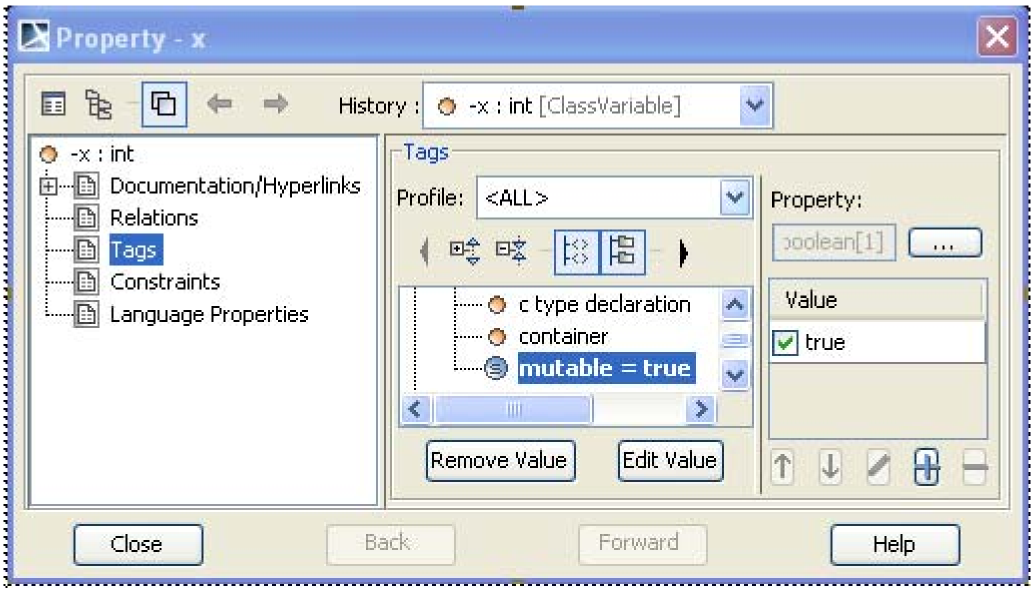
Variable modifiers

Mutable variable modifiers are mapped to UML attribute’s language properties *Mutable*.

Constraint: only member variable can be mutable (global variable cannot).

Code	MD-UML
`mutable int x;`	

Copyright © 1998-2015 No Magic, Inc.



Property - x

History : -x : int [ClassVariable]

- -x : int
  - Documentation/Hyperlinks
  - Relations
  - Tags
  - Constraints
  - Language Properties

Tags

Profile: <ALL>

- c type declaration
- container
- mutable = true

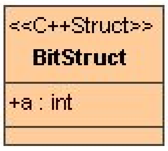
Property: boolean[1]

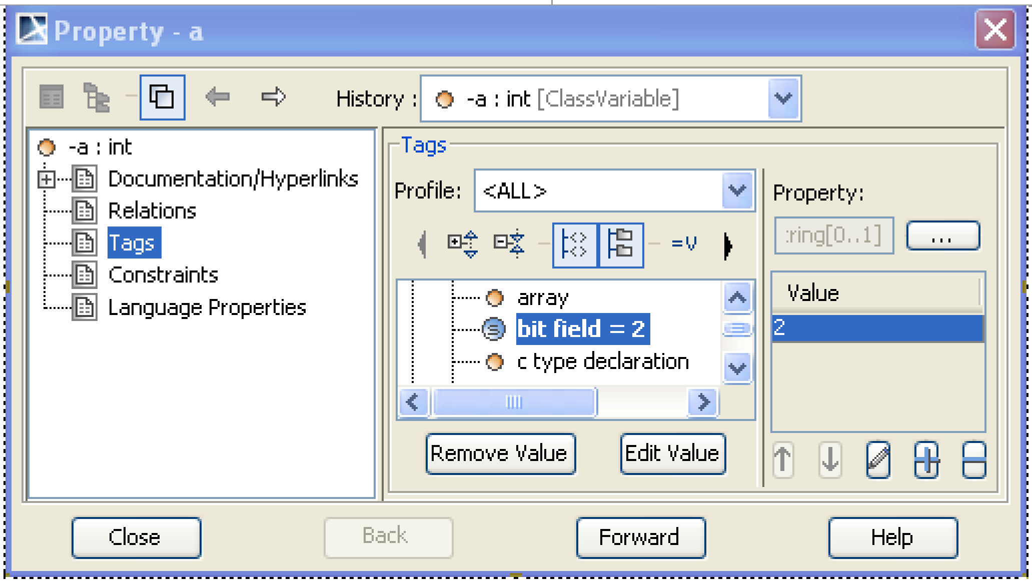
Value

✓ true

Bit field is mapped to *Bit field* tag value.

Code	MD-UML

`struct BitStruct {`
`      int a:2;`
`};`
	



Property - a

History : -a : int [ClassVariable]

- -a : int
  - Documentation/Hyperlinks
  - Relations
  - Tags
  - Constraints
  - Language Properties

Tags

Profile: <ALL>

- array
- bit field = 2
- c type declaration

Property: String[0..1]

Value

2

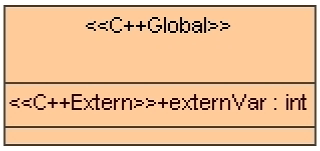
Variable extern

C++ extern variable is mapped to «C++Extern» stereotype. Linkage tag value is used to specify the kind of link-age “C” or “C++”, if linkage is empty (or without value) extern without linkage is generated.

See C++Extern for more info.

Code	MD-UML

Copyright © 1998-2015 No Magic, Inc.

`extern int externVar;`	

Variable default value

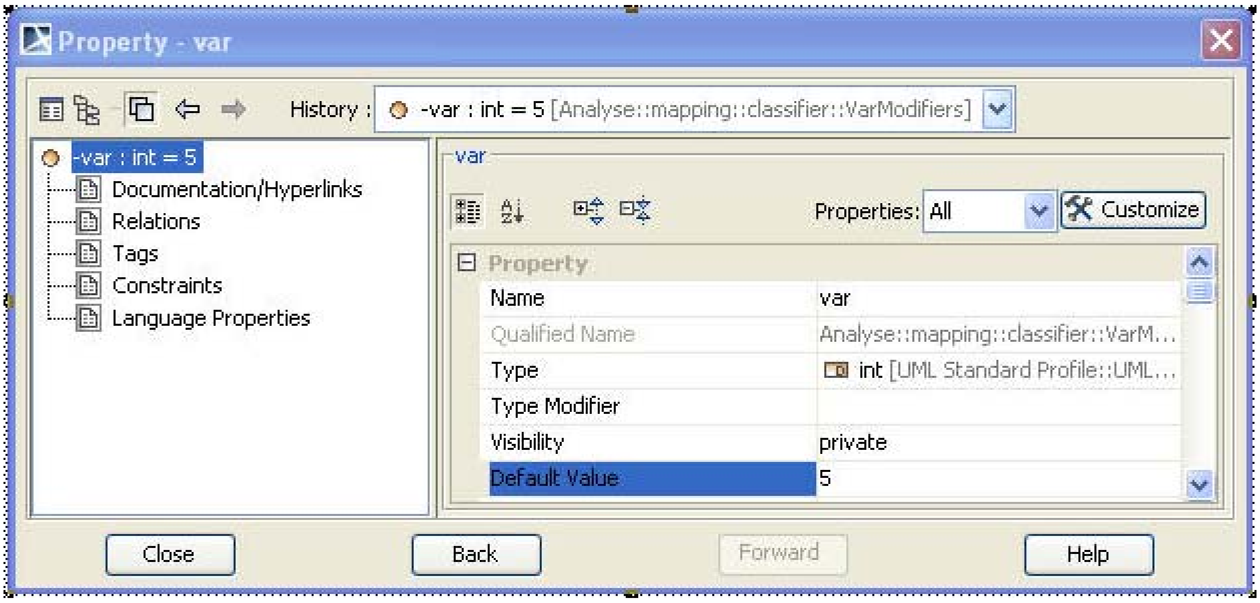
Variable initial value is mapped to UML attribute’s default value.

Variable initial value set using function style method is mapped to UML attribute’s default value and attribute’s language property *Abbreviated Initialization* set to true.

Constraint: Only “static const” member variables can be initialized, and they cannot be initialized using function style method.

Code	MD-UML

`int var = 5;`
`int var2(10);`	



Property - var

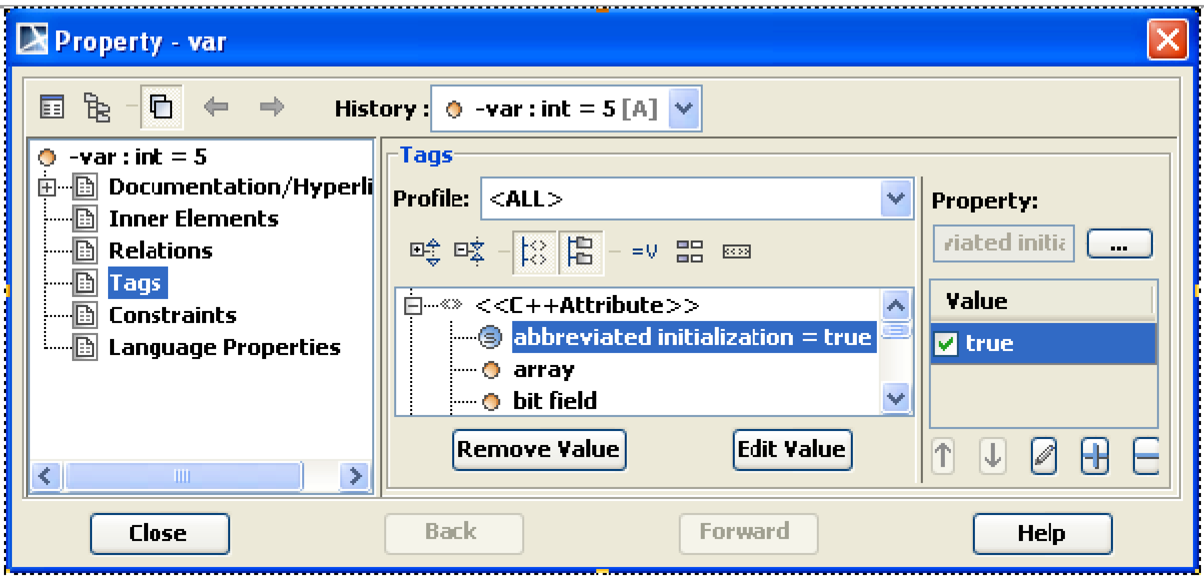
History : -var : int = 5 [Analyse::mapping::classifier::VarModifiers]

-var : int = 5
- Documentation/Hyperlinks
- Relations
- Tags
- Constraints
- Language Properties

var — Properties: All — Customize

Property	
Name	var
Qualified Name	Analyse::mapping::classifier::VarM...
Type	int [UML Standard Profile::UML...
Type Modifier	
Visibility	private
Default Value	5

Close — Back — Forward — Help



Property - var

History : -var : int = 5 [A]

-var : int = 5
- Documentation/Hyperli
- Inner Elements
- Relations
- Tags
- Constraints
- Language Properties

Tags — Profile: <ALL>

<<C++Attribute>>
- abbreviated initialization = true
- array
- bit field

Property: viated initia

Value: ✓ true

Remove Value — Edit Value

Close — Back — Forward — Help

Copyright © 1998-2015 No Magic, Inc.

Const Volatile qualified type

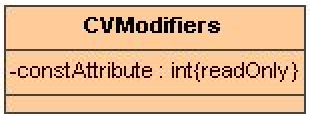
C++ const and volatile modifiers for attribute/function parameter are mapped to *Type Modifiers* properties.

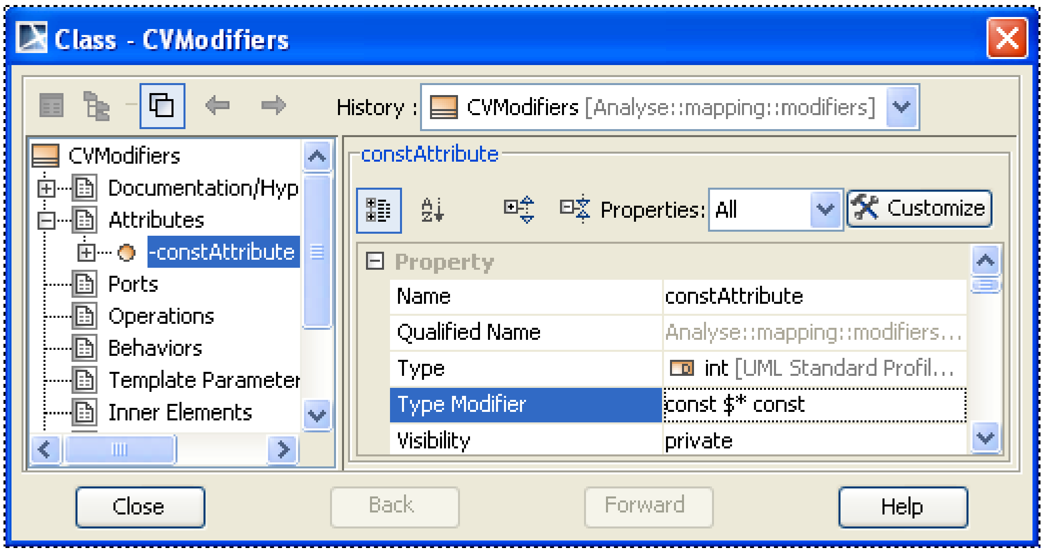
For const attribute, the property *Is Read Only* is set to true during reverse.

The character $ into Type Modifier value is replaced by the type name.

Constraint : If the property *Is Read Only* is set and *Type Modifiers* is not set to const or const volatile (set to const, or an error message will display during syntax check)

Code	**MD-UML**

`class CVModifiers {`
`     const int* const constAttribute;`
`}`
	



Property	
Name	constAttribute
Qualified Name	Analyse::mapping::modifiers...
Type	int [UML Standard Profil...
Type Modifier	const $* const
Visibility	private

Function

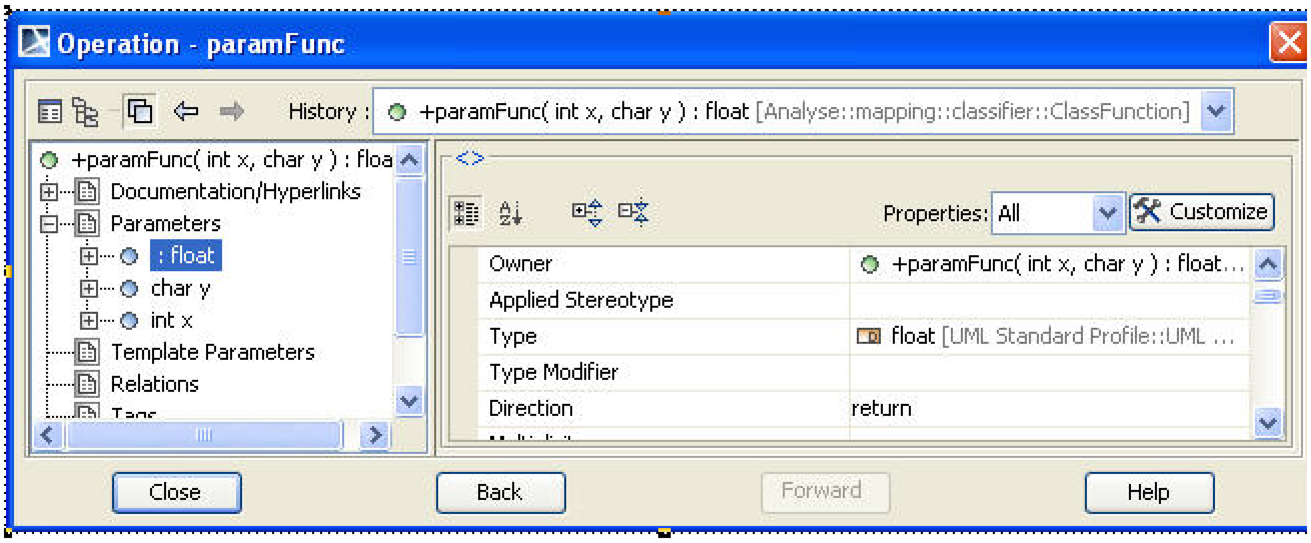
C++ function is mapped to UML operation, parameter of function is mapped to UML parameter with property *direction* set to “inout”, return type of function is mapped to UML parameter with property *direction* set to “return”. Type of parameter is mapped to type of UML parameter.

C++ default parameter value is mapped to *defaultValue* property of UML parameter.

Pointer, reference and array type of parameter are mapped to property *Type Modifier* of parameter. See *Variable modifiers* for more info.

Code	**MD-UML**
`float paramFunc(int x, char x);`	

Copyright © 1998-2015 No Magic, Inc.


Operation - paramFunc

History : +paramFunc(int x, char y) : float [Analyse::mapping::classifier::ClassFunction]

+paramFunc(int x, char y) : floa
Documentation/Hyperlinks
Parameters
: float
char y
int x
Template Parameters
Relations
Tags

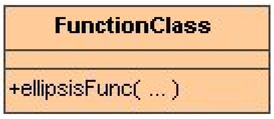
Properties: All

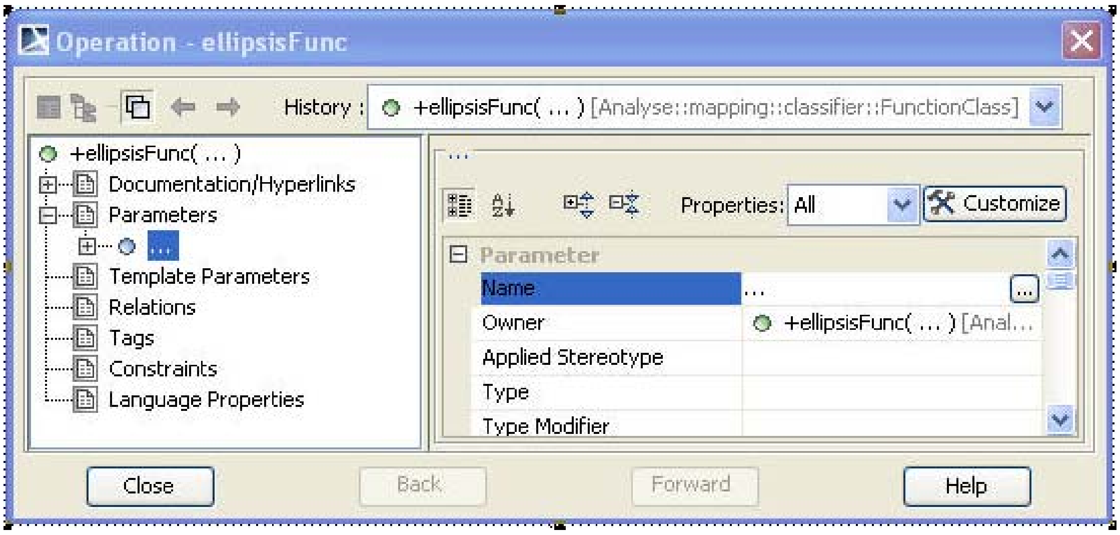
Owner	+paramFunc(int x, char y) : float...
Applied Stereotype	
Type	float [UML Standard Profile::UML ...
Type Modifier	
Direction	return

Function variable-length parameter list

C++ function variable-length parameter list is mapped to a UML parameter with name "..." (dot 3 times) and without type.

Code	MD-UML

```
Class FunctionClass {
public:
    EllipsisFunc(...);
};
```
	


Operation - ellipsisFunc

History : +ellipsisFunc(...) [Analyse::mapping::classifier::FunctionClass]

+ellipsisFunc(...)
Documentation/Hyperlinks
Parameters
...
Template Parameters
Relations
Tags
Constraints
Language Properties

Properties: All

Parameter

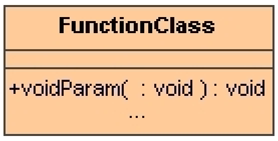
Name	...
Owner	+ellipsisFunc(...) [Anal...
Applied Stereotype	
Type	
Type Modifier	

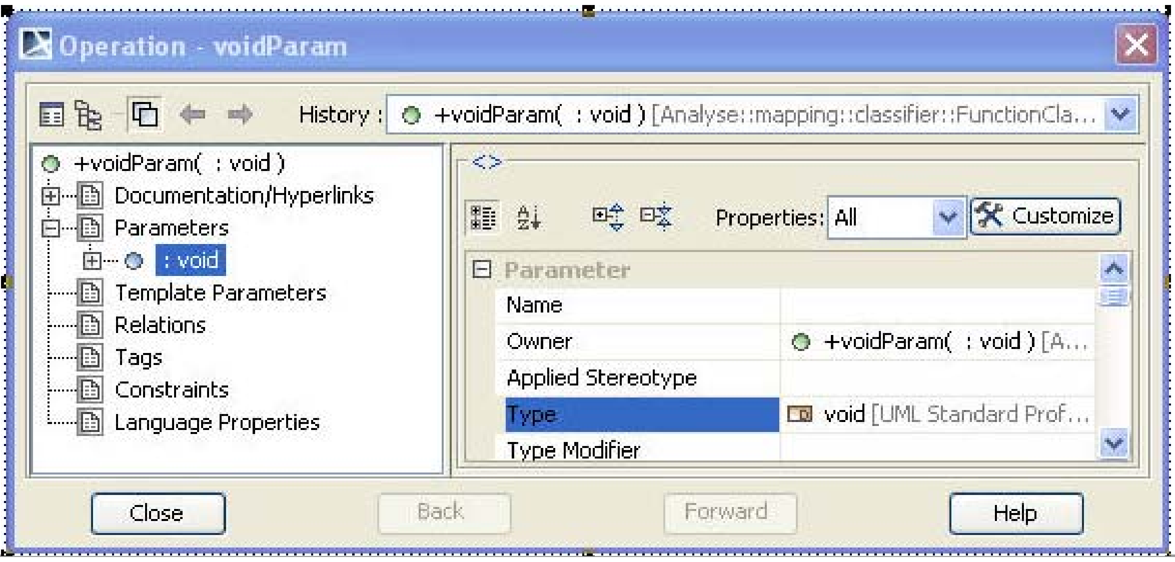
void parameter

C++ void function parameter is mapped to a UML parameter without name and with type "void".

Code	MD-UML

Copyright © 1998-2015 No Magic, Inc.

`Class FunctionClass {`
`public:`
`     void voidParam(void);`
`};`
	



Operation - voidParam

History : +voidParam(: void) [Analyse::mapping::classifier::FunctionCla...

- +voidParam(: void)
  - Documentation/Hyperlinks
  - Parameters
    - : void
  - Template Parameters
  - Relations
  - Tags
  - Constraints
  - Language Properties

Properties: All

Parameter	
Name	
Owner	+voidParam(: void) [A...
Applied Stereotype	
Type	void [UML Standard Prof...
Type Modifier	

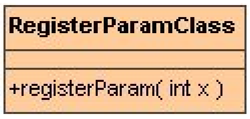
Close | Back | Forward | Help

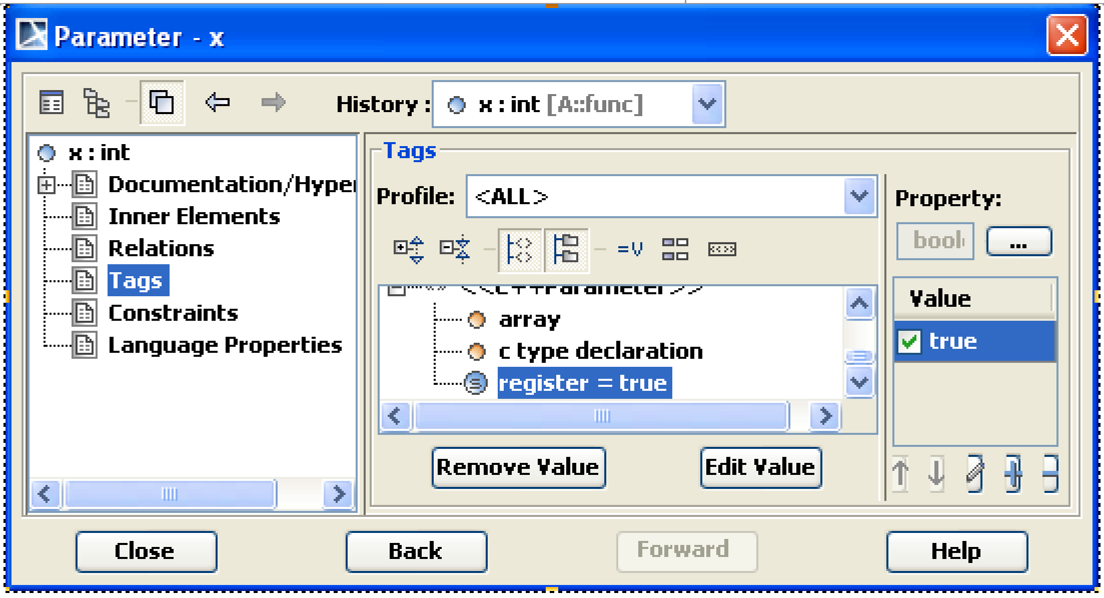
Register parameter

C++ register parameter is mapped to UML parameter language property Register

Depending on the compiler, register can be limited on some types (int, char).

Code	MD-UML

`class RegisterParamClass {`
`     void registerParam(register int x);`
`};`
	



Parameter - x

History : x : int [A::func]

- x : int
  - Documentation/Hyper
  - Inner Elements
  - Relations
  - Tags
  - Constraints
  - Language Properties

Tags

Profile: <ALL>

- <<C++Parameter>>
  - array
  - c type declaration
  - register = true

Property: bool

Value
✓ true

Remove Value | Edit Value

Close | Back | Forward | Help

Copyright © 1998-2015 No Magic, Inc.

Function modifiers

C++ function modifiers are mapped to Language properties of Operation.

Virtual function is mapped to *Virtual modifier* property.

Inline function is mapped to *Inline modifier* property.

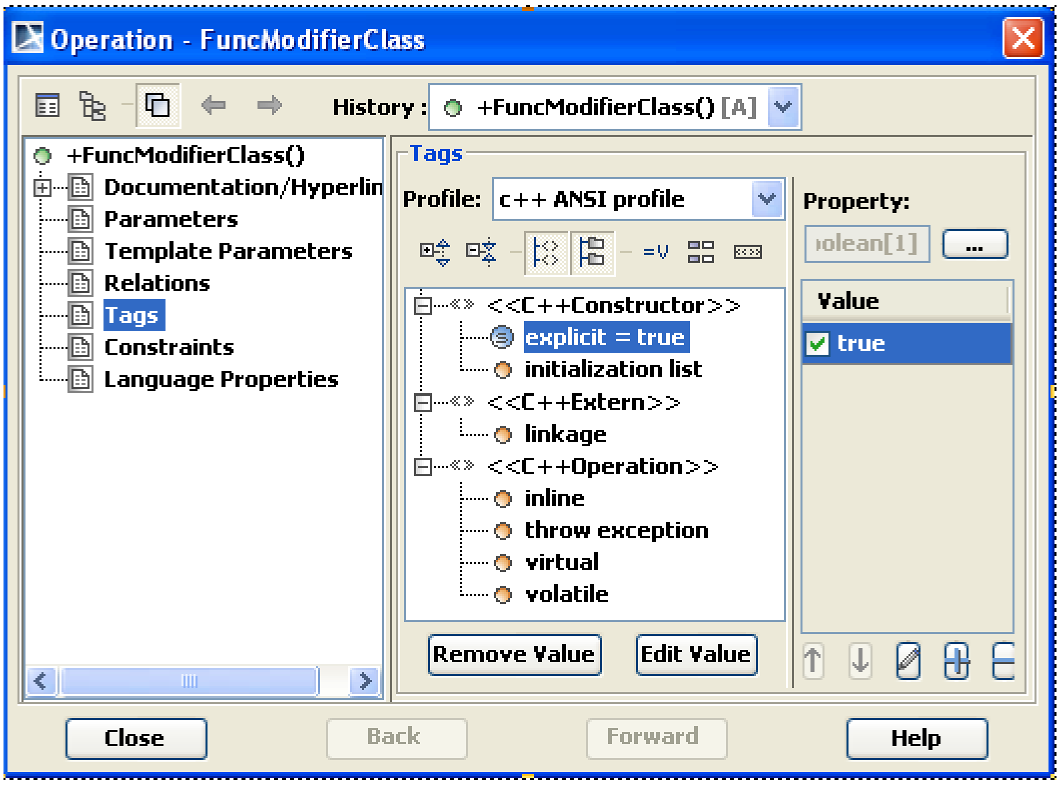
Explicit function is mapped to *Explicit modifier* property. Constraint: explicit is only valid for constructor.

Const function is mapped to UML operation *Is Query* property.

Volatile function is mapped to Tag value *volatile*.

Code	**MD-UML**

`class FuncModifierClass {`
`    explicit FuncModifierClass();`
 	
`};`	


Operation - FuncModifierClass

History : +FuncModifierClass() [A]

+FuncModifierClass()
- Documentation/Hyperlin
- Parameters
- Template Parameters
- Relations
- Tags
- Constraints
- Language Properties

Tags

Profile: c++ ANSI profile

Property:
olean[1]

```
<<C++Constructor>>
    explicit = true
    initialization list
<<C++Extern>>
    linkage
<<C++Operation>>
    inline
    throw exception
    virtual
    volatile
```

Value
✓ true

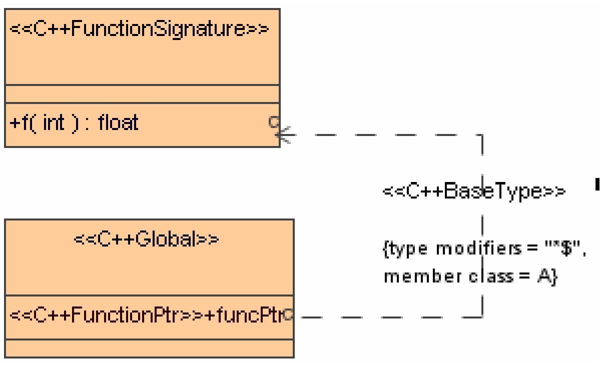
Function pointer

C++ function pointer type is mapped to attribute/parameter with «C++FunctionPtr» stereotype, a dependency with «C++BaseType» stereotype link from the attribute/parameter to the operation in a «C++FunctionSignature» class, and type modifiers of the dependency is set to *$.

Member function pointer use the same mapping, and member class tag of «C++BaseType» stereotype point to a class.

Code	**MD-UML**

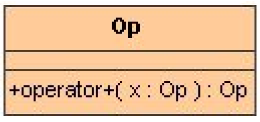
Copyright © 1998-2015 No Magic, Inc.

`float (A*funcPtr)(int);`	 <<C++FunctionSignature>> +f(int) : float <<C++Global>> <<C++FunctionPtr>>+funcPtr <<C++BaseType>> {type modifiers = ""$", member class = A}

Function operator

C++ function operator is mapped to normal function with the C++ operator name mapped to UML operation name. See *C++Operator* for more info.

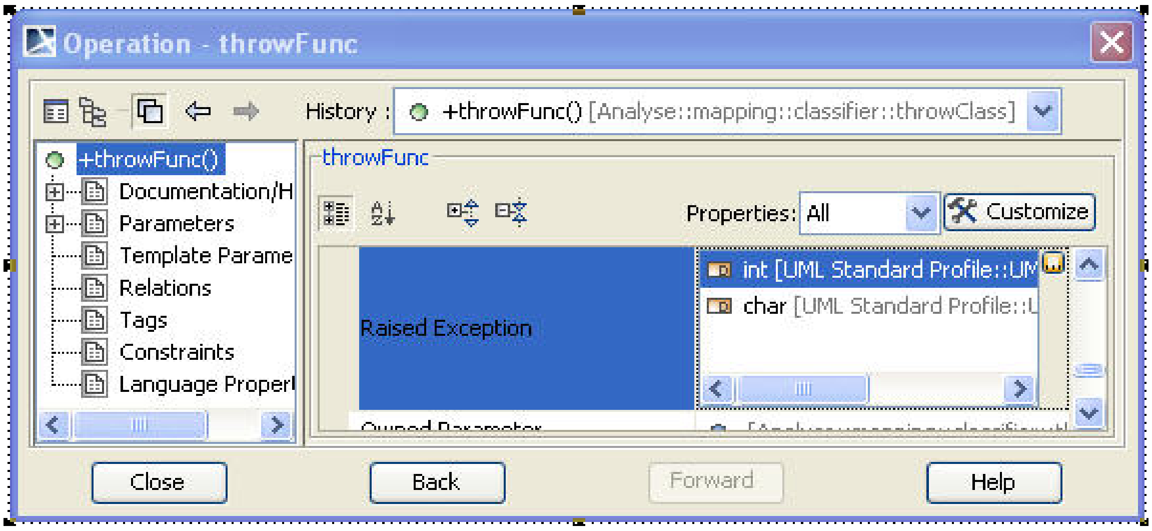
Code	MD-UML

`Class Op {`
`Public:`
`     Op operator+(Op x);`
`};`
	

Exception

C++ exception is mapped to UML operation's *raised exception* properties. If *raisedExpression* is empty, and *throw exception* tag is set to *none* a throw without parameter is generated. If *raisedExpression* is empty, and *throw exception* tag is set to *any* throw keyword is not generated. If the tag *throw exception* is not set, then generate specific *raisedExpression*, or do not generate throw if *raisedExpression* is empty.

Code	MD-UML
`void throwFunc() throw (int,char);`	



Operation - throwFunc

History : +throwFunc() [Analyse::mapping::classifier::throwClass]

Raised Exception: int [UML Standard Profile::UM; char [UML Standard Profile::U

Copyright © 1998-2015 No Magic, Inc.

Visibility

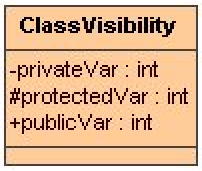
Variables and function visibility are mapped using the UML *visibility* property.

Members of C++ class without access visibility specified are private.

Members of C++ struct or union without access visibility specified are public.

Variables and functions outside a class/struct/union are public.

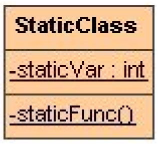
Code	**MD-UML**

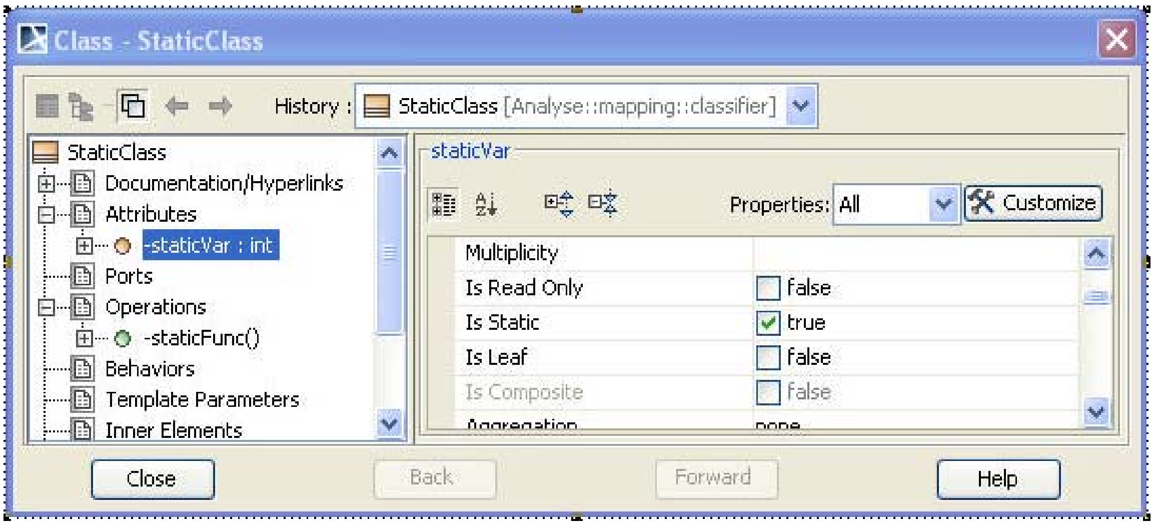
`class ClassVisibility {`
`     int privateVar;`
`protected:`
`     int protectedVar;`
`public:`
`     int publicVar;`
`};`
	 ClassVisibility
	-privateVar : int
	#protectedVar : int
	+publicVar : int

Static members

Static variables and functions are mapped to UML *Is Static* property.

Code	**MD-UML**

`class StaticClass {`
`     static int staticVar;`
`     static void staticFunc();`
`};`
	



Properties: All

Multiplicity	
Is Read Only	☐ false
Is Static	✓ true
Is Leaf	☐ false
Is Composite	☐ false
Aggregation	none

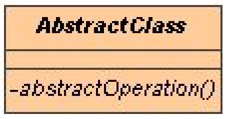
Pure virtual function and abstract class

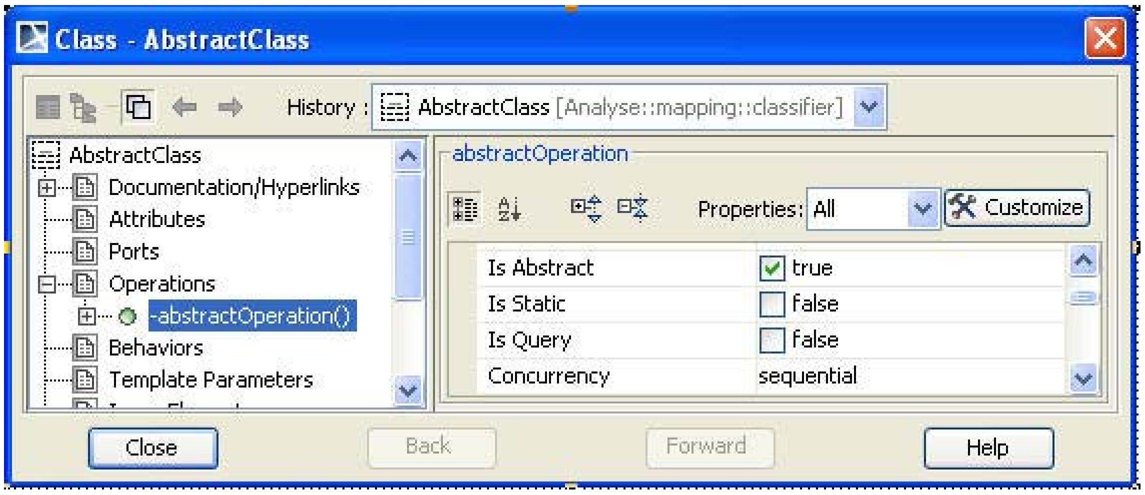
Pure virtual C++ function is mapped to UML operation with property *Is Abstract* set to true. If one or more functions are abstract in a class, the property *Is Abstract* of the UML class is set to true.

Copyright © 1998-2015 No Magic, Inc.

Constraint: if no operation is abstract, the class cannot be abstract.

Code	MD-UML

```
class AbstractClass {
    virtual abstractOperation()=0;
};
```
	


Class - AbstractClass

History : AbstractClass [Analyse::mapping::classifier]

AbstractClass
Documentation/Hyperlinks
Attributes
Ports
Operations
-abstractOperation()
Behaviors
Template Parameters

abstractOperation

Properties: All   Customize

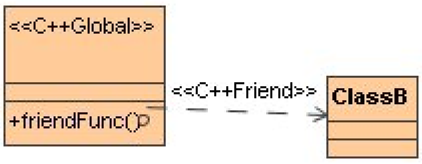
Is Abstract	✓ true
Is Static	false
Is Query	false
Concurrency	sequential

Close   Back   Forward   Help

Friend declaration

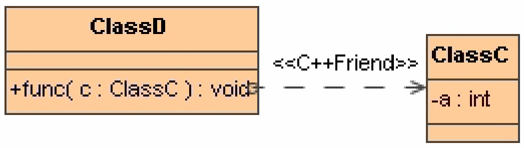
C++ friend function is mapped with a «C++Friend» stereotyped dependency relationship between the function (an UML operation) and the friendClass. This relationship grants the friendship to the friendClass. See *C++Friend* for more info.

Code	MD-UML

```
class ClassB {
public:
    friend void friendFunc();
};
void friendFunc();
```
	

C++ friend member function is mapped with a «C++Friend» stereotyped dependency relationship between the member function and the friend class. This relationship grants the friendship to the friend class.

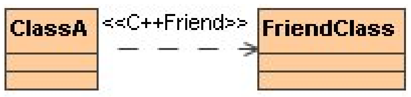
Code	MD-UML

```
class ClassD {
    void func(ClassC c);
};
class ClassC {
    friend void ClassD::func(ClassC
c);
};
```
	 ClassD
	+func(c : ClassC) : void
	<<C++Friend>>
	ClassC
	-a : int

Copyright © 1998-2015 No Magic, Inc.

C++ friend class are mapped with a «C++Friend» stereotyped dependency relationship between the class and the friendClass. This relationship grants the friendship to the friend class.

Code	MD-UML

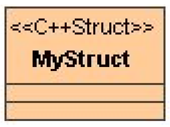
`class FriendClass {`
`public:`
`    friend class ClassA;`
`};`
`class ClassA {`
`};`
	

Struct

C++ struct are mapped to a UML class with stereotype «C++Struct». See *C++Struct* for more info.

NOTE The current version of MD use class's language property "Class Key"

Code	MD-UML

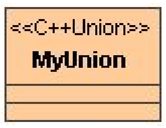
`struct MyStruc {`
`};`
	

Union

C++ union is mapped to a UML class with stereotype «C++Union». See *C++Union* for more info.

NOTE The current version of MD use class's language property "Class Key"

Code	MD-UML

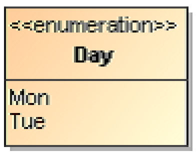
`union MyUnion {`
`};`
	

Enumeration

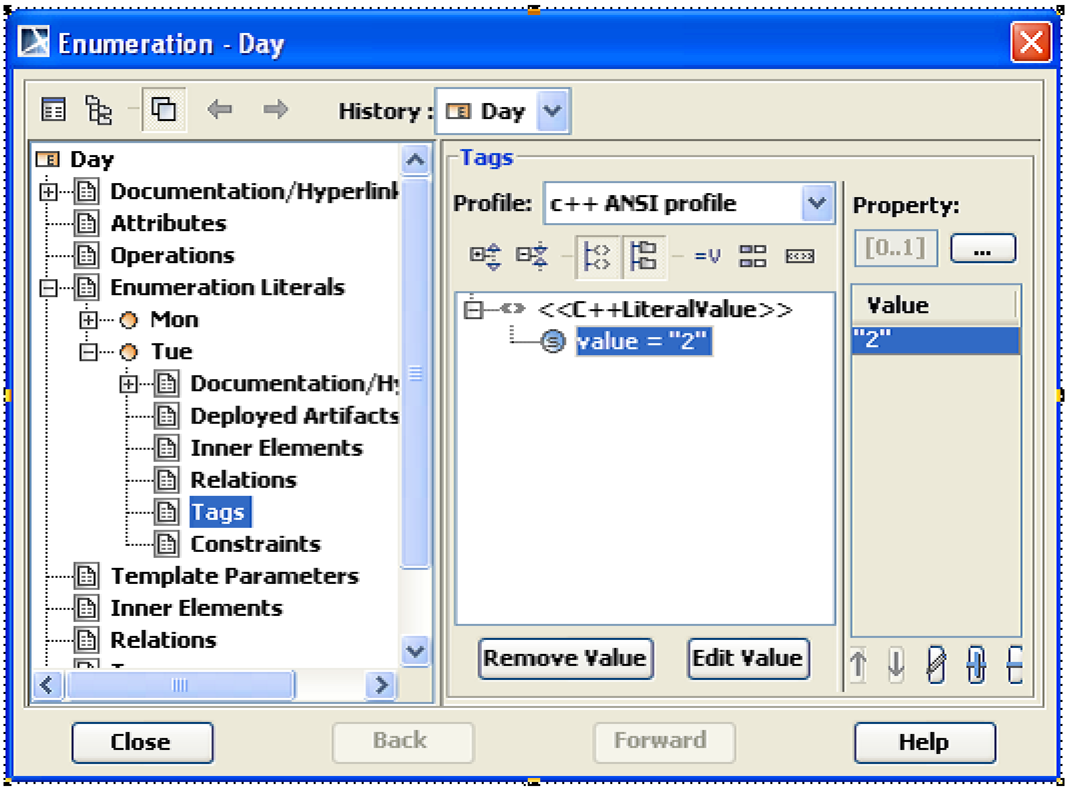
C++ enum is mapped to UML enumeration. C++ enum fields are mapped to UML enumeration literals.

C++ enum field with a specified value is mapped to tag value of «C++LiteralValue» stereotype.

Code	MD-UML

`enum Day {`
`    Mon,`
`    Tue=2`
`};`
	

Copyright © 1998-2015 No Magic, Inc.



Profile: c++ ANSI profile

<<C++LiteralValue>>
value = "2"

Value
"2"

Typedef

C++ typedef is mapped to a class with «C++Typedef» stereotype. A «C++BaseType» dependency links to the original type.

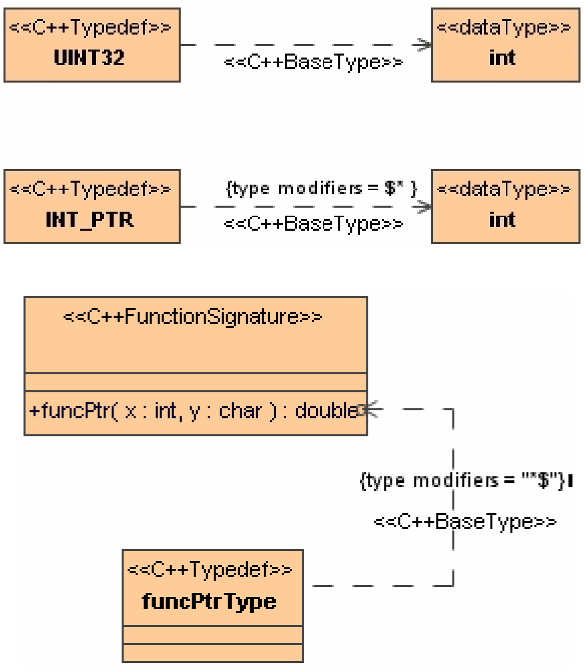
Type modifiers tag of «C++BaseType» dependency is used to define type modifiers. $ character is replaced by the type name.

A typedef on a function pointer is mapped by linking a «C++BaseType» dependency to an operation and type modifiers tag of «C++BaseType» dependency is set to *$. Operation signature can be stored in a «C++FunctionSignature» class.

See *C++Typedef* for more info.

Code	MD-UML

Copyright © 1998-2015 No Magic, Inc.

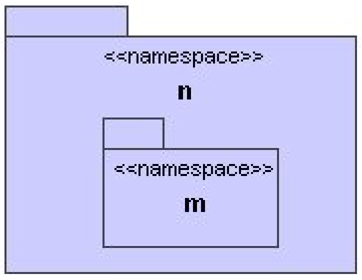
```
typedef int UINT32;
typedef int* INT_PTR;
typedef double (*funcPtrType)(int, char);
;
```
	 <<C++Typedef>>
	UINT32
	<<C++BaseType>>
	<<dataType>>
	int
	<<C++Typedef>>
	INT_PTR
	{type modifiers = $* }
	<<C++BaseType>>
	<<dataType>>
	int
	<<C++FunctionSignature>>
	+funcPtr(x : int, y : char) : double
	{type modifiers = "*$"}
	<<C++BaseType>>
	<<C++Typedef>>
	funcPtrType

Namespace

C++ namespace is mapped to a UML package with the stereotype «C++Namespace». See *C++Namespace* for more info.

Unnamed namespace is named unnamed+index number of unnamed namespace (start at 1), and *unique namespace name* tag is set to the source file path+:+index number of unnamed namespace (start at 0).

Code	MD-UML

```
namespace n {
    namespace m {
    }
}
```
	

[1]Global functions and variables

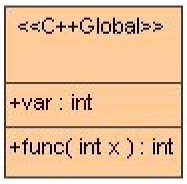
Global functions and variables are mapped to operations and attributes into an unnamed class with stereotype «C++Global». «C++Global» class resides in its respective namespace, or in a top package.

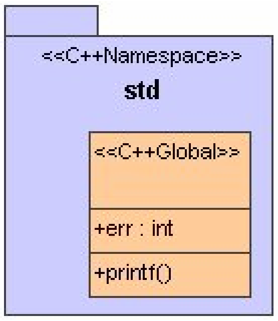
See *C++Global* for more info.

1.

Copyright © 1998-2015 No Magic, Inc.

Code	MD-UML

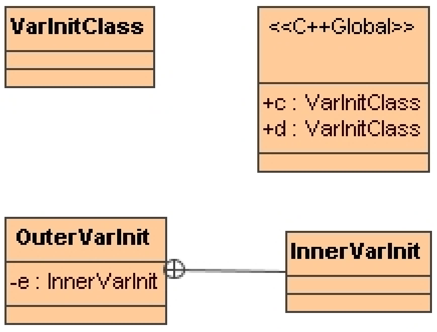
`int var;`
`int func(int x);`
	

`namespace std {`
`     int err;`
`     void printf();`
`}`
	 <<C++Namespace>>
	std
	<<C++Global>>
	+err : int
	+printf()

Class definition

Variables can be created after a class/struct/union declaration. These variables are mapped to UML attribute, and placed in their respective namespace/global/class container.

Code	MD-UML

`class VarInitClass {`
`} c, d;`
`class OuterVarInit {`
`     class InnerVarInit {`
`     } e;`
`};`
	 VarInitClass
	<<C++Global>>
	+c : VarInitClass
	+d : VarInitClass
	OuterVarInit
	-e : InnerVarInit
	InnerVarInit

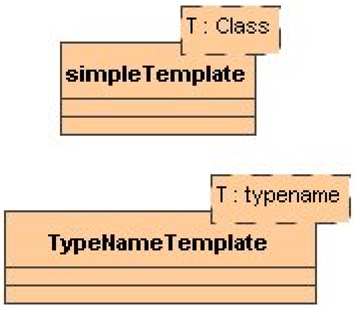
Class Template Definition

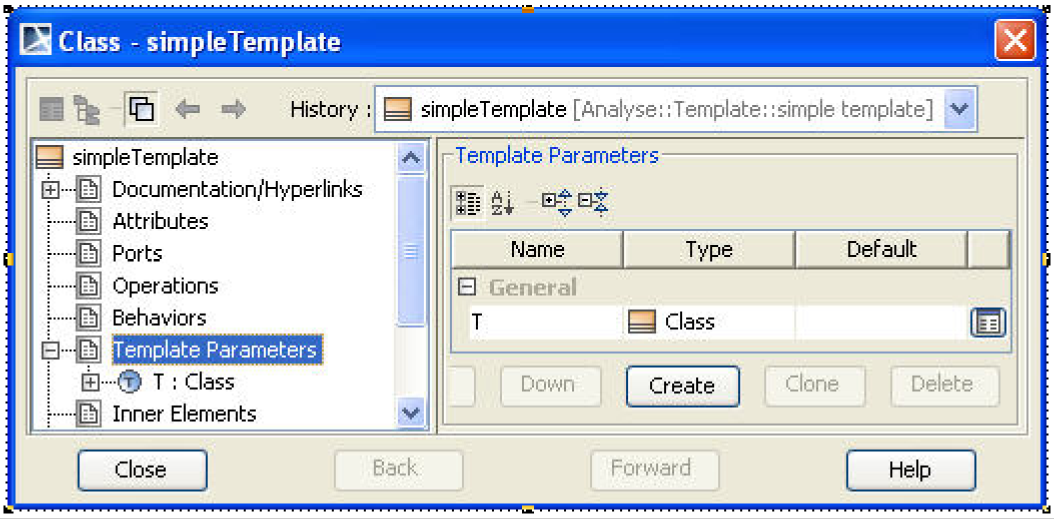
C++ template class is mapped to UML class with template parameters properties added.

Type of template parameter is always set to UML Class. To generate/reverse typename keyword, *type keyword* tag is set to *typename*.

Copyright © 1998-2015 No Magic, Inc.

Code	MD-UML

`template <class T>`
`class simpleTemplate {`
`};`
`template <typename T>`
`class TypeNameTemplate {`
`};`
	 T : Class
	simpleTemplate
	T : typename
	TypeNameTemplate

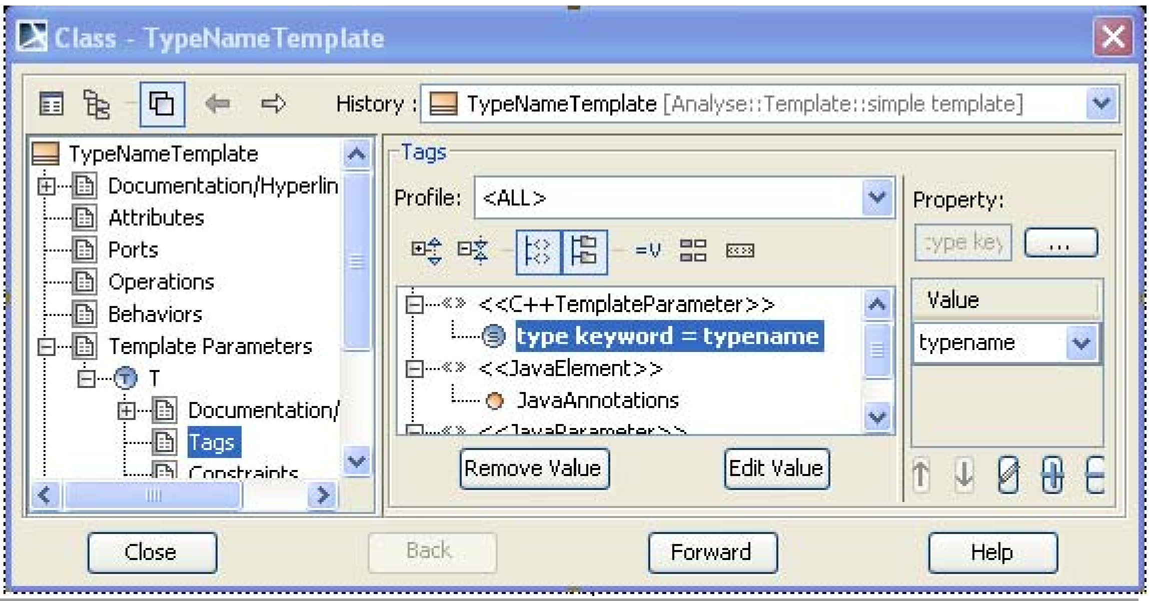


Class - simpleTemplate

History : simpleTemplate [Analyse::Template::simple template]

Template Parameters

Name	Type	Default
General		
T	Class	



Class - TypeNameTemplate

History : TypeNameTemplate [Analyse::Template::simple template]

Tags

Profile: <ALL>

<<C++TemplateParameter>>
type keyword = typename
<<JavaElement>>
JavaAnnotations

Property: type key

Value: typename

Function Template Definition

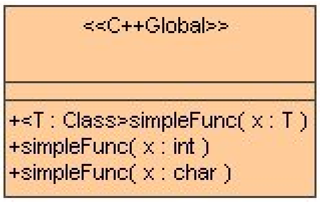
C++ template function is mapped to UML operation with template parameters properties added.

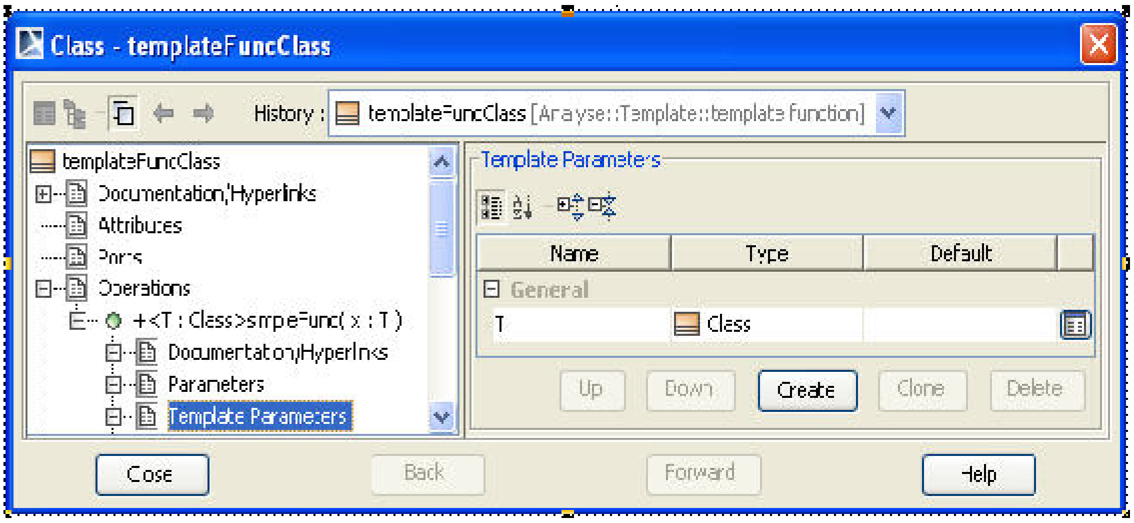
C++ template function overload is mapped to a normal function. (the same name with the same number of parameter, but different type of parameter)

Copyright © 1998-2015 No Magic, Inc.

New style of template function overloading is mapped to a normal function. (the same name with the same number of parameter, but different type of parameter) and a template binding relationship is created between the overload operation and the template operation, with specific template parameter substitutions.

Code	MD-UML

`template <class T>`
`void simpleFunc(T x);`
`// overload old style`
`void simpleFunc(int x);`
`// overload new style`
`template<>`
`void simpleFunc<char>(char x);`
	 <<C++Global>>
	+<T : Class>simpleFunc(x : T)
	+simpleFunc(x : int)
	+simpleFunc(x : char)

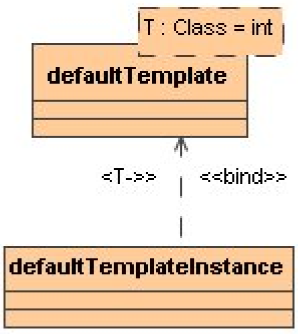


Default template parameter

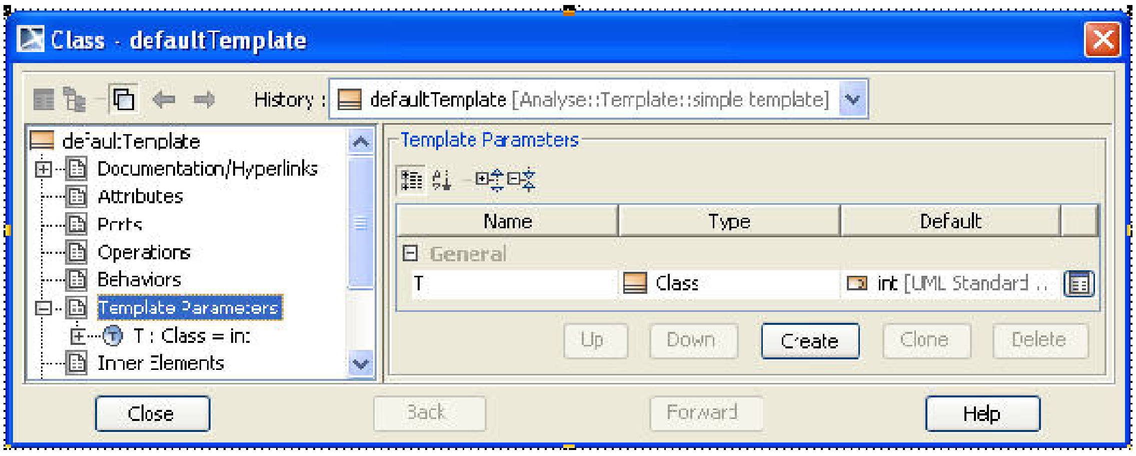
C++ default template parameter is mapped to UML default template parameters.

Instantiation using the default template parameter is mapped using a template binding relationship with an empty *actual* property for the *template parameter substitution*.

Code	MD-UML

`template <class T=int>`
`class defaultTemplate {`
`};`
	 T : Class = int
	defaultTemplate
	<T->
	<<bind>>
	defaultTemplateInstance

Copyright © 1998-2015 No Magic, Inc.



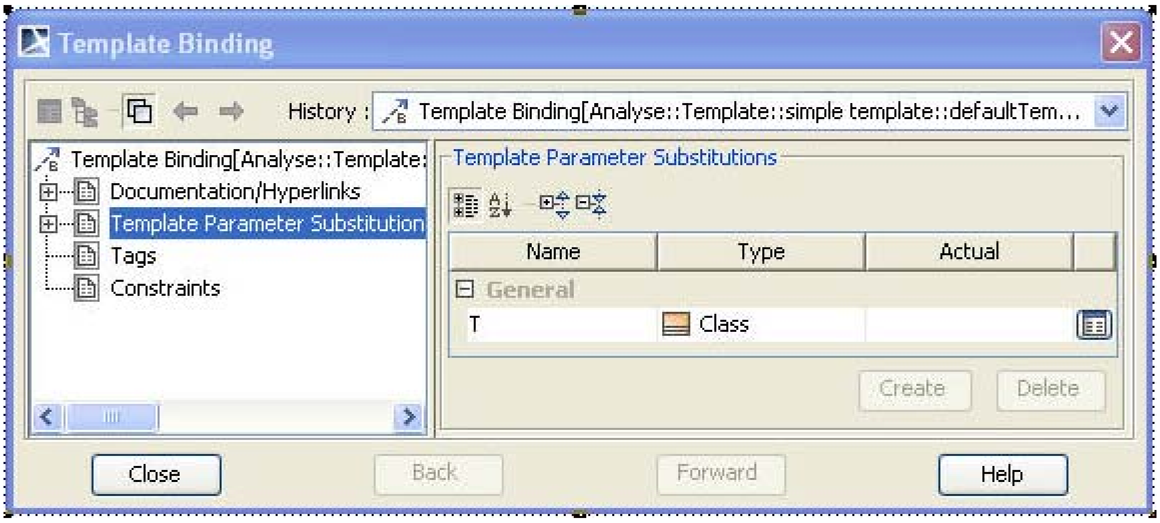
Class - defaultTemplate

History : defaultTemplate [Analyse::Template::simple template]

- defaultTemplate
  - Documentation/Hyperlinks
  - Attributes
  - Ports
  - Operations
  - Behaviors
  - Template Parameters
    - T : Class = int
  - Inner Elements

Template Parameters

Name	Type	Default
General		
T	Class	int [UML Standard ...



Template Binding

History : Template Binding[Analyse::Template::simple template::defaultTem...

- Template Binding[Analyse::Template:
  - Documentation/Hyperlinks
  - Template Parameter Substitution
  - Tags
  - Constraints

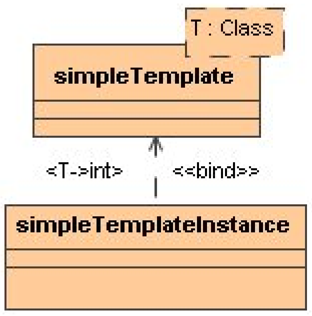
Template Parameter Substitutions

Name	Type	Actual
General		
T	Class	

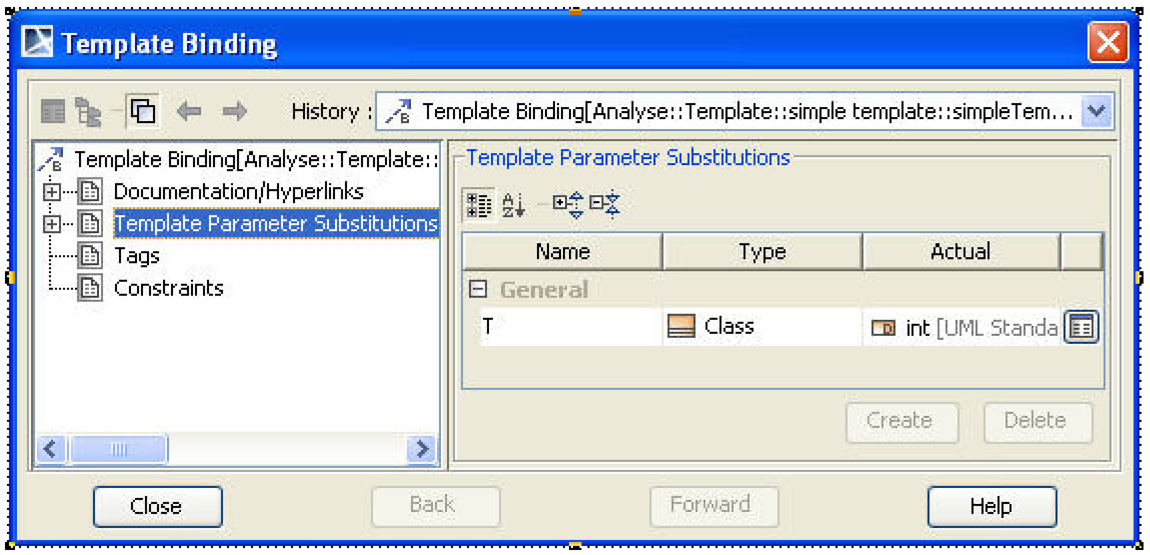
Template instantiation

Template instantiation are mapped to template binding relationship between the template class and the instantiate class, the *template parameter substitution* of the binding relationship is set using the template argument.

Code	**MD-UML**

```
template <class T>
class simpleTemplate {
};

simpleTemplate<int> simpleTemplateInstance;
```
	 T : Class
	simpleTemplate
	<T->int>
	<<bind>>
	simpleTemplateInstance

Copyright © 1998-2015 No Magic, Inc.

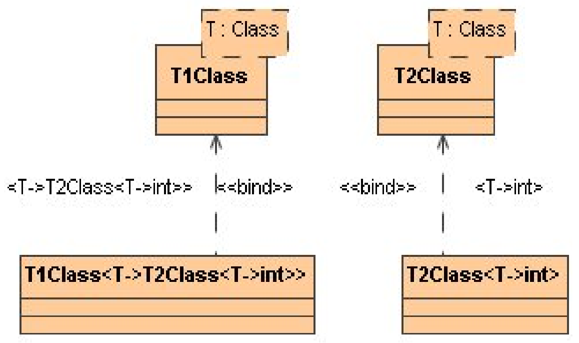

Template Binding
History : Template Binding[Analyse::Template::simple template::simpleTem...
Template Binding[Analyse::Template::
Documentation/Hyperlinks
Template Parameter Substitutions
Tags
Constraints
Template Parameter Substitutions
Create
Delete
Close
Back
Forward
Help

Name	Type	Actual
General		
T	Class	int [UML Standa

For template argument using template instantiation as argument, an intermediate class is created with the specific binding

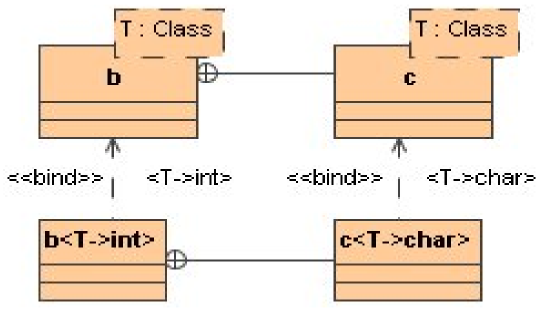
.

Code	MD-UML

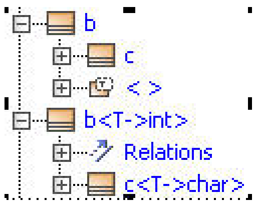
`template <class T>`
`class T1Class {`
`};`
`template <class T>`
`class T2Class {`
`};`
 	
`T1Class<T2Class<int>> ...`
	 T : Class
	T1Class
	T : Class
	T2Class
	<T->T2Class<T->int>>
	<<bind>>
	<<bind>>
	<T->int>
	T1Class<T->T2Class<T->int>>
	T2Class<T->int>

For template argument using multiple template instantiations in an inner class (b<int>::c<char>), the intermediate class instance is created in the outer class instance.

Code	MD-UML

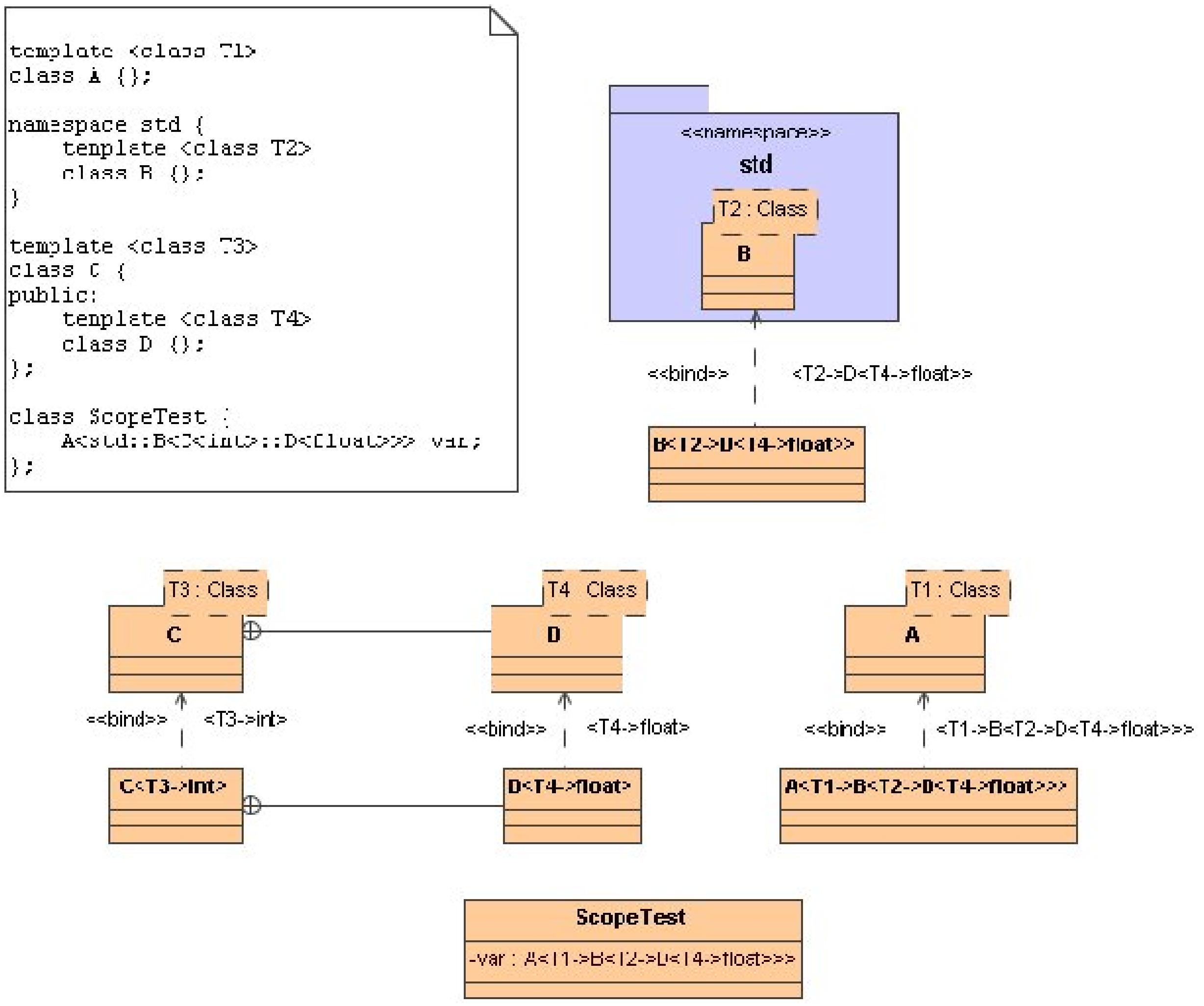
`template <class T>`
`class b {`
`     template <class T>`
`     class c {`
`     };`
`};`
 	
`b<int>::c<char> ...`
	 T : Class
	b
	T : Class
	c
	<<bind>>
	<T->int>
	<<bind>>
	<T->char>
	b<T->int>
	c<T->char>

Copyright © 1998-2015 No Magic, Inc.

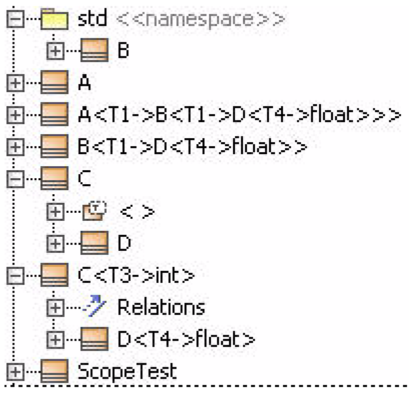
	

b
- c
- < >
b<T->int>
- Relations
- c<T->char>

Example of complex template instantiation. Containment relationship are placed on diagram for information only, these relationships are not created during a reverse process. Containment relationship is modeled by placing a class into a specific class/package. See Containment tree below the diagram.



```
template <class T1>
class A {};

namespace std {
    template <class T2>
    class B {};
}

template <class T3>
class C {
public:
    template <class T4>
    class D {};
};

class ScopeTest {
    A<std::B<C<int>::D<float>>> var;
};
```



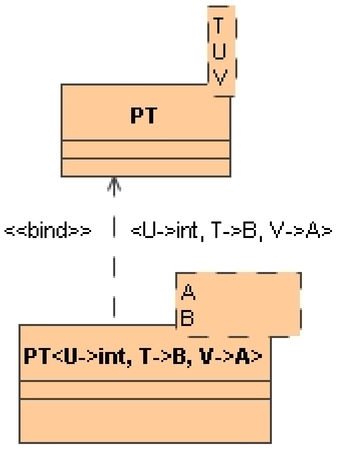
std <<namespace>>
- B
A
A<T1->B<T1->D<T4->float>>>
B<T1->D<T4->float>>
C
- < >
- D
C<T3->int>
- Relations
- D<T4->float>
ScopeTest

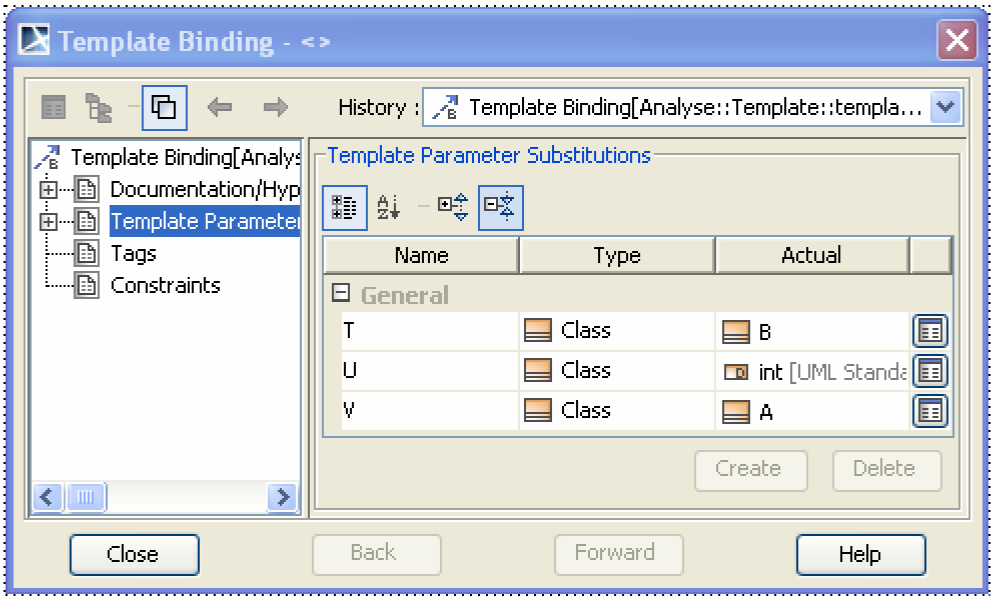
Copyright © 1998-2015 No Magic, Inc.

Partial template instantiation

C++ partial template instantiation use the same mapping as Template Instantiation and the unbinded parameter is binded to the specific template parameter class.

Code	MD-UML

`template <class T,class U,class V>`
`class PT {};`
`template <class A,class B>`
`class PT<B, int, A> {};`
	 T
	U
	V
	PT
	<<bind>>
	<U->int, T->B, V->A>
	A
	B
	PT<U->int, T->B, V->A>



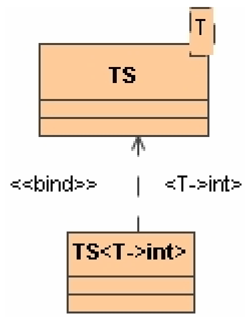
Name	Type	Actual
General		
T	Class	B
U	Class	int [UML Standa
V	Class	A

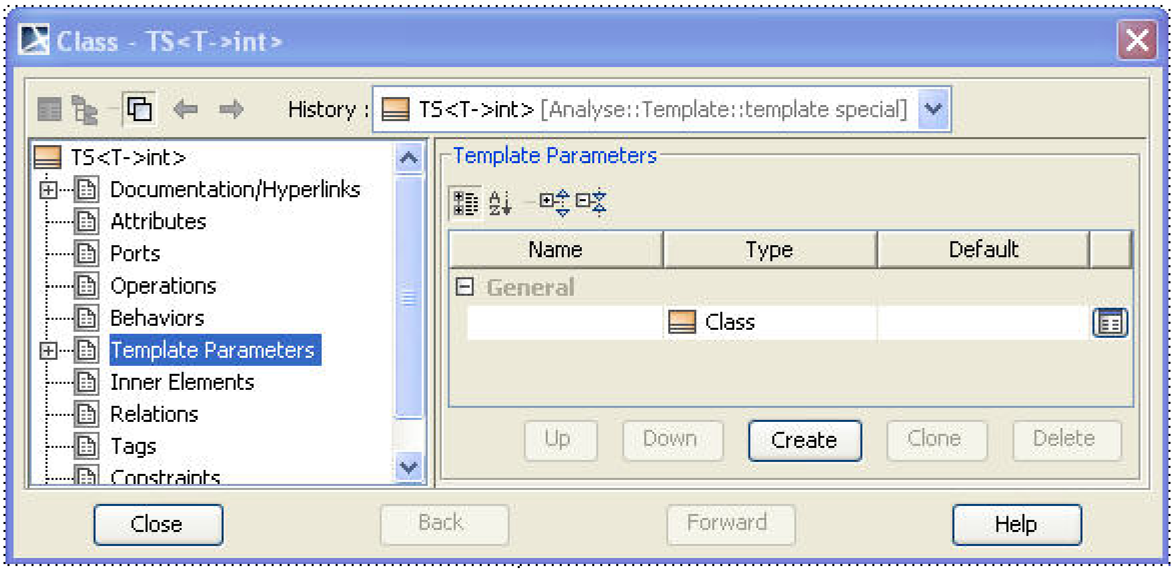
Template specialization

C++ Template specialization uses the same mapping as Template Instantiation.

Copyright © 1998-2015 No Magic, Inc.

Code	MD-UML

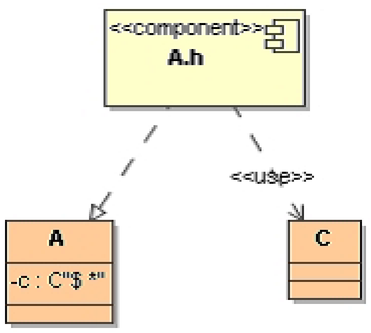
`template <class T>`
`class TS {};`
`template <>`
`class TS<int> {};`
	



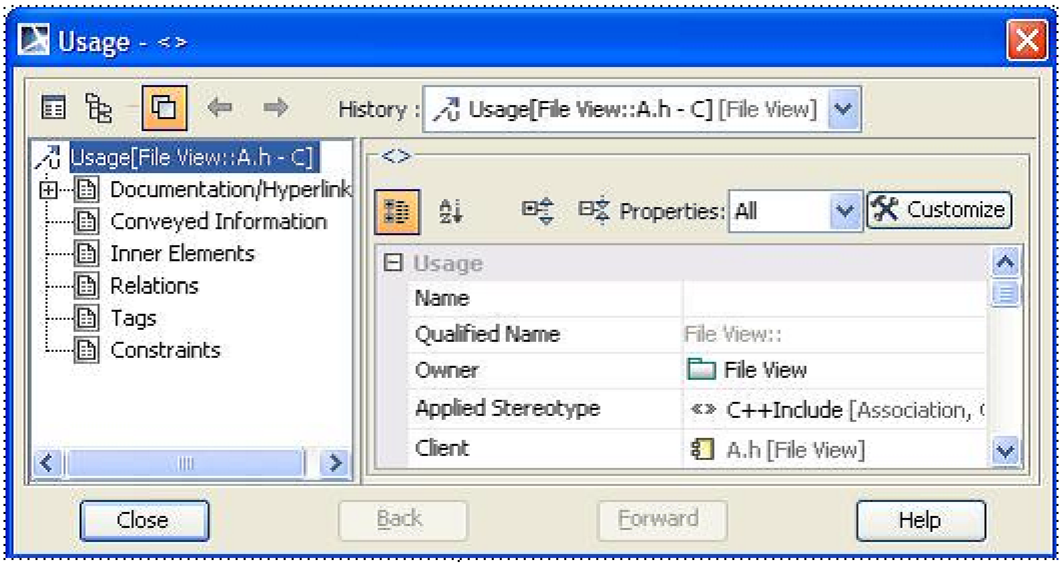
Forward class declaration

The example code is declared in A.h file. The file component A.h has the «use» association applied by «C++Include» stereotype with "Class Forward" tag value.

Code	MD-UML

`class C;`
`class A {`
`     private:`
`      C* c;`
`};`
	

Copyright © 1998-2015 No Magic, Inc.



Usage - <>

History : Usage[File View::A.h - C] [File View]

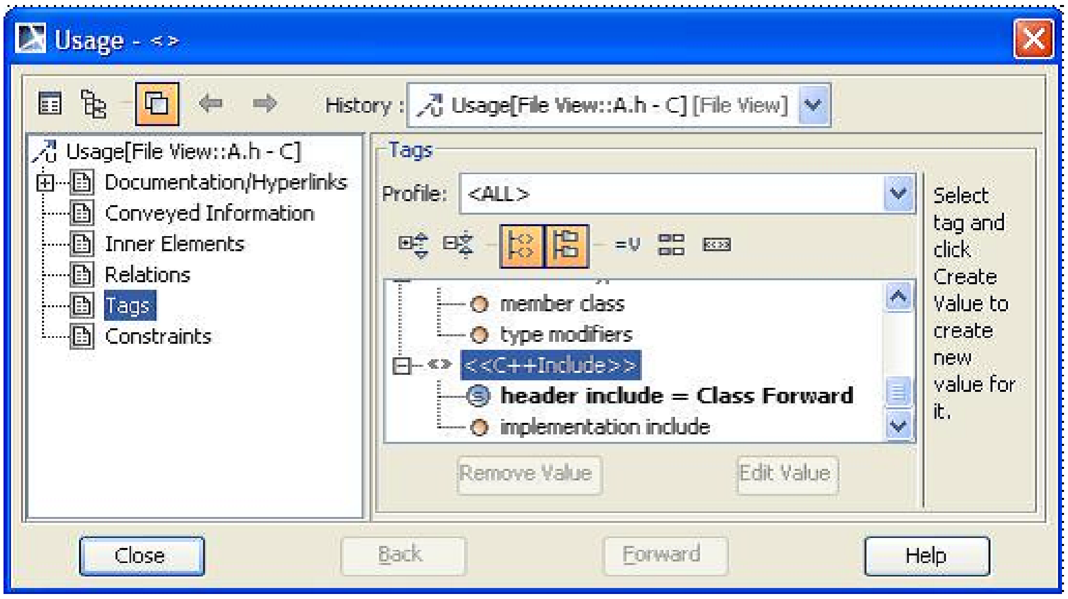
- Usage[File View::A.h - C]
  - Documentation/Hyperlink
  - Conveyed Information
  - Inner Elements
  - Relations
  - Tags
  - Constraints

<>

Properties: All

Usage	
Name	
Qualified Name	File View::
Owner	File View
Applied Stereotype	<> C++Include [Association, (
Client	A.h [File View]

Close Back Forward Help



Usage - <>

History : Usage[File View::A.h - C] [File View]

- Usage[File View::A.h - C]
  - Documentation/Hyperlinks
  - Conveyed Information
  - Inner Elements
  - Relations
  - Tags
  - Constraints

Tags

Profile: <ALL>

```
member class
type modifiers
<> <<C++Include>>
   header include = Class Forward
   implementation include
```

Remove Value   Edit Value

Select tag and click Create Value to create new value for it.

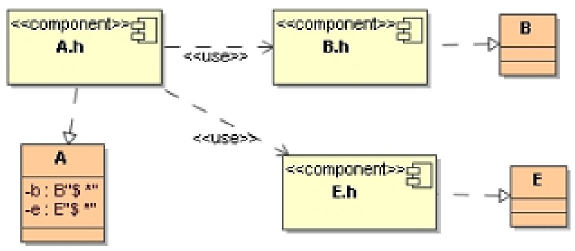
Close Back Forward Help

Include declaration

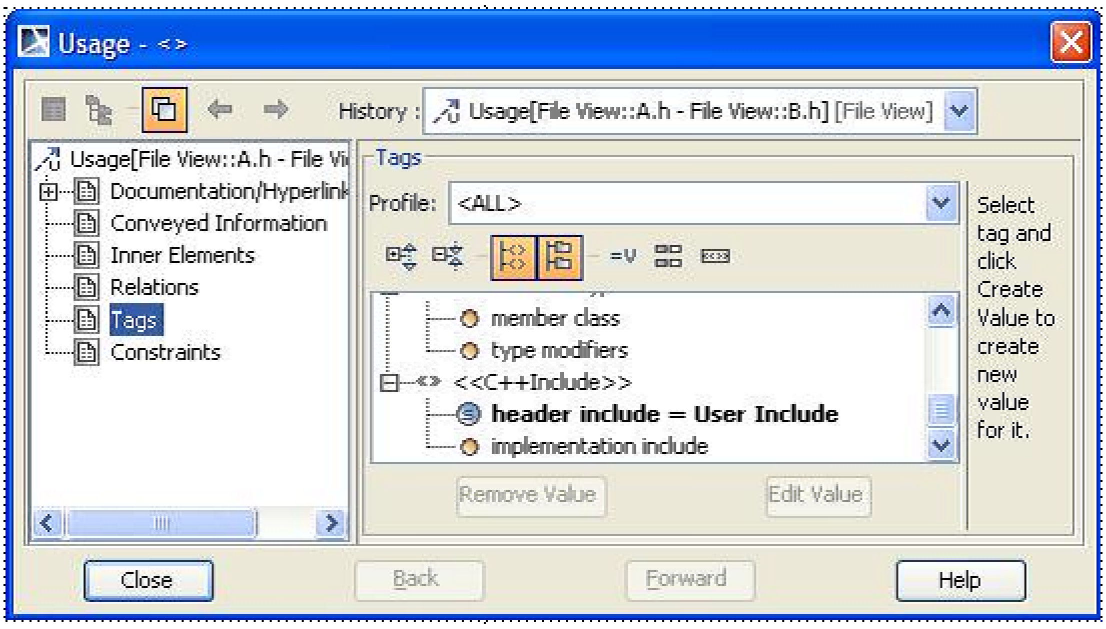
The example code is declared in A.h file. The «use» association is also applied to C++Include stereotype shown in section 3.36 Forward class declaration.

Copyright © 1998-2015 No Magic, Inc.

Code	MD-UML

`#include "B.h"`
`#include <E.h>`
`class A {`
`     private:`
`      B* b;`
`      E* e;`
`};`
	 <<component>>
	A.h
	<<use>>
	<<component>>
	B.h
	B
	A
	-b : B"$ *"
	-e : E"$ *"
	<<use>>
	<<component>>
	E.h
	E

```
Specification for #include "B.h"
```



Usage - < >

History : Usage[File View::A.h - File View::B.h] [File View]

- Usage[File View::A.h - File Vi
  - Documentation/Hyperlink
  - Conveyed Information
  - Inner Elements
  - Relations
  - Tags
  - Constraints

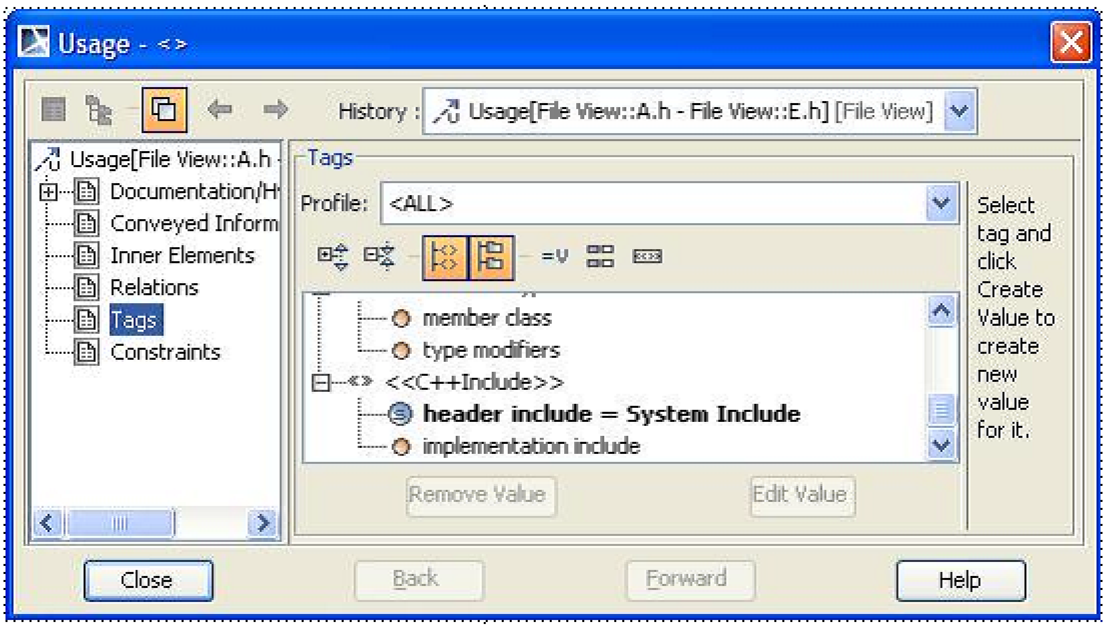
Tags

Profile: <ALL>

- member class
- type modifiers
- <<C++Include>>
  - **header include = User Include**
  - implementation include

Select tag and click Create Value to create new value for it.

```
Specification for #include <E.h>
```



Usage - < >

History : Usage[File View::A.h - File View::E.h] [File View]

- Usage[File View::A.h -
  - Documentation/H
  - Conveyed Inform
  - Inner Elements
  - Relations
  - Tags
  - Constraints

Tags

Profile: <ALL>

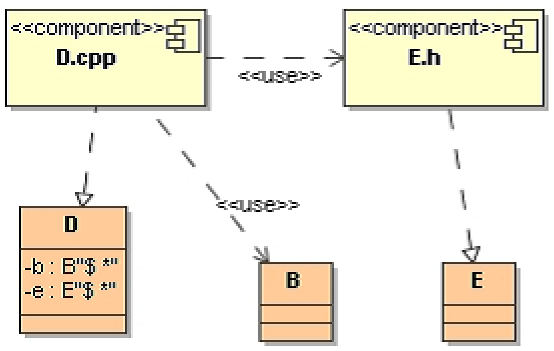
- member class
- type modifiers
- <<C++Include>>
  - **header include = System Include**
  - implementation include

Select tag and click Create Value to create new value for it.

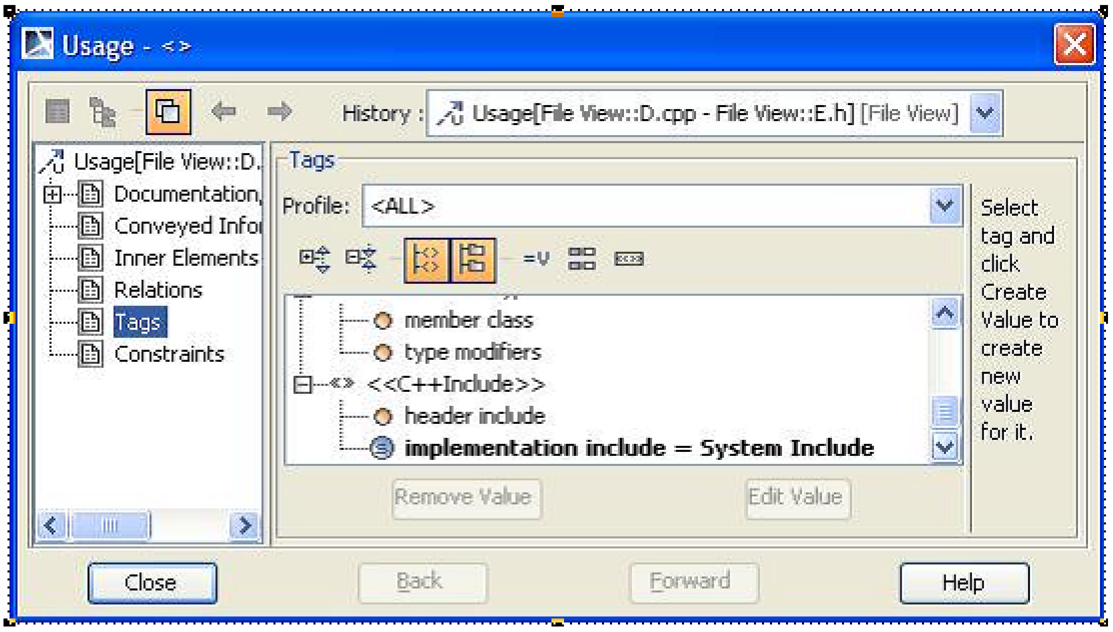
Copyright © 1998-2015 No Magic, Inc.

The example code is declared in D.cpp file.

Code	**MD-UML**

`#include <E.h>`
`class B;`
`class D {`
`private:B* b;`
`E* e;`
`}`
	 <<component>>
	D.cpp
	<<component>>
	E.h
	<<use>>
	<<use>>
	D
	-b : B"$ *"
	-e : E"$ *"
	B
	E

```
Specification for #include <E.h>
```


Usage - < >

History : Usage[File View::D.cpp - File View::E.h] [File View]

Usage[File View::D.
- Documentation
- Conveyed Infor
- Inner Elements
- Relations
- Tags
- Constraints

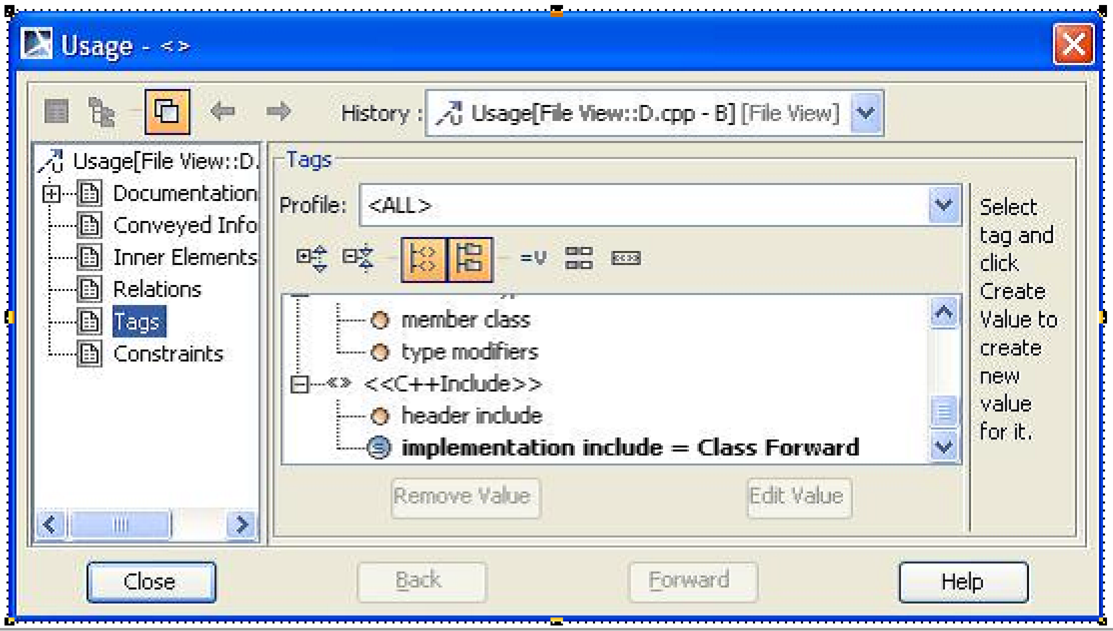
Tags

Profile: <ALL>

member class
type modifiers
<<C++Include>>
header include
implementation include = System Include

Select tag and click Create Value to create new value for it.

```
Specification for class B;
```


Usage - < >

History : Usage[File View::D.cpp - B] [File View]

Usage[File View::D.
- Documentation
- Conveyed Info
- Inner Elements
- Relations
- Tags
- Constraints

Tags

Profile: <ALL>

member class
type modifiers
<<C++Include>>
header include
implementation include = Class Forward

Select tag and click Create Value to create new value for it.

Copyright © 1998-2015 No Magic, Inc.

Conversion from old project version

This chapter describes the changes applied when loading a MD project version <= 11.6.

Translation Activity Diagram

There are projects that use C++ language properties or C++ profile or have type modifier, which need to be translated with version of MagicDraw project less than or equal 11.6.

Open local project

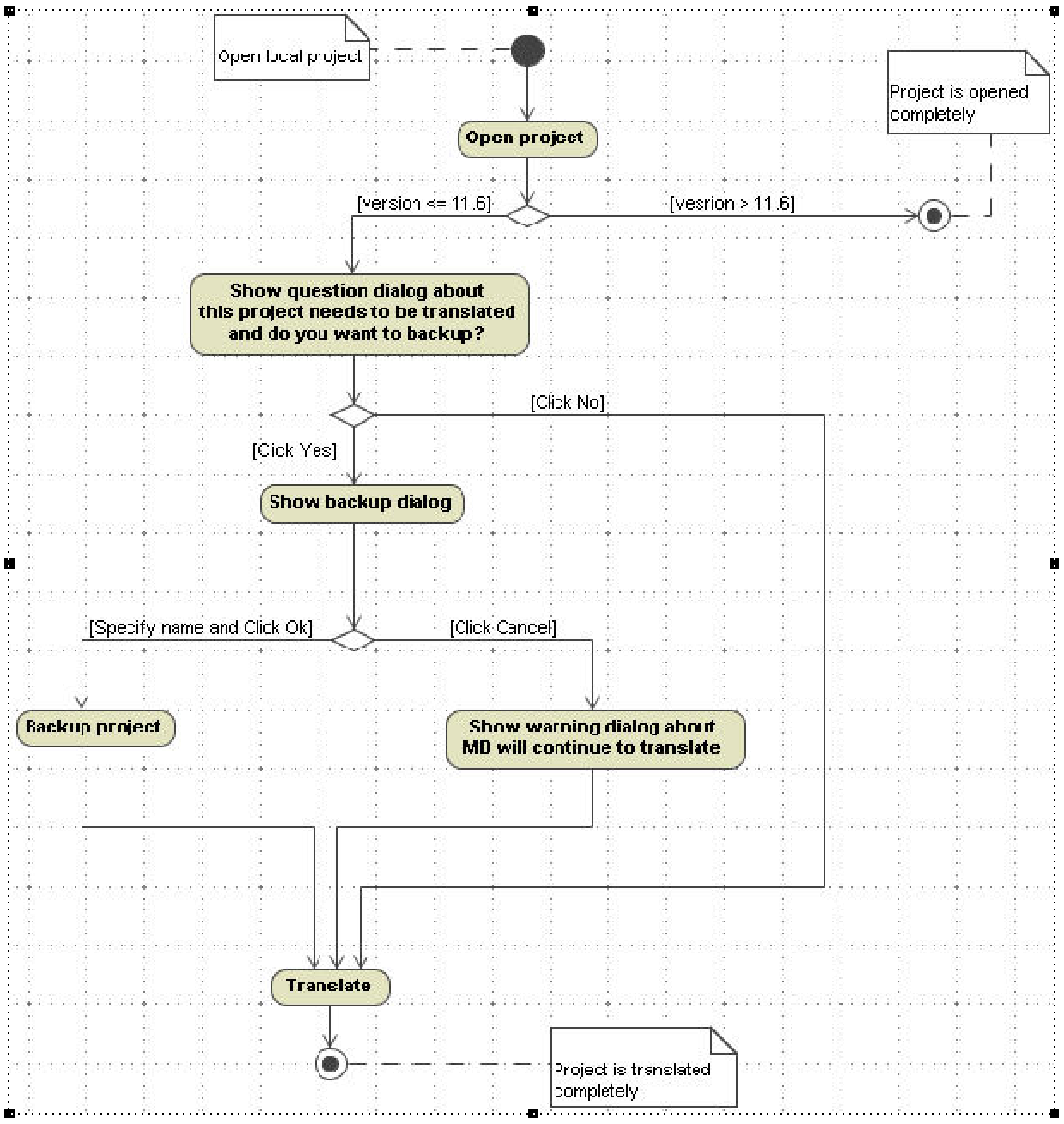

Open local project
Project is opened completely
Open project
[version <= 11.6]
[vesrion > 11.6]
Show question dialog about this project needs to be translated and do you want to backup?
[Click No]
[Cick Yes]
Show backup dialog
[Specify name and Click Ok]
[Click Cancel]
Backup project
Show warning dialog about MD will continue to translate
Translate
Project is translated completely

Figure 54 -- Open local project Activity Diagram

Copyright © 1998-2015 No Magic, Inc.

Open teamwork project

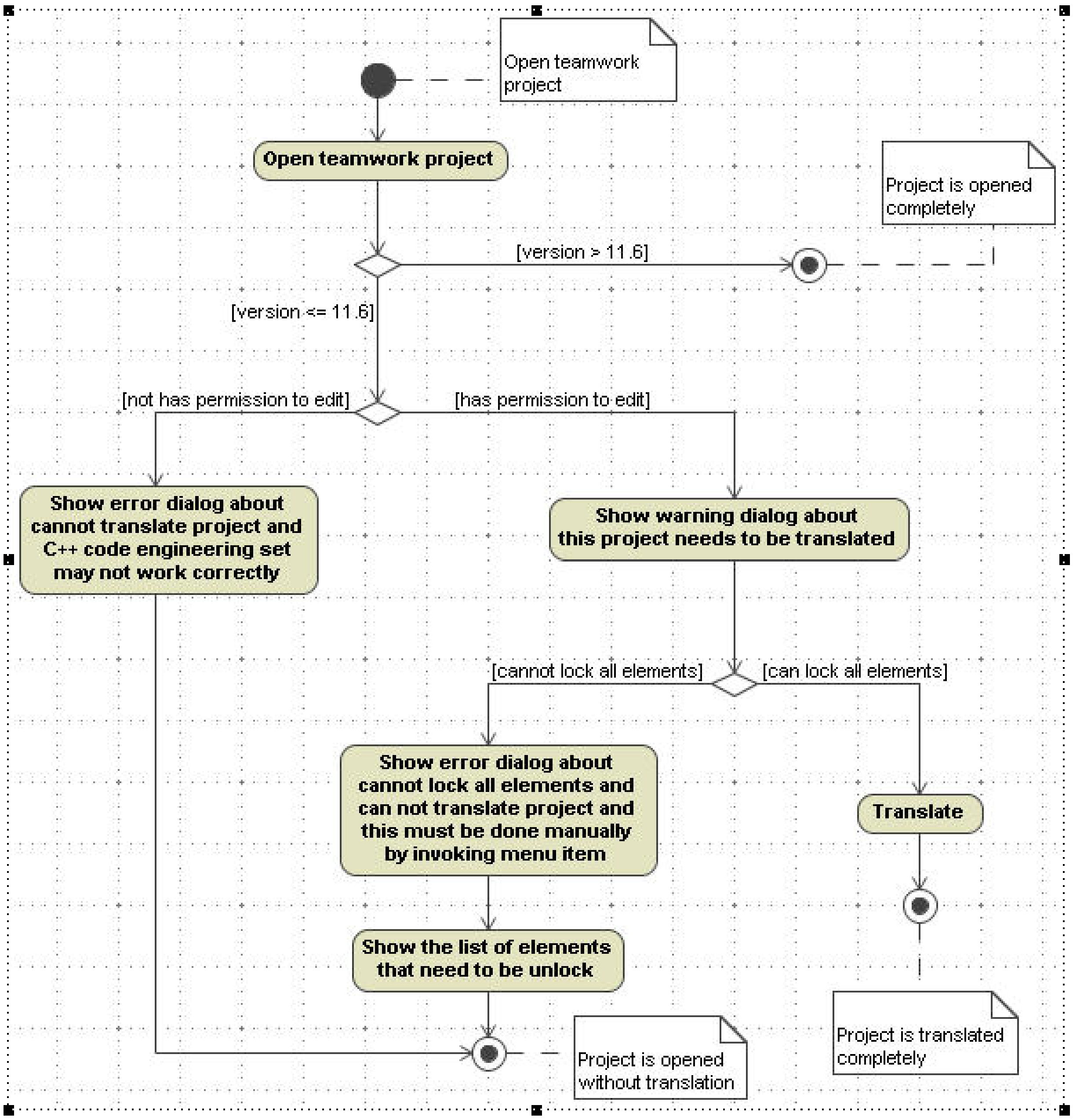

Open teamwork project
Open teamwork project
Project is opened completely
[version > 11.6]
[version <= 11.6]
[not has permission to edit]
[has permission to edit]
Show error dialog about cannot translate project and C++ code engineering set may not work correctly
Show warning dialog about this project needs to be translated
[cannot lock all elements]
[can lock all elements]
Show error dialog about cannot lock all elements and can not translate project and this must be done manually by invoking menu item
Translate
Show the list of elements that need to be unlock
Project is opened without translation
Project is translated completely

Figure 55 -- Open teamwork project Activity Diagram

Copyright © 1998-2015 No Magic, Inc.

Import MagicDraw project

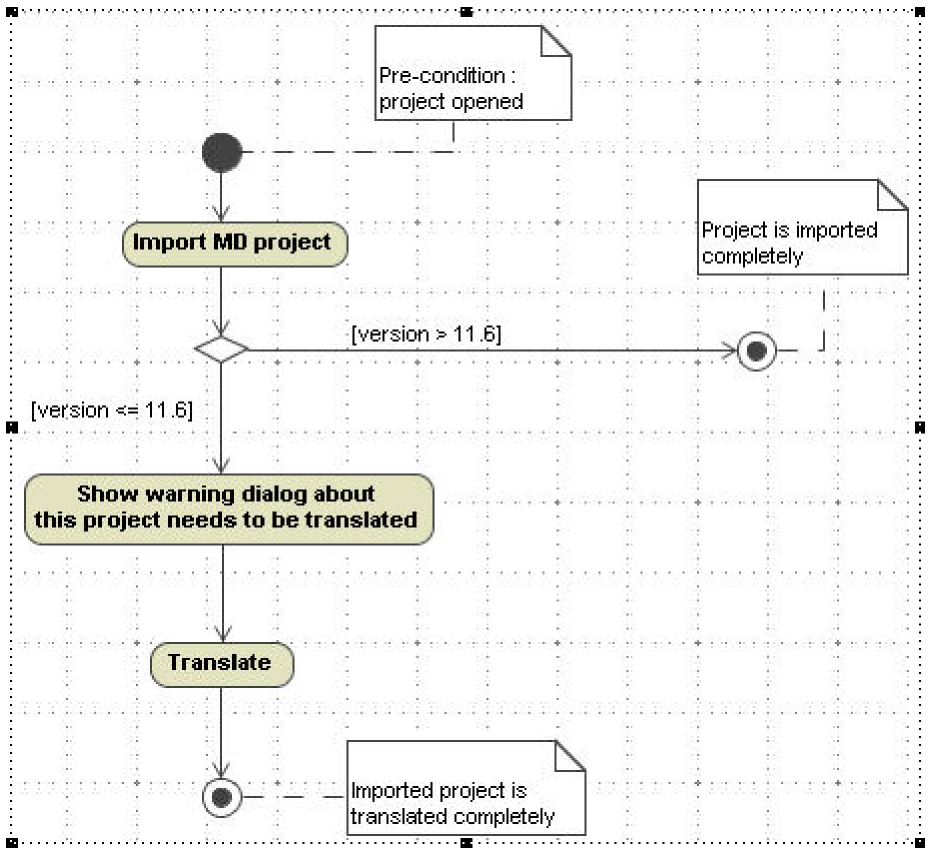

Pre-condition : project opened
Import MD project
Project is imported completely
[version > 11.6]
[version <= 11.6]
Show warning dialog about this project needs to be translated
Translate
Imported project is translated completely

Figure 56 -- Import MagicDraw project Activity Diagram

Copyright © 1998-2015 No Magic, Inc.

Use module

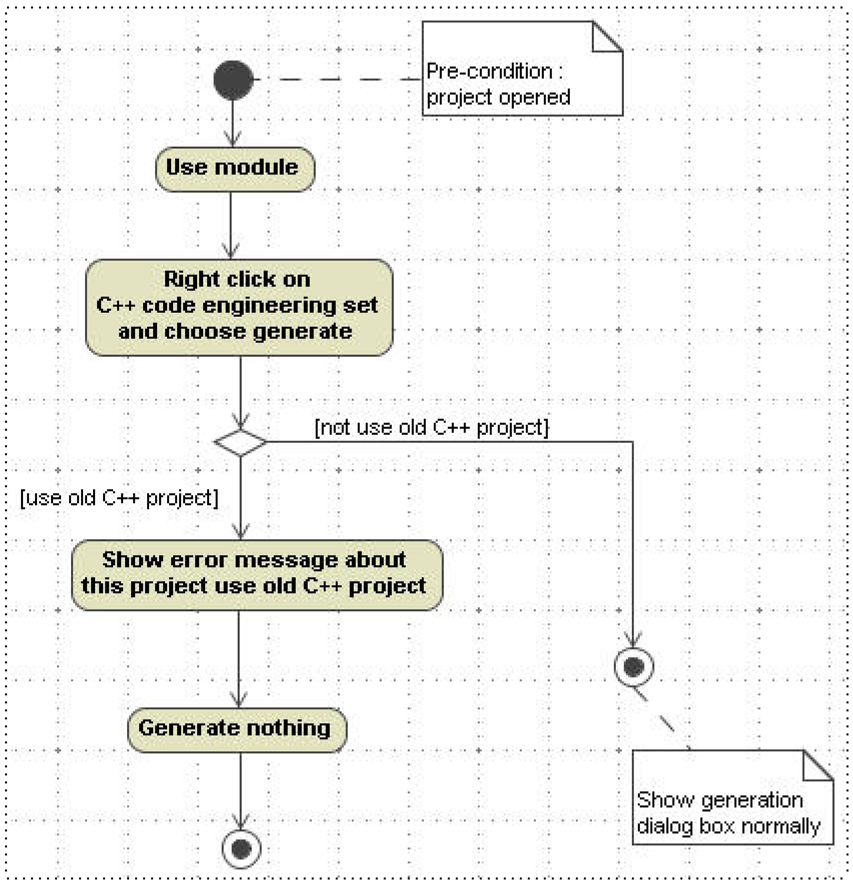

Pre-condition : project opened
Use module
Right click on C++ code engineering set and choose generate
[not use old C++ project]
[use old C++ project]
Show error message about this project use old C++ project
Generate nothing
Show generation dialog box normally

Figure 57 -- Use module Activity Diagram

Copyright © 1998-2015 No Magic, Inc.

Update C++ Language Properties and Profiles

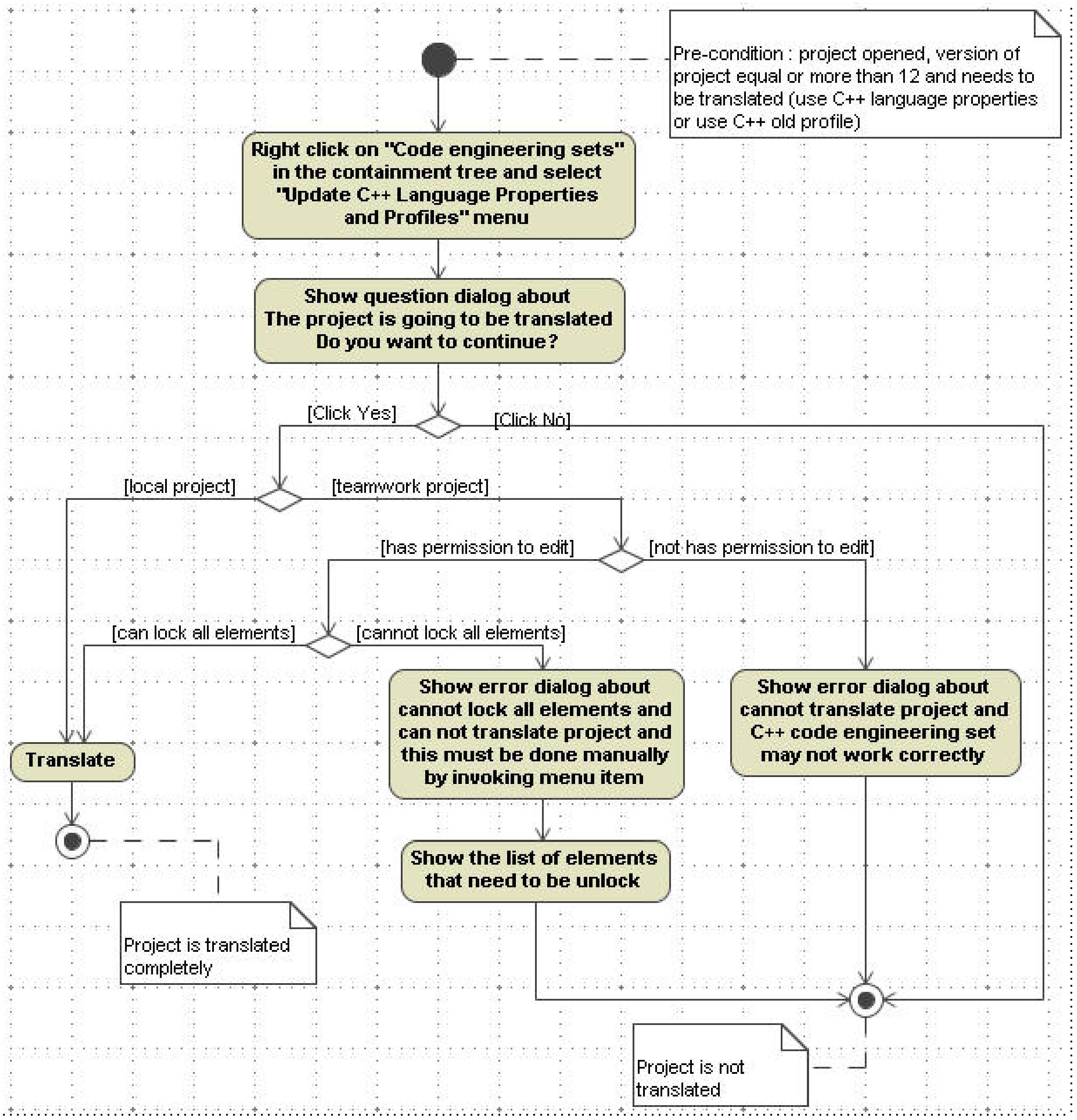

Pre-condition : project opened, version of project equal or more than 12 and needs to be translated (use C++ language properties or use C++ old profile)
Right click on "Code engineering sets" in the containment tree and select "Update C++ Language Properties and Profiles" menu
Show question dialog about The project is going to be translated Do you want to continue?
[Click Yes]
[Click No]
[local project]
[teamwork project]
[has permission to edit]
[not has permission to edit]
[can lock all elements]
[cannot lock all elements]
Show error dialog about cannot lock all elements and can not translate project and this must be done manually by invoking menu item
Show error dialog about cannot translate project and C++ code engineering set may not work correctly
Translate
Show the list of elements that need to be unlock
Project is translated completely
Project is not translated

Figure 58 -- Update C++ Language Properties and Profiles Activity Diagram

Language properties

Until MD version 11.6, language properties are stored in a specific format, since MD version 12 language properties are moved to stereotype’s tag value.

Copyright © 1998-2015 No Magic, Inc.

Class

There are **class**, **struct** and **union** class key in Class Language Properties.

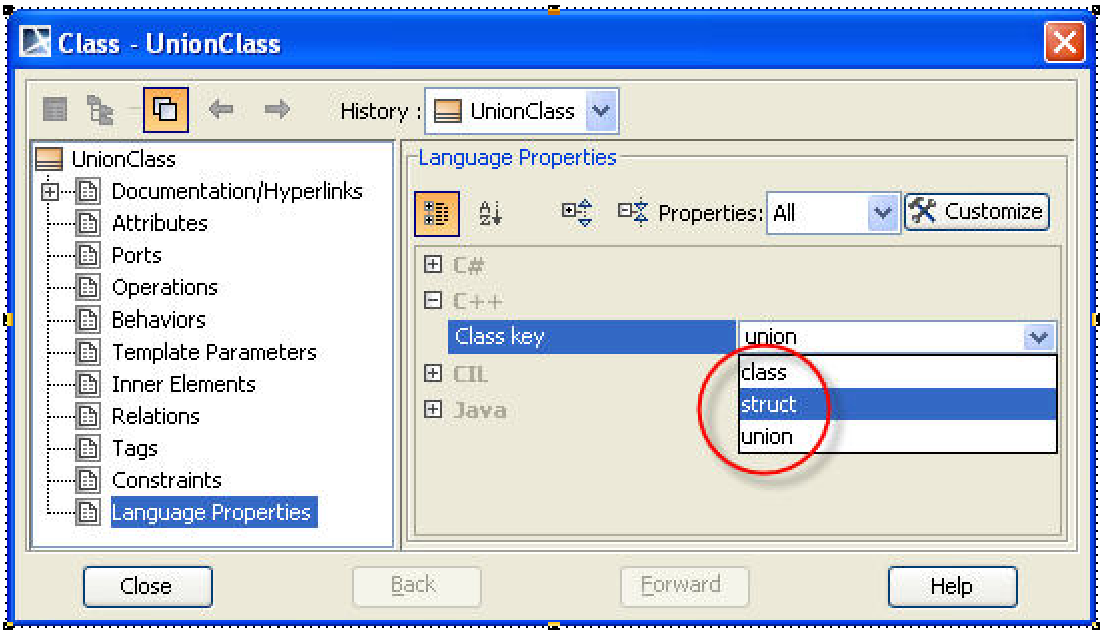
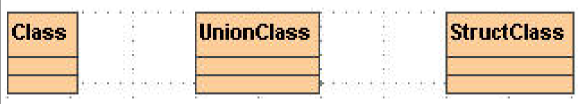
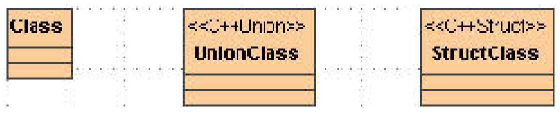
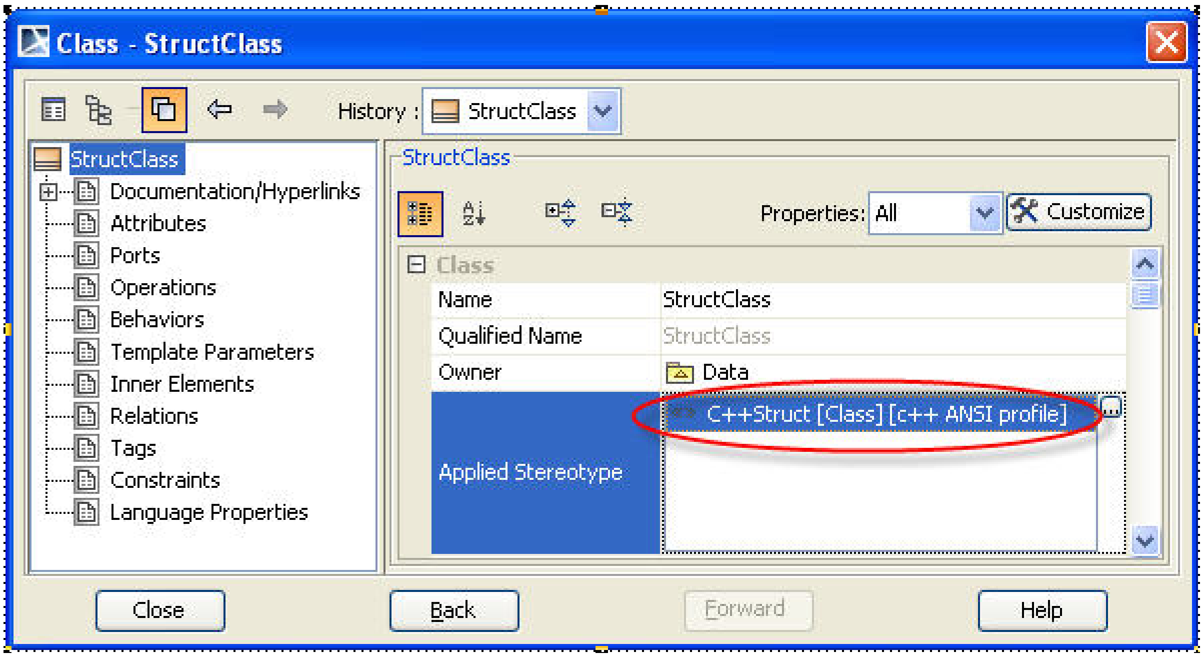


Figure 59 -- Class Language Properties

Copyright © 1998-2015 No Magic, Inc.

Class - Class key

Old value	Translation
	
`class`	no change.
`struct`	Apply the **«C++Struct»** stereotype.



Class - StructClass

History : StructClass

StructClass
- Documentation/Hyperlinks
- Attributes
- Ports
- Operations
- Behaviors
- Template Parameters
- Inner Elements
- Relations
- Tags
- Constraints
- Language Properties

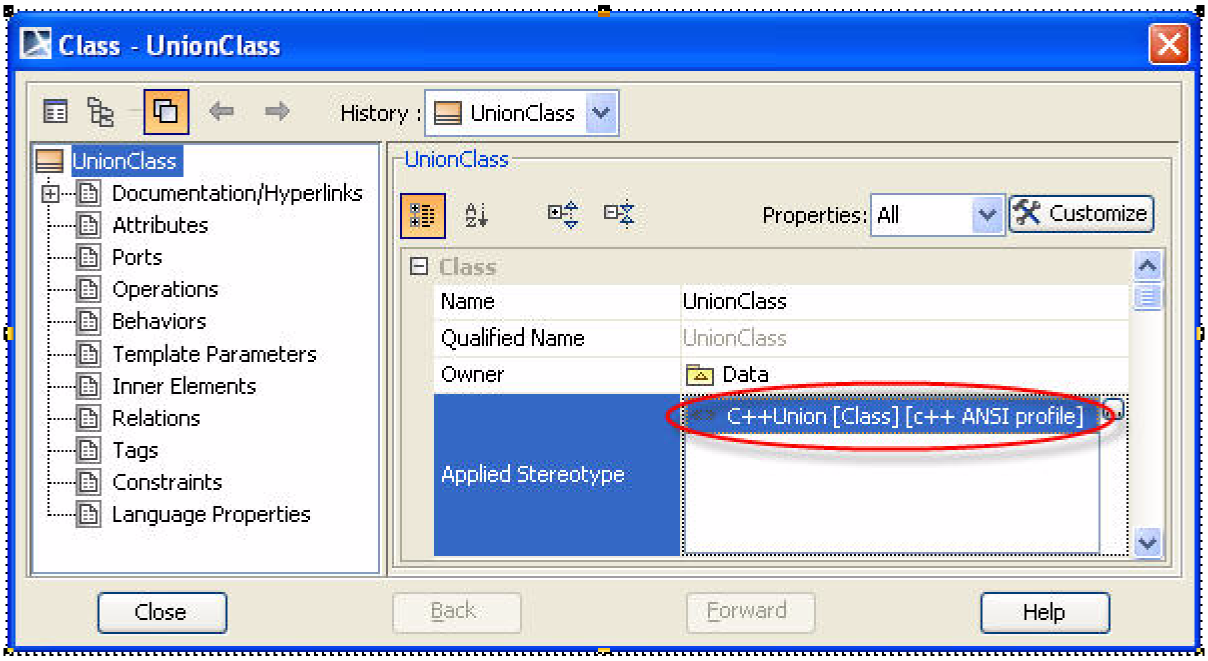
StructClass

Properties: All — Customize

Class

Name	StructClass
Qualified Name	StructClass
Owner	Data
Applied Stereotype	C++Struct [Class] [c++ ANSI profile]

`union`	Apply the **«C++Union»** stereotype.



Class - UnionClass

History : UnionClass

UnionClass
- Documentation/Hyperlinks
- Attributes
- Ports
- Operations
- Behaviors
- Template Parameters
- Inner Elements
- Relations
- Tags
- Constraints
- Language Properties

UnionClass

Properties: All — Customize

Class

Name	UnionClass
Qualified Name	UnionClass
Owner	Data
Applied Stereotype	C++Union [Class] [c++ ANSI profile]

Copyright © 1998-2015 No Magic, Inc.

Operation

This example is **OperationClass** class that has operation named **OperationClass()** which is constructor and operation named **myOperation**.

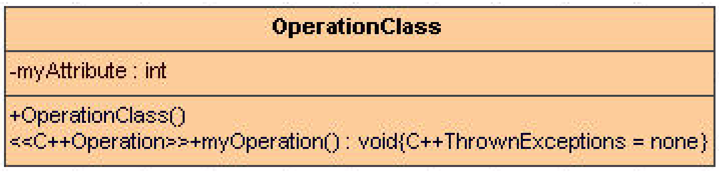

OperationClass
-myAttribute : int
+OperationClass()
<<C++Operation>>+myOperation() : void{C++ThrownExceptions = none}

Figure 60 -- Operation Example in Class Diagram

The Model that is being shown in the figure below is a translation.

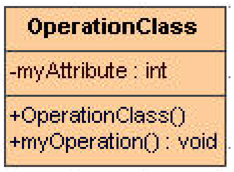
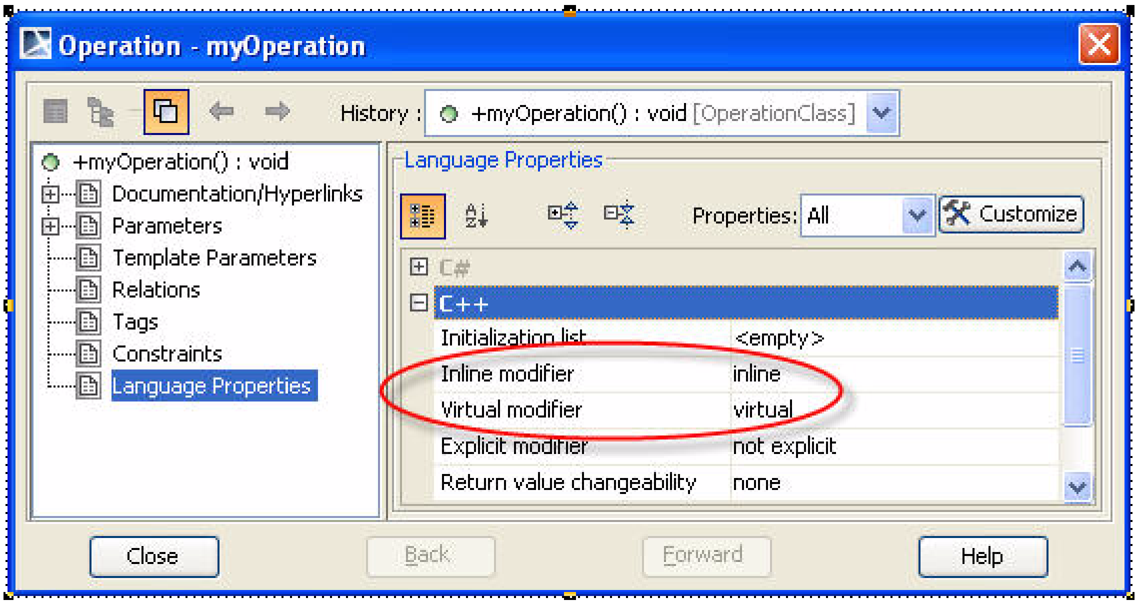


Figure 61 -- Translated Operation in Class Diagram

There are **Inline modifier** and **Virtual modifier** in Operation Language Properties that need to be translated and apply the **«C++Operation»** stereotype.



Initialization list	<empty>
Inline modifier	inline
Virtual modifier	virtual
Explicit modifier	not explicit
Return value changeability	none

Figure 62 -- Operation Language Properties (Operation)

There are **Initialization list** and **Explicit modifier** in Operation Language Properties that need to be translated and apply the **«C++Constructor»** stereotype.

Copyright © 1998-2015 No Magic, Inc.

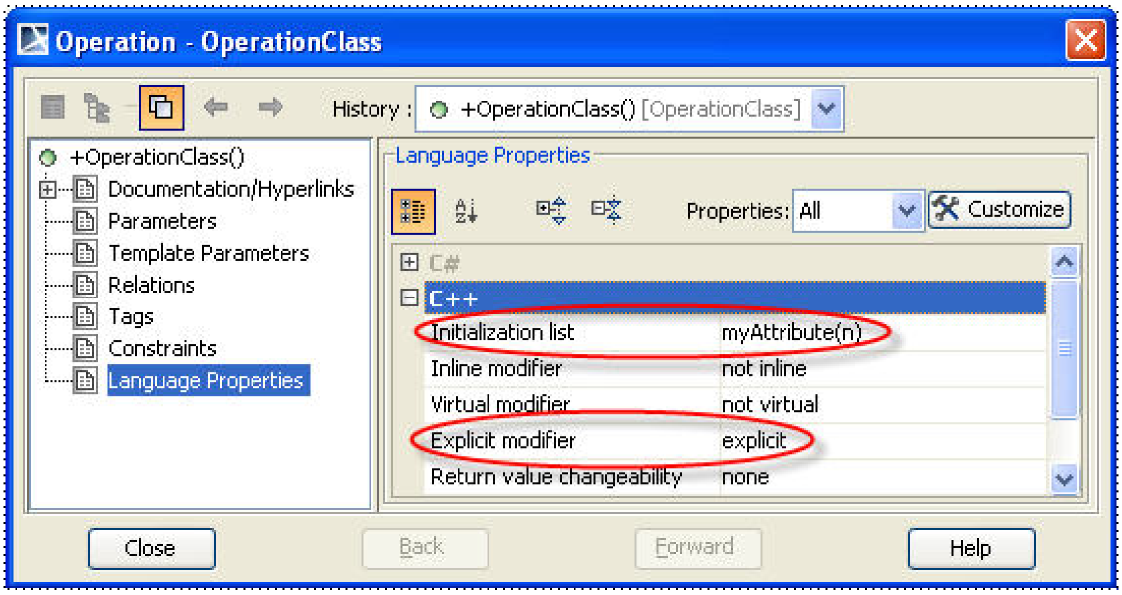


Figure 63 -- Operation Language Properties (Constructor)

Note: Initialization list and Explicit modifier will be translated when it was set in Constructor. The Constructor is an operation that has the same name as its owner or applies the «constructor» stereotype in UML Standard Profile.

Operation - Initialization List

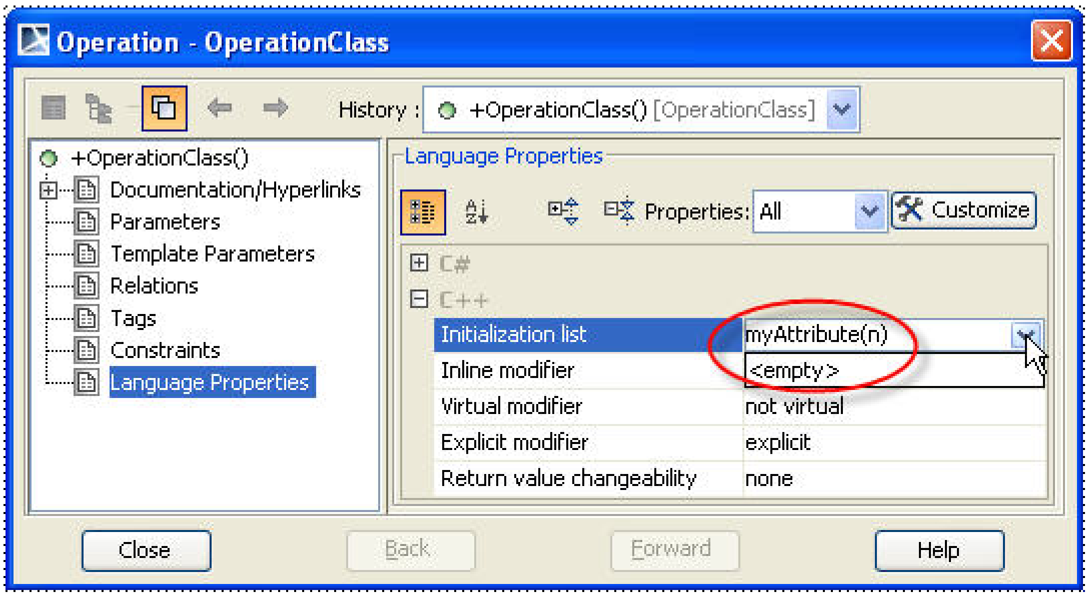
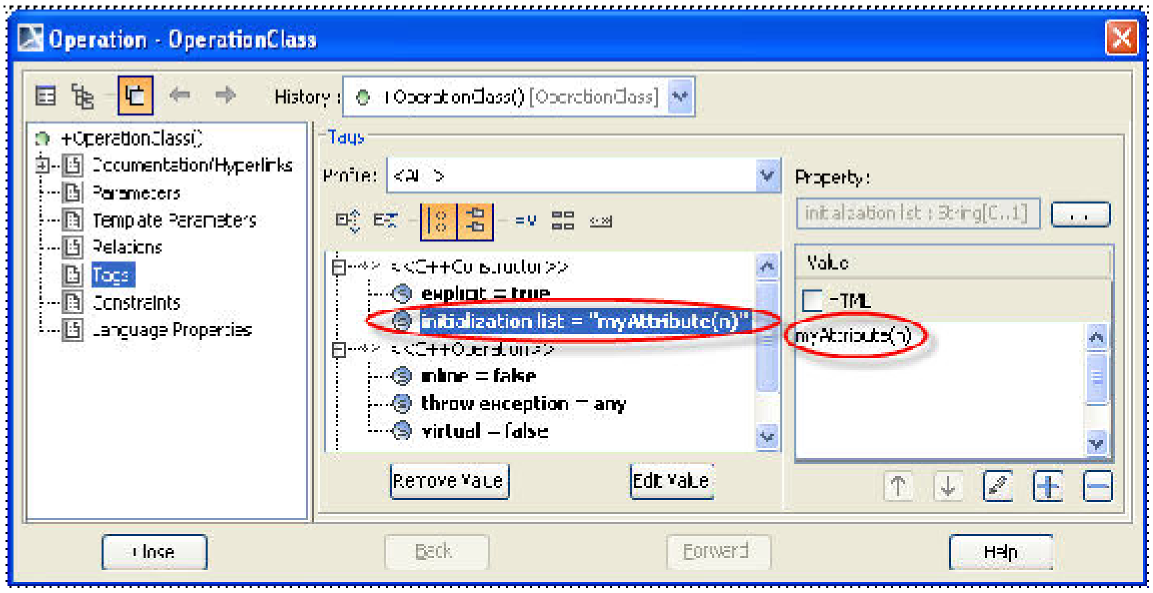


Figure 64 -- Operation Initialization list

Old value	**Translation**
`<empty>`	no change.
`myAttribute(n)`	Apply the **«C++Constructor»** stereotype and set **initialization list** tag value to **myAttribute(n)**.

Copyright © 1998-2015 No Magic, Inc.



Operation - Inline Modifier

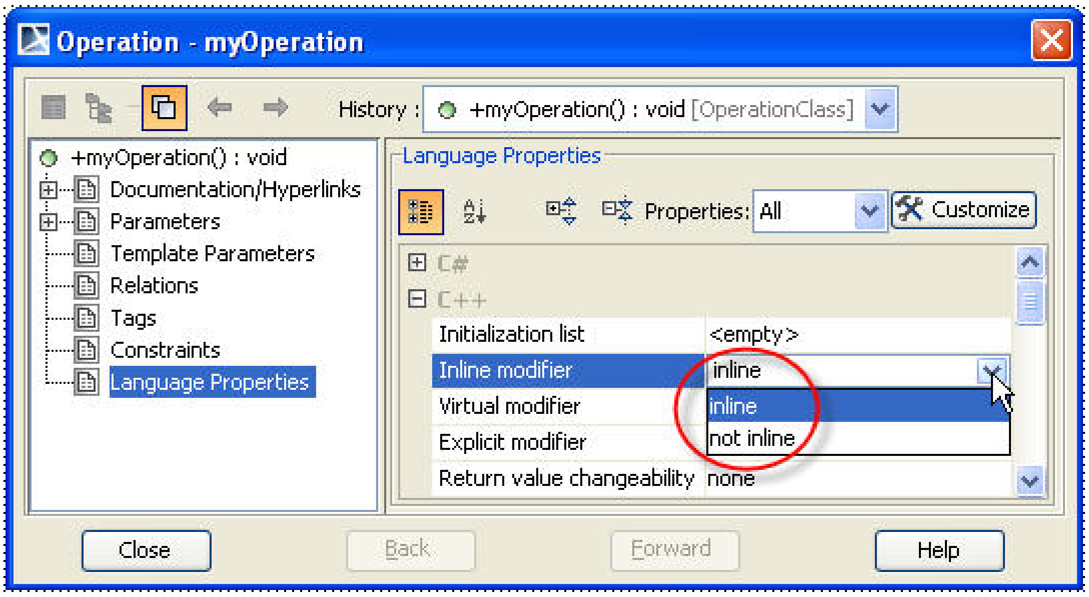
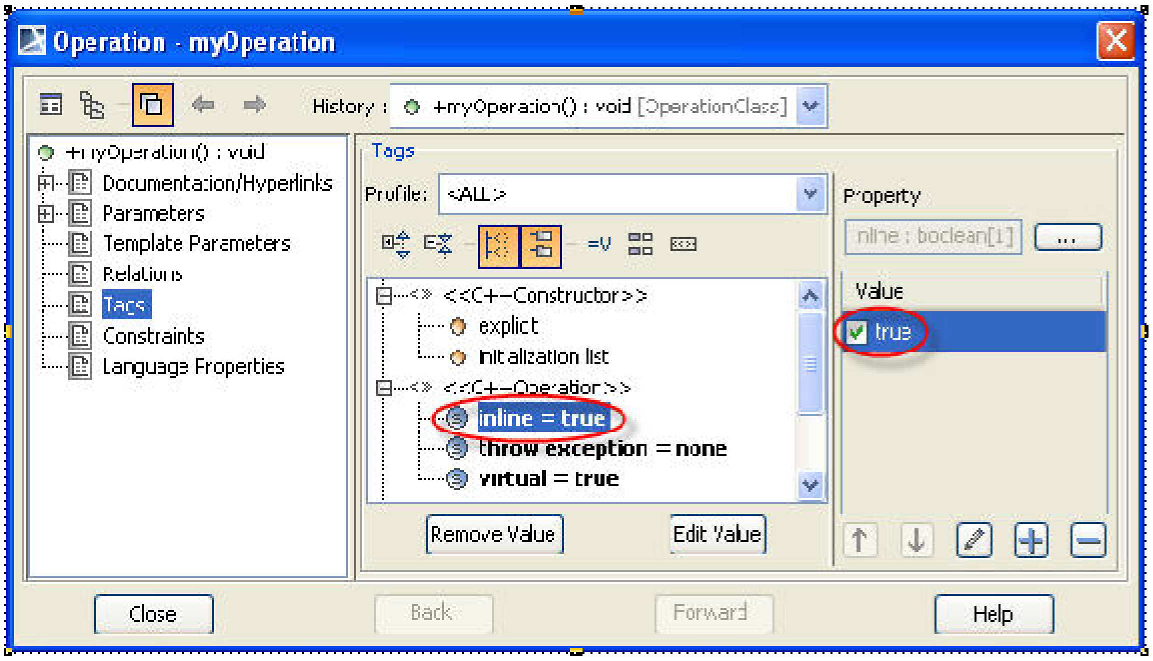


Figure 65 -- Operation Inline modifier

Old value	**Translation**
`not inline`	Apply the **«C++Operation»** stereotype and set **inline** tag value to **false**.
`inline`	Apply the **«C++Operation»** stereotype and set **inline** tag value to **true**.

Copyright © 1998-2015 No Magic, Inc.



Operation - Virtual Modifier

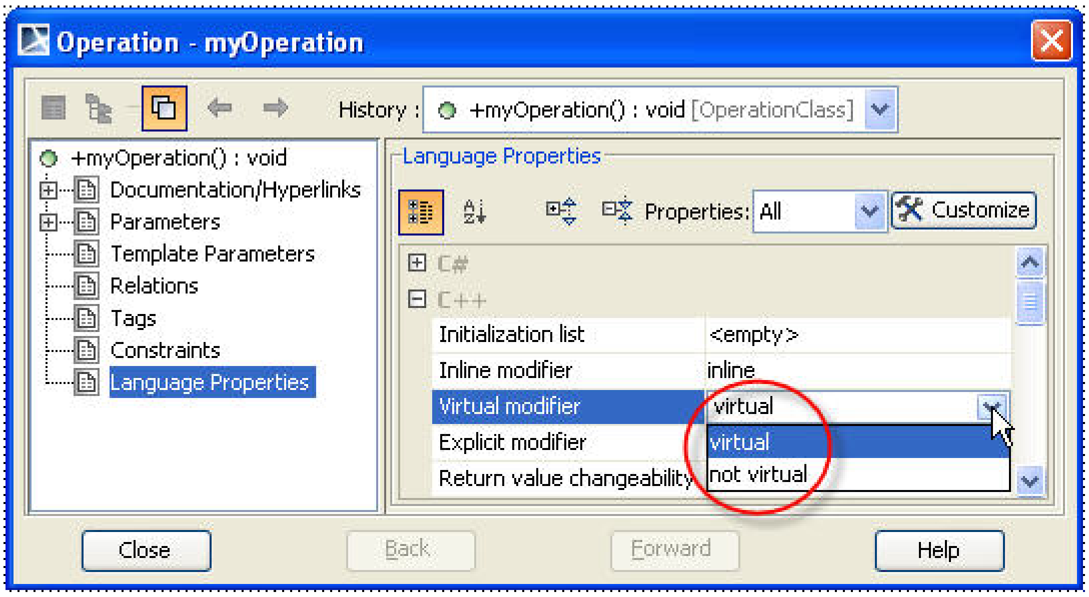
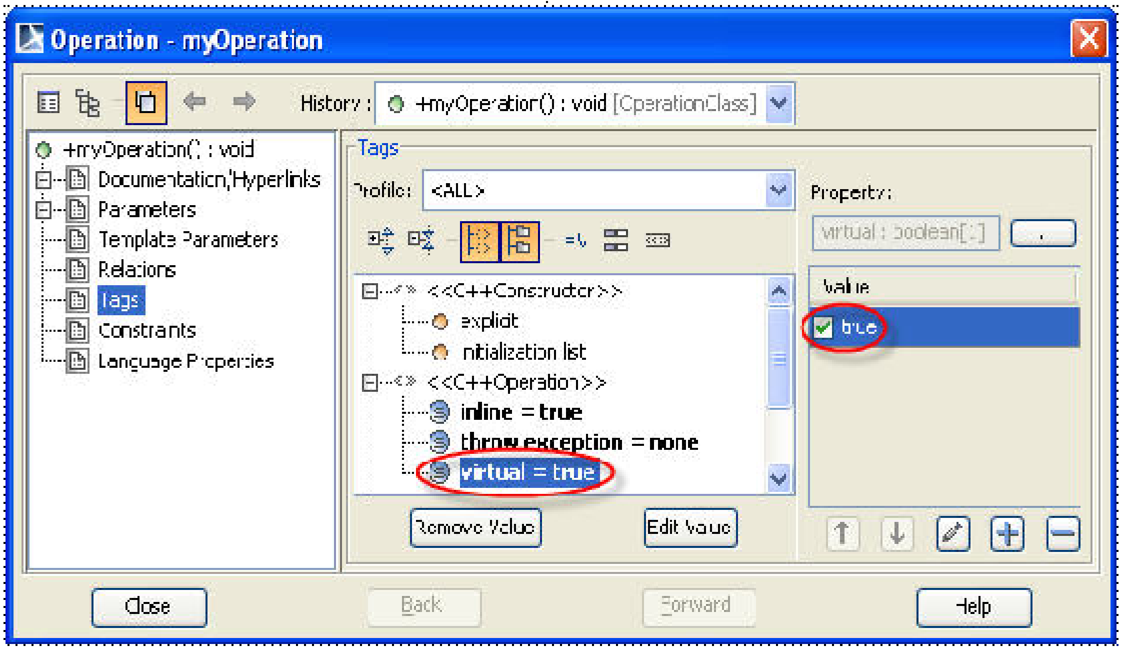


Figure 66 -- Operation Virtual modifier

Old value	Translation
`not virtual`	Apply the **«C++Operation»** stereotype and set **virtual** tag value to **false**.
`virtual`	Apply the **«C++Operation»** stereotype and set **virtual** tag value to **true**.

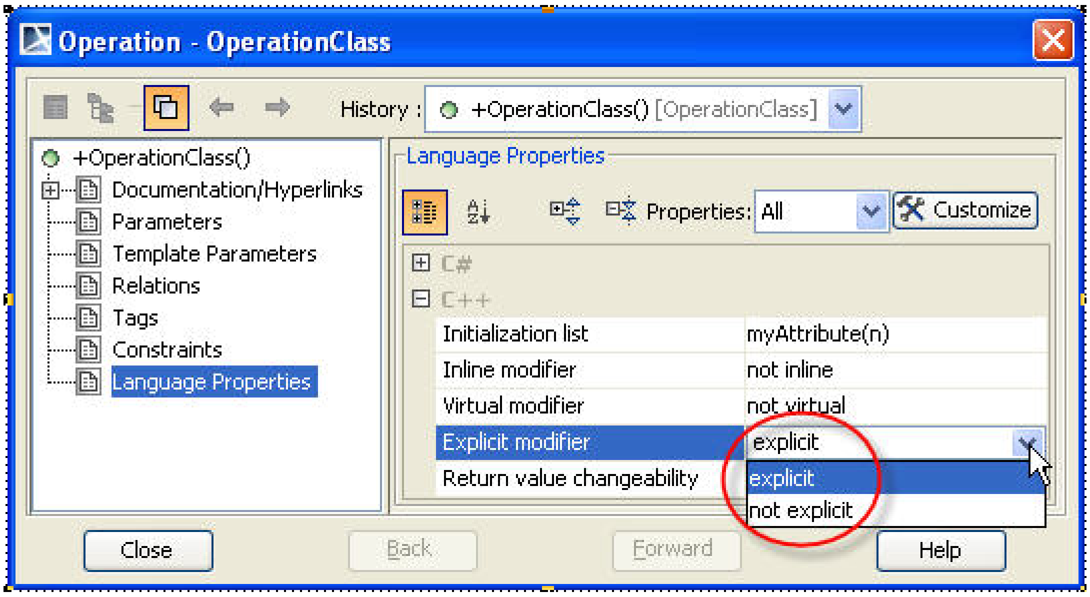
Copyright © 1998-2015 No Magic, Inc.



<<C++Constructor>>
explicit
initialization list
<<C++Operation>>
inline = true
throw exception = none
virtual = true

Value: true

Operation - Explicit Modifier



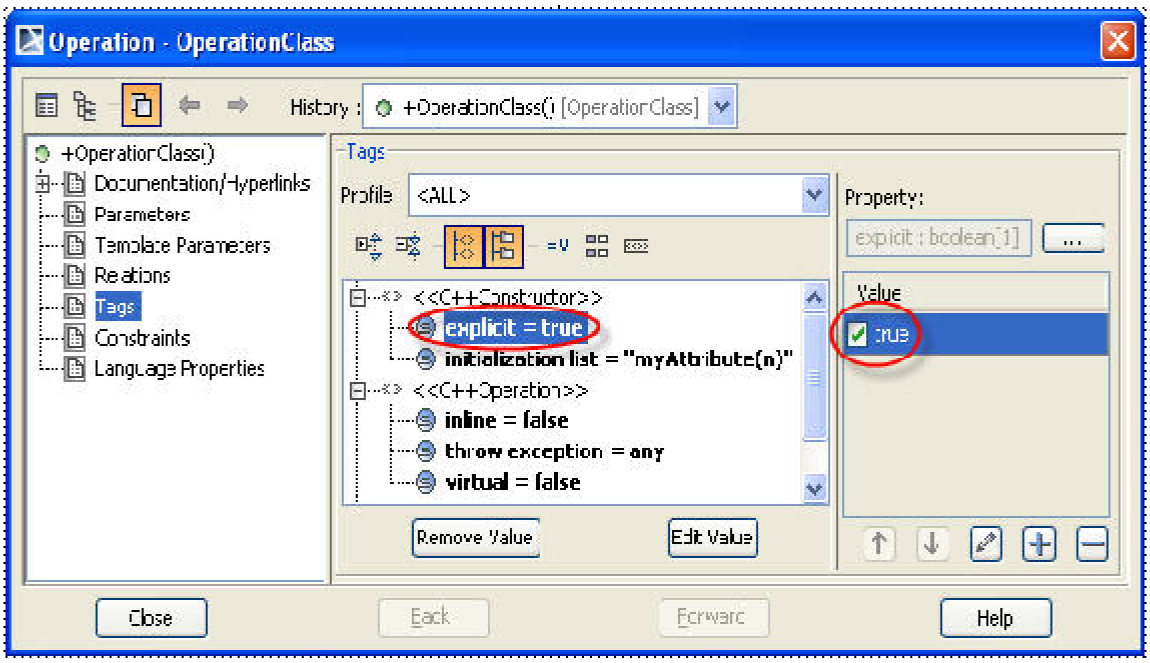
Initialization list	myAttribute(n)
Inline modifier	not inline
Virtual modifier	not virtual
Explicit modifier	explicit
Return value changeability	

explicit
not explicit

Figure 67 -- Operation Explicit modifier

Old value	**Translation**
`not explicit`	Apply the **«C++Constructor»** stereotype and set **explicit** tag value to **false**.
`explicit`	Apply the **«C++Constructor»** stereotype and set **explicit** tag value to **true**.

Copyright © 1998-2015 No Magic, Inc.



<<C++Constructor>>
explicit = true
initialization list = "myAttribute(n)"
<<C++Operation>>
inline = false
throw exception = any
virtual = false

Operation - Return value changeability

This Example is **ReturnValueChangeabilityClass** class that has three operations and all operations have set **return value changeability** in Language Properties to **const** as below.

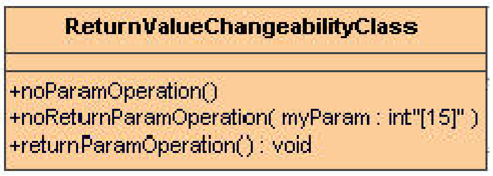

ReturnValueChangeabilityClass
+noParamOperation()
+noReturnParamOperation(myParam : int"[15]")
+returnParamOperation() : void

Figure 68 -- Return value changeability Example in Class Diagram

The Model that is being shown in the figure below is a translation.

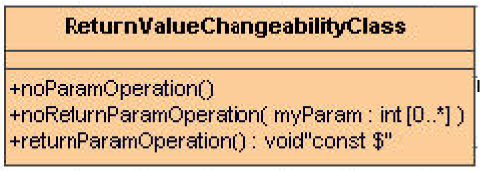
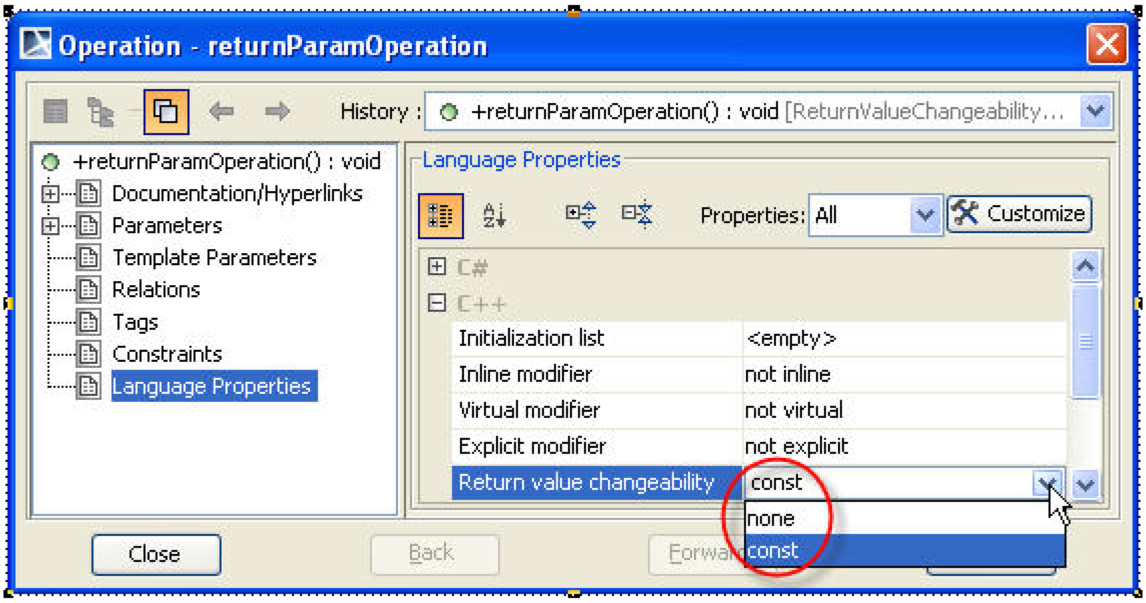

ReturnValueChangeabilityClass
+noParamOperation()
+noReturnParamOperation(myParam : int [0..*])
+returnParamOperation() : void"const $"

Figure 69 -- Translated Return value changeability in Class Diagram

The Return value changeability Language Properties is being shown in the figure below.

Copyright © 1998-2015 No Magic, Inc.


Operation - returnParamOperation

History : +returnParamOperation() : void [ReturnValueChangeability…

+returnParamOperation() : void
Documentation/Hyperlinks
Parameters
Template Parameters
Relations
Tags
Constraints
Language Properties

Language Properties

Properties: All

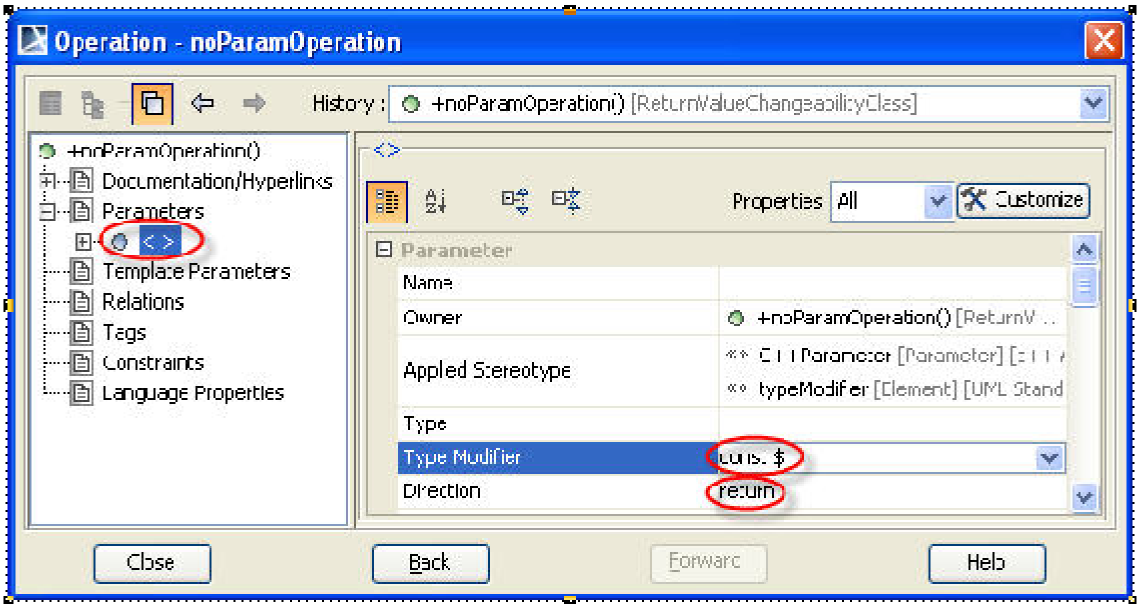
C#
C++

Initialization list	<empty>
Inline modifier	not inline
Virtual modifier	not virtual
Explicit modifier	not explicit
Return value changeability	const

none
const

Figure 70 -- Operation Return value changeability

Old value	Translation
`none`	no change.
`const``, operation doesn't has parameter.`	Create one return type parameter and set **Type Modifier** to **const $**.


Operation - noParamOperation

History : +noParamOperation() [ReturnValueChangeabilityClass]

+noParamOperation()
Documentation/Hyperlinks
Parameters
<>
Template Parameters
Relations
Tags
Constraints
Language Properties

<>

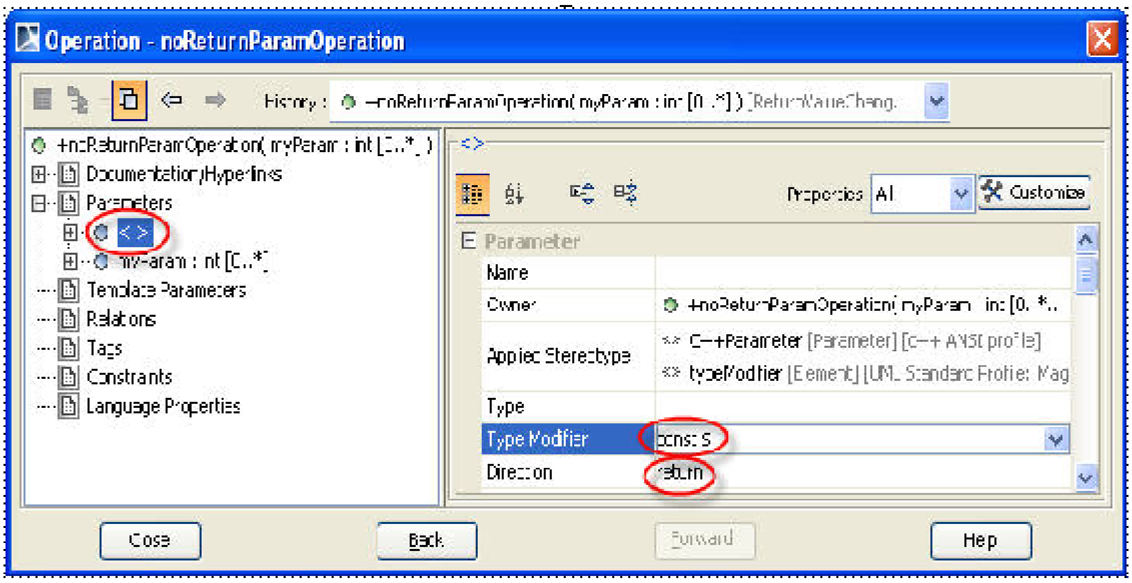
Properties: All

Parameter

Name	
Owner	+noParamOperation() [ReturnV…
Applied Stereotype	«» C++Parameter [Parameter] [C++ A… «» typeModifier [Element] [UML Stand…
Type	
Type Modifier	const $
Direction	return

`const``, operation has parameter but does not have return type parameter.`	Create one return type parameter and set **Type Modifier** to **const $**.

Copyright © 1998-2015 No Magic, Inc.



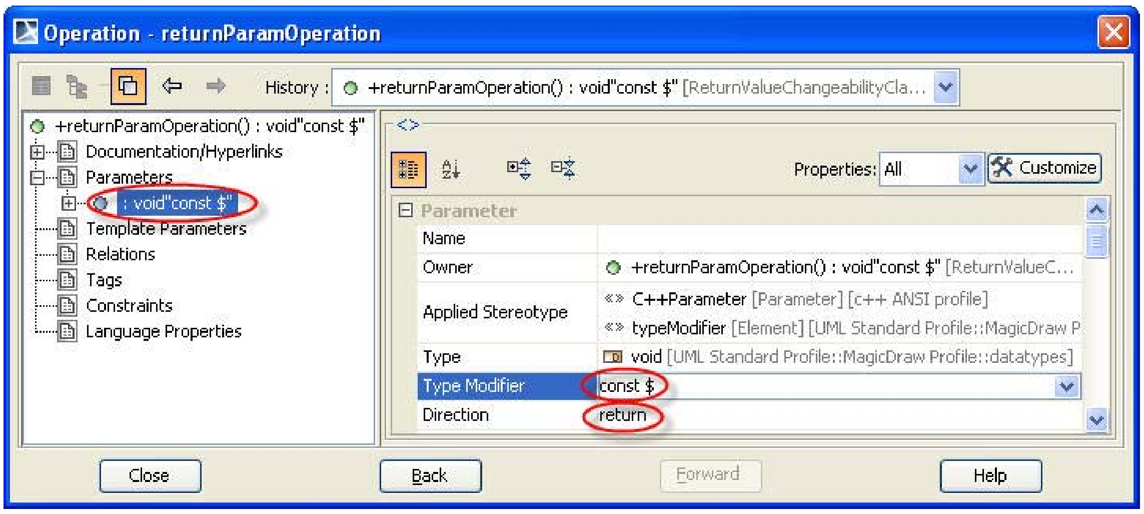
+noReturnParamOperation(myParam : int [0..*]) [ReturnValueChang...

- +noReturnParamOperation(myParam : int [0..*])
  - Documentation/Hyperlinks
  - Parameters
    - <>
    - myParam : int [0..*]
  - Template Parameters
  - Relations
  - Tags
  - Constraints
  - Language Properties

<>

Parameter	
Name	
Owner	+noReturnParamOperation(myParam : int [0..*..
Applied Stereotype	«» C++Parameter [Parameter] [c++ ANSI profile] «» typeModifier [Element] [UML Standard Profile::Mag
Type	
Type Modifier	const $
Direction	return

`const, operation has return type parameter.`	Set **Type Modifier** in return type parameter to **const $**.



+returnParamOperation() : void"const $" [ReturnValueChangeabilityCla...

- +returnParamOperation() : void"const $"
  - Documentation/Hyperlinks
  - Parameters
    - : void"const $"
  - Template Parameters
  - Relations
  - Tags
  - Constraints
  - Language Properties

<>

Parameter	
Name	
Owner	+returnParamOperation() : void"const $" [ReturnValueC...
Applied Stereotype	«» C++Parameter [Parameter] [c++ ANSI profile] «» typeModifier [Element] [UML Standard Profile::MagicDraw P
Type	void [UML Standard Profile::MagicDraw Profile::datatypes]
Type Modifier	const $
Direction	return

Attribute

This example is **AttributeClass** class that has attribute named **myAttribute**, return type is **int** and type modifier is **[15]**.

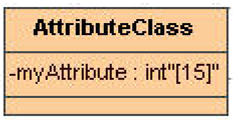


Figure 71 -- Attribute Example in Class Diagram

The Model that is being shown in the figure below is translation.

Copyright © 1998-2015 No Magic, Inc.

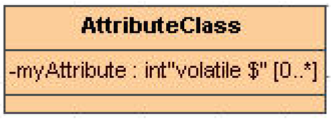
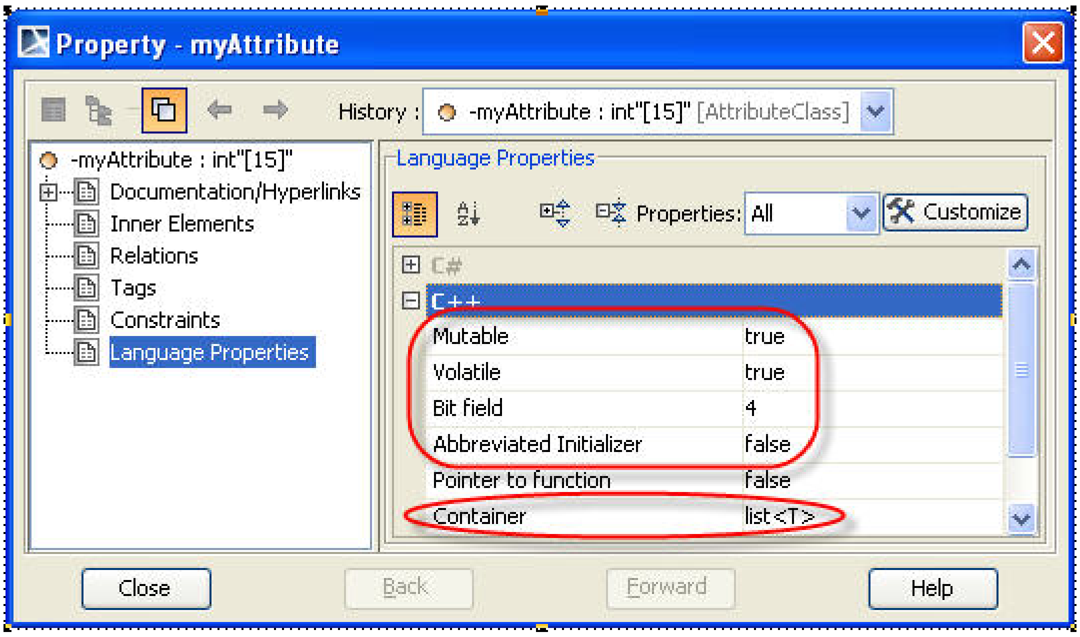


Figure 72 -- Translated Attribute in Class Diagram

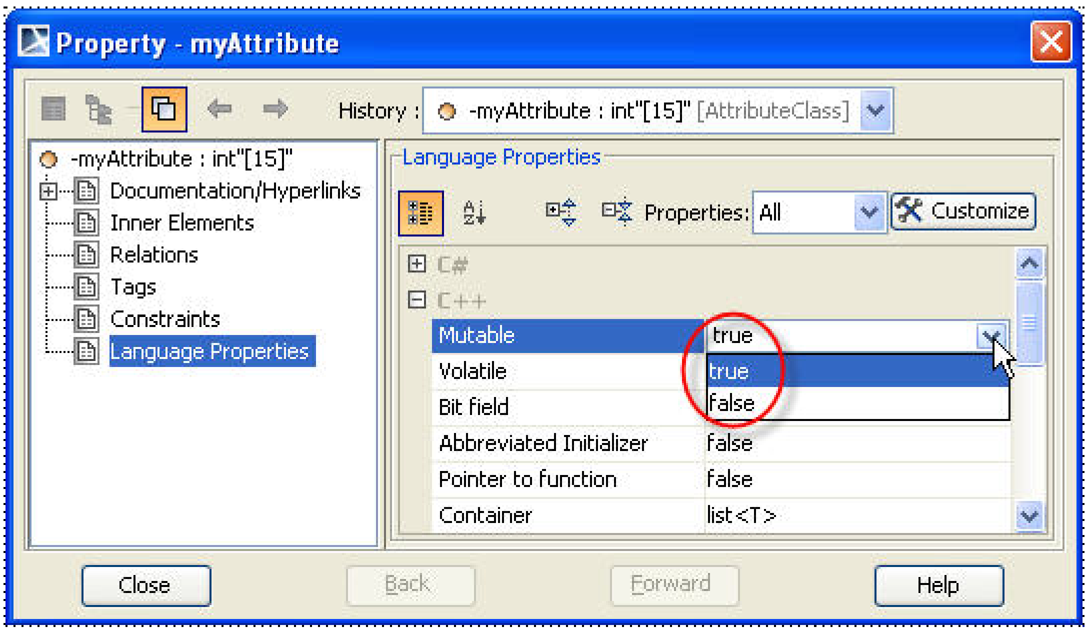
There are **Mutable**, **Bit field**, **Abbreviated Initializer** and **Container** in Attribute Language Properties that need to be translated and apply the **«C++Attribute»** stereotype. A **Volatile** in Attribute Language Properties will move to **Type Modifier**.



Mutable	true
Volatile	true
Bit field	4
Abbreviated Initializer	false
Pointer to function	false
Container	list<T>

Figure 73 -- Attribute Language Properties

Attribute - Mutable

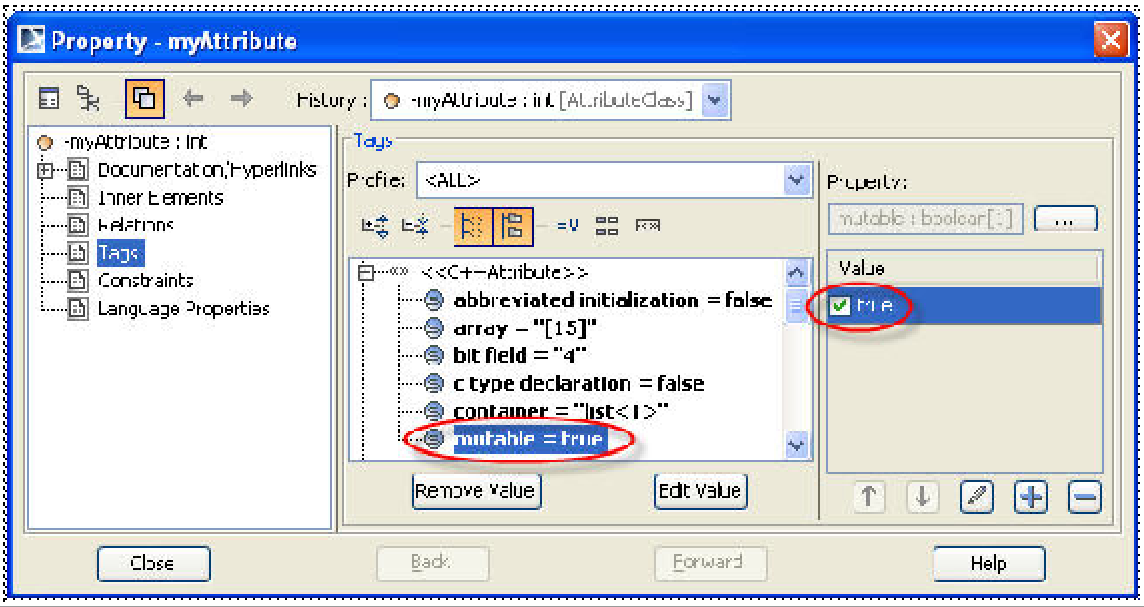


Mutable	true
Volatile	
Bit field	
Abbreviated Initializer	false
Pointer to function	false
Container	list<T>

Figure 74 -- Attribute Mutable

Copyright © 1998-2015 No Magic, Inc.

Old value	Translation
`false`	Apply the **«C++Attribute»** stereotype and set **mutable** tag value to **false**.
`true`	Apply the **«C++Attribute»** stereotype and set **mutable** tag value to **true**.


Property - myAttribute

History : -myAttribute : int [AttributeClass]

- -myAttribute : int
  - Documentation/Hyperlinks
  - Inner Elements
  - Relations
  - Tags
  - Constraints
  - Language Properties

Tags

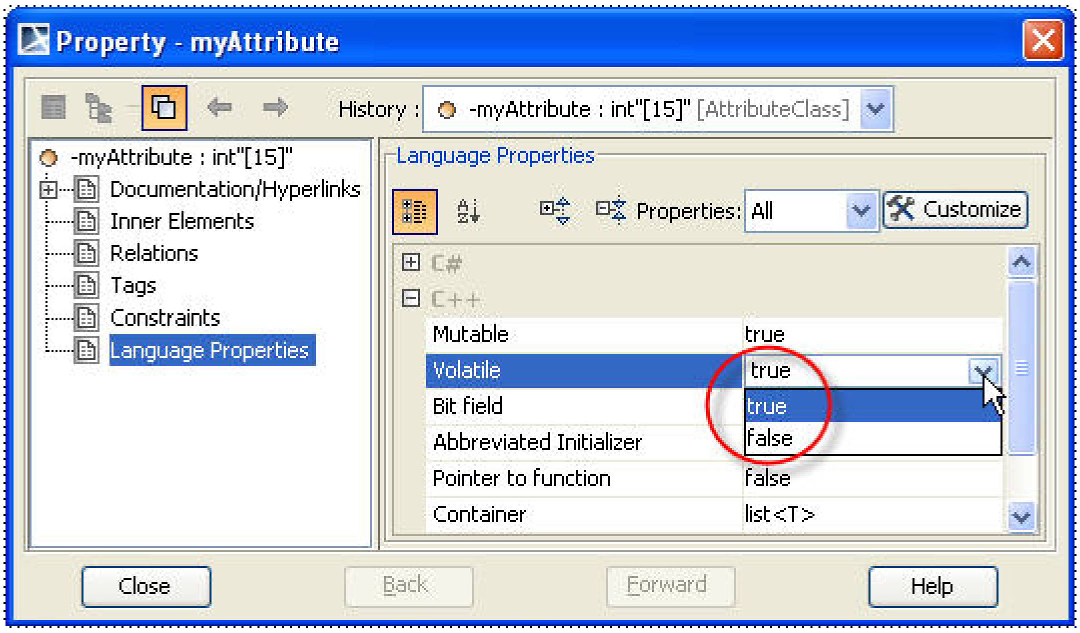
Profile: <ALL>

- <<C++Attribute>>
  - abbreviated initialization = false
  - array = "[15]"
  - bit field = "4"
  - c type declaration = false
  - container = "list<T>"
  - mutable = true

Property: mutable : boolean[1]

Value: ✓ true

Attribute - Volatile


Property - myAttribute

History : -myAttribute : int"[15]" [AttributeClass]

- -myAttribute : int"[15]"
  - Documentation/Hyperlinks
  - Inner Elements
  - Relations
  - Tags
  - Constraints
  - Language Properties

Language Properties

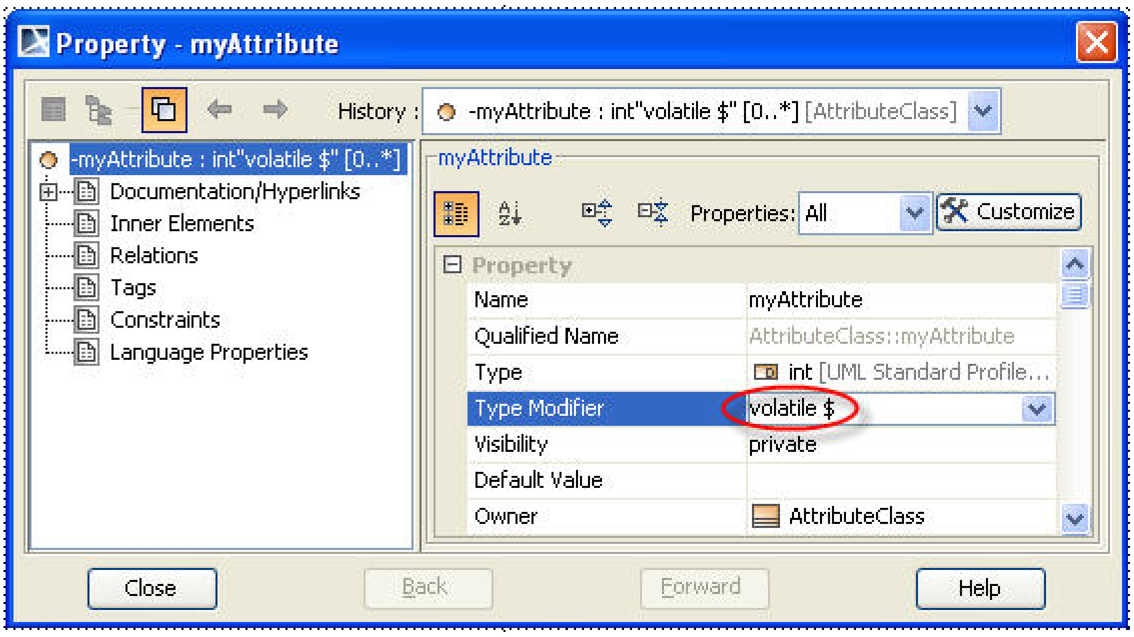
Properties: All

C#	
C++	
Mutable	true
Volatile	true
Bit field	true / false
Abbreviated Initializer	
Pointer to function	false
Container	list<T>

Figure 75 -- Attribute Volatile

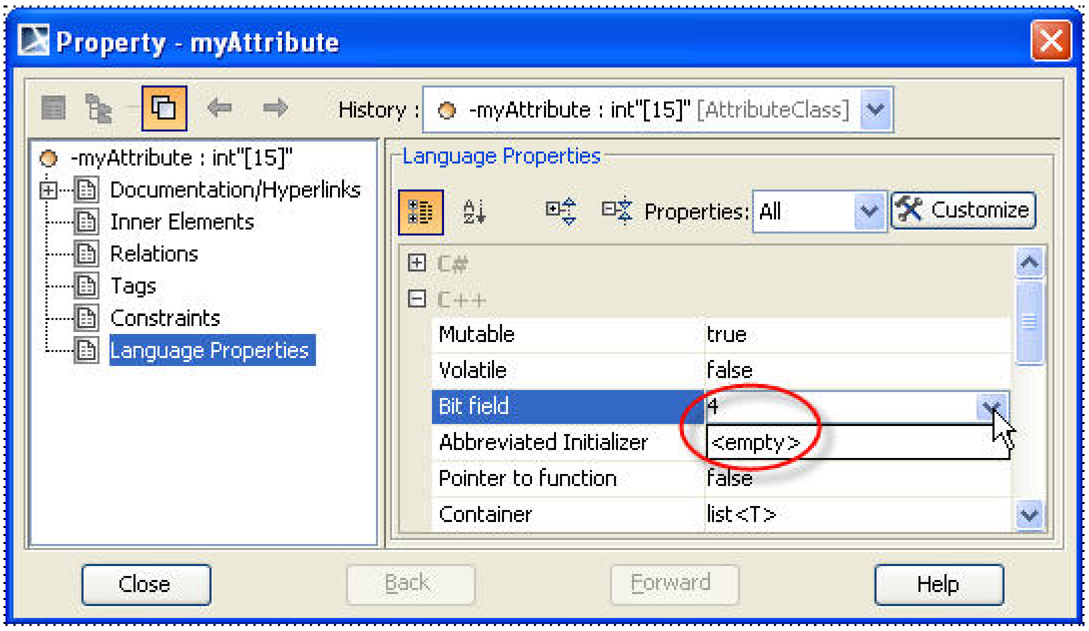
Old value	Translation
`false`	no change.
`true`	Set **Type Modifier** to **volatile $**.

Copyright © 1998-2015 No Magic, Inc.



Name	myAttribute
Qualified Name	AttributeClass::myAttribute
Type	int [UML Standard Profile…
Type Modifier	volatile $
Visibility	private
Default Value	
Owner	AttributeClass

Attribute -Bit field

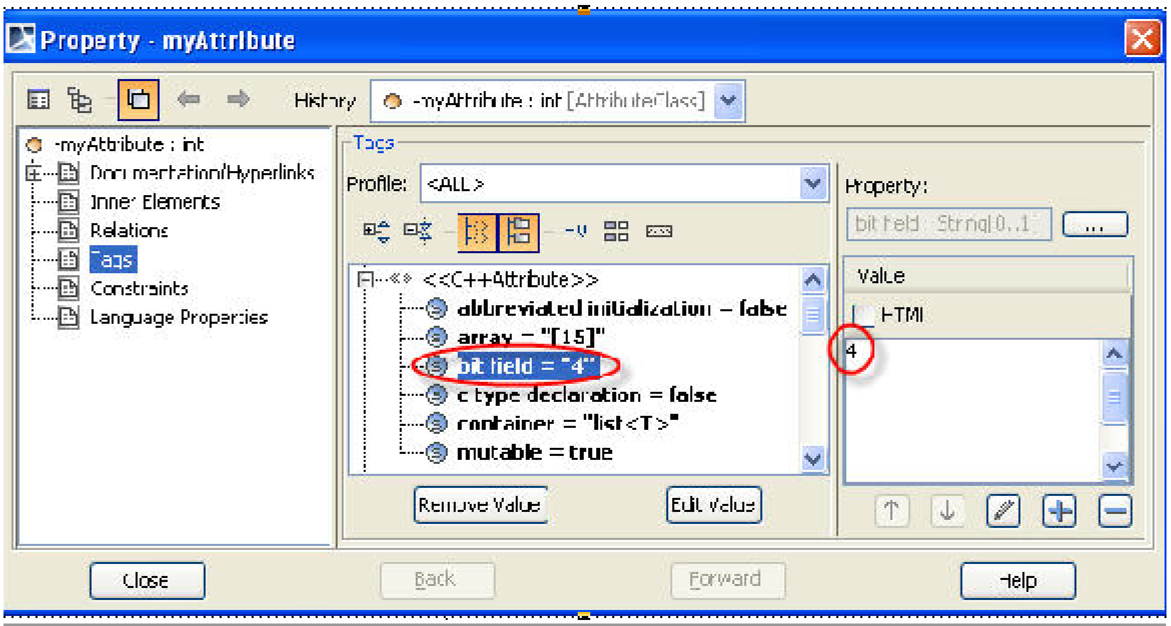


Mutable	true
Volatile	false
Bit field	4
Abbreviated Initializer	
Pointer to function	false
Container	list<T>

Figure 76 -- Attribute Bit field

Old value	Translation
`<empty>`	no change.
`4`	Apply the **«C++Attribute»** stereotype and set **bit field** tag value to **4**.

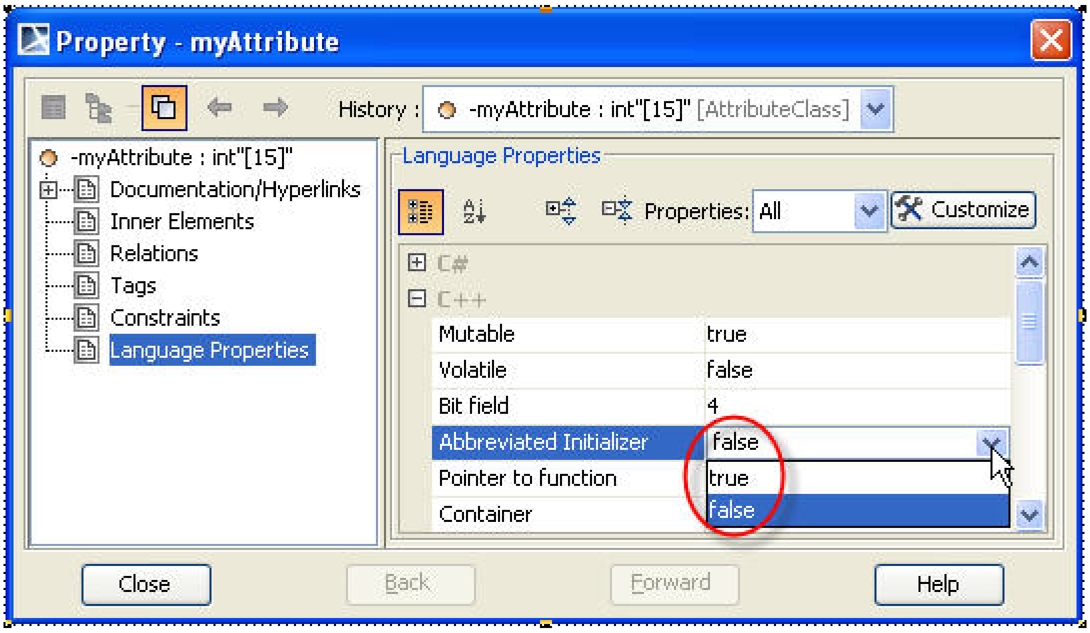
Copyright © 1998-2015 No Magic, Inc.



<<C++Attribute>>
- abbreviated initialization = false
- array = "[15]"
- bit field = "4"
- c type declaration = false
- container = "list<T>"
- mutable = true

Value: 4

Attribute -Abbreviated initializer

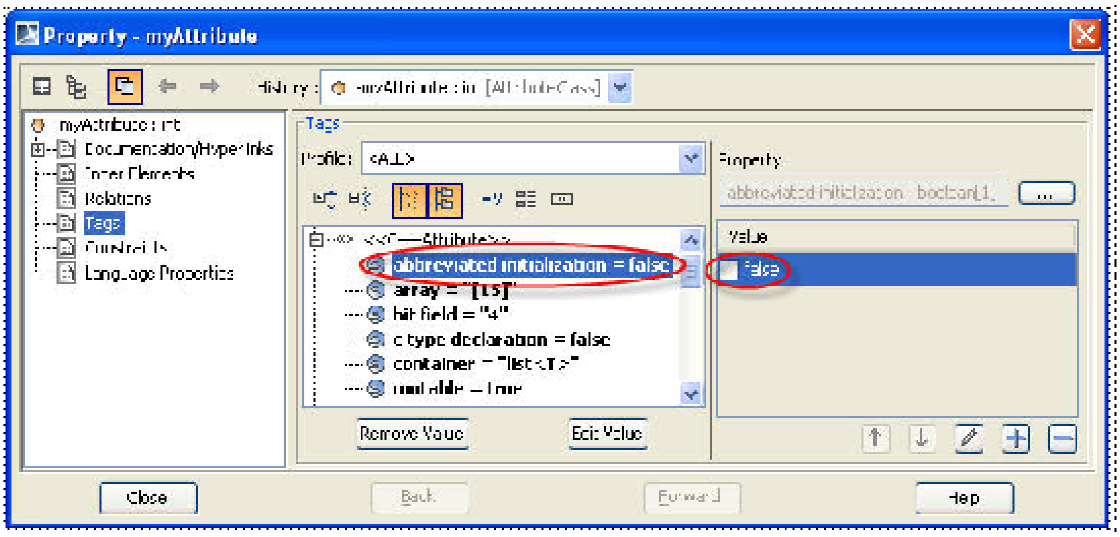


Mutable	true
Volatile	false
Bit field	4
Abbreviated Initializer	false
Pointer to function	true
Container	false

Figure 77 -- Attribute Abbreviated Initializer

Old value	Translation
`false`	Apply the **«C++Attribute»** stereotype and set **abbreviated initialization** tag value to **false**.
`true`	Apply the **«C++Attribute»** stereotype and set **abbreviated initialization** tag value to **true**.

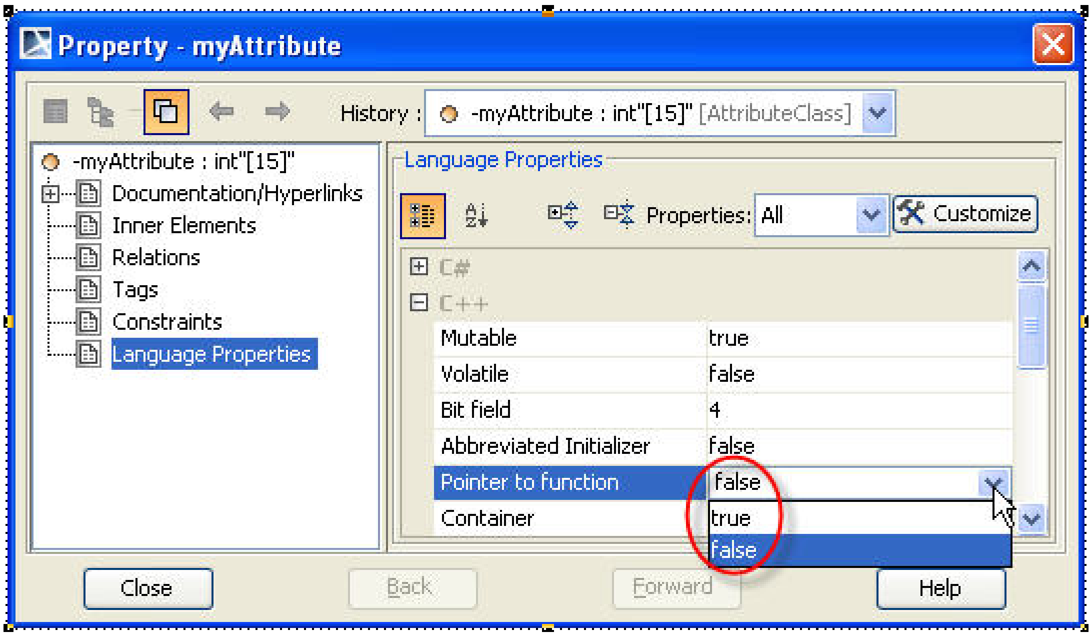
Copyright © 1998-2015 No Magic, Inc.



<<C++Attribute>>
abbreviated initialization = false
array = "[15]"
bit field = "4"
c type declaration = false
container = "list<T>"
mutable = true

Value: false

Attribute - Pointer to function



Mutable	true
Volatile	false
Bit field	4
Abbreviated Initializer	false
Pointer to function	false
Container	

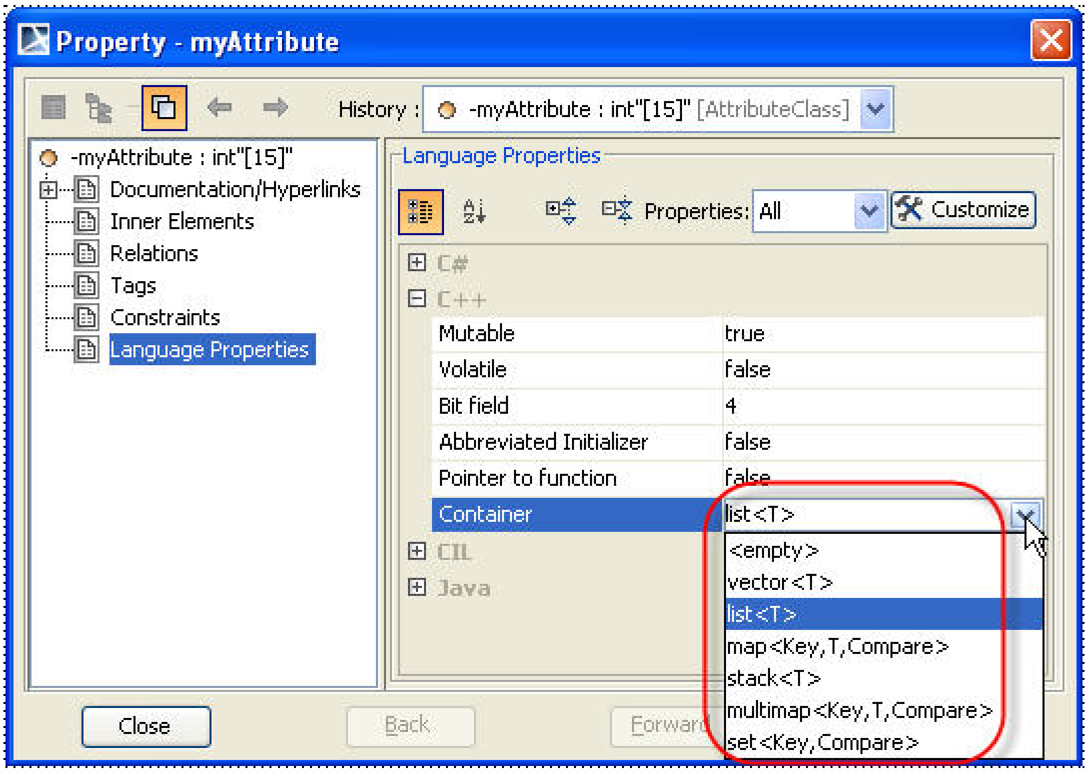
Dropdown: true / false

Figure 78 -- Attribute Pointer to function

Old value	Translation
`false`	no change.
`true`	no change.

Copyright © 1998-2015 No Magic, Inc.

Attribute - Container



Language Properties

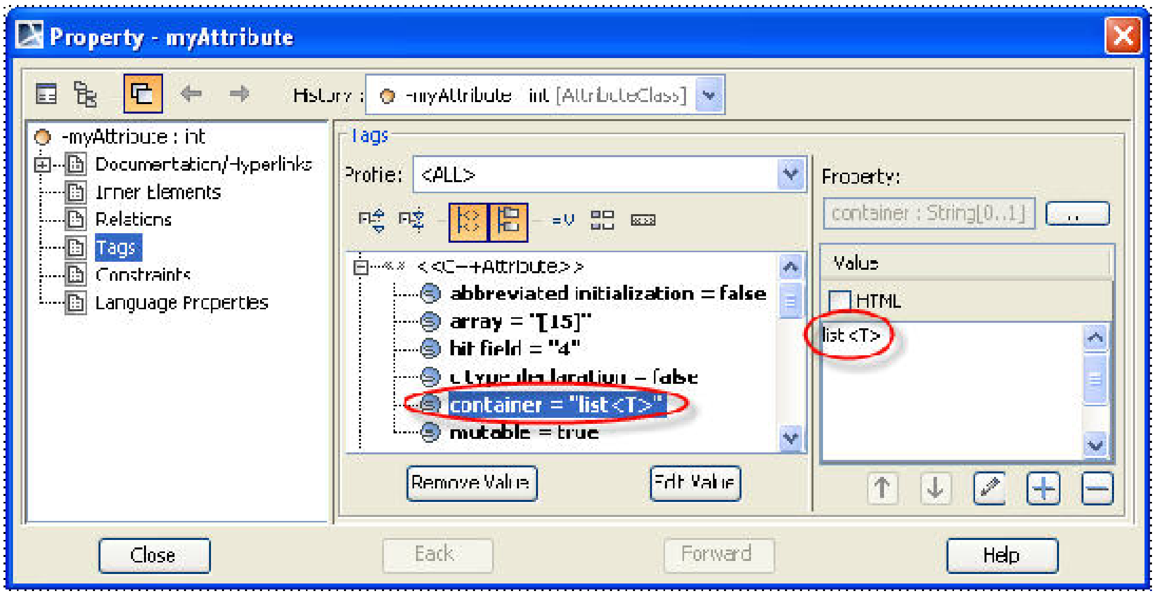
Mutable	true
Volatile	false
Bit field	4
Abbreviated Initializer	false
Pointer to function	false
Container	list<T>

<empty>
vector<T>
list<T>
map<Key,T,Compare>
stack<T>
multimap<Key,T,Compare>
set<Key,Compare>

Figure 79 -- Attribute Container

Old value	**Translation**
`<empty>`	no change.
`vector<T>`	Apply the **«C++Attribute»** stereotype and set **container** tag value to **vector<T>**.
`list<T>`	Apply the **«C++Attribute»** stereotype and set **container** tag value to **list<T>**.
`map<Key, T, Compare>`	Apply the **«C++Attribute»** stereotype and set **container** tag value to **map<Key, T, Compare>**.
`stack<T>`	Apply the **«C++Attribute»** stereotype and set **container** tag value to **stack<T>**.
`multimap<Key, T, Compare>`	Apply the **«C++Attribute»** stereotype and set **container** tag value to **multimap<Key, T, Compare>**.
`set<Key, Compare>`	Apply the **«C++Attribute»** stereotype and set **container** tag value to **set<Key, Compare>**.

Copyright © 1998-2015 No Magic, Inc.


Property - myAttribute

History : -myAttribute : int [AttributeClass]

- -myAttribute : int
  - Documentation/Hyperlinks
  - Inner Elements
  - Relations
  - Tags
  - Constraints
  - Language Properties

Tags

Profile: <ALL>

- <<C++Attribute>>
  - abbreviated initialization = false
  - array = "[15]"
  - bit field = "4"
  - c type declaration = false
  - container = "list<T>"
  - mutable = true

Property: container : String[0..1]

Value

HTML

list<T>

Generalization

These examples are **ParentClass** class and **childClass** class that extends from **ParentClass** class.

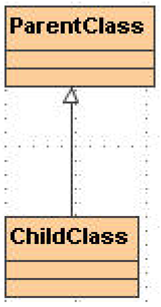
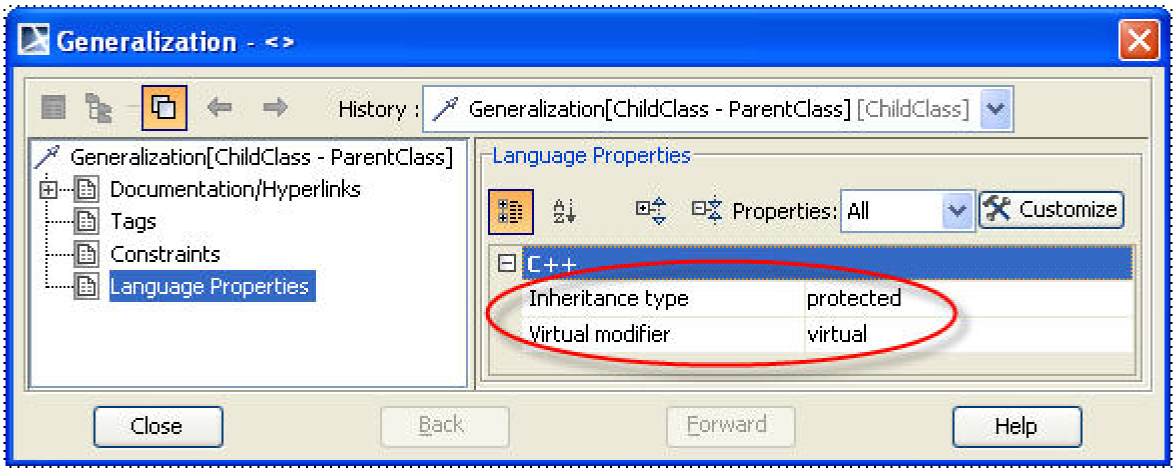


Figure 80 -- Generalization Example in Class Diagram

There are **Inheritance type** and **Virtual modifier** in Generalization Language Properties that need to be translated and apply the **«C++Generalization»** stereotype.


Generalization - <>

History : Generalization[ChildClass - ParentClass] [ChildClass]

- Generalization[ChildClass - ParentClass]
  - Documentation/Hyperlinks
  - Tags
  - Constraints
  - Language Properties

Language Properties

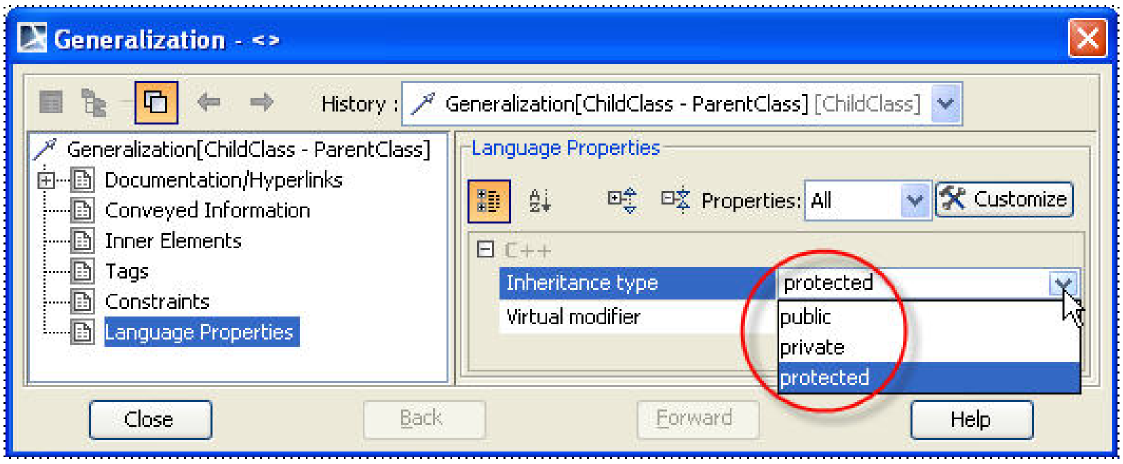
Properties: All

C++	
Inheritance type	protected
Virtual modifier	virtual

Figure 81 -- Generalization Language Properties

Copyright © 1998-2015 No Magic, Inc.

Generalization - Inheritance type

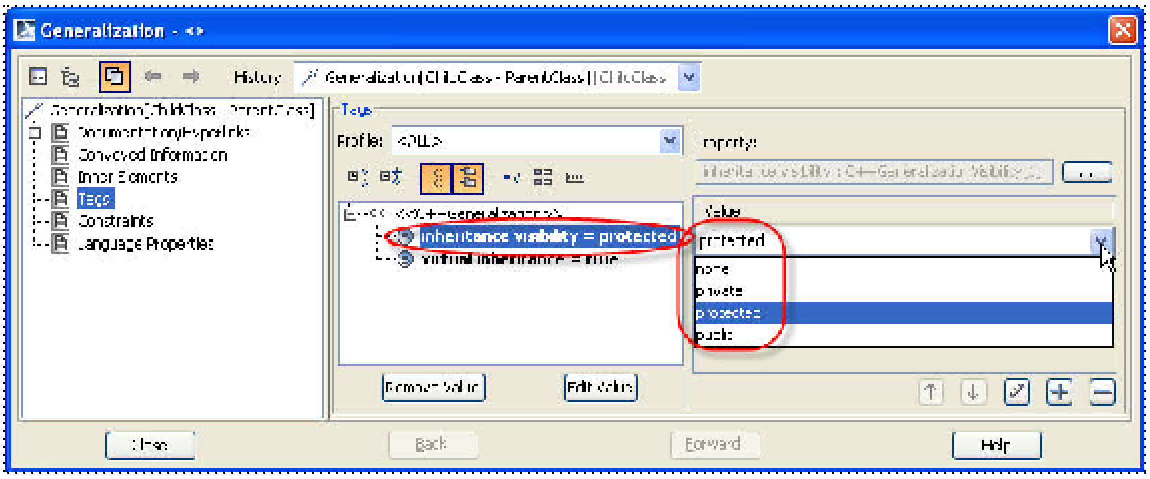


Language Properties

Inheritance type	protected
Virtual modifier	

Figure 82 -- Generalization Inheritance type

Old value	Translation
`public`	Apply the **«C++Generalization»** stereotype and set **inheritance visibility** tag value to **public**.
`protected`	Apply the **«C++Generalization»** stereotype and set **inheritance visibility** tag value to **protected**.
`private`	Apply the **«C++Generalization»** stereotype and set **inheritance visibility** tag value to **private**.



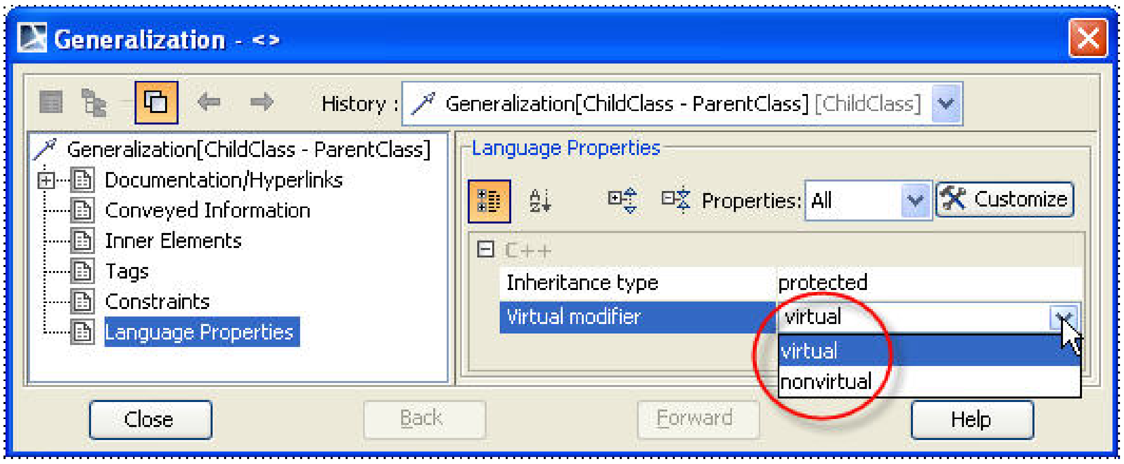
Tags

«C++Generalization»
inheritance visibility = protected
virtual inheritance = true

Value: protected

Copyright © 1998-2015 No Magic, Inc.

Generalization - Virtual modifier



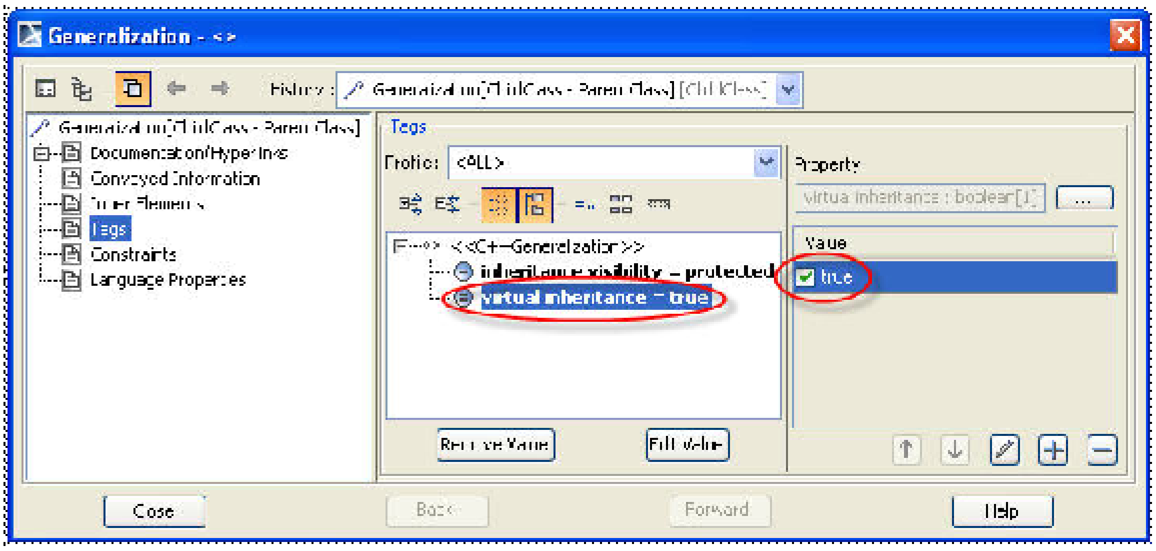
Language Properties

Inheritance type	protected
Virtual modifier	virtual

virtual
nonvirtual

Figure 83 -- Generalization Virtual modifier

Old value	Translation
`nonvirtual`	Apply the **«C++Generalization»** stereotype and set **virtual inheritance** tag value to **false**.
`virtual`	Apply the **«C++Generalization»** stereotype and set **virtual inheritance** tag value to **true**.



Tags

<<C++Generalization>>
inheritance visibility = protected
virtual inheritance = true

Property: virtual inheritance : boolean[1]

Value: ✓ true

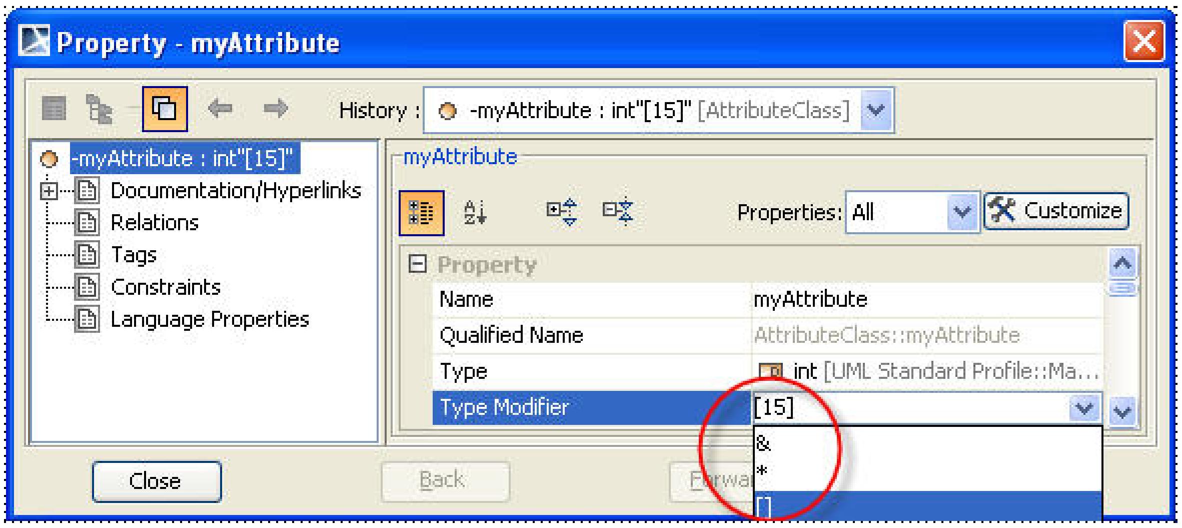
Type Modifiers

In version 12, type modifiers use the $ character to specify the type modifiers construct. This allows mapping of complex type modifiers. For example, `const int* const` is mapped to const $* const.

Array

The Attribute Type Modifier is being shown in the figure below.

Copyright © 1998-2015 No Magic, Inc.



Property - myAttribute

History : -myAttribute : int"[15]" [AttributeClass]

- -myAttribute : int"[15]"
  - Documentation/Hyperlinks
  - Relations
  - Tags
  - Constraints
  - Language Properties

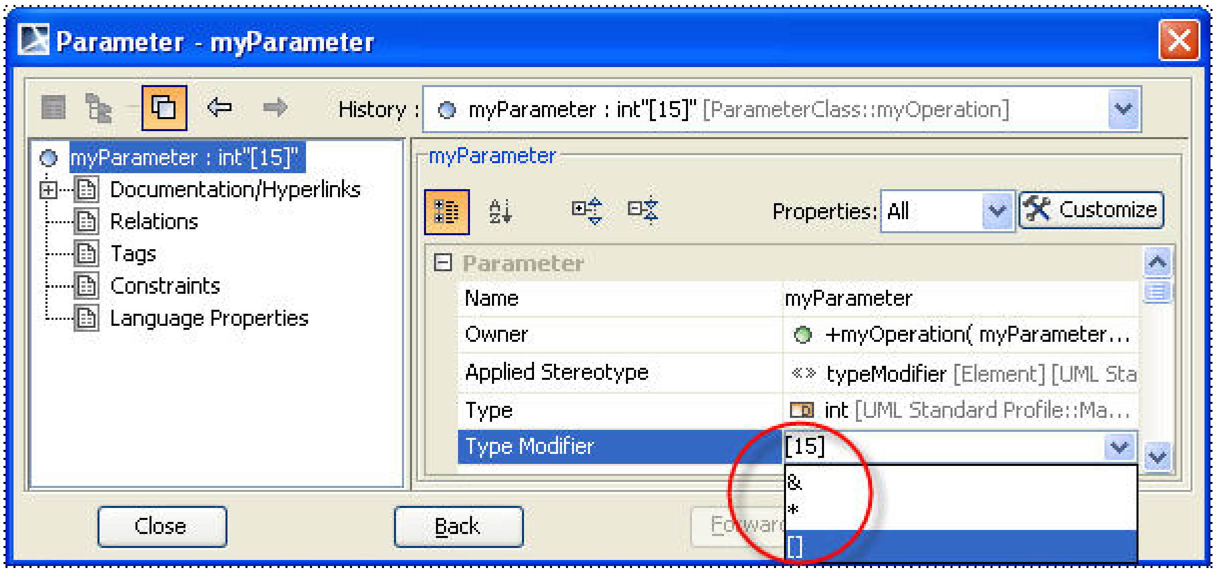
myAttribute

Properties: All

Property	
Name	myAttribute
Qualified Name	AttributeClass::myAttribute
Type	int [UML Standard Profile::Ma...
Type Modifier	[15]

Figure 84 -- Array type modifier in Attribute

The Parameter Type Modifier is being shown in the figure below.



Parameter - myParameter

History : myParameter : int"[15]" [ParameterClass::myOperation]

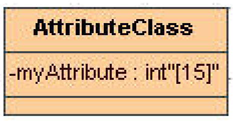
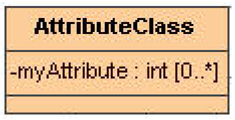
- myParameter : int"[15]"
  - Documentation/Hyperlinks
  - Relations
  - Tags
  - Constraints
  - Language Properties

myParameter

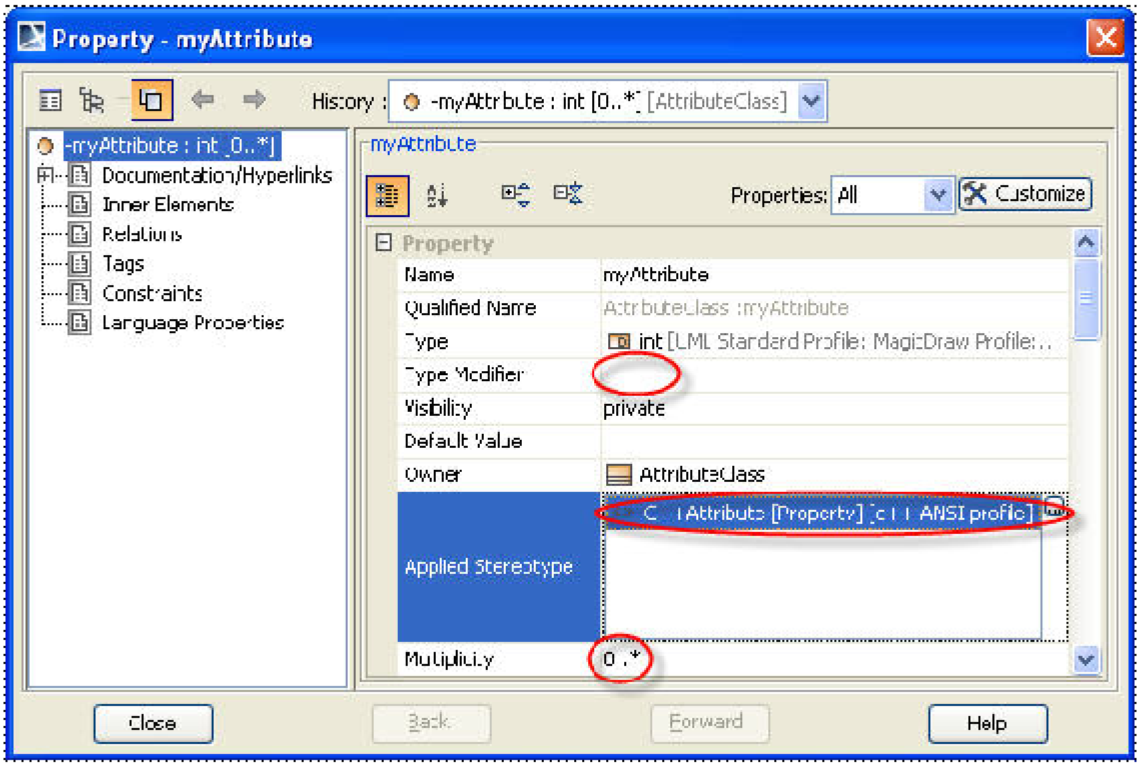
Properties: All

Parameter	
Name	myParameter
Owner	+myOperation(myParameter...
Applied Stereotype	«» typeModifier [Element] [UML Sta
Type	int [UML Standard Profile::Ma...
Type Modifier	[15]

Figure 85 -- Array type modifier in Parameter

Old value	Translation
	
`[15]`	Apply the **«C++Attribute»** stereotype and set **array** tag value to **[15]**. Set **Multiplicity** field to **0..***. Remove **[15]** from **type modifier** field.

Copyright © 1998-2015 No Magic, Inc.


Property - myAttribute

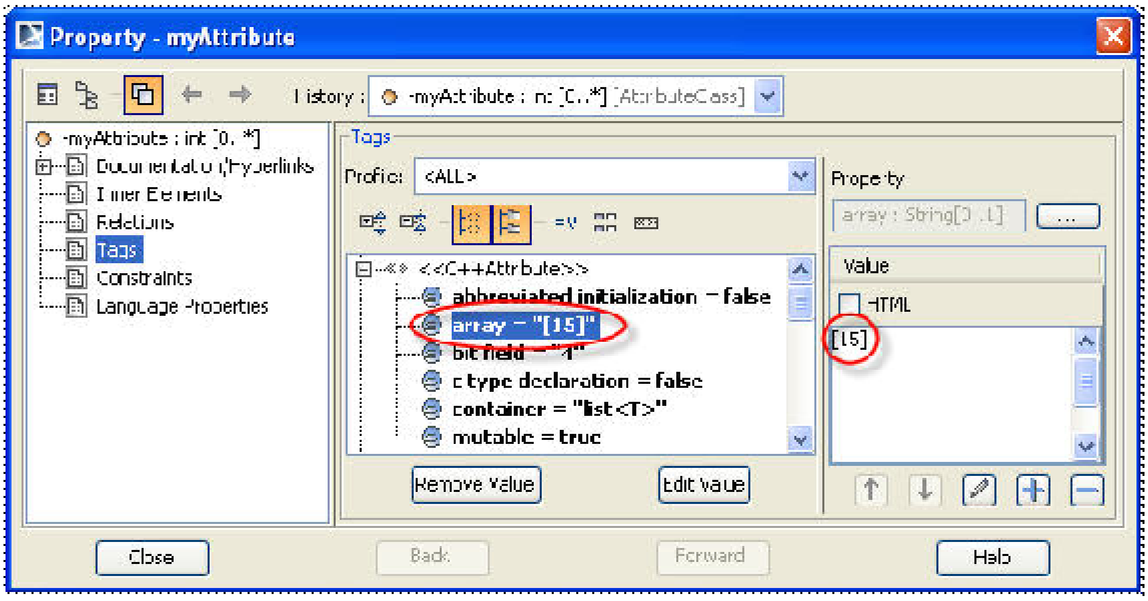
History : -myAttribute : int [0..*] [AttributeClass]

-myAttribute : int [0..*]
Documentation/Hyperlinks
Inner Elements
Relations
Tags
Constraints
Language Properties

myAttribute
Properties: All
Customize

Property

Name	myAttribute
Qualified Name	AttributeClass::myAttribute
Type	int [UML Standard Profile::MagicDraw Profile::...
Type Modifier	
Visibility	private
Default Value	
Owner	AttributeClass
Applied Stereotype	C++Attribute [Property] [C++ ANSI profile]
Multiplicity	0..*


Property - myAttribute

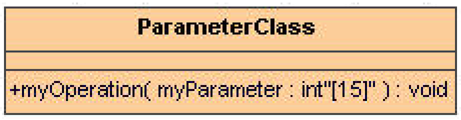
History : -myAttribute : int [0..*] [AttributeClass]

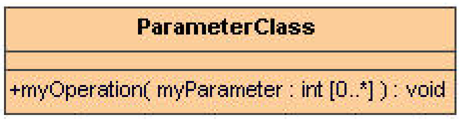
-myAttribute : int [0..*]
Documentation/Hyperlinks
Inner Elements
Relations
Tags
Constraints
Language Properties

Tags
Profile: <ALL>

<<C++Attribute>>
abbreviated initialization = false
array = "[15]"
bit field = "-1"
c type declaration = false
container = "list<T>"
mutable = true

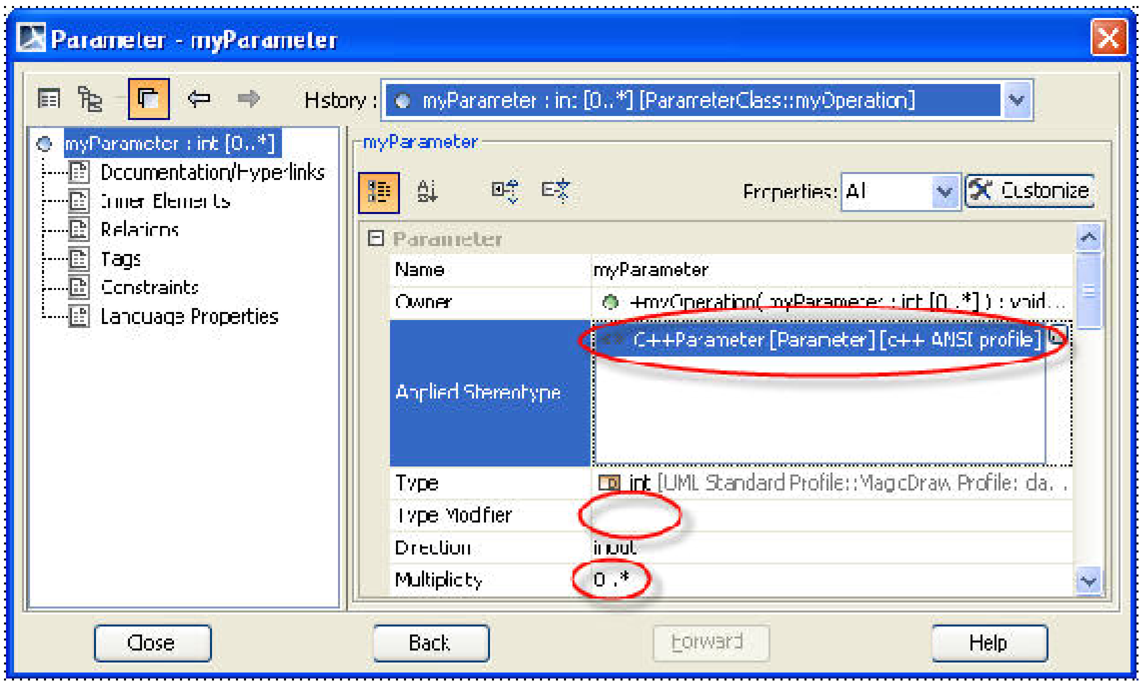
Property
array : String[0..1]
Value
HTML
[15]





`[15]`	Apply the **«C++Parameter»** stereotype and set **array** tag value to **[15]**. Set **Multiplicity** field to **0..***. Remove **[15]** from **type modifier** field.

Copyright © 1998-2015 No Magic, Inc.



Parameter - myParameter

History : myParameter : in: [0..*] [ParameterClass::myOperation]

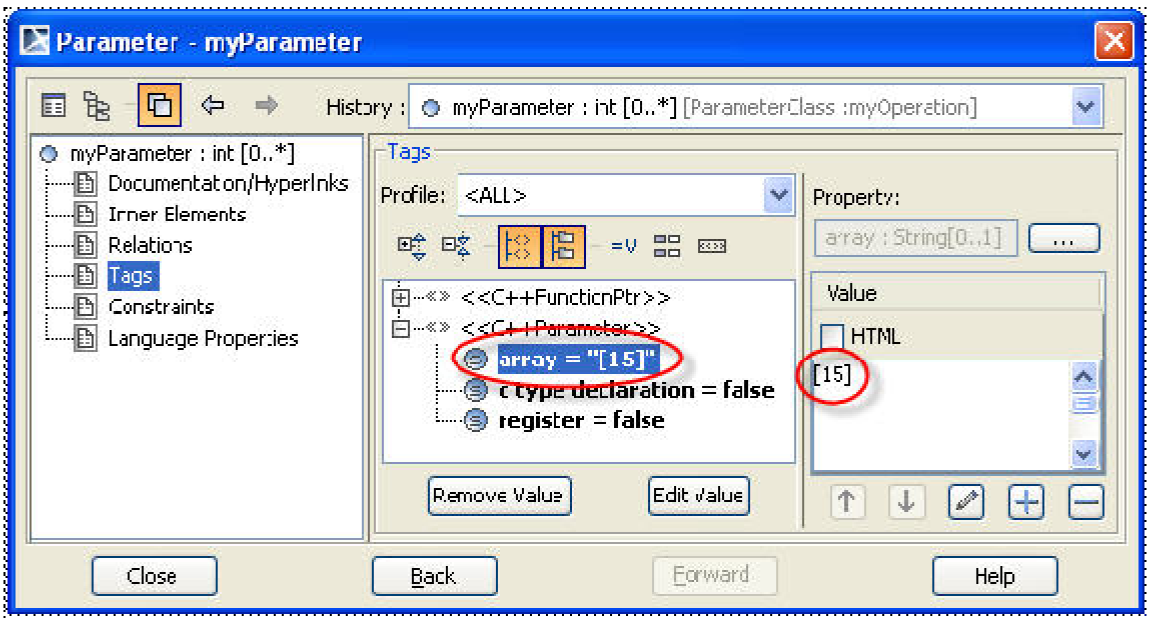
- myParameter : int [0..*]
  - Documentation/Hyperlinks
  - Inner Elements
  - Relations
  - Tags
  - Constraints
  - Language Properties

myParameter

Properties: All

Parameter	
Name	myParameter
Owner	+myOperation(myParameter : int [0..*]) : void...
Applied Stereotype	C++Parameter [Parameter] [C++ ANSI profile]
Type	int [UML Standard Profile::MagicDraw Profile::da...
Type Modifier	
Direction	inout
Multiplicity	0..*

Close Back Forward Help



Parameter - myParameter

History : myParameter : int [0..*] [ParameterClass::myOperation]

- myParameter : int [0..*]
  - Documentation/Hyperlinks
  - Inner Elements
  - Relations
  - Tags
  - Constraints
  - Language Properties

Tags

Profile: <ALL>

- <<C++FunctionPtr>>
- <<C++Parameter>>
  - array = "[15]"
  - c type declaration = false
  - register = false

Property: array : String[0..1]

Value
HTML
[15]

Remove Value Edit Value

Close Back Forward Help

Stereotypes

«C++EnumerationLiteral»

This example is **EnumClass** class that has literal value named **literal**. This literal value applies the **«C++EnumerationLiteral»** stereotype and sets **C++Initializer** tag value to **0**.

Copyright © 1998-2015 No Magic, Inc.

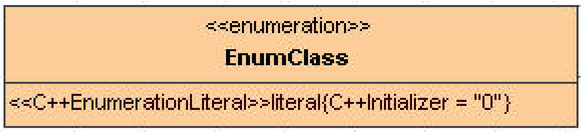
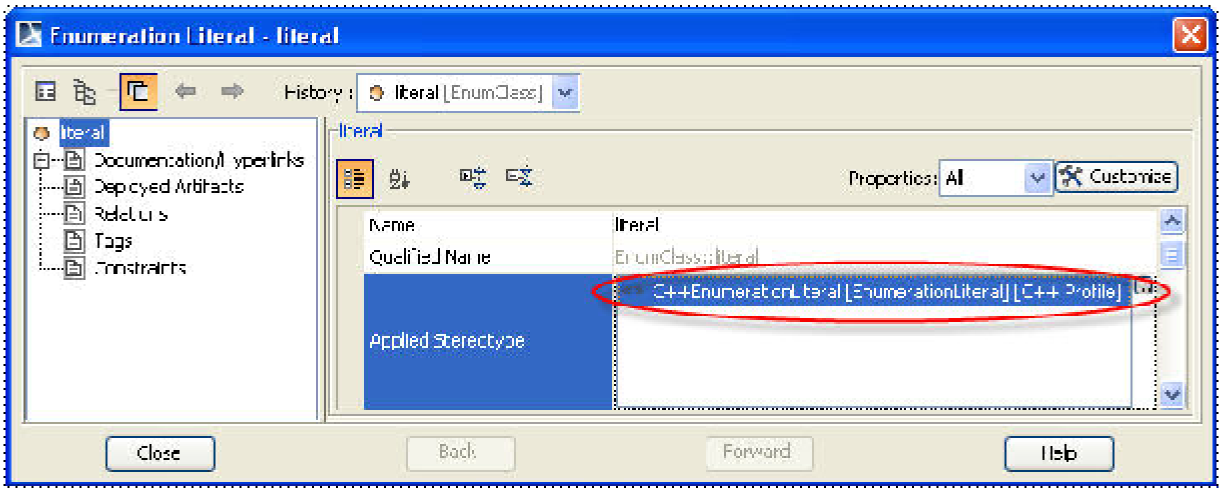


Figure 86 -- «C++EnumerationLiteral» stereotype Example in Class Diagram

The **«C++EnumerationLiteral»** stereotype is being shown in the figure below.



Name: literal
Qualified Name: EnumClass::literal
Applied Stereotype: C++EnumerationLiteral [EnumerationLiteral] [C++ Profile]

Figure 87 -- «C++EnumerationLiteral» stereotype

The **C++Initializer** tag value is being shown in the figure below.

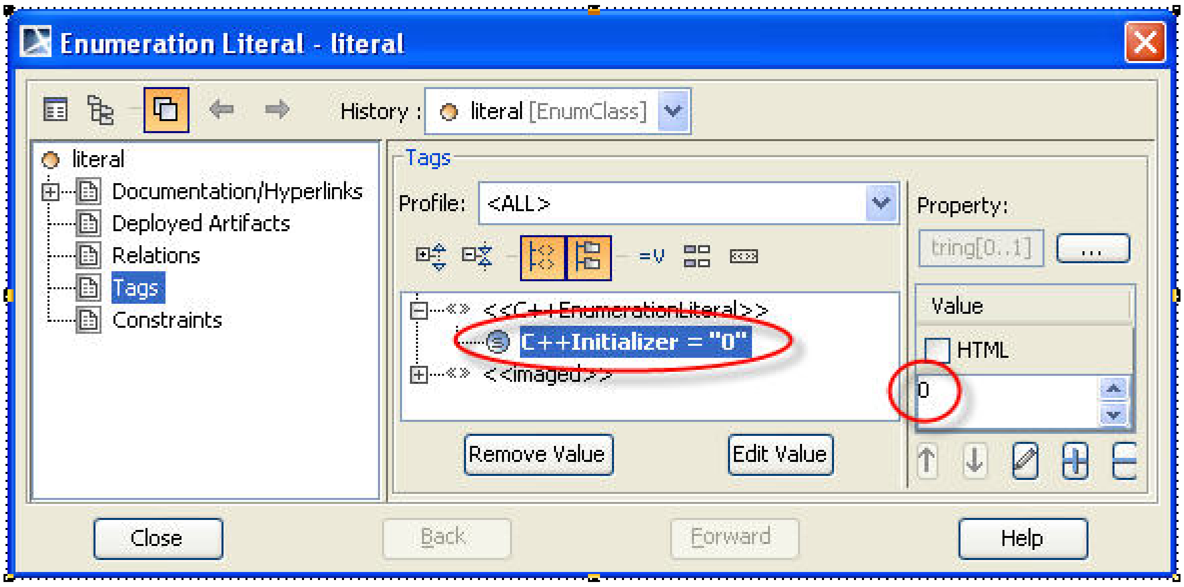
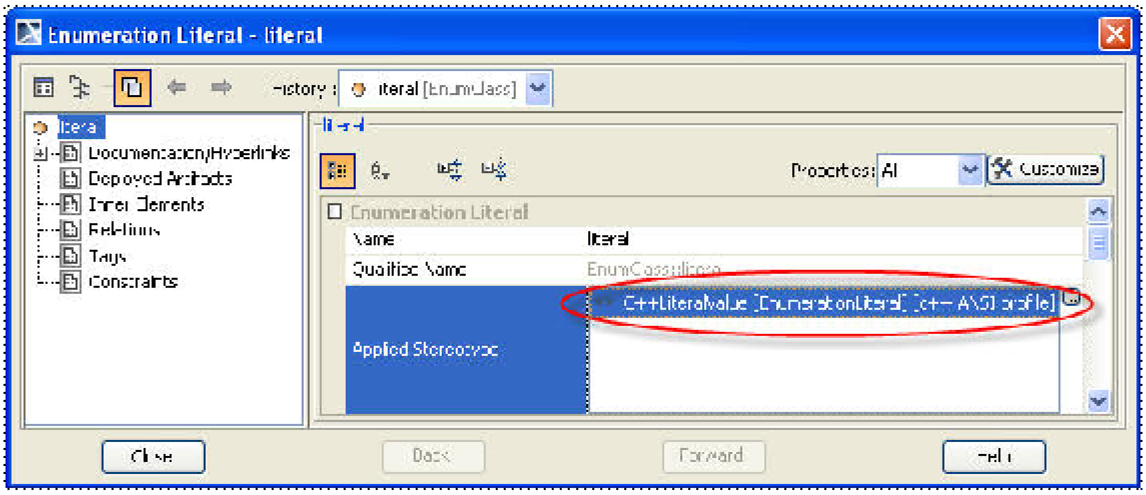


Figure 88 -- Enumeration Literal Tag Value

Old value	**Translation**
Enumeration Literal applies the **«C++EnumerationLiteral»** stereotype and sets **C++Initializer** tag value to **0**.	Apply the **«C++LiteralValue»** and set **value** tag value to **0**. Remove the **«C++EnumerationLiteral»** stereotype and **C++Initializer** tag value from Enumeration Literal.

Copyright © 1998-2015 No Magic, Inc.



Enumeration Literal - literal

History : literal [EnumClass]

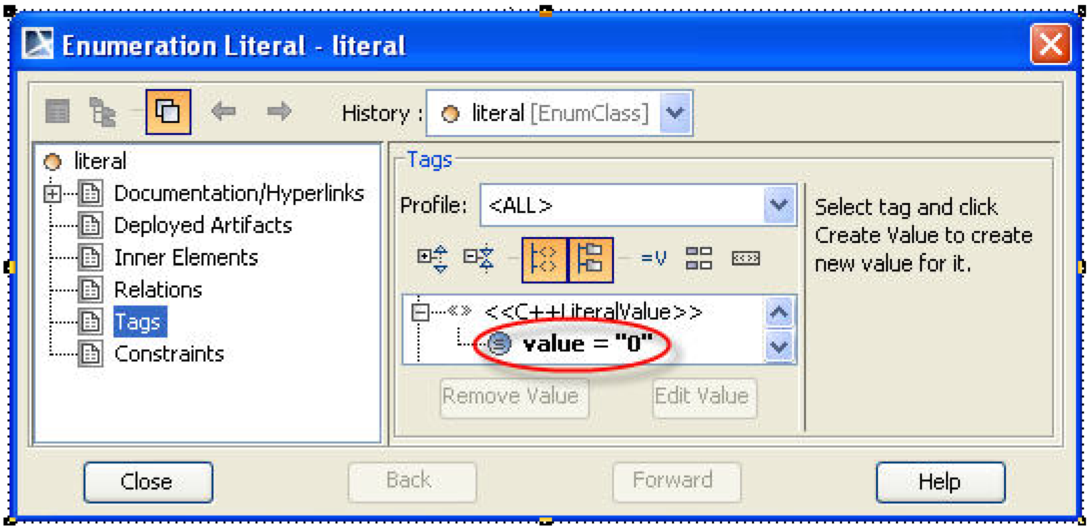
- literal
  - Documentation/Hyperlinks
  - Deployed Artifacts
  - Inner Elements
  - Relations
  - Tags
  - Constraints

literal

Properties: All

Enumeration Literal

Name	literal
Qualified Name	EnumClass::literal
Applied Stereotype	C++LiteralValue [EnumerationLiteral] [C++ ANSI profile]



Enumeration Literal - literal

History : literal [EnumClass]

- literal
  - Documentation/Hyperlinks
  - Deployed Artifacts
  - Inner Elements
  - Relations
  - Tags
  - Constraints

Tags

Profile: <ALL>

<<C++LiteralValue>>
value = "0"

Select tag and click Create Value to create new value for it.

The Model that is being shown in the figure below is a translation.

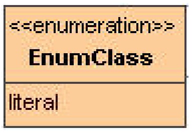


Figure 89 -- Translated Enumeration Literal in Class Diagram

«C++Namespace»

This example is **MyPackage** package that applies the **«C++Namespace»** stereotype in old profile and sets **unique namespace name** tag value to **myNamespace**.

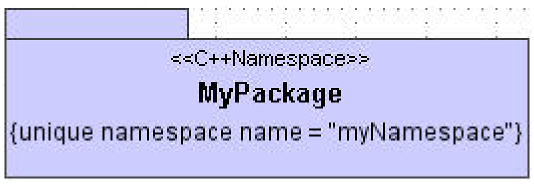
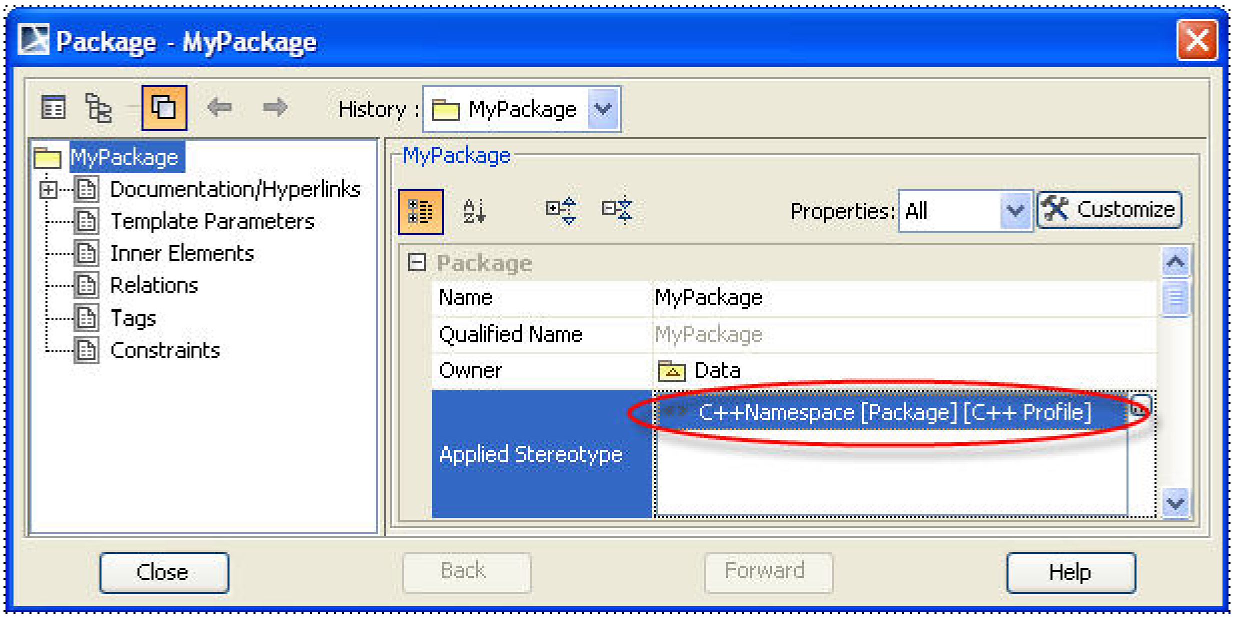


Figure 90 -- «C++Namespace» stereotype Example in Class Diagram

Copyright © 1998-2015 No Magic, Inc.

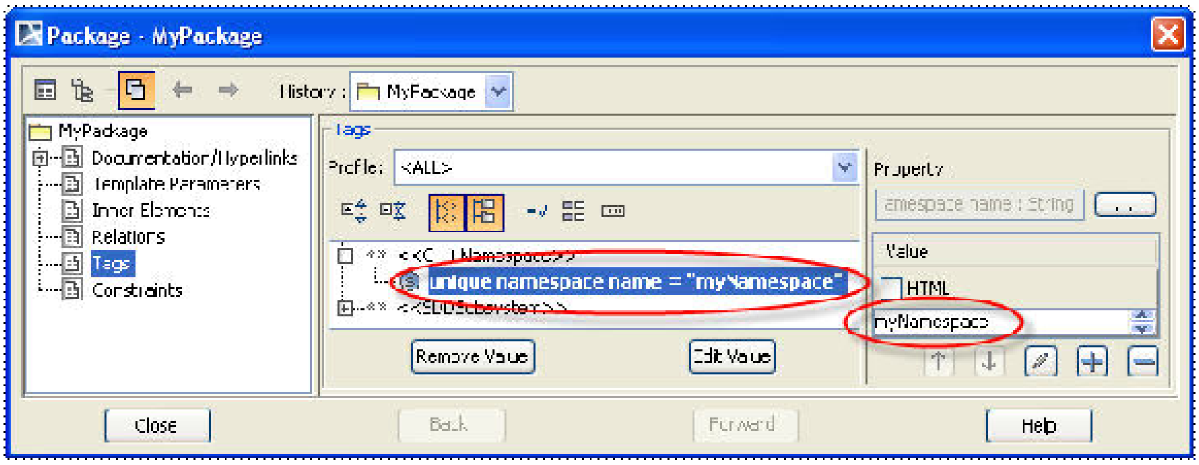
The **«C++Namespace»** stereotype is being shown in the figure below.



Package
Name: MyPackage
Qualified Name: MyPackage
Owner: Data
Applied Stereotype: C++Namespace [Package] [C++ Profile]

Figure 91 -- «C++Namespace» stereotype

The **unique namespace name** tag value is being shown in the figure below.

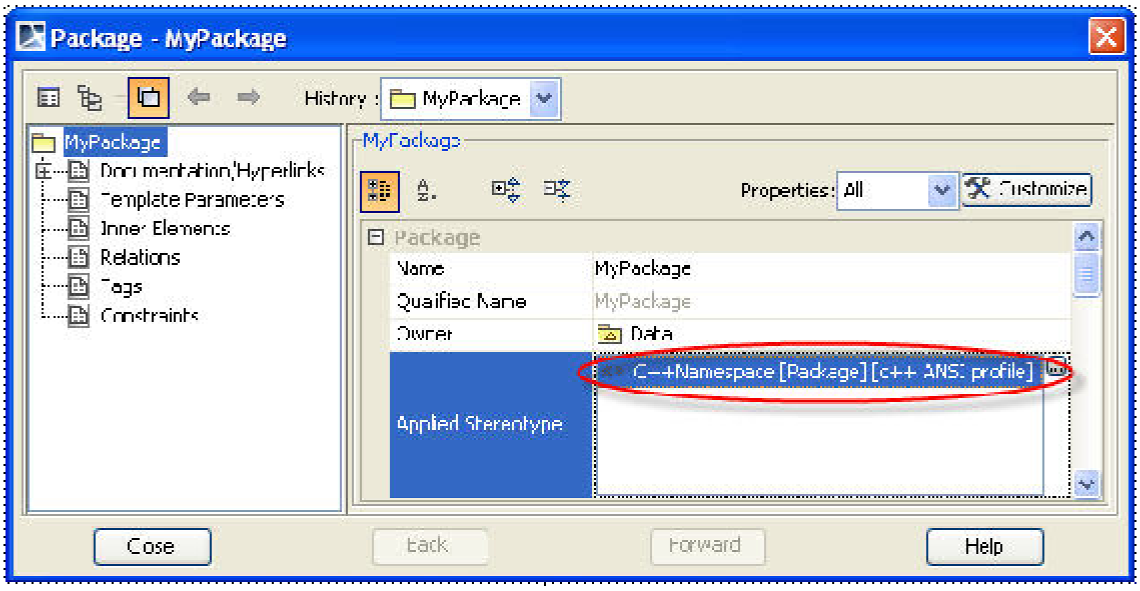


Tags
Profile: <ALL>
<<C++Namespace>>
unique namespace name = "myNamespace"
<<SDDSubsystem>>
Property: namespace name : String
Value: myNamespace

Figure 92 -- unique namespace name tag value

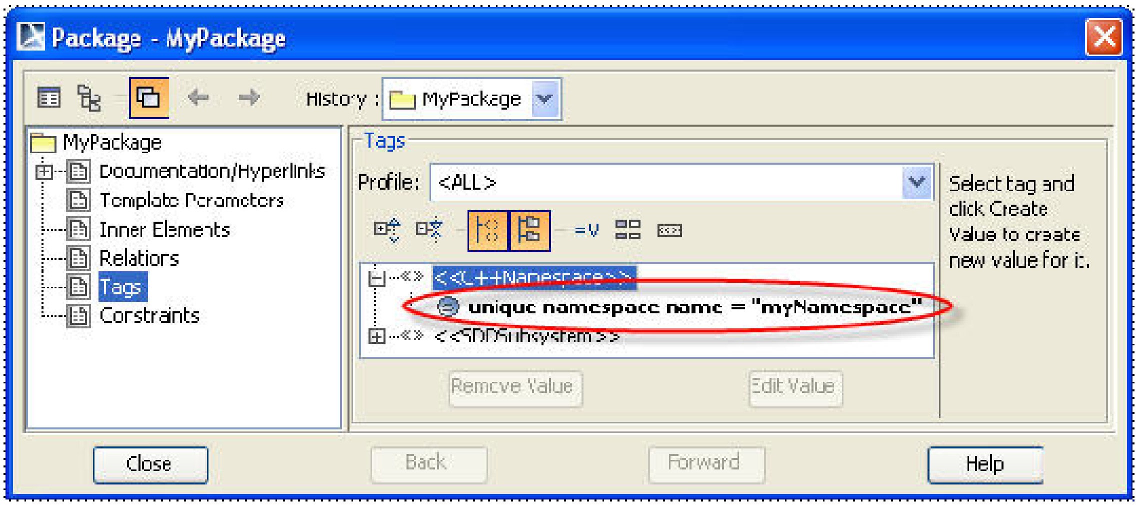
Old value	Translation
Package applies the **«C++Namespace»** stereotype in old profile (C++ Profile) and sets **unique namespace name** tag value to **myNamespace**.	Apply the **«C++Namespace»** stereotype in new profile (c++ ANSI profile) and set **unique namespace name** tag value to **myNamespace**. Remove the **«C++Namespace»** stereotype (C++ Profile) and **unique namespace name** tag value.

Copyright © 1998-2015 No Magic, Inc.



Package

Name	MyPackage
Qualified Name	MyPackage
Owner	Data
Applied Stereotype	C++Namespace [Package] [c++ ANSI profile]



Tags

Profile: <ALL>

<<C++Namespace>>
unique namespace name = "myNamespace"
<<SDDSubsystem>>

Select tag and click Create Value to create new value for it.

«C++Typename»

The **«C++Typename»** stereotype can apply to Template Parameters. Therefore, Elements that have template parameters could apply this stereotype.

Copyright © 1998-2015 No Magic, Inc.

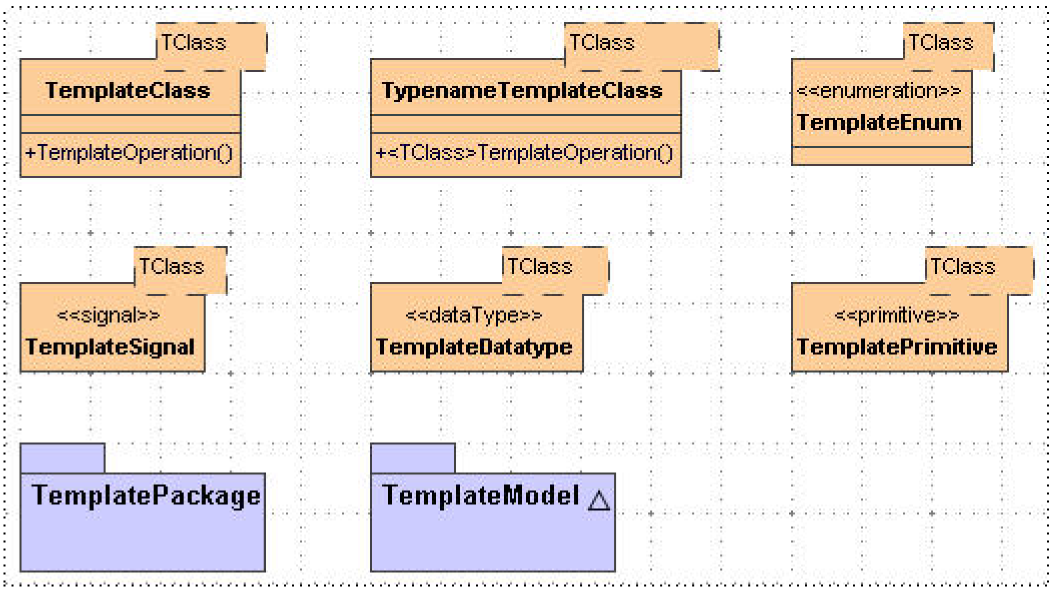
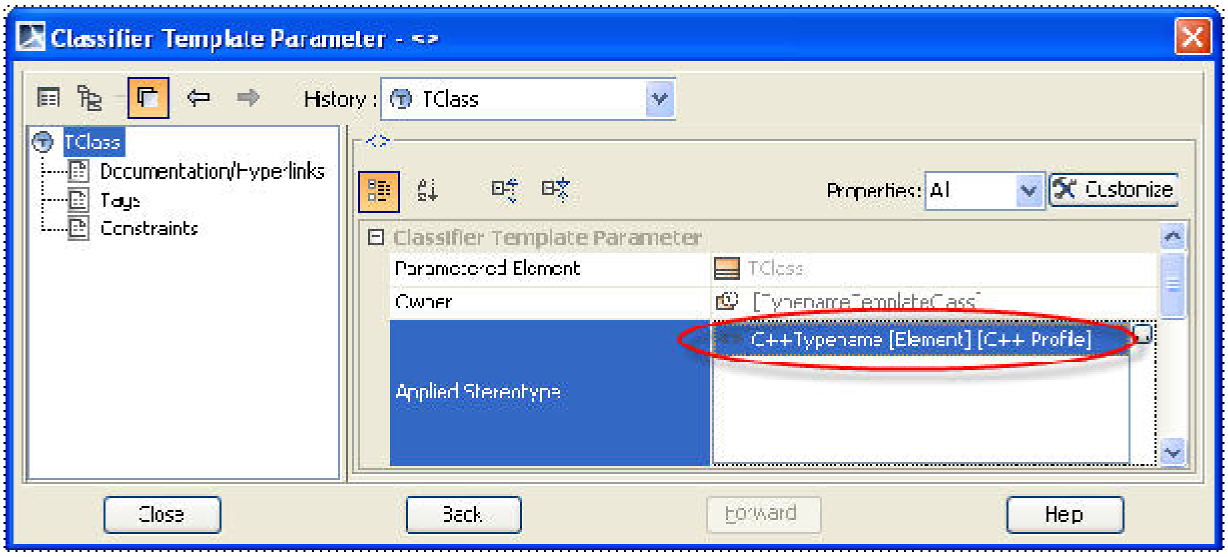


Figure 93 -- Elements that can have template parameters

In version 12.0, all elements that have template parameters must apply the **«C++TemplateParameter»** stereotype.

If elements from old version apply the **«C++Typename»** stereotype, the translation will apply the **«C++TemplateParameter»** stereotype and set **type keyword** tag value to **typename**. If not, the translation will apply the **«C++TemplateParameter»** stereotype and set **type keyword** tag value to **class**.



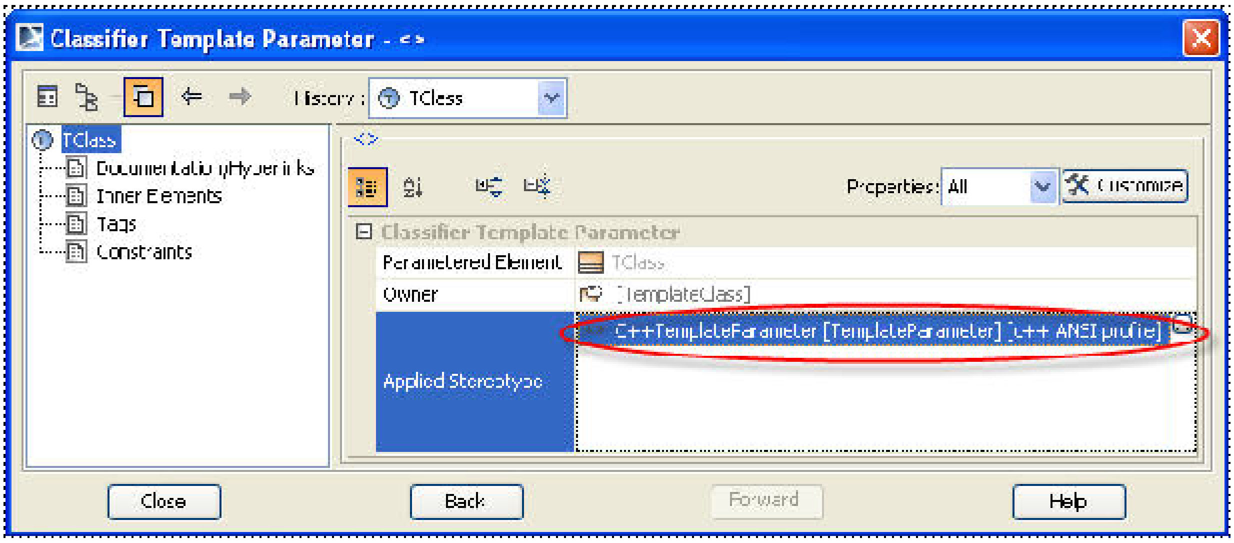
Classifier Template Parameter

Parametered Element	TClass
Owner	TypenameTemplateClass
Applied Stereotype	C++Typename [Element] [C++ Profile]

Figure 94 -- «C++Typename» stereotype

Copyright © 1998-2015 No Magic, Inc.

The **«C++TemplateParameter»** stereotype is being shown in the figure below.

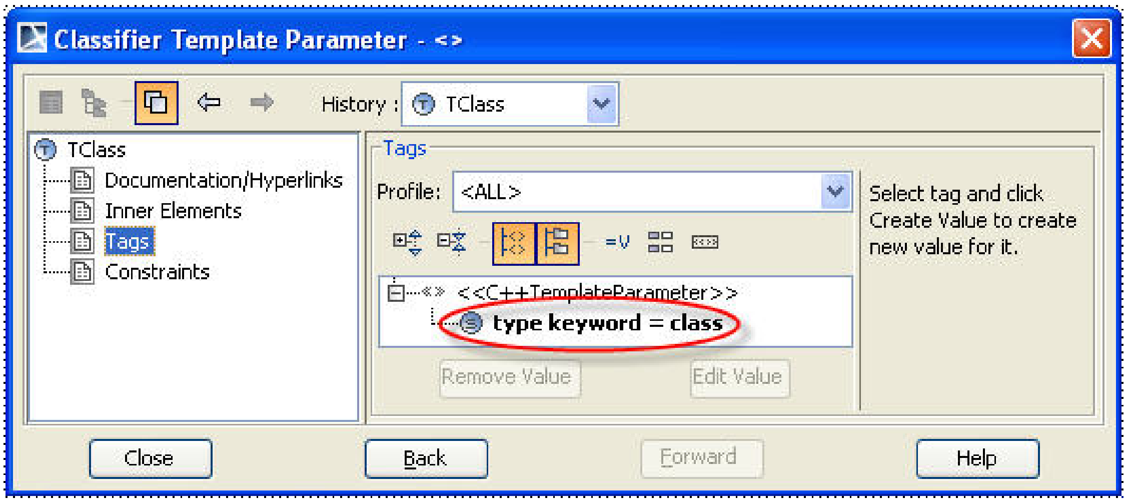


Classifier Template Parameter

Parametered Element	TClass
Owner	[TemplateClass]
Applied Stereotype	C++TemplateParameter [TemplateParameter] [C++ ANSI profile]

Figure 95 -- «C++TemplateParamater» stereotype

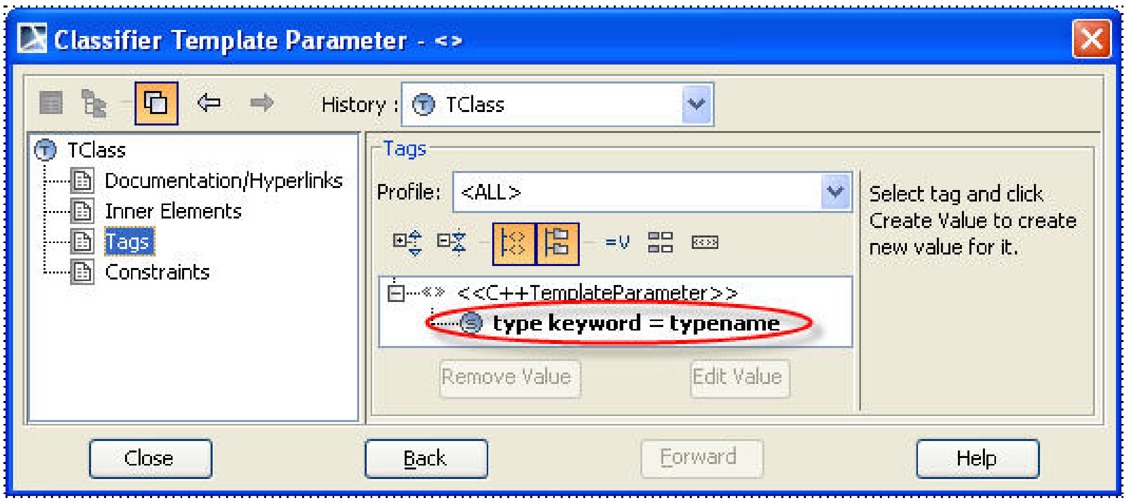
Old value	Translation
Template Parameter **does not** apply the **«C++Typename»** stereotype.	Apply the **«C++TemplateParameter»** stereotype and set **type keyword** tag value to **class**.



<<C++TemplateParameter>>
type keyword = class

Template Parameter applies the **«C++Typename»** stereotype.	Apply the **«C++TemplateParameter»** stereotype and set **type keyword** tag value to **typename**. Remove the **«C++Typename»** stereotype.

Copyright © 1998-2015 No Magic, Inc.



Classifier Template Parameter - <>

History : TClass

- TClass
  - Documentation/Hyperlinks
  - Inner Elements
  - Tags
  - Constraints

Tags

Profile: <ALL>

<<C++TemplateParameter>>
type keyword = typename

Select tag and click Create Value to create new value for it.

Remove Value | Edit Value

Close | Back | Forward | Help

«constructor» and «destructor» in UML Standard Profile

These examples are UMLStandardConstructorClass class that has an operation named myOperation() which applies the «constructor» stereotype in UML Standard Profile and UMLStandardDestructorClass class that has an operation named myOperation() which applies the «destructor» stereotype in UML Standard Profile.

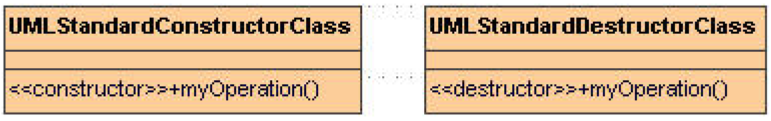
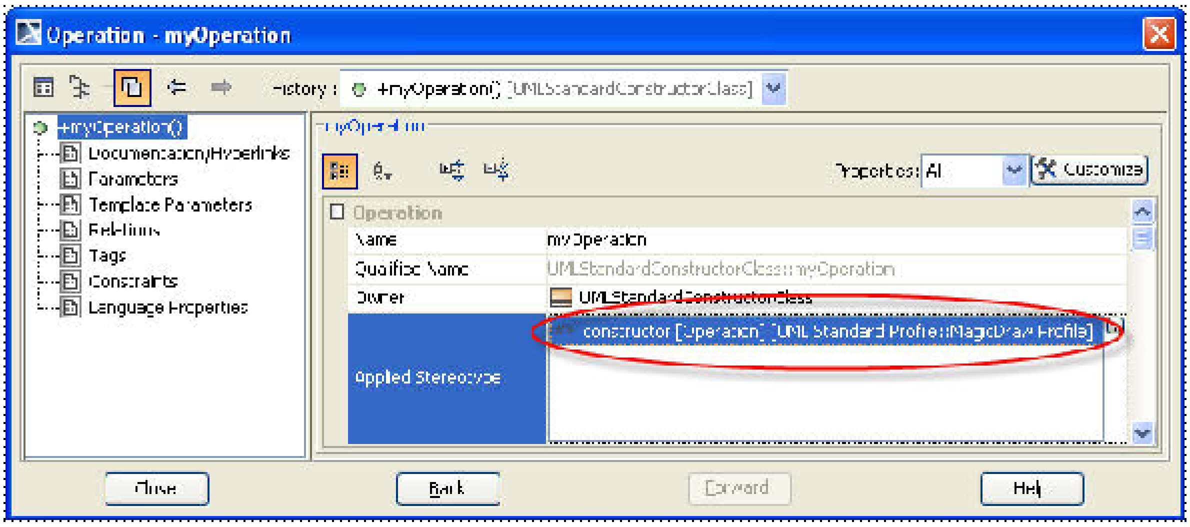

UMLStandardConstructorClass
<<constructor>>+myOperation()
UMLStandardDestructorClass
<<destructor>>+myOperation()

Figure 96 -- «constructor» and «destructor» stereotype Example in Class Diagram

«constructor»

The **«constructor»** stereotype is being shown in the figure below.



Operation - myOperation

History : +myOperation() [UMLStandardConstructorClass]

- +myOperation()
  - Documentation/Hyperlinks
  - Parameters
  - Template Parameters
  - Relations
  - Tags
  - Constraints
  - Language Properties

myOperation

Properties: All | Customize

Operation

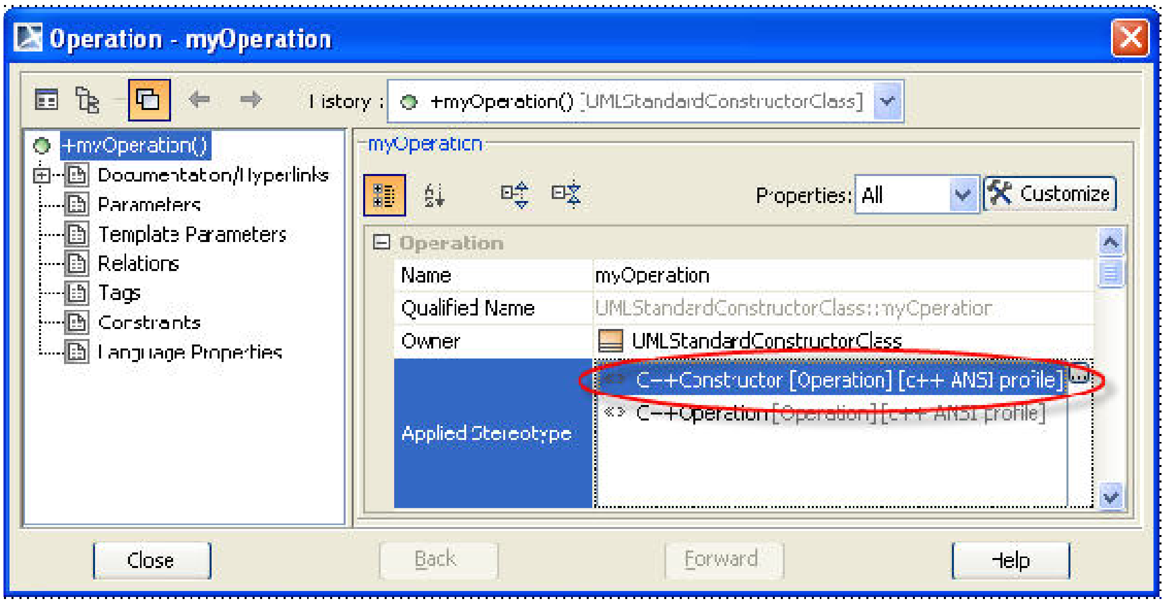
Name	myOperation
Qualified Name	UMLStandardConstructorClass::myOperation
Owner	UMLStandardConstructorClass
Applied Stereotype	constructor [Operation] [UML Standard Profile::MagicDraw Profile]

Close | Back | Forward | Help

Figure 97 -- «constructor» stereotype

Copyright © 1998-2015 No Magic, Inc.

Old value	Translation
Operation applies the **«constructor»** stereotype in UML Standard Profile.	Apply the **«C++Constructor»** stereotype (c++ ANSI profile). Remove the **«constructor»** stereotype (UML Standard Profile).


Operation - myOperation

History : +myOperation() [UMLStandardConstructorClass]

+myOperation()
- Documentation/Hyperlinks
- Parameters
- Template Parameters
- Relations
- Tags
- Constraints
- Language Properties

myOperation

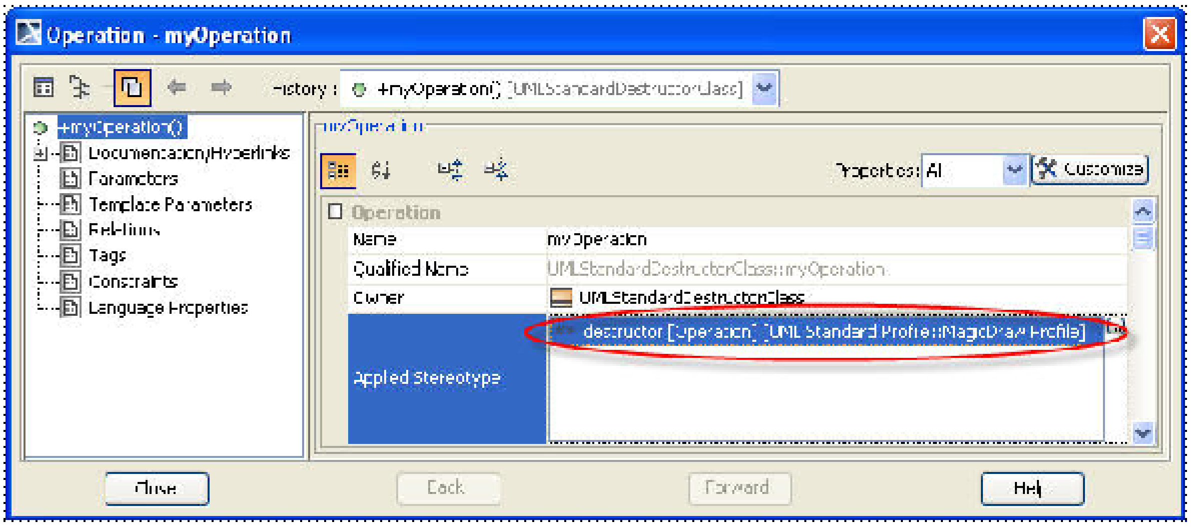
Properties: All

Operation

Name	myOperation
Qualified Name	UMLStandardConstructorClass::myOperation
Owner	UMLStandardConstructorClass
Applied Stereotype	C++Constructor [Operation] [c++ ANSI profile] C++Operation [Operation] [c++ ANSI profile]

«destructor»

The **«destructor»** stereotype is being shown in the figure below.


Operation - myOperation

History : +myOperation() [UMLStandardDestructorClass]

+myOperation()
- Documentation/Hyperlinks
- Parameters
- Template Parameters
- Relations
- Tags
- Constraints
- Language Properties

myOperation

Properties: All

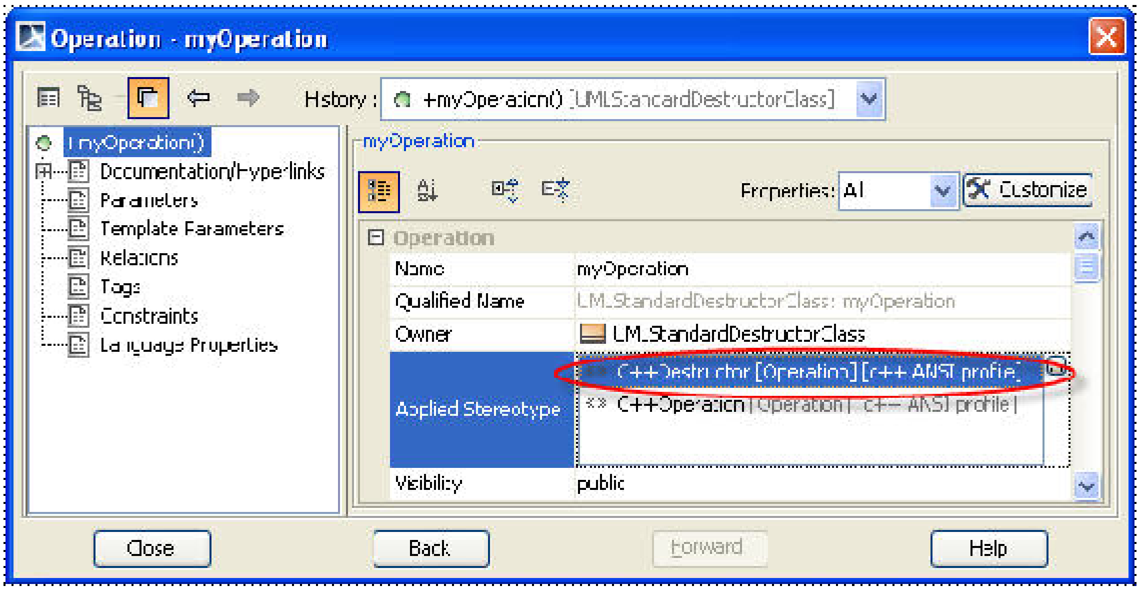
Operation

Name	myOperation
Qualified Name	UMLStandardDestructorClass::myOperation
Owner	UMLStandardDestructorClass
Applied Stereotype	destructor [Operation] [UML Standard Profile::MagicDraw Profile]

Figure 98 -- «destructor» stereotype

Copyright © 1998-2015 No Magic, Inc.

Old value	Translation
Operation applies the «**destructor**» stereotype in UML Standard Profile.	Apply the «**C++Destructor**» stereotype (c++ ANSI profile). Remove the «**destructor**» stereotype (UML Standard Profile).



Operation

Name	myOperation
Qualified Name	UMLStandardDestructorClass::myOperation
Owner	UMLStandardDestructorClass
Applied Stereotype	«» C++Destructor [Operation] [c++ ANSI profile] «» C++Operation [Operation] [c++ ANSI profile]
Visibility	public

Tag Value

C++ThrownExceptions

The «**C++Operation**» stereotype can apply to an operation.

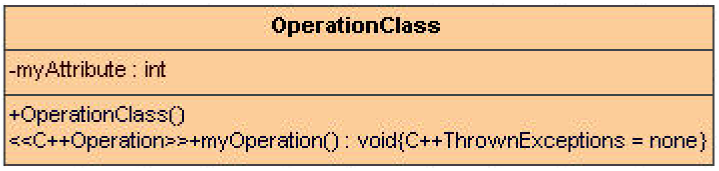

OperationClass
-myAttribute : int
+OperationClass()
<<C++Operation>>+myOperation() : void{C++ThrownExceptions = none}

Figure 99 -- C++ThrownExceptions Example in Class Diagram

In version 12.0, all operations must apply «**C++Operation**» stereotype and default **throw exception** tag value is **any**.

The **C++ThrownExceptions** tag value is being shown in the figure below.

Copyright © 1998-2015 No Magic, Inc.

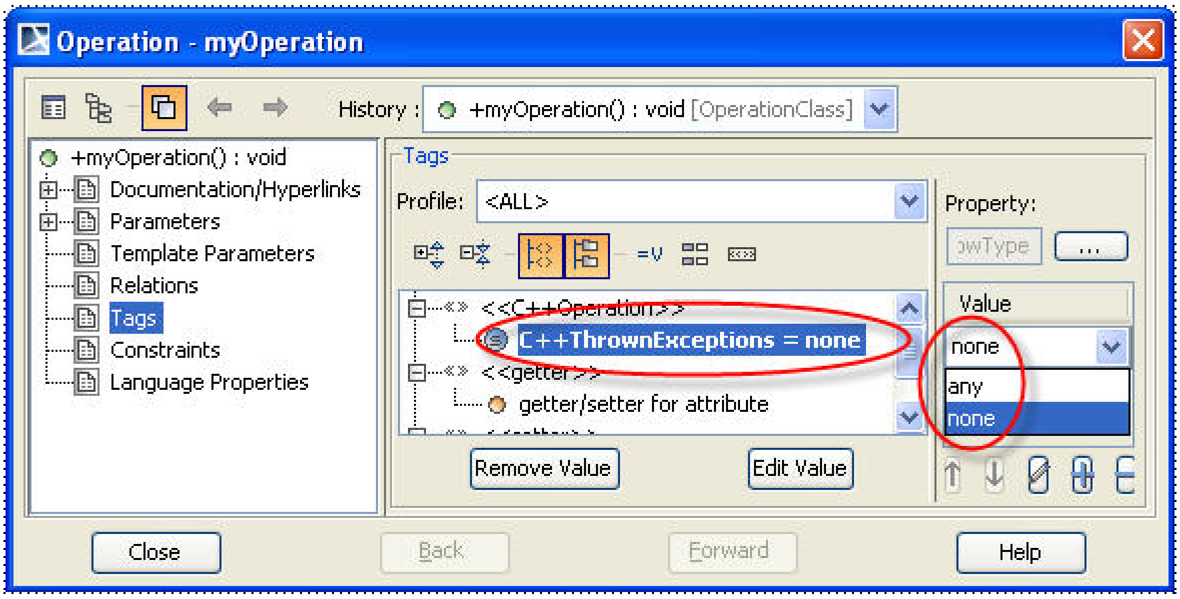
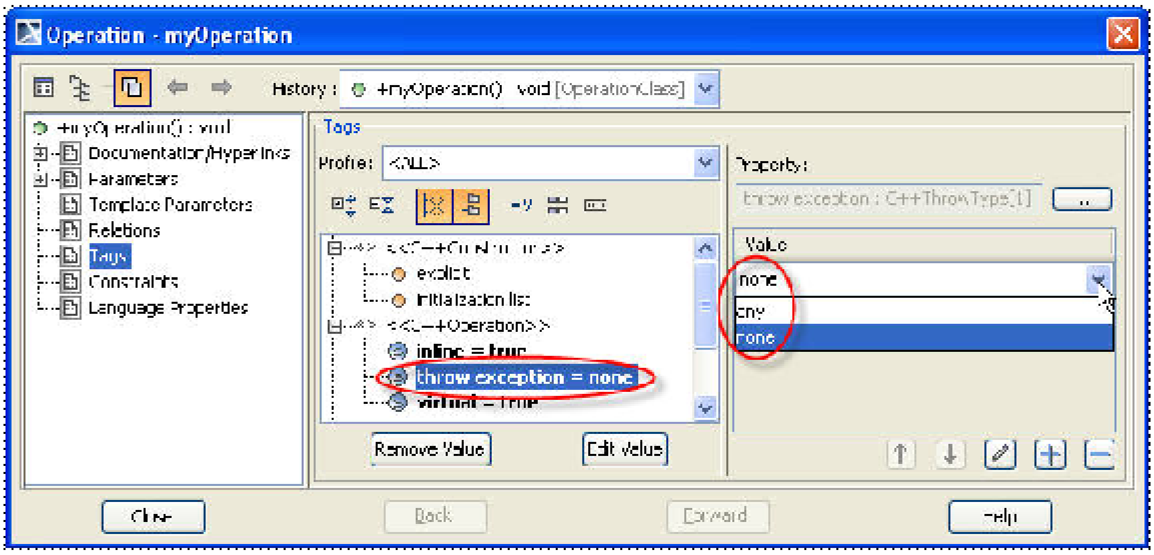


Figure 100 -- C++ThrownExceptions Tag Value

Old value	**Translation**
Operation **does not** apply the **«C++Operation»** stereotype in old profile (C++ Profile).	Apply the **«C++Operation»** stereotype in new profile (c++ ANSI profile) and set **throw exception** tag value to **any**.
Operation applies the **«C++Operation»** stereotype in old profile (C++ Profile) and set **C++ThrownExceptions** tag value to **any**.	Apply the **«C++Operation»** stereotype in new profile (c++ ANSI profile) and set **throw exception** tag value to **any**. Remove the **«C++Operation»** stereotype (old profile) and **C++ThrownExceptions** tag value.
Operation applies the **«C++Operation»** stereotype in old profile (C++ Profile) and set **C++ThrownExceptions** tag value to **none**.	Apply the **«C++Operation»** stereotype in new profile (c++ ANSI profile) and set **throw exception** tag value to **none**. Remove the **«C++Operation»** stereotype (old profile) and **C++ThrownExceptions** tag value.

Copyright © 1998-2015 No Magic, Inc.



<<C++Constructor>>
explicit
initialization list
<<C++Operation>>
inline = true
throw exception = none
virtual = true

throw exception : C++ThrowType[1]
none
any
none

Constructor and Destructor name

This example is **myClass** class that has operations named **myClass()** which is the constructor and **~myClass()** which is the destructor.

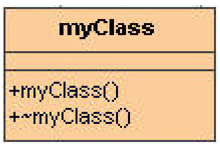
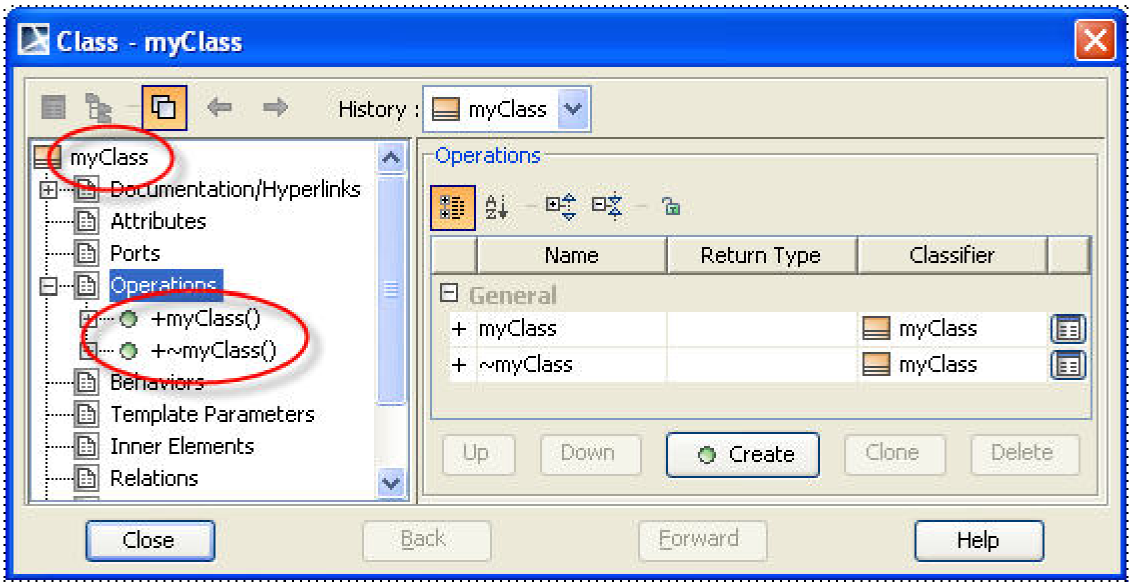


Figure 101 -- Constructor and Destructor name Example in Class Diagram

The Constructor and Destructor name are being shown in the figure below.

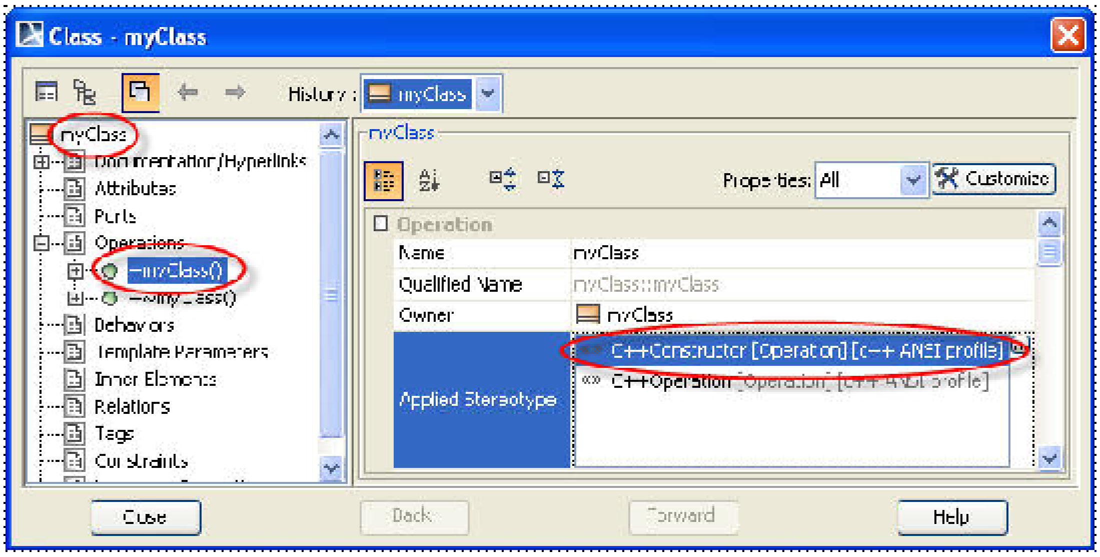


Name	Return Type	Classifier
General		
+ myClass		myClass
+ ~myClass		myClass

Figure 102 -- Constructor and Destructor name

Old value	Translation
Operation name is the same as the owner's name.	Apply the **«C++Constructor»** stereotype.

Copyright © 1998-2015 No Magic, Inc.

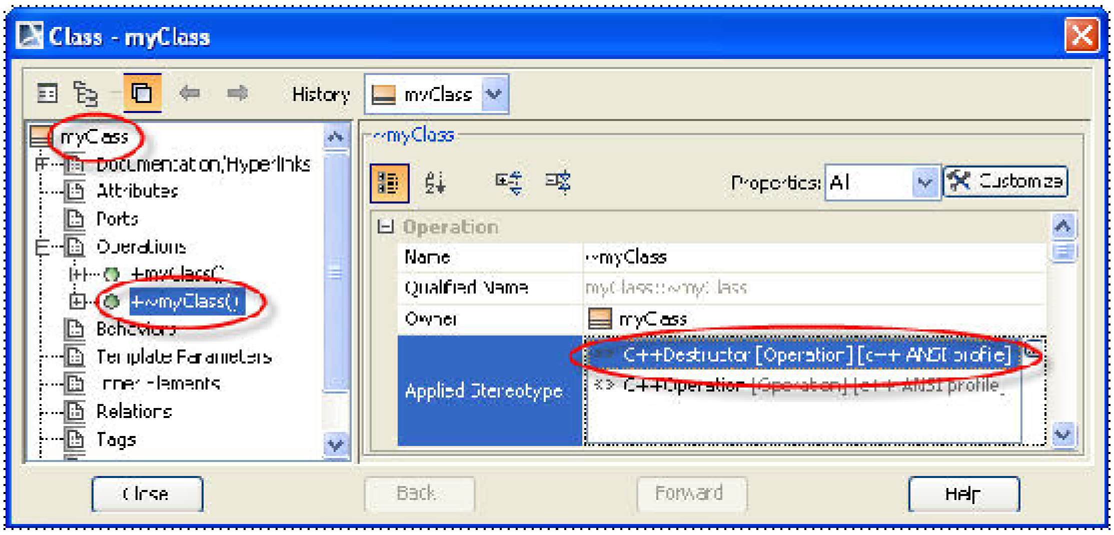


Class - myClass

Name	myClass
Qualified Name	myClass::myClass
Owner	myClass
Applied Stereotype	C++Constructor [Operation] [c++ ANSI profile]
	C++Operation [Operation] [c++ ANSI profile]

Operation name is '~' + the same name as the owner's name.

Apply the **«C++Destructor»** stereotype.



Class - myClass

Name	~myClass
Qualified Name	myClass::~myClass
Owner	myClass
Applied Stereotype	C++Destructor [Operation] [c++ ANSI profile]
	C++Operation [Operation] [c++ ANSI profile]

Copyright © 1998-2015 No Magic, Inc.

Data type

bool

There are five cases of using bool data type such as Attribute type, Parameter type, Return type, Actual in Template Parameter Substitution and Default in Classifier Template Parameter. MagicDraw will change old bool data type to new bool data type in new profile.

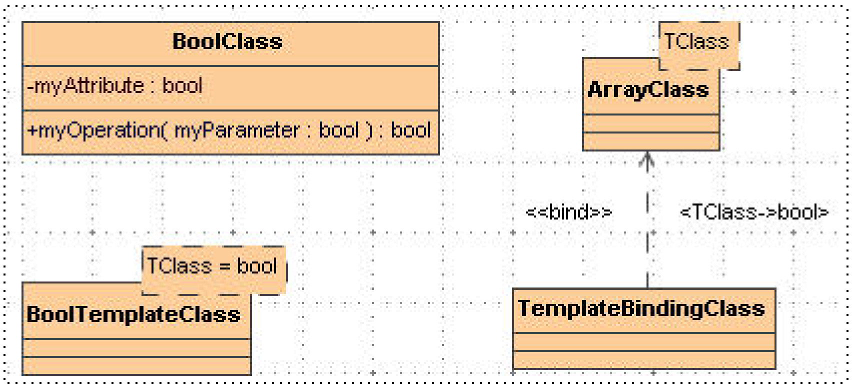
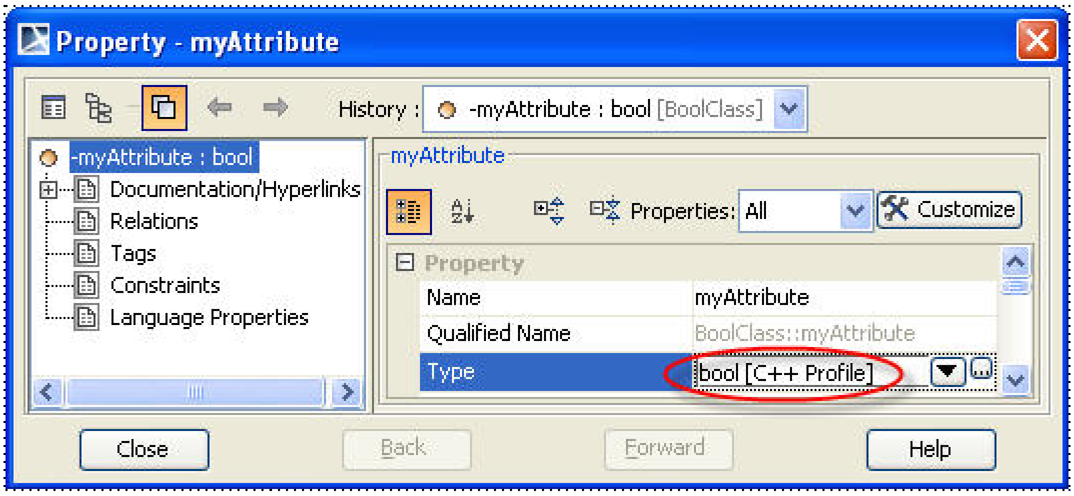


Figure 103 -- bool data type Example in Class Diagram

Attribute type

The old bool data type in Attribute type is being shown in the figure below.


Property - myAttribute

History : -myAttribute : bool [BoolClass]

- -myAttribute : bool
  - Documentation/Hyperlinks
  - Relations
  - Tags
  - Constraints
  - Language Properties

myAttribute

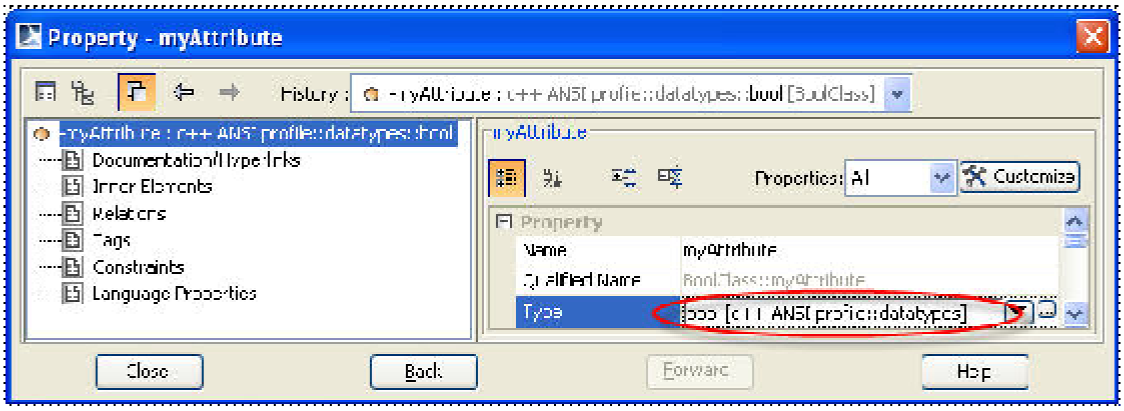
Properties: All | Customize

Property	
Name	myAttribute
Qualified Name	BoolClass::myAttribute
Type	bool [C++ Profile]

Figure 104 -- old bool data type in Attribute type

Old value	Translation
Old bool data type in old profile (C++ Profile).	Change to the new bool data type in new profile (c++ ANSI profile).

Copyright © 1998-2015 No Magic, Inc.



Property - myAttribute

History : myAttribute : c++ ANSI profile::datatypes::bool [BoolClass]

myAttribute : c++ ANSI profile::datatypes::bool
- Documentation/Hyperlinks
- Inner Elements
- Relations
- Tags
- Constraints
- Language Properties

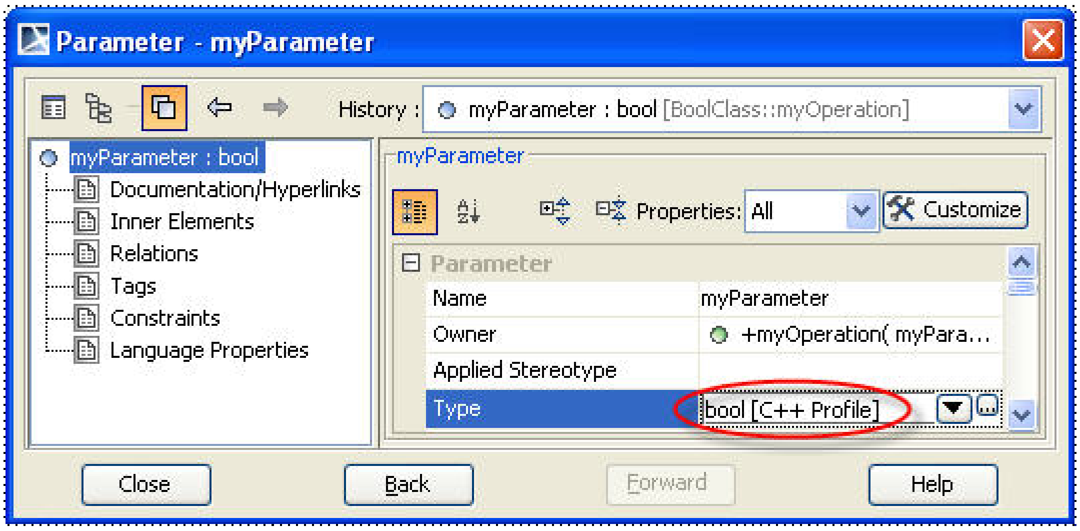
myAttribute

Properties: All

Property	
Name	myAttribute
Qualified Name	BoolClass::myAttribute
Type	bool [c++ ANSI profile::datatypes]

Parameter type

The old bool data type in Parameter type is being shown in the figure below.



Parameter - myParameter

History : myParameter : bool [BoolClass::myOperation]

myParameter : bool
- Documentation/Hyperlinks
- Inner Elements
- Relations
- Tags
- Constraints
- Language Properties

myParameter

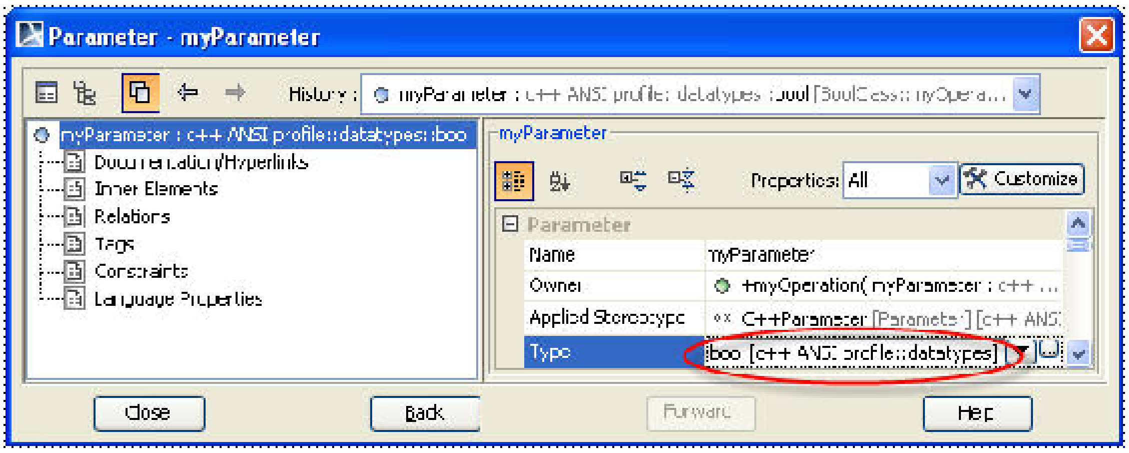
Properties: All

Parameter	
Name	myParameter
Owner	+myOperation(myPara...
Applied Stereotype	
Type	bool [C++ Profile]

Figure 105 -- old bool data type in Parameter type

Old value	**Translation**
Old bool data type in old profile (C++ Profile).	Change to the new bool data type in new profile (c++ ANSI profile).

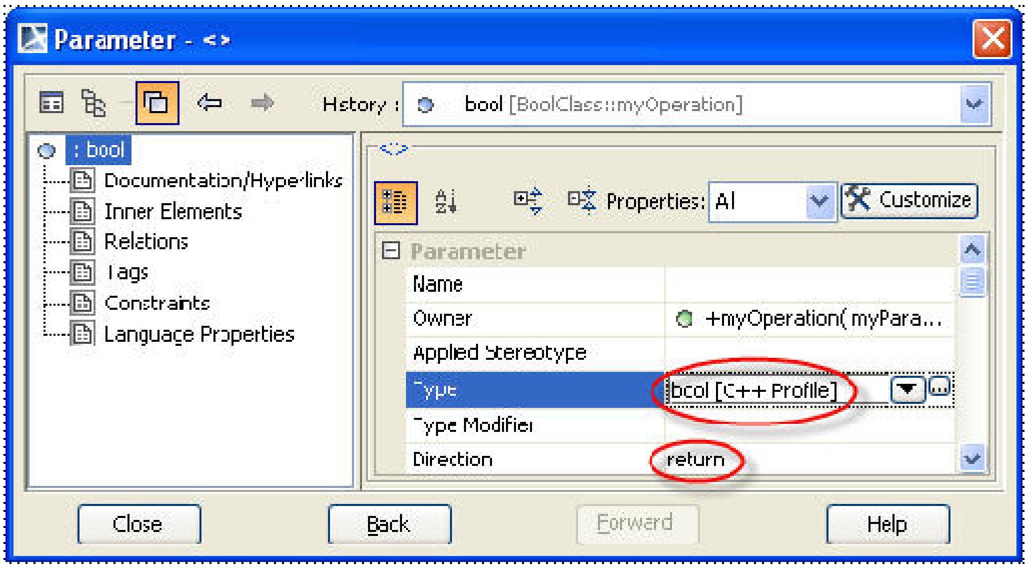
Copyright © 1998-2015 No Magic, Inc.



Parameter	
Name	myParameter
Owner	+myOperation(myParameter : c++ ...
Applied Stereotype	C++Parameter [Parameter] [c++ ANSI
Type	bool [c++ ANSI profile::datatypes]

Return type

The old bool data type in Return type is being shown in the figure below.

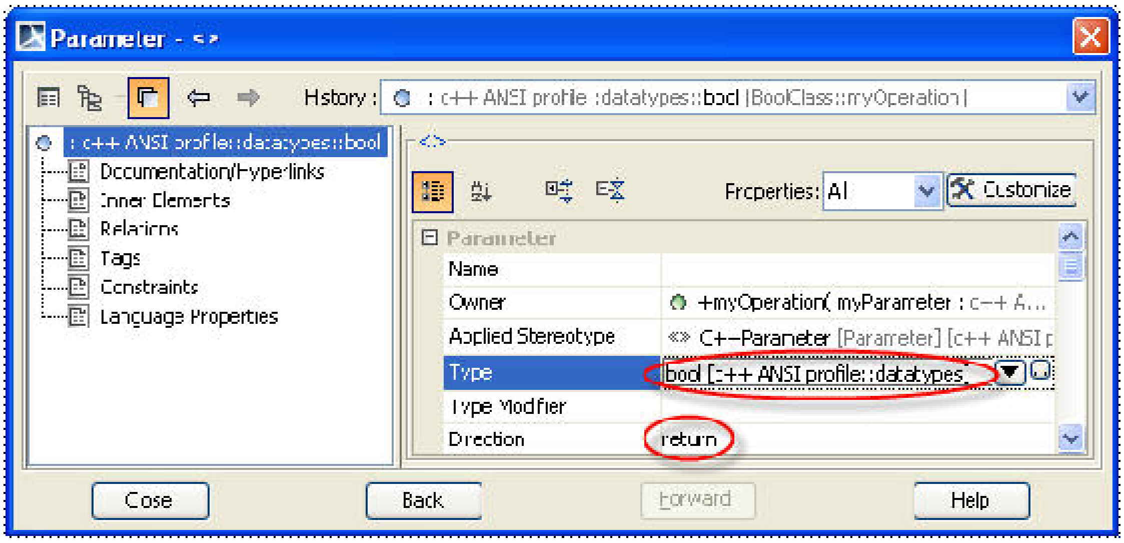


Parameter	
Name	
Owner	+myOperation(myPara...
Applied Stereotype	
Type	bool [C++ Profile]
Type Modifier	
Direction	return

Figure 106 -- Figure 57 -- old bool type in Return type

Old value	Translation
Old bool data type in old profile (C++ Profile).	Change to the new bool data type in new profile (c++ ANSI profile).

Copyright © 1998-2015 No Magic, Inc.



History : : c++ ANSI profile::datatypes::bool [BoolClass::myOperation]

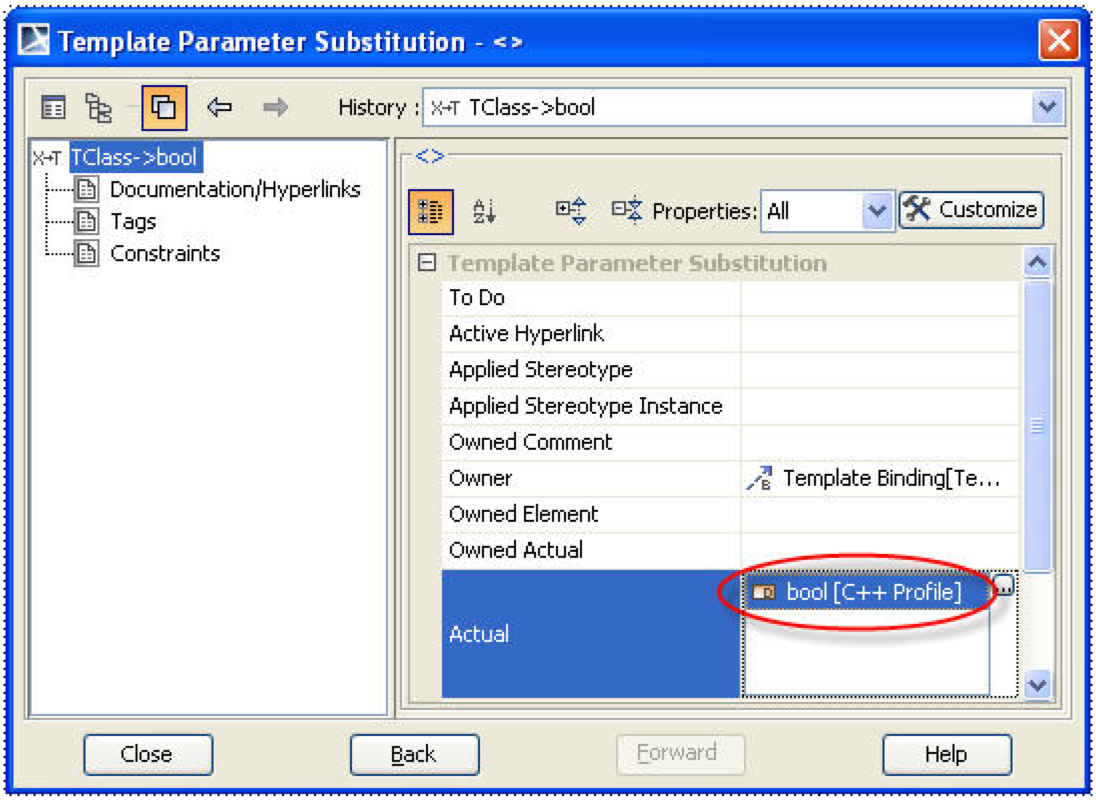
: c++ ANSI profile::datatypes::bool
Documentation/Hyperlinks
Inner Elements
Relations
Tags
Constraints
Language Properties

Properties: All

Parameter	
Name	
Owner	+myOperation(myParameter : c++ A...
Applied Stereotype	C++Parameter [Parameter] [c++ ANSI p
Type	bool [c++ ANSI profile::datatypes
Type Modifier	
Direction	return

Actual in Template Parameter Substitution

The old bool data type in Actual in Template Parameter Substitution is being shown in the figure below.



History : TClass->bool

TClass->bool
Documentation/Hyperlinks
Tags
Constraints

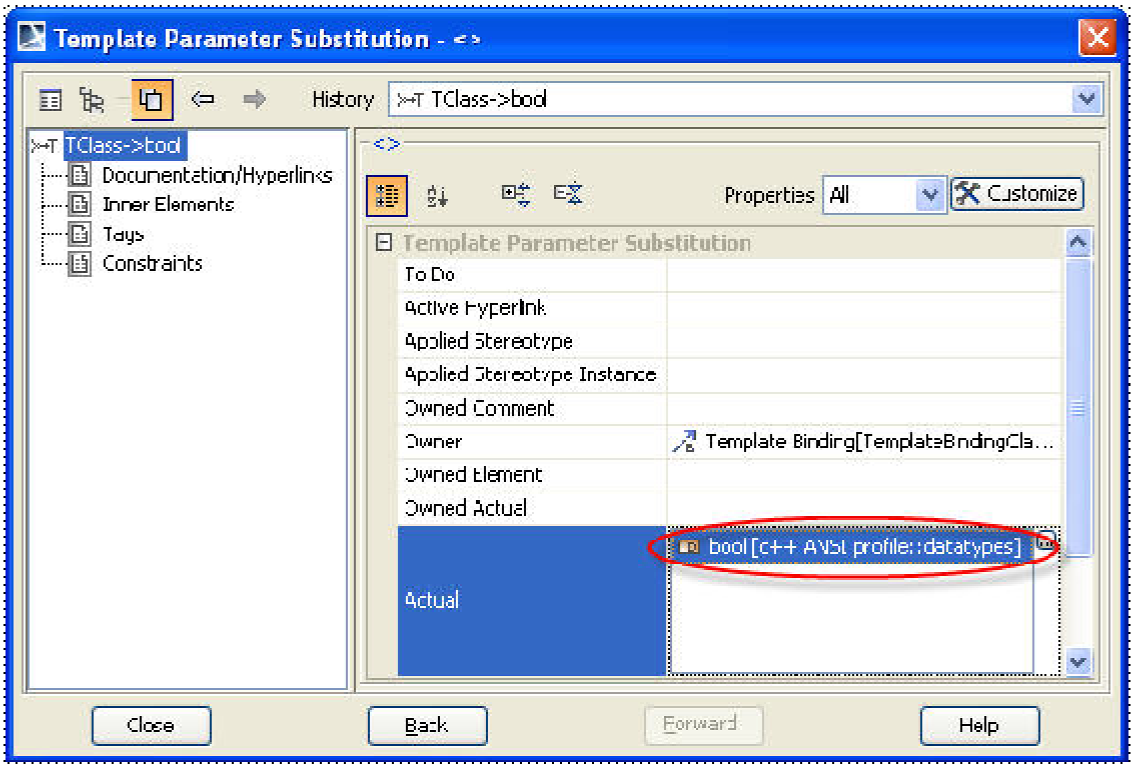
Properties: All

Template Parameter Substitution	
To Do	
Active Hyperlink	
Applied Stereotype	
Applied Stereotype Instance	
Owned Comment	
Owner	Template Binding[Te...
Owned Element	
Owned Actual	
Actual	bool [C++ Profile]

Figure 107 -- old bool data type in Actual in Template Parameter Substitution

Old value	**Translation**
Old bool data type in old profile (C++ Profile).	Change to the new bool data type in new profile (c++ ANSI profile).

Copyright © 1998-2015 No Magic, Inc.



Template Parameter Substitution - <>

History: TClass->bool

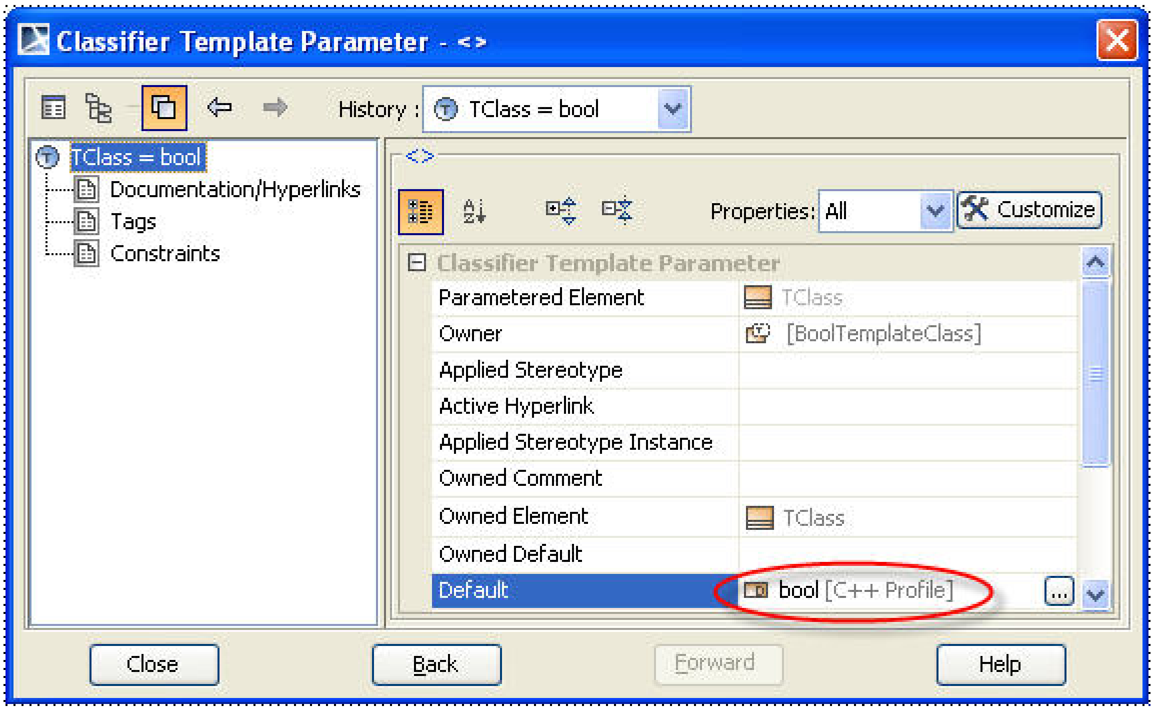
TClass->bool
- Documentation/Hyperlinks
- Inner Elements
- Tags
- Constraints

Properties: All

Template Parameter Substitution	
To Do	
Active Hyperlink	
Applied Stereotype	
Applied Stereotype Instance	
Owned Comment	
Owner	Template Binding[TemplateBindingCla...
Owned Element	
Owned Actual	
Actual	bool [c++ ANSI profile::datatypes]

Default in Classifier Template Parameter

The old bool data type in Default in Classifier Template Parameter is being shown in the figure below.



Classifier Template Parameter - <>

History : TClass = bool

TClass = bool
- Documentation/Hyperlinks
- Tags
- Constraints

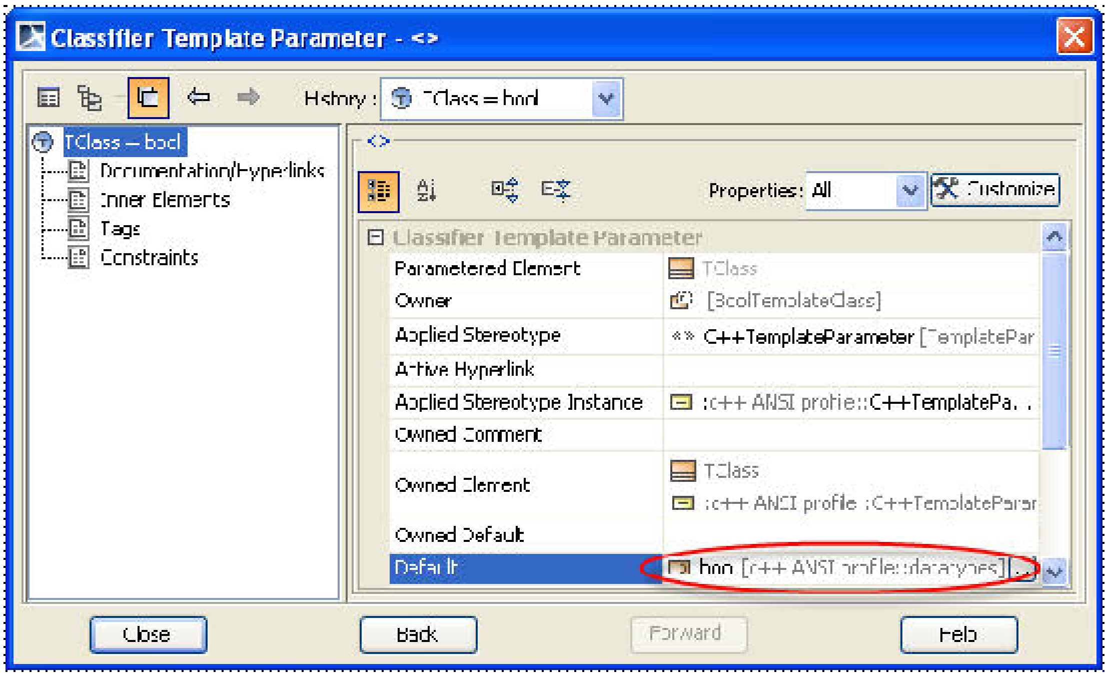
Properties: All

Classifier Template Parameter	
Parametered Element	TClass
Owner	[BoolTemplateClass]
Applied Stereotype	
Active Hyperlink	
Applied Stereotype Instance	
Owned Comment	
Owned Element	TClass
Owned Default	
Default	bool [C++ Profile]

Figure 108 -- old bool data type in Default in Classifier Template Parameter

Copyright © 1998-2015 No Magic, Inc.

Old value	Translation
Old bool data type in old profile (C++ Profile).	Change to the new bool data type in new profile (c++ ANSI profile).



Properties: All

Classifier Template Parameter	
Parametered Element	TClass
Owner	[BoolTemplateClass]
Applied Stereotype	«» C++TemplateParameter [TemplatePar
Active Hyperlink	
Applied Stereotype Instance	:c++ ANSI profile::C++TemplatePa. .
Owned Comment	
Owned Element	TClass :c++ ANSI profile :C++TemplatePar
Owned Default	
Default	bool [c++ ANSI profile::datatypes]

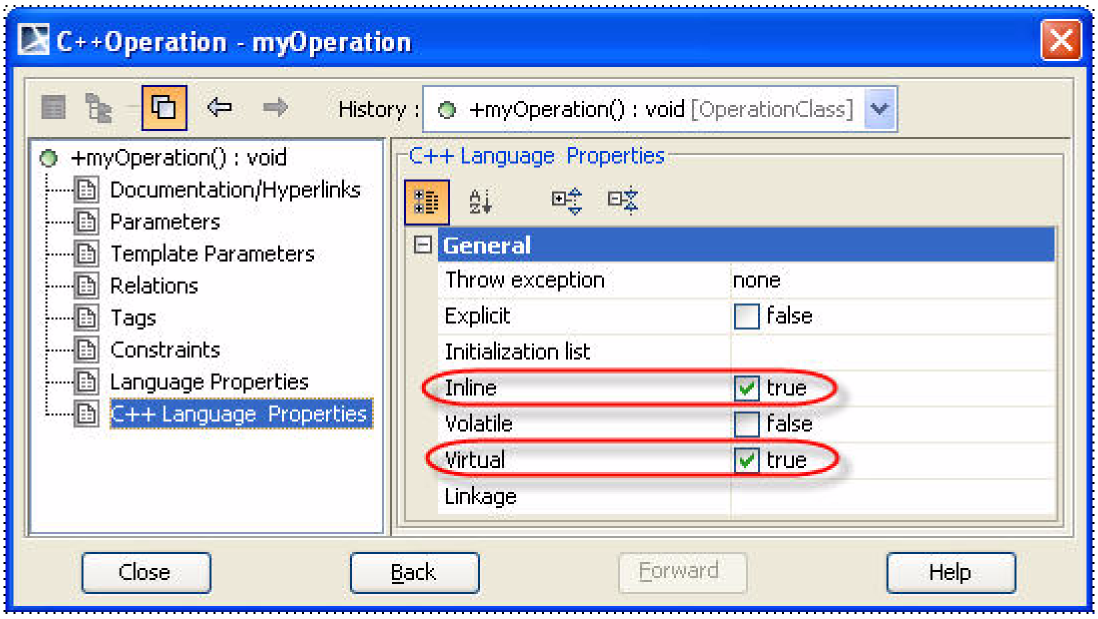
Copyright © 1998-2015 No Magic, Inc.

DSL customization

This chapter describes how to customize tag values that transform from stereotypes in old C++ profile and language properties.

Operation and Constructor

There are **Inline** and **Virtual** in Operation Language Properties that translate to Operation tag values.



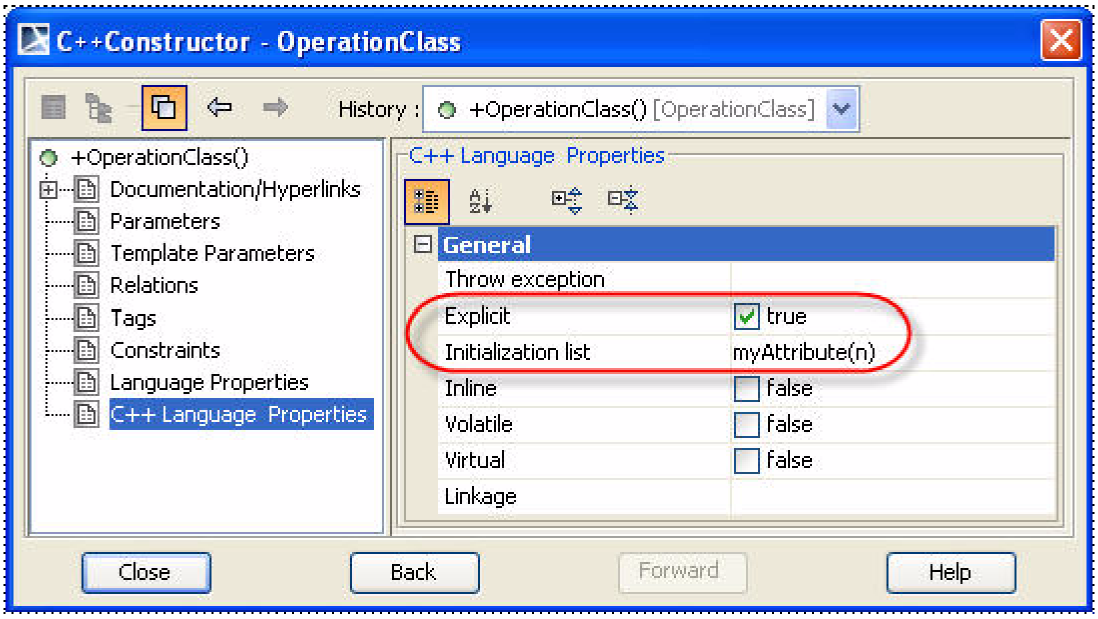
C++ Language Properties

General

Throw exception	none
Explicit	false
Initialization list	
Inline	true
Volatile	false
Virtual	true
Linkage	

Figure 109 -- tag values in Operation

There are **Explicit** and **Initialization list** in Operation Language Properties that translate to Constructor tag values.



C++ Language Properties

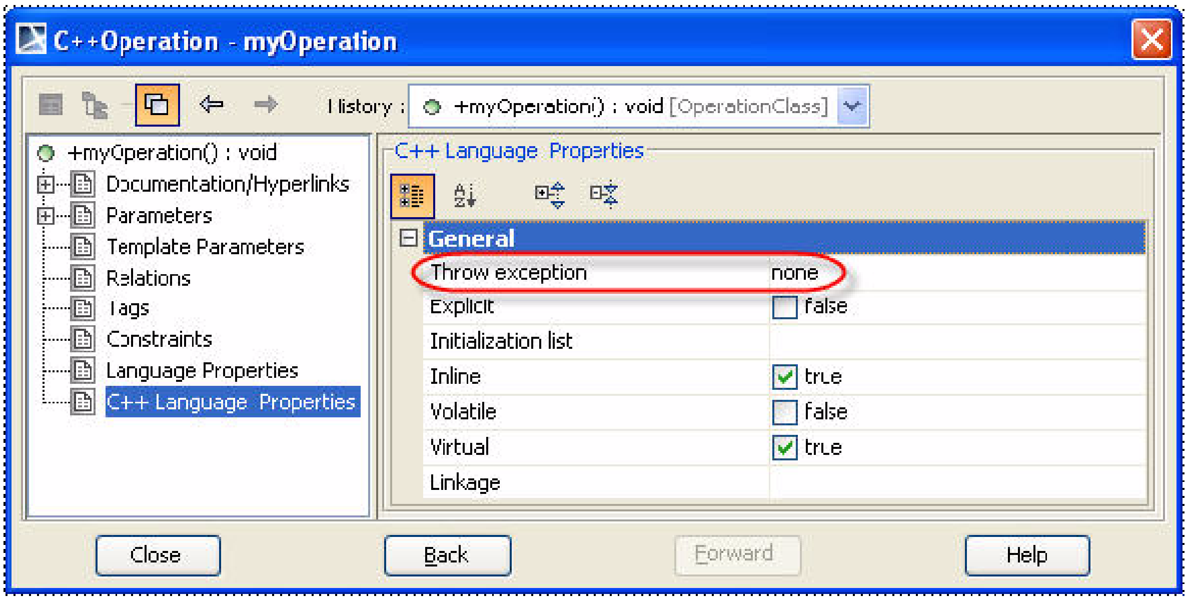
General

Throw exception	
Explicit	true
Initialization list	myAttribute(n)
Inline	false
Volatile	false
Virtual	false
Linkage	

Figure 110 -- tag values in Constructor

Copyright © 1998-2015 No Magic, Inc.

There is **C++ThrownExceptions** tag value that translate to Operation tag value.



C++ Language Properties

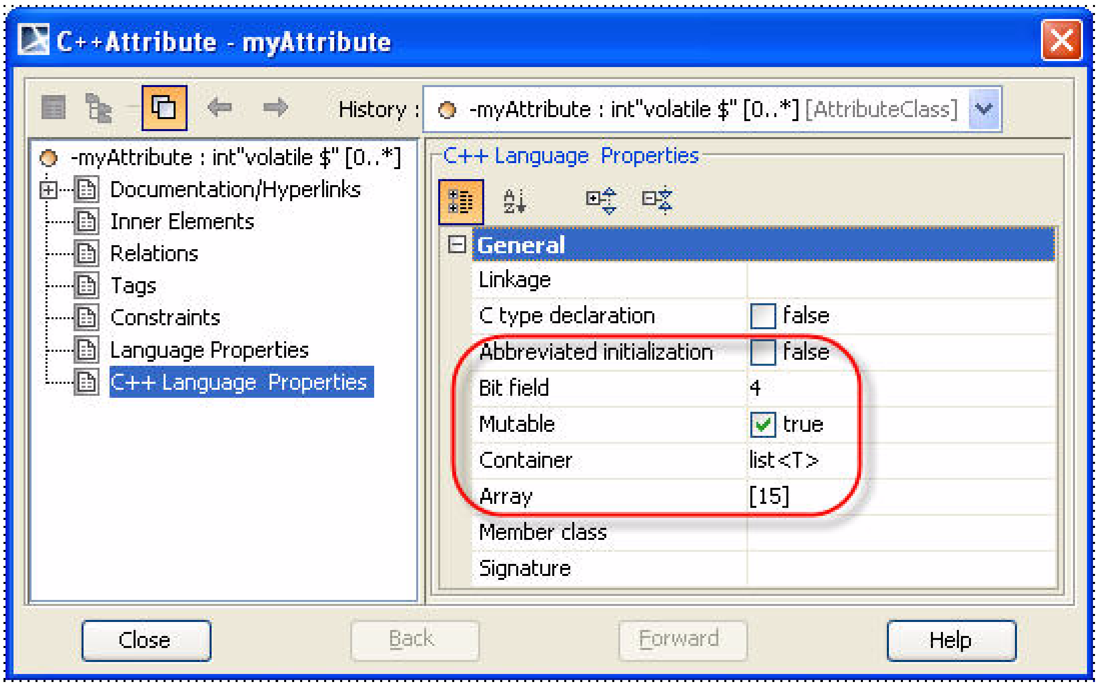
General

Throw exception	none
Explicit	false
Initialization list	
Inline	✓ true
Volatile	false
Virtual	✓ true
Linkage	

Figure 111 -- Throw exception tag value in Operation

Attribute

There are **Abbreviated initialization**, **Bit field**, **Mutable** and **Container** in Attribute Language Properties and **Array** from **Type Modifier** that translate to Attribute tag values.


C++Attribute - myAttribute
History : -myAttribute : int"volatile $" [0..*] [AttributeClass]
-myAttribute : int"volatile $" [0..*]
Documentation/Hyperlinks
Inner Elements
Relations
Tags
Constraints
Language Properties
C++ Language Properties
Close
Back
Forward
Help

C++ Language Properties

General

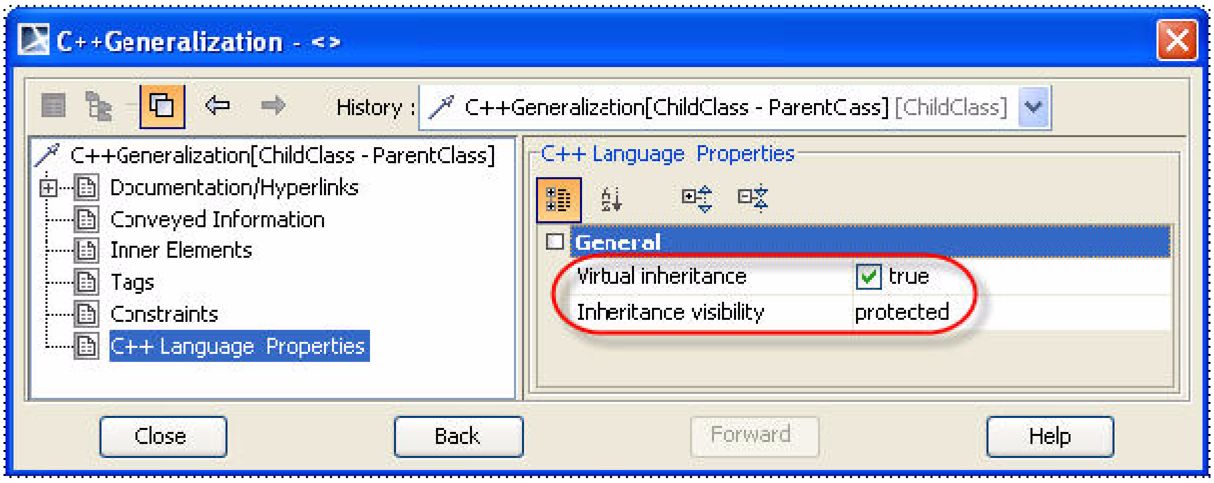
Linkage	
C type declaration	false
Abbreviated initialization	false
Bit field	4
Mutable	✓ true
Container	list<T>
Array	[15]
Member class	
Signature	

Figure 112 -- tag values in Attribute

Copyright © 1998-2015 No Magic, Inc.

Generalization

There are **Virtual inheritance** and **Inheritance visibility** in Generalization Language Properties that translate to Generalization tag values.


C++Generalization - <>

History : C++Generalization[ChildClass - ParentCass] [ChildClass]

- C++Generalization[ChildClass - ParentClass]
  - Documentation/Hyperlinks
  - Conveyed Information
  - Inner Elements
  - Tags
  - Constraints
  - C++ Language Properties

C++ Language Properties

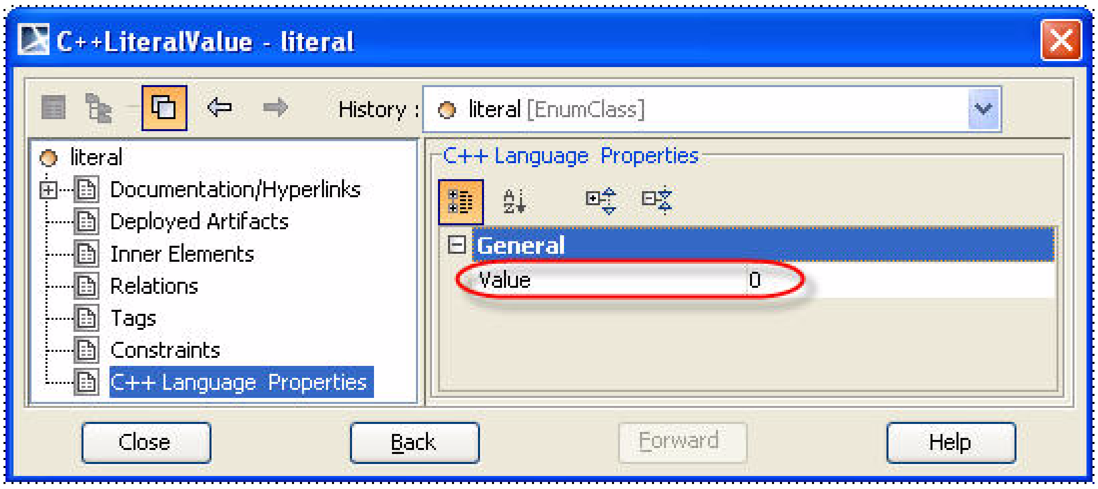
General

Virtual inheritance	✓ true
Inheritance visibility	protected

Figure 113 -- tag values in Generalization

Enumeration literal

There is value from **«C++EnumerationLiteral»** stereotype that translate to Enumeration literal tag value.


C++LiteralValue - literal

History : literal [EnumClass]

- literal
  - Documentation/Hyperlinks
  - Deployed Artifacts
  - Inner Elements
  - Relations
  - Tags
  - Constraints
  - C++ Language Properties

C++ Language Properties

General

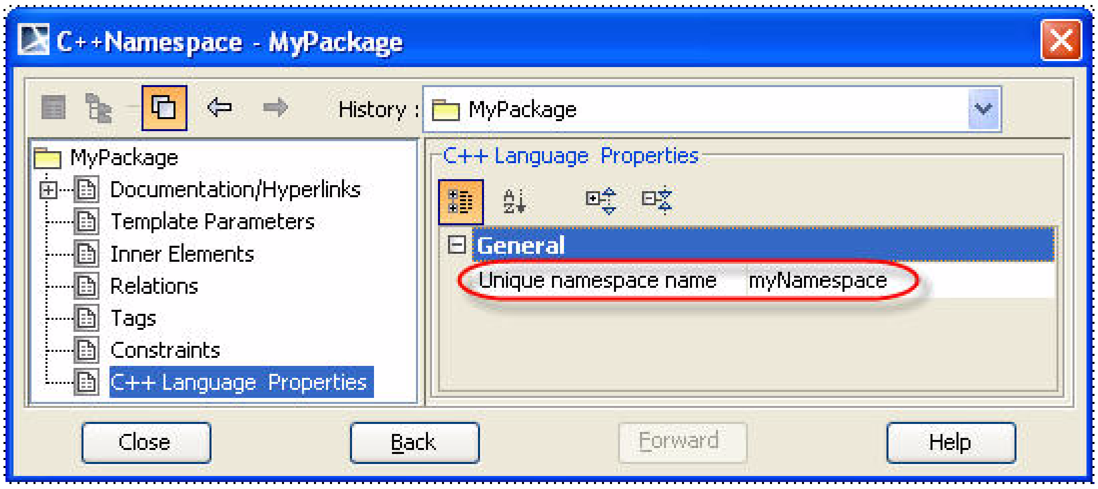
Value	0

Figure 114 -- tag value in Enumeration literal

Namespace

There is value from **«C++Namespace»** stereotype that translate to Namespace tag value.

Copyright © 1998-2015 No Magic, Inc.

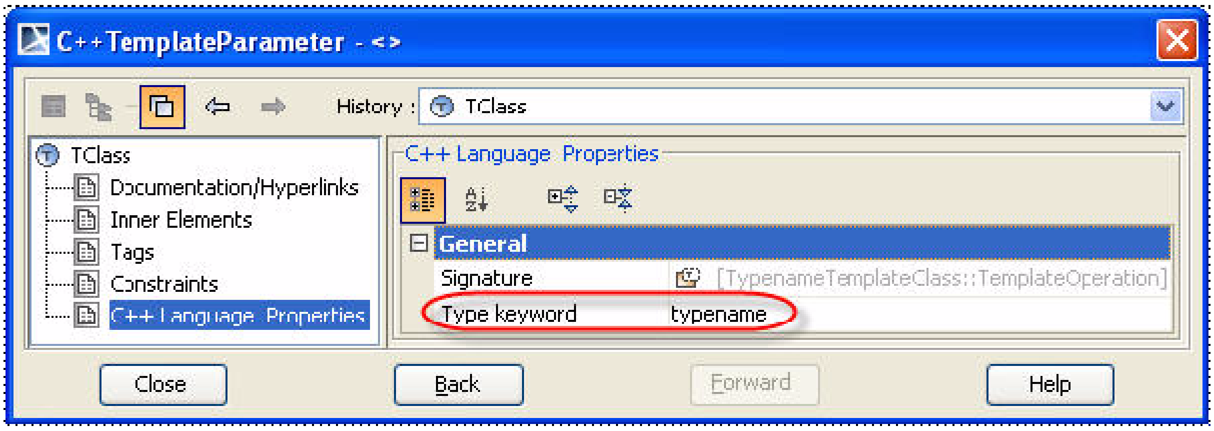


Unique namespace name: myNamespace

Figure 115 -- tag value in Namespace

Template parameter

There is value from **«C++Typename»** stereotype that translate to Template parameter tag value.



Signature: [TypenameTemplateClass::TemplateOperation]
Type keyword: typename

Figure 116 -- tag value in Template parameter

Profile constraints

This chapter describes all constraints added by the ANSI profile.

Operation

- **isQuery** set to true can only apply to member operation (non global)
- **virtual** tag set to true can only apply to non static member operation.

Constructor

operation with **«C++Constructor»** stereotype

- Name should be the same as the owner’s name.

Copyright © 1998-2015 No Magic, Inc.

Destructor

operation with «C++Destructor» stereotype

- Name should be '~' + the same name as the owner's name.

Global

Class with **«C++Global»** stereotype:

- Name length should be 0.
- All operations and attributes should be public.
- Only one global class by namespace.
- Owner of global class can only be package

Typedef

Class with **«C++Typedef»** stereotype:

- One and only one **«C++BaseType»** dependency from a **«C++Typedef»** class.
- **«C++BaseType»** supplier dependency can only be Classifier or Operation.
- **«C++Typedef»** class cannot contain attribute and operation.
- Only one inner Class or inner Enumeration is valid.

Friend

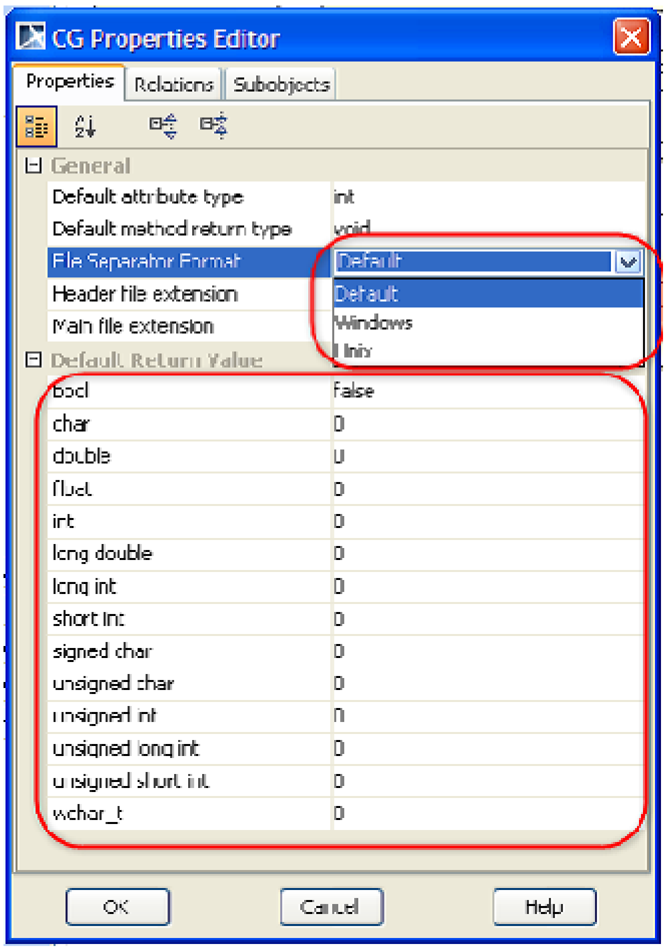
Dependency with **«C++Friend»** stereotype:

- **«C++Friend»** client dependency can only be Class or Operation.
- **«C++Friend»** supplier dependency can only be Class.

Copyright © 1998-2015 No Magic, Inc.

New in MagicDraw 12.1

CG Properties Editor



General	
Default attribute type	int
Default method return type	void
File Separator Format	Default
Header file extension	
Main file extension	
Default Return Value	
bool	false
char	0
double	0
float	0
int	0
long double	0
long int	0
short int	0
signed char	0
unsigned char	0
unsigned int	0
unsigned long int	0
unsigned short int	0
wchar_t	0

Figure 117 -- New properties

File Separator Format

This property is used to indicate the format of file separator used in code generation.

Default Return Value

Now you can set the default return value, which will be used for generating function body for new function. For example, if the default return value of bool is set to false and generated the function shown below will be created in .cpp file.

```
bool func(); in .h file,
bool func()
{
return false;
}
```

Roundtrip on #include statement and forward class declaration

From the previous version of MagicDraw, the #include statement will be generated only at the first generation of a newly created file. According to this, it prevents user from doing roundtrip code engineering. Therefore,

Copyright © 1998-2015 No Magic, Inc.

#include statement and forward class declaration are mapped to the model in MD 12.1. See the mapping section for more information.

At the generation time, the required #include statement will be generated by collecting data from model. Include generator will collect include information from

-Property

-Generalization

-Parameter

-Template Parameter Substitution

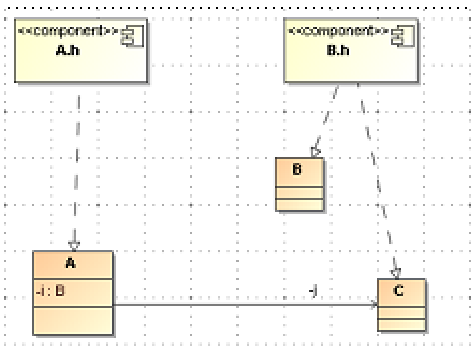
-Template Parameter default value

-Usage

Note: The usage relationship corresponding to the #include statement is also created in the model

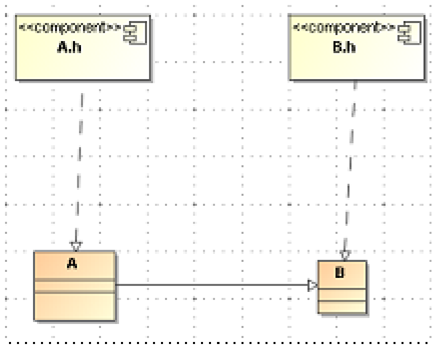
Property

Model	Code

 «component» A.h
«component» B.h
B
A
-i : B
-j
C
	`#include "B.h"`
	`class A{`
	`     B i;`
	`     C* j;`
	`};`

Include information will be collected from both normal property (i) and association end property (j).Generalization

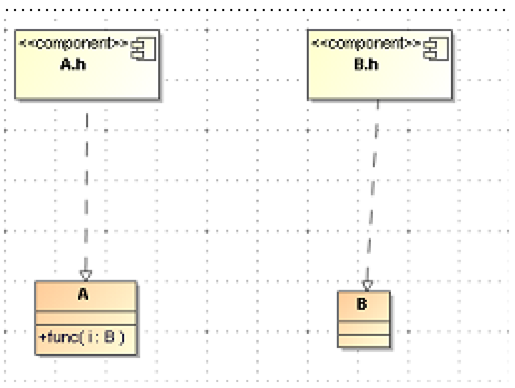
Model	Code

	`#include "B.h"`
	`class A:B{`
	
	`};`

Copyright © 1998-2015 No Magic, Inc.

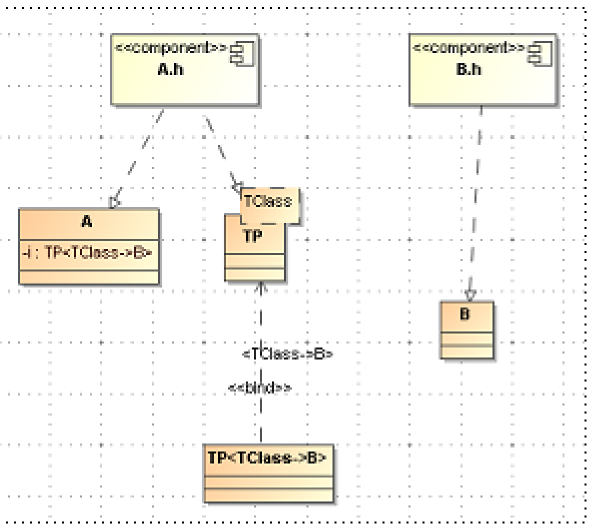
Parameter

Model	Code

 <<component>>
A.h
<<component>>
B.h
A
+func(i : B)
B
	`#include "B.h"`
	`class A{`
	`     public:`
	`            void func(B i);`
	`};`

Template Parameter Substitution

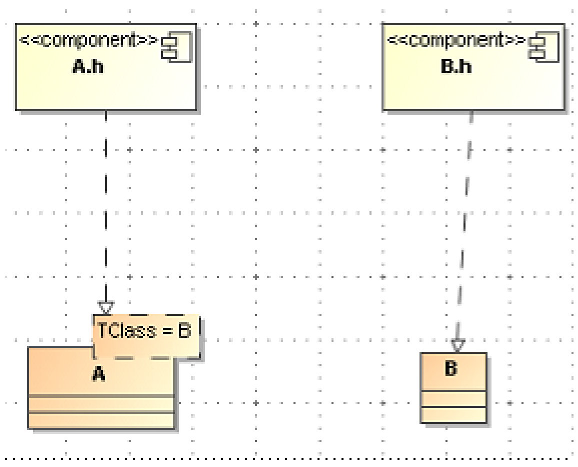
Model	Code

 <<component>>
A.h
<<component>>
B.h
A
-i : TP<TClass->B>
TClass
TP
B
<TClass->B>
<<bind>>
TP<TClass->B>
	`#include "B.h"`
	`template <class Tclass>`
	`class TP{};`
	`class A{`
	`     TP<B> i;`
	`};`

Copyright © 1998-2015 No Magic, Inc.

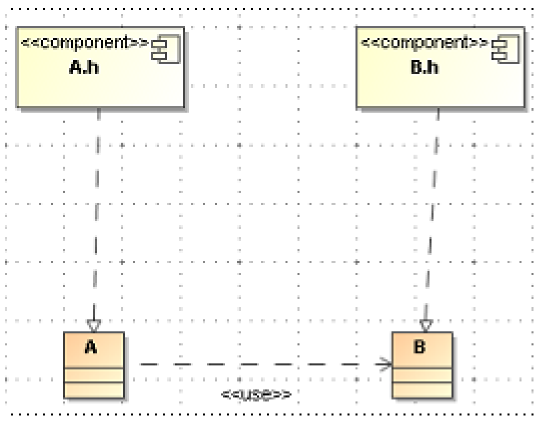
Template Parameter Default Value

Model	Code

 <<component>>
A.h
<<component>>
B.h
TClass = B
A
B
	`#include "B.h"`
	`template <class TClass=B>`
	`class A{};`

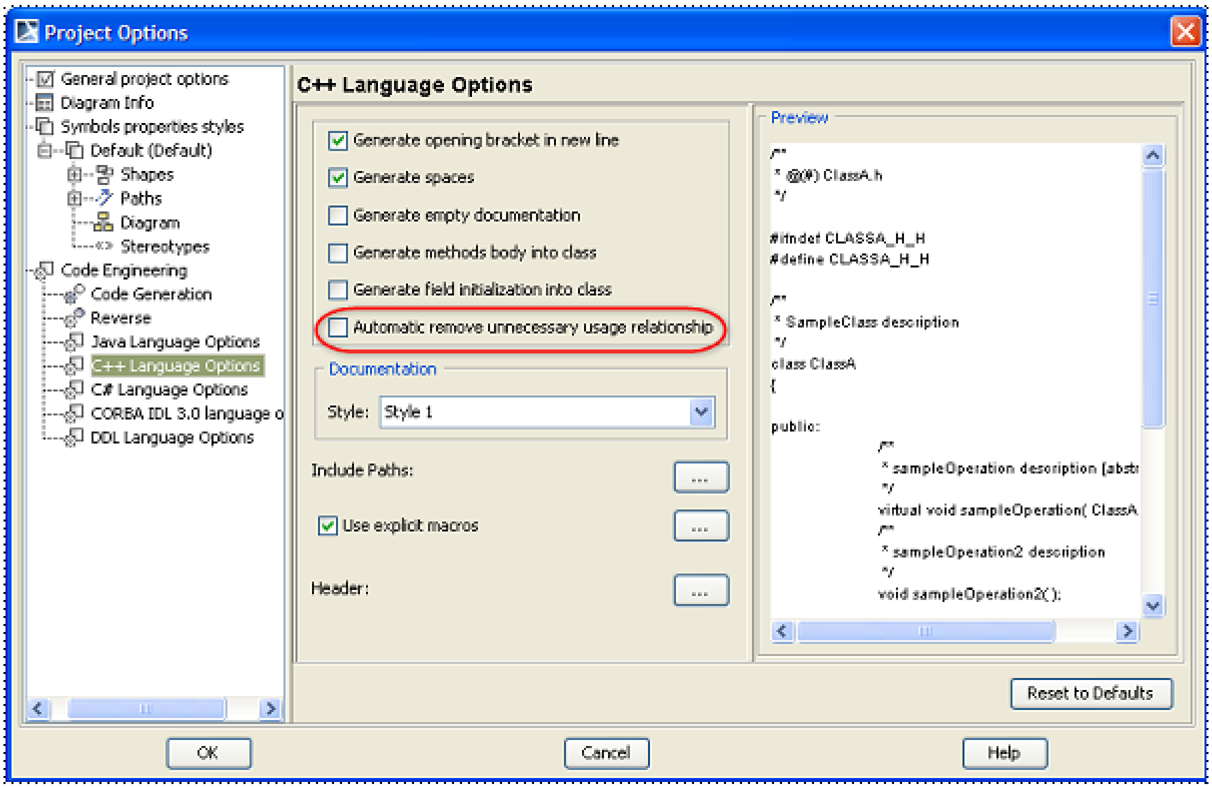
Usage

Model	Code

	`#include "B.h"`
	`class A{`
	
	`};`

Copyright © 1998-2015 No Magic, Inc.

Project Option and Code Generation Options



- [✓] Generate opening bracket in new line
- [✓] Generate spaces
- [ ] Generate empty documentation
- [ ] Generate methods body into class
- [ ] Generate field initialization into class
- [ ] Automatic remove unnecessary usage relationship

Style: Style 1

Include Paths:

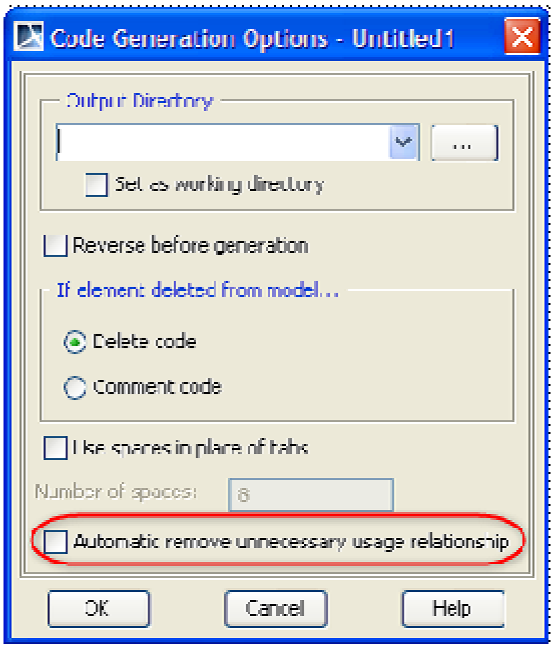
[✓] Use explicit macros

Header:

```
/**
 * @(#) ClassA.h
 */

#ifndef CLASSA_H_H
#define CLASSA_H_H

/**
 * SampleClass description
 */
class ClassA
{

public:
                /**
                 * sampleOperation description (abstr
                 */
                virtual void sampleOperation( ClassA
                /**
                 * sampleOperation2 description
                 */
                void sampleOperation2( );
```

Figure 118 -- New Option in Project Options

If this new option is selected, the same option in Code Generation Option dialog will be selected as shown below.


Code Generation Options - Untitled1
Output Directory
...
If element deleted from model...
OK
Cancel
Help

- [ ] Set as working directory
- [ ] Reverse before generation
- [✓ Delete code] / Comment code
- [ ] Use spaces in place of tabs

Number of spaces: 8

- [ ] Automatic remove unnecessary usage relationship

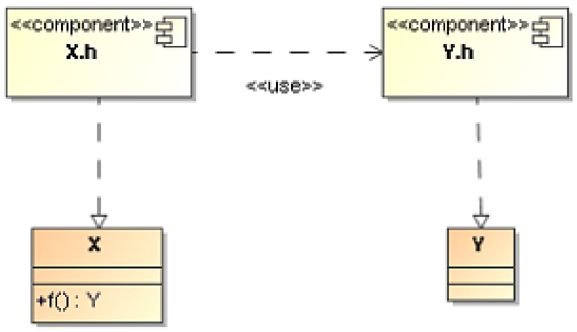
Figure 119 -- New option in Code Generation Options

If the user generates code with this option selected, code generator will automatically detect and remove unnecessary usage relationship from model. See the example below for better understanding

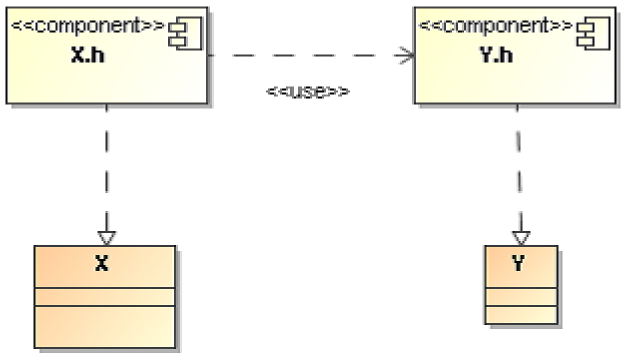
Copyright © 1998-2015 No Magic, Inc.

.

Model	**Code**

	`//X.h`
	`#include "Y.h"`
	`class X{`
	`Y f();`
	`};`

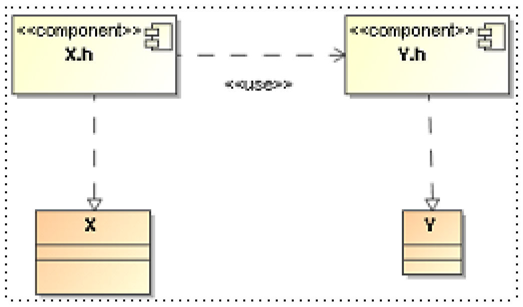
From model above, if the user removes function f():Y from class X, in the model point of view, the usage from X.h to Y.h is not necessary.

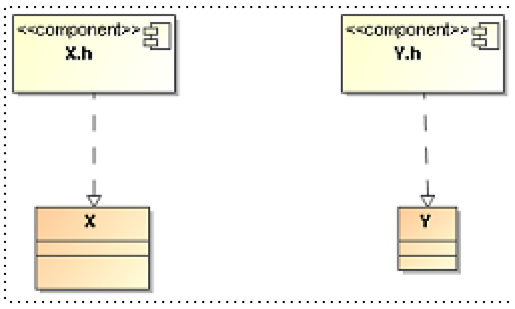


Copyright © 1998-2015 No Magic, Inc.

See the difference between outcomes when selecting and not selecting Automatic remove unnecessary usage relationship.

Automatic remove unnecessary usage relationship	**Model**	**Code**

Selected		`//X.h`
		`#include "Y.h"`
		`class X{};`

Not Selected		`//X.h`
		`class X{};`

Copyright © 1998-2015 No Magic, Inc.

New in MagicDraw 14.0

Support C++ dialects

C++ code engineering set in MagicDraw 14.0 supports three dialects

- ANSI – conform to ISO/IEC 14882 specification for C++ programming language
- CLI – conform to ECMA-372 C++/CLI Language Specification base on Microsoft Visual studio 2005
- Managed – conform to Managed Extensions for C++ Specification introduced in Microsoft Visual Studio 2003

Note	The syntax under Managed dialect is deprecated in Microsoft Visual studio 2005.

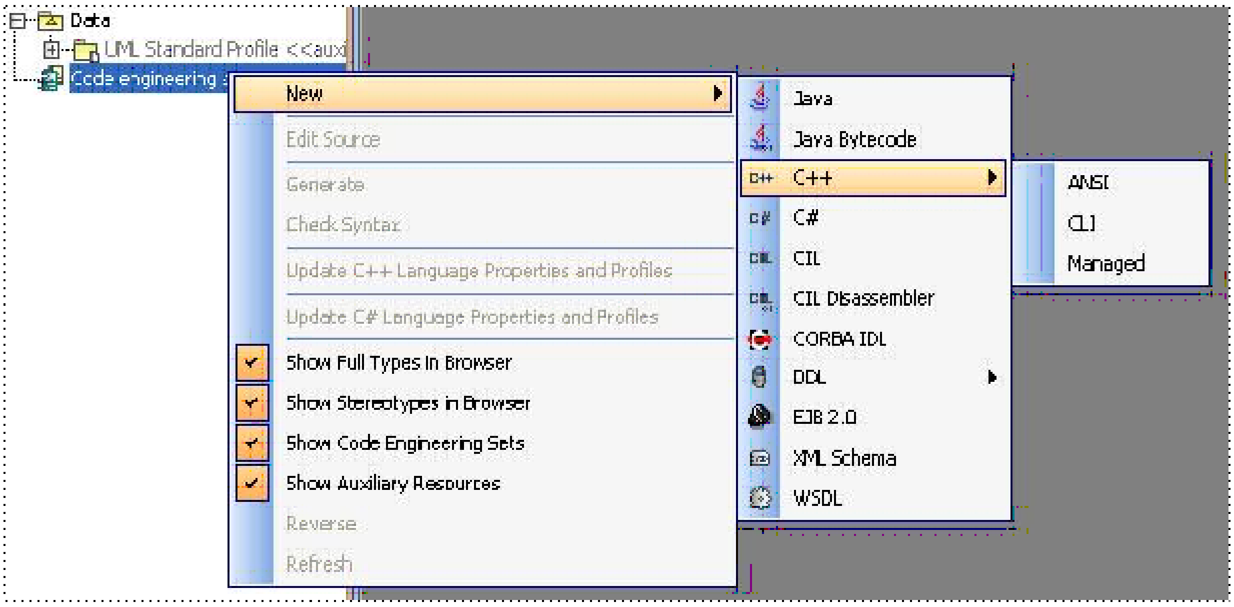
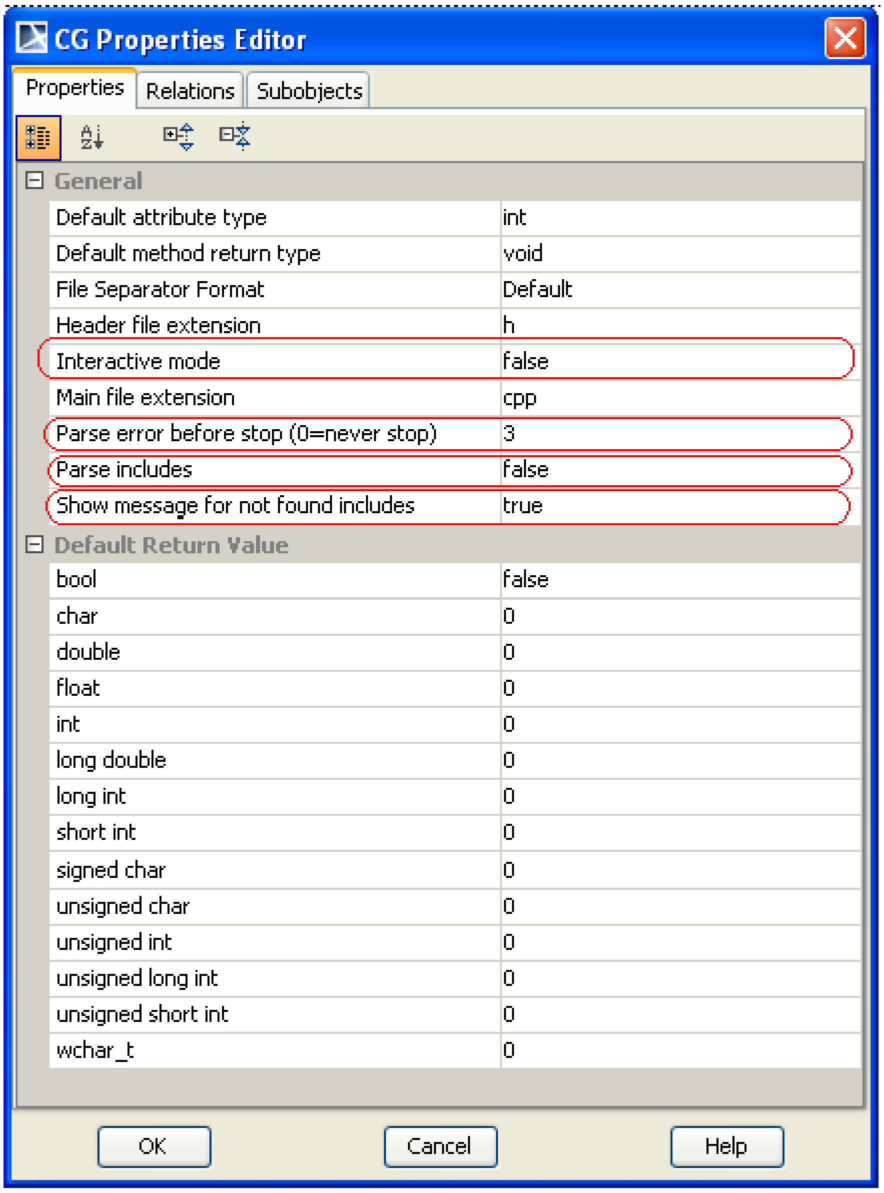

Data
UML Standard Profile <<aux
Code engineering
New
Edit Source
Generate
Check Syntax
Update C++ Language Properties and Profiles
Update C# Language Properties and Profiles
Show Full Types In Browser
Show Stereotypes in Browser
Show Code Engineering Sets
Show Auxiliary Resources
Reverse
Refresh
Java
Java Bytecode
C++
C#
CIL
CIL Disassembler
CORBA IDL
DDL
EJB 2.0
XML Schema
WSDL
ANSI
CLI
Managed

Figure 120 -- New dialects in C++ CE

Copyright © 1998-2015 No Magic, Inc.

CG Properties Editor

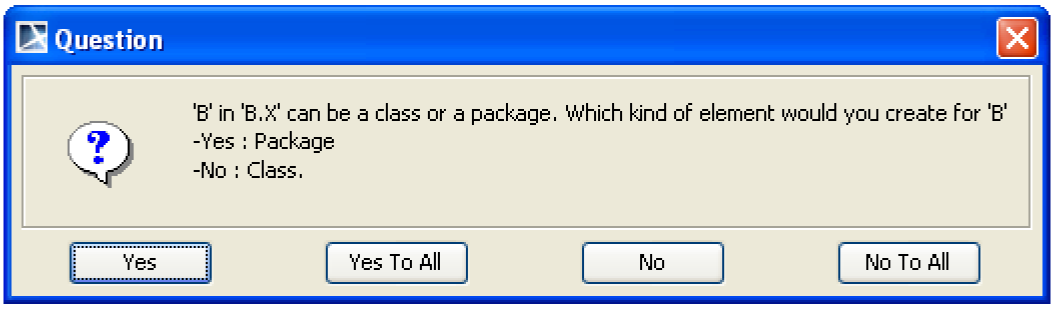


General	
Default attribute type	int
Default method return type	void
File Separator Format	Default
Header file extension	h
Interactive mode	false
Main file extension	cpp
Parse error before stop (0=never stop)	3
Parse includes	false
Show message for not found includes	true
Default Return Value	
bool	false
char	0
double	0
float	0
int	0
long double	0
long int	0
short int	0
signed char	0
unsigned char	0
unsigned int	0
unsigned long int	0
unsigned short int	0
wchar_t	0

Figure 121 -- New properties

Copyright © 1998-2015 No Magic, Inc.

Property name	Description

Interactive mode	If enabled, user will be questioned during the reverse process when the parser cannot decide whether the symbol is a class or namespace.
	
	For example
	
	If you reverse the following code
	`class A {`
	`        B::X i;`
	`};`
	The parser does not have enough information to decide whether B is class or namespace.
	
	Question
	'B' in 'B.X' can be a class or a package. Which kind of element would you create for 'B'
	-Yes : Package
	-No : Class.
	
	Figure 122 -- Question dialog for interactive mode
	NOTE If the default mode, not interactive mode, the reverse module will automatically do the best guess according to the information in code.

Parse error before stop	This property allows users to specify the number of errors to ignore before the reverse process will be terminated. The user can set the property to zero to allow CE to reverse all code before stopping.
Parse Includes	If set to false, the reverse module will reverse only the selected files and ignore all #include statements.
Show message for not found includes	Used to display messages in the message window when the reverse module cannot find the files included in the #include statement. **NOTE** You have to set the property **Parse Includes** to true in order to use the property **Show message for not found includes**.

New in MagicDraw 16.8

Doxygen-after-member documentation

NOTE	This feature only works for import-code-only mode.

A doxygen documentation after a member is reversed now, the comment after the member should use the special doxygen tag used to mark a documentation after a member.

- '/**<'
- '///<'

Copyright © 1998-2015 No Magic, Inc.

- '//!<'

@see on documentation

NOTE	This feature only works for import-code-only mode.

Documentation with doxygne @see tag is reversed with a link to the model. The documentation should use one of the following special doxygen tag to describe an @see element.

- @see
- \see
- @sa
- \sa

NOTE	The documentation must be in the form of “@see element”. Only one element follows @see will be reversed as an element link in model documentation.

Symbian macro

The following macro definitions have been added to the explicit macro list.

```
// Symbian macro
#define _L(string) TPtrC((const TText*) string)
#define _LIT(name,s) const static TListC<sizeof(s)> name = {sizeof(s) -1,s}
#define _LIT8(name,s) const static TListC<sizeof(s)> name = {sizeof(s) -1,s}

#define IMPORT_C /*__declspec(dllimport)*/
#define EXPORT_C /*__declspec(dllexport)*/

#ut NONSHARABLE_CLASS(x) class x
#define NONSHARABLE_STRUCT(x) struct x

#define GLREF_D extern
#define GLDEF_D
#define LOCAL_D static
#define GLREF_C extern
#define GLDEF_C
#define LOCAL_C static
```

Tutorial

Type Modifier

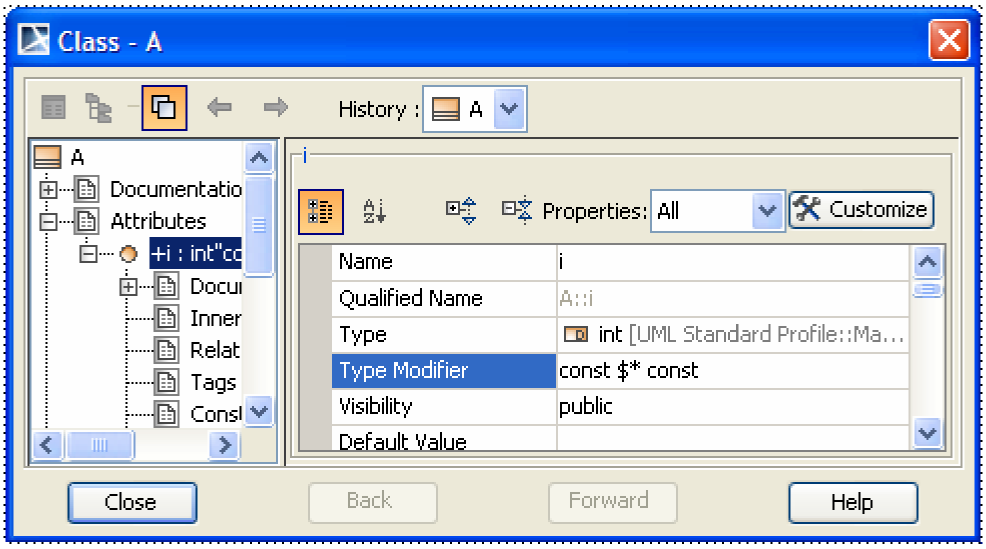
In order to correctly generate code, the user has to put type modifier in the correct format. The dollar sign, $, will be replaced with Type.

For example, If you want to generate the following code,

```
class A
{
        const int* const i;
};
```

Copyright © 1998-2015 No Magic, Inc.

type modifier must be set to “const $* const” as shown in the figure below.


Class - A

History : A

Properties: All

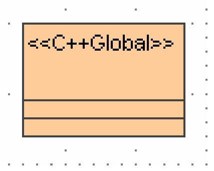
Name	i
Qualified Name	A::i
Type	int [UML Standard Profile::Ma...
Type Modifier	const $* const
Visibility	public
Default Value	

Figure 123 -- Type Modifier Example

Global Member

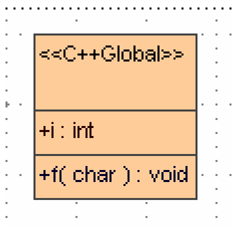
The new C++ ANSI profile allows you to model C++ global member. In order to do this, please follow the steps below.

1. Create a class
2. Apply stereotype «C++Global»



Note	You can leave the class unnamed. Name of «C++Global» class does not have any effect on code generation.

3. Create members, attribute and function in this class. All members should set their visibilities to Public.



See "Global functions and variables", on page 88 for example.

Typedef

With new C++ ANSI profile, you can model both normal Typedef and Typedef of function pointer.

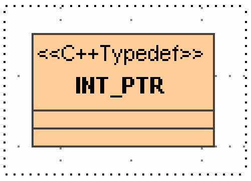
Copyright © 1998-2015 No Magic, Inc.

Normal Typedef

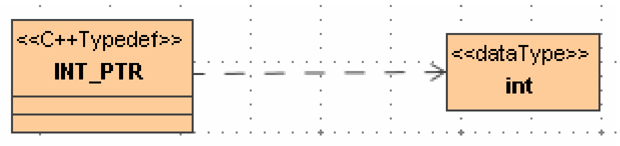
The steps below show you how to create the model that can be used to generate the following code.

```
typedef int* INT_PTR;
```

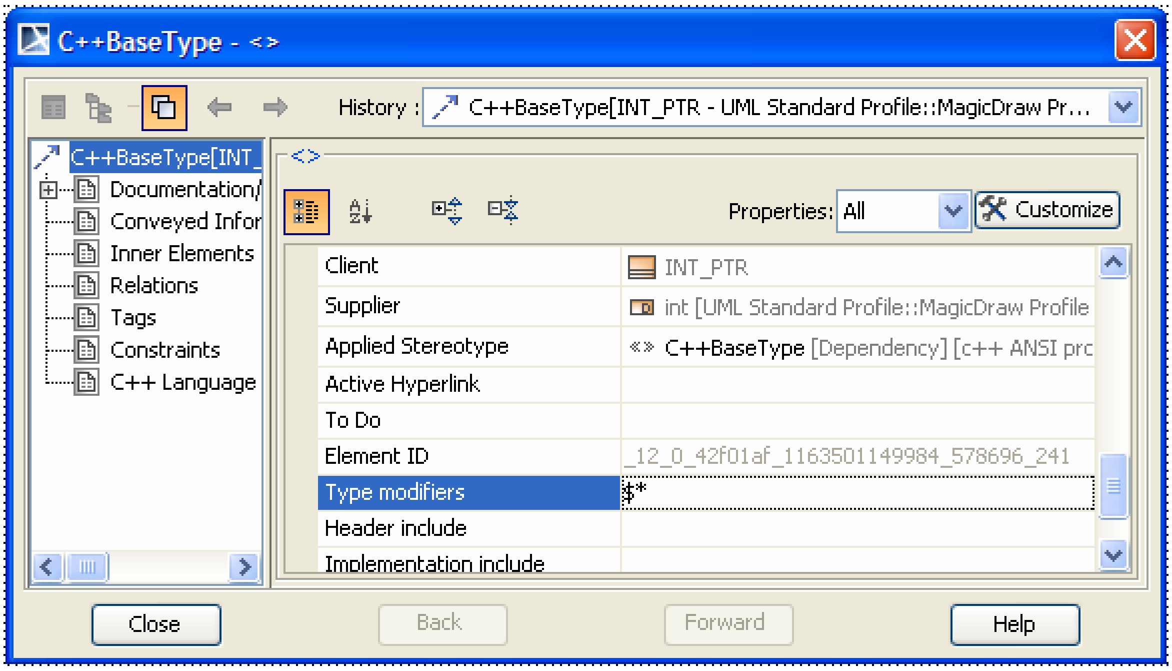
1. Create a class, apply stereotype «C++Typedef» and name it with typedef name. In this case, INT_PTR.



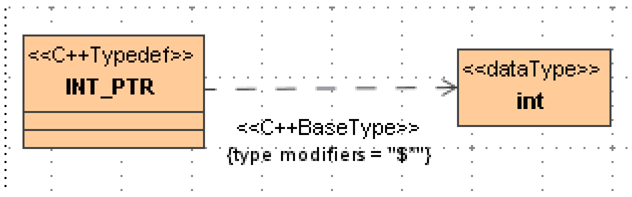
2. Draw Dependency relationship from such class to base type element. In this case, int.



3. Apply the Dependency with «C++BaseType» and set type modifier value of Dependency to "$*".



Client	INT_PTR
Supplier	int [UML Standard Profile::MagicDraw Profile
Applied Stereotype	«» C++BaseType [Dependency] [c++ ANSI prc
Active Hyperlink	
To Do	
Element ID	_12_0_42f01af_1163501149984_578696_241
Type modifiers	$*
Header include	
Implementation include	



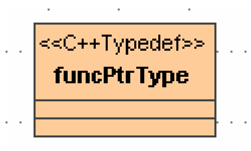
Typedef of function pointer

The steps below show you how to create the model that can be used to generate the following code.

Copyright © 1998-2015 No Magic, Inc.

```
typedef double (*funcPtrType)(int, char);
```

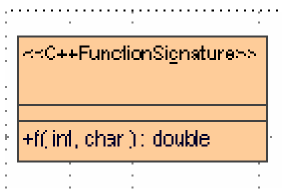
1. Create a class, apply stereotype «C++Typedef» and name it with typedef name. In this case, funcPtrType.



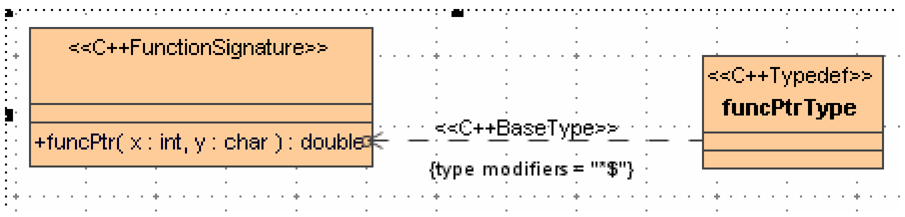
2. Create an operation with function signature type of function pointer in «C++FunctionSignature» class. The operation name does not have any effect on code generation.

Note If «C++FunctionSignature» class does not exist, create a new one.

.



3. Draw Dependency relationship from **funcPtrType** class to f**(int, char): double,** apply «C++BaseType» to the Dependency and set type modifier to *$.



See Typedef for mapping example.

Function Pointer

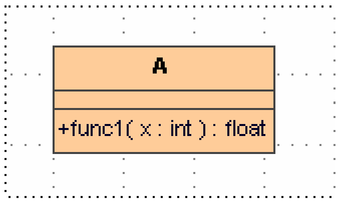
The steps below show you how to create the model that can be used to generate the following code.

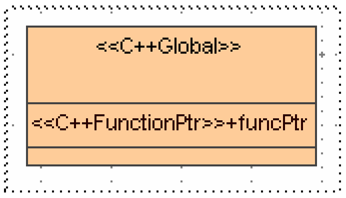
```
class A
{
 public:
       float func1(int x);
};
float (A*funcPtr)(int);
```

Suppose that the model for class A is already created.

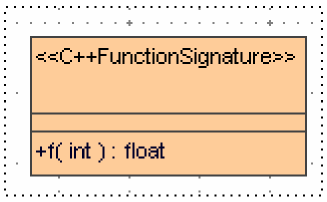
Copyright © 1998-2015 No Magic, Inc.

1. Create attribute with name, in this case, funcPtr and apply «C++FunctionPtr» stereotype. In this example, funcPtr is created in «C++Global» class means that funcPtr is global member.

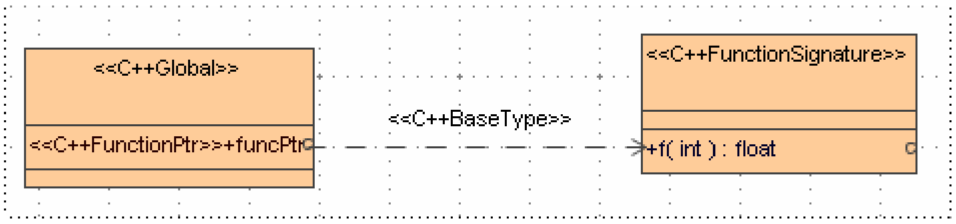




2. Create an operation with function signature type of function pointer in «C++FunctionSignature» class. The operation name does not have any effect on code generation.

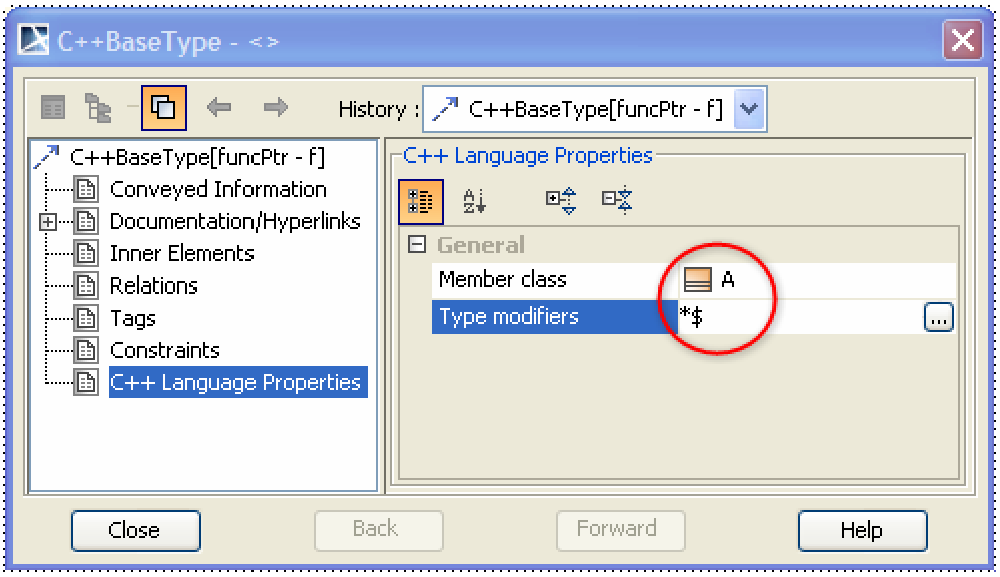


3. Draw a Dependency from funcPtr to f that is just created in «C++FunctionSignature» class and set «C++BaseType» stereotype to the Dependency.



Copyright © 1998-2015 No Magic, Inc.

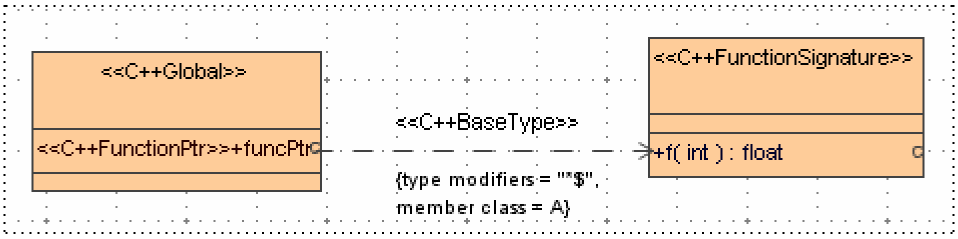
4. Open specification dialog of «C++BaseType» Dependency.
   Put *$ as value of Type Modifier in C++ Language Properties.
   Set Member class to class A.



C++ Language Properties

General

Member class	A
Type modifiers	*$



See 3.16 Function pointer for mapping example.

Friend

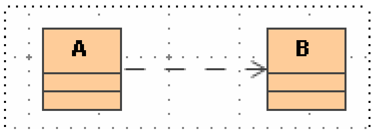
With new C++ ANSI profile, you can model C++ friend, both friend class and friend function.

Friend Class

The steps below show you how to create the model that can be used to generate the following code.

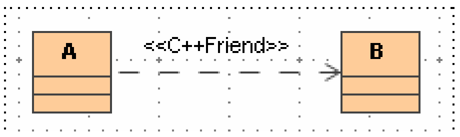
```
class A{};
class B
{
     friend class A;
};
```

1. From the existing classes, A and B, draw dependency from class **A** to class **B**.



Copyright © 1998-2015 No Magic, Inc.

2. Apply stereotype «C++Friend» to the dependency

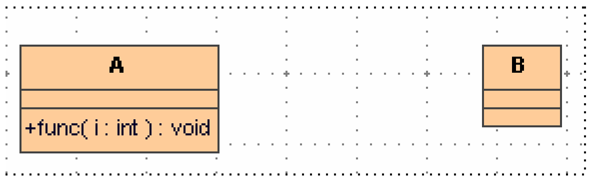


Friend Function

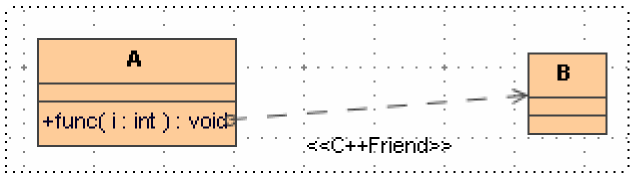
To model friend function, do the same steps as modeling friend class.

```
class A
{
        void func(int i);
};
class B
{
        friend void A::func(int i);
};
```

1. From the existing classes, A and B, draw dependency from **func** in class **A** to class **B**.



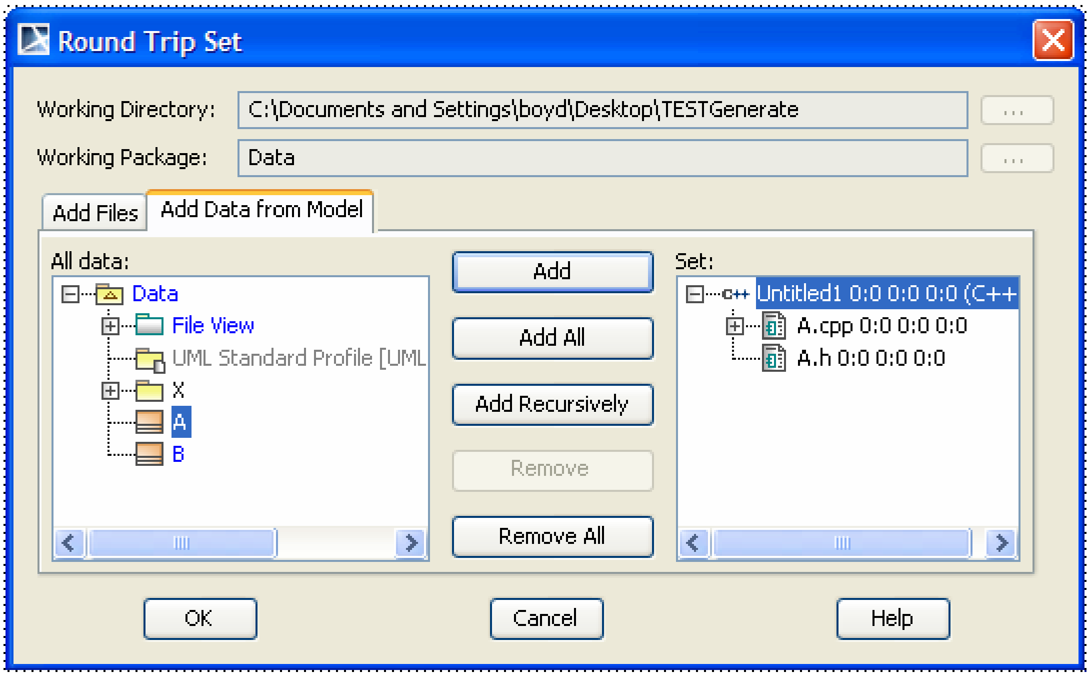
2. Apply stereotype «C++Friend» to the dependency.



See Friend declaration for mapping example.

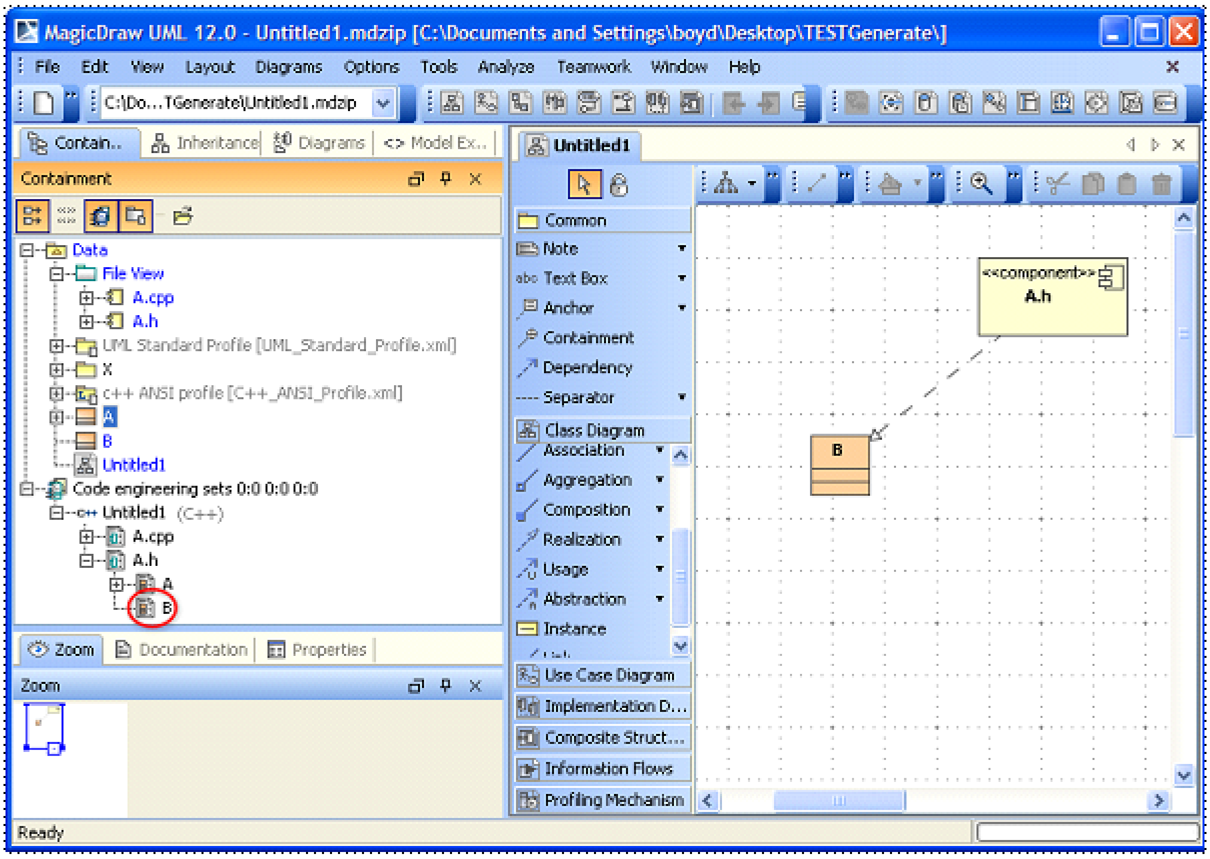
How to specify component to generate code to

For general case, if the user adds data from a model in Round Trip Set (as shown below), the default component, in this case **A.cpp** and **A.h**, will be added to code engineering set, which means that on code generation, class A and its members will be created.



However, MagicDraw allows user to specify the component to generate the model into by using Realization.

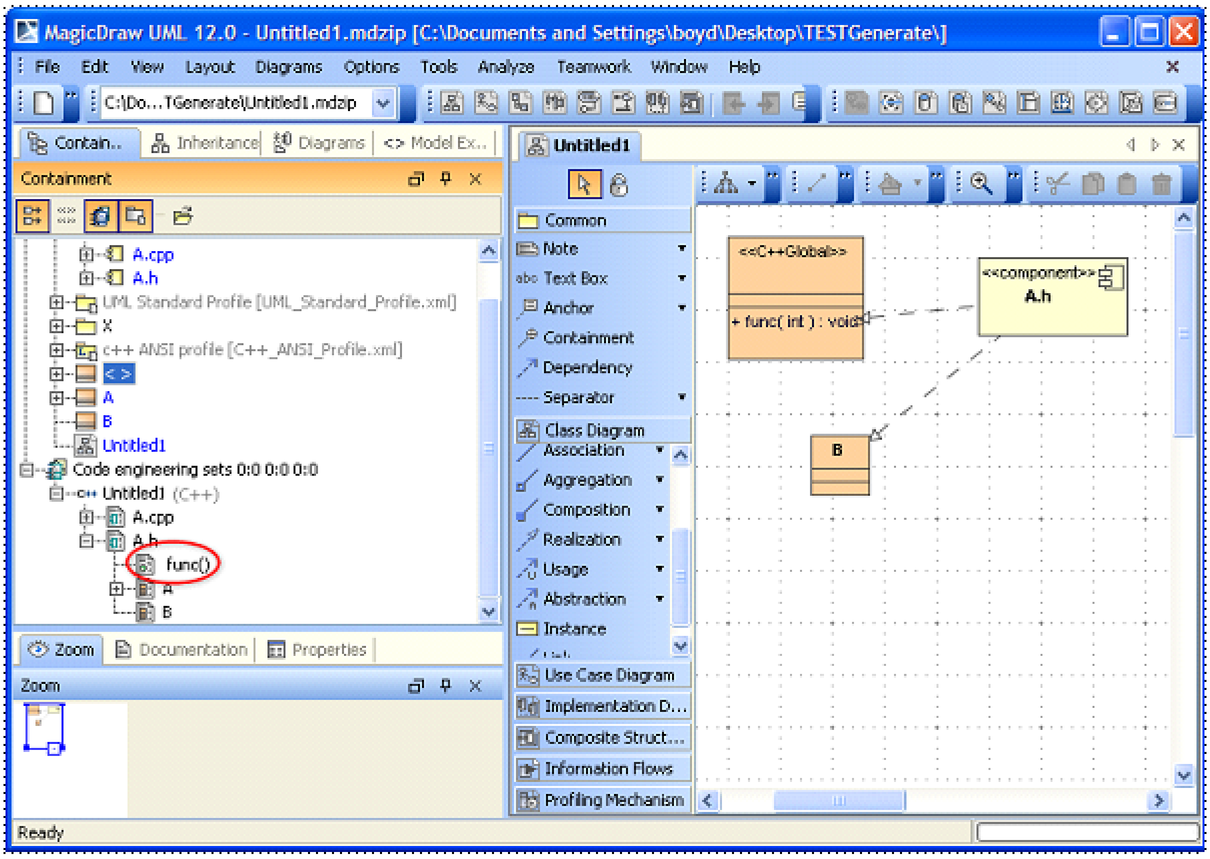
See the figure below for better understanding.


MagicDraw UML 12.0 - Untitled1.mdzip [C:\Documents and Settings\boyd\Desktop\TESTGenerate\]
File Edit View Layout Diagrams Options Tools Analyze Teamwork Window Help
C:\Do...TGenerate\Untitled1.mdzip
Contain.. Inheritance Diagrams Model Ex..
Containment
Data
File View
A.cpp
A.h
UML Standard Profile [UML_Standard_Profile.xml]
X
c++ ANSI profile [C++_ANSI_Profile.xml]
A
B
Untitled1
Code engineering sets 0:0 0:0 0:0
Untitled1 (C++)
A.cpp
A.h
A
B
Zoom Documentation Properties
Zoom
Ready
Untitled1
Common
Note
Text Box
Anchor
Containment
Dependency
Separator
Class Diagram
Association
Aggregation
Composition
Realization
Usage
Abstraction
Instance
Use Case Diagram
Implementation D...
Composite Struct...
Information Flows
Profiling Mechanism
«component»
A.h
B

Copyright © 1998-2015 No Magic, Inc.

When drawing the Realization from component A.h to class B. Class B will be added to code engineering set as you can see in the red circle, which means that class B will be generated in A.h.

You can also specify component to generate global member as shown in the figure below.



@see support for import-code-only mode

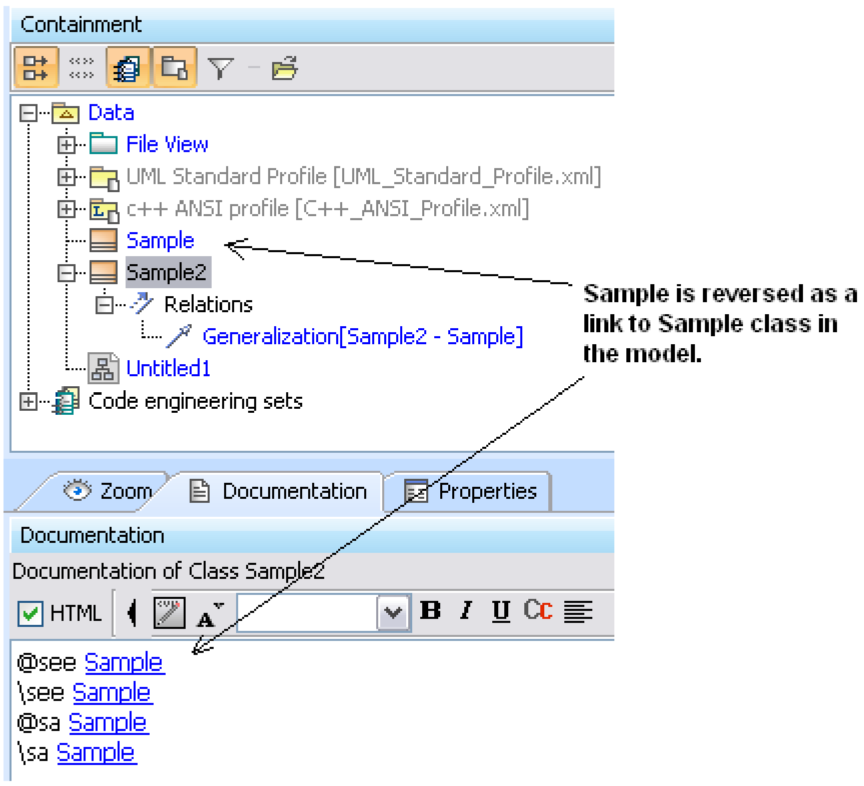
Documentation with doxygne @see tag is reversed with a link to the model. The documentation should use one of the following special doxygen tag to describe an @see element.

- @see
- \see
- @sa
- \sa

Copyright © 1998-2015 No Magic, Inc.

Example:

```
/**
 * Sample class
  */
class Sample {};

/**
 * @see Sample
 * \see Sample
 * @sa Sample
 * \sa Sample
 */
class Sample2: Sample {};
```


Containment
Data
File View
UML Standard Profile [UML_Standard_Profile.xml]
c++ ANSI profile [C++_ANSI_Profile.xml]
Sample
Sample2
Relations
Generalization[Sample2 - Sample]
Untitled1
Code engineering sets
Zoom
Documentation
Properties
Documentation
Documentation of Class Sample2
HTML
Sample is reversed as a link to Sample class in the model.

@see Sample
\see Sample
@sa Sample
\sa Sample

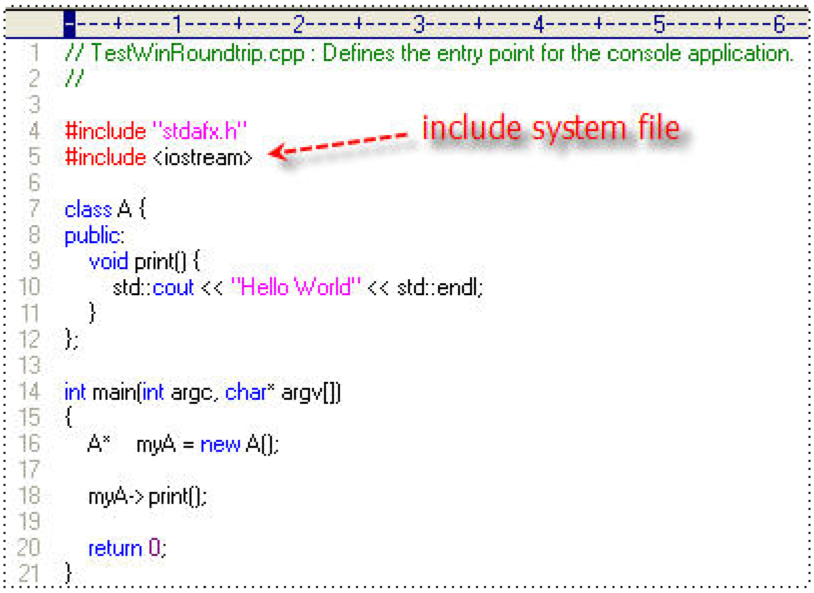
Navigable short cut from model link in documentation

- **Ctrl + click**
  MagicDraw will open the specification dialog of the model that the link point to.
- **Ctrl + Alt + click**
  MagicDraw will navigate to the documentation of the model that the link point to.

Copyright © 1998-2015 No Magic, Inc.

Project constraint

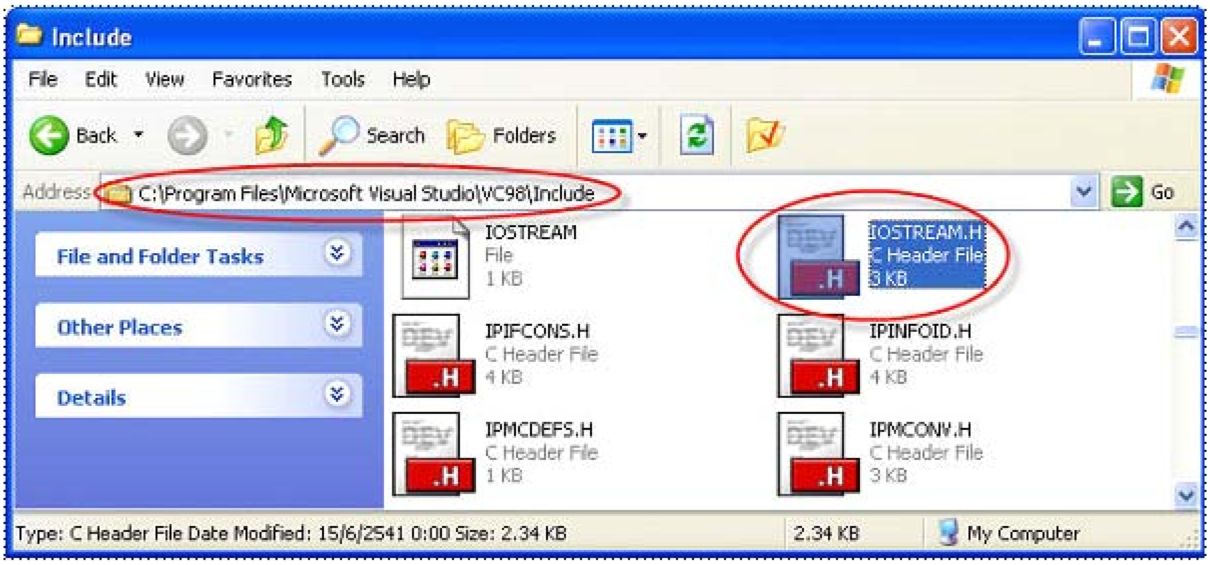
This chapter describes how to set **include paths** for include **system file**.



```
// TestWinRoundtrip.cpp : Defines the entry point for the console application.
//

#include "stdafx.h"
#include <iostream>

class A {
public:
    void print() {
        std::cout << "Hello World" << std::endl;
    }
};

int main(int argc, char* argv[])
{
    A*    myA = new A();

    myA->print();

    return 0;
}
```

Figure 124 -- Include system file example

If you include any system file, you need to set system file path as well.



Include
File Edit View Favorites Tools Help
Back Search Folders Go
Address C:\Program Files\Microsoft Visual Studio\VC98\Include
File and Folder Tasks
Other Places
Details
IOSTREAM File 1 KB
IOSTREAM.H C Header File 3 KB
IPIFCONS.H C Header File 4 KB
IPINFOID.H C Header File 4 KB
IPMCDEFS.H C Header File 1 KB
IPMCONV.H C Header File 3 KB
Type: C Header File Date Modified: 15/6/2541 0:00 Size: 2.34 KB
2.34 KB
My Computer

Figure 125 -- System file paths

To set **include path**

1. From the **Options** menu choose **Project**.
2. Expand **Code Engineering**, choose **C++ Language Options**. Select the Include path check-box. **Use include paths** button (**…**) appears on the right side of the Project options dialog box. Click this button.
3. The **Set Include Path** dialog appears.

Copyright © 1998-2015 No Magic, Inc.

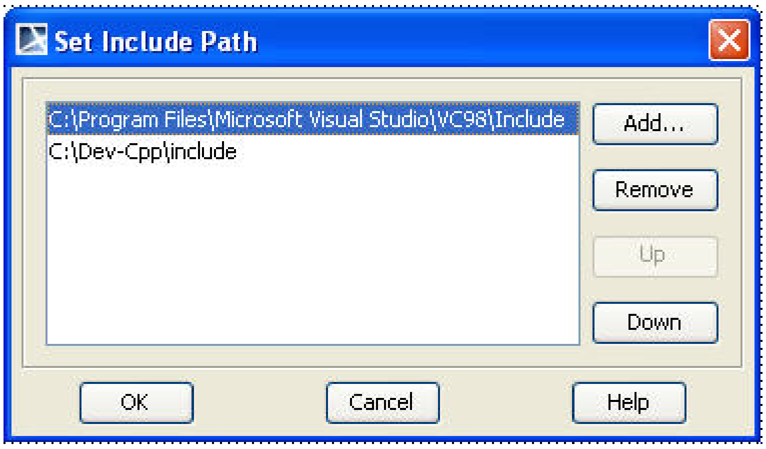
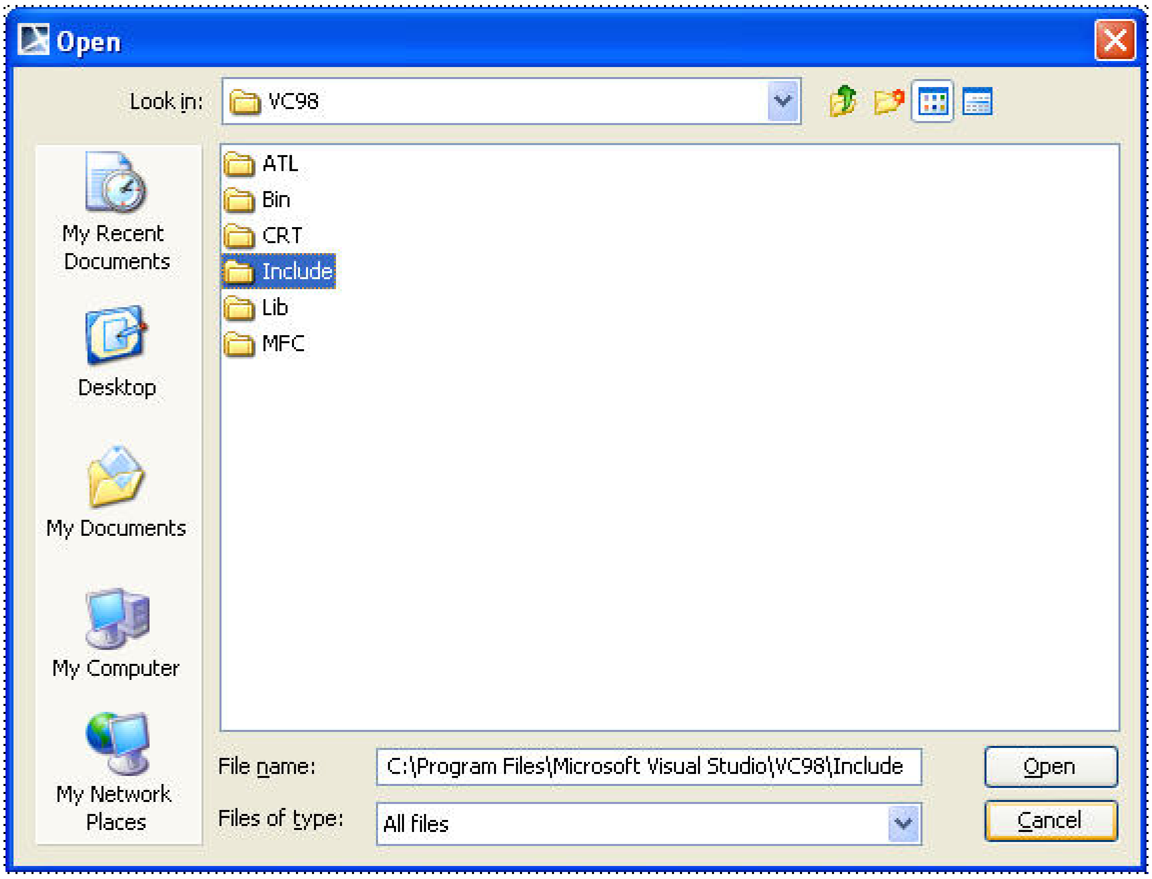


Figure 126 -- Set Include Path dialog box

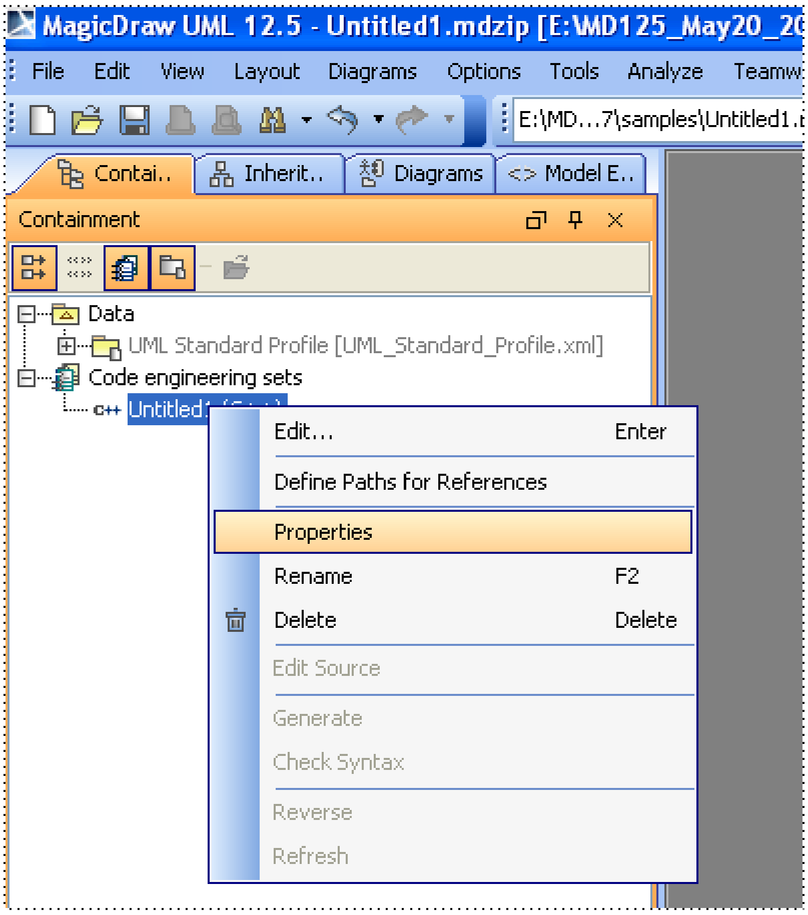
Button	**Action**
Add	The **Open** dialog box appears for select directory path.



Remove	Remove the selected Include path.
Up	Move the selected Include path upward.
Down	Move the selected Include path downward.

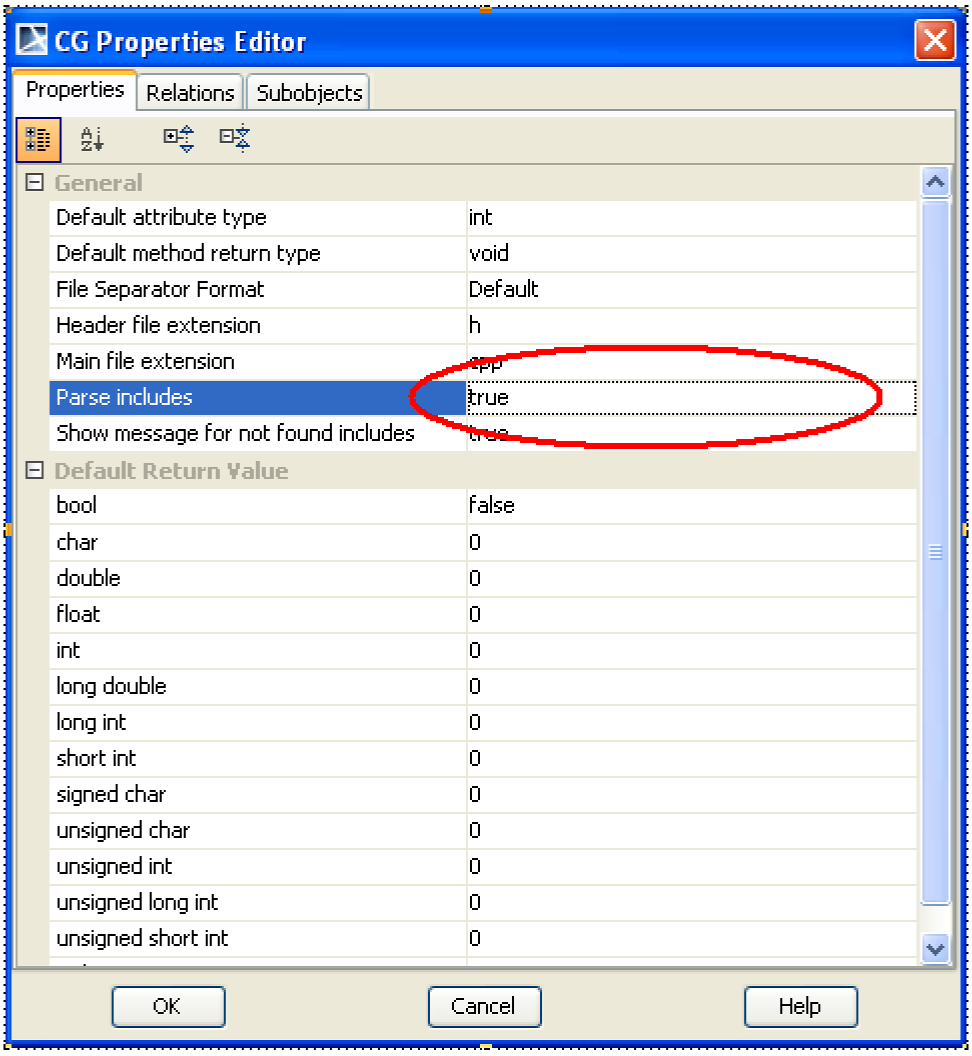
Then select the code-engineering set and then select **Properties**.

Copyright © 1998-2015 No Magic, Inc.



Copyright © 1998-2015 No Magic, Inc.

Finally set the “Parse includes” property to “true”.



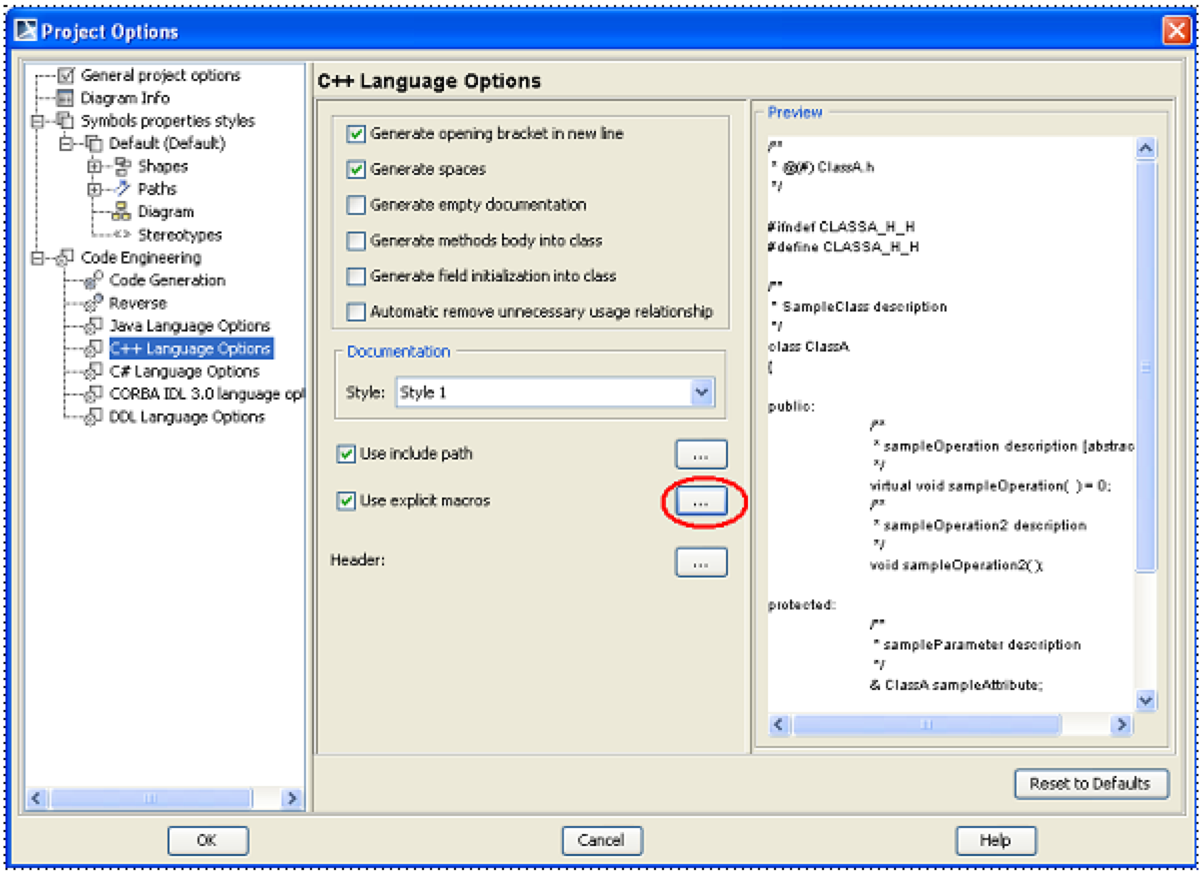
General	
Default attribute type	int
Default method return type	void
File Separator Format	Default
Header file extension	h
Main file extension	cpp
Parse includes	true
Show message for not found includes	true
Default Return Value	
bool	false
char	0
double	0
float	0
int	0
long double	0
long int	0
short int	0
signed char	0
unsigned char	0
unsigned int	0
unsigned long int	0
unsigned short int	0

Note	If the user sets the “Parse Includes” property to “false”, then during reverse engineering of the C++ code, MagicDraw will not parse inside the header file and all the unknown type will be created in the “Default” package of the model. This way the user can solve some keyword specific problems in the library, speed up the reverse process, and remove unnecessary model inside the header file.

Copyright © 1998-2015 No Magic, Inc.

Working with QT

To reverse engineer QT code, we recommend you set **Parse Includes** option to "false". You will also need to set MagicDraw preprocessor to skip some QT macros ("Options"-> "Project"-> "C++ Language Options"-> "Use explicit macros").



- [✓] Generate opening bracket in new line
- [✓] Generate spaces
- [ ] Generate empty documentation
- [ ] Generate methods body into class
- [ ] Generate field initialization into class
- [ ] Automatic remove unnecessary usage relationship

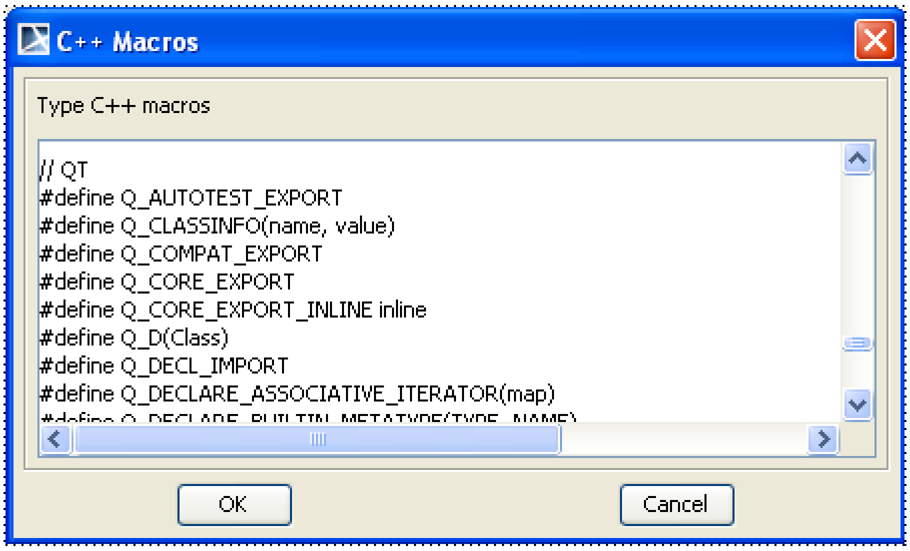
Documentation — Style: Style 1

- [✓] Use include path ...
- [✓] Use explicit macros ...

Header: ...

```
/**
 * @(#) ClassA.h
 */

#ifndef CLASSA_H_H
#define CLASSA_H_H

/**
 * SampleClass description
 */
class ClassA
{

public:
              /**
               * sampleOperation description [abstrac
               */
              virtual void sampleOperation( )= 0;
              /**
               * sampleOperation2 description
               */
              void sampleOperation2( );

protected:
              /**
               * sampleParameter description
               */
              & ClassA sampleAttribute;
```

MagicDraw has already defined some default skipped macros for QT.



Type C++ macros

```
// QT
#define Q_AUTOTEST_EXPORT
#define Q_CLASSINFO(name, value)
#define Q_COMPAT_EXPORT
#define Q_CORE_EXPORT
#define Q_CORE_EXPORT_INLINE inline
#define Q_D(Class)
#define Q_DECL_IMPORT
#define Q_DECLARE_ASSOCIATIVE_ITERATOR(map)
```

Copyright © 1998-2015 No Magic, Inc.

Microsoft C++ Profiles

Support for Microsoft specific structure and keywords, MagicDraw will provide new profiles named “Microsoft Visual C++ Profile”, “C++/CLI Profile”, and “C++ Managed Profile”.

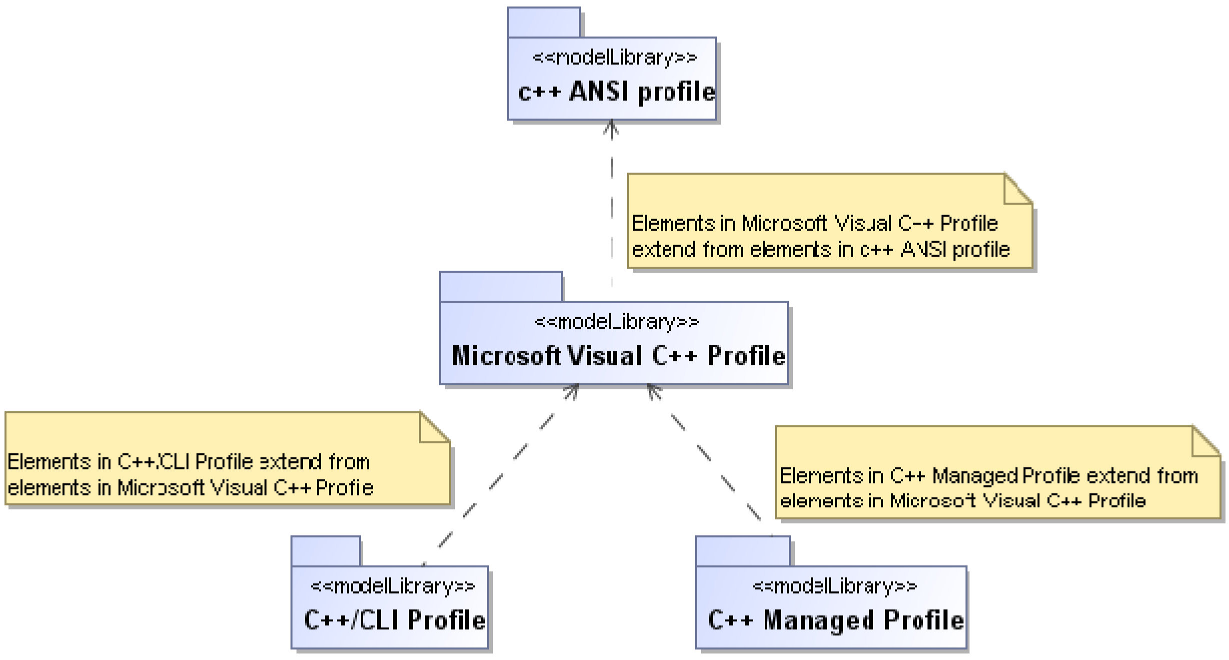

<<modelLibrary>>
c++ ANSI profile
Elements in Microsoft Visual C++ Profile extend from elements in c++ ANSI profile
<<model_library>>
Microsoft Visual C++ Profile
Elements in C++/CLI Profile extend from elements in Microsoft Visual C++ Profile
Elements in C++ Managed Profile extend from elements in Microsoft Visual C++ Profile
<<modelLibrary>>
C++/CLI Profile
<<modelLibrary>>
C++ Managed Profile

Figure 127 -- Profile Dependency

Microsoft Visual C++ Profile provides stereotypes and tagged values for modeling Microsoft Visual C++, which has additional specific keywords and constructs. It conforms to Microsoft Visual C++ 6.0 and later. However, it does not include keywords and constructs related to Managed features.

Note	Microsoft Visual C++ with Managed features has been called Managed C++, Visual Studio 2003. Later in Visual Studio 2005, the set of keywords and constructs related to Managed features changed and is also known as C++/CLI.

Microsoft Visual C++ Profile

Profile Name: **Microsoft Visual C++ Profile**

Module Name: **C++_MS_Profile.xml**

Data type

The Microsoft Visual C++ profile includes only the data types that do not exist in the ANSI C++ profile.

Microsoft C/C++ supports sized integer types. Declaration of 8-, 16-, 32-, or 64-bit integer variables can be done by using the **__int***n* type specifier, where *n* is 8, 16, 32, or 64.

Copyright © 1998-2015 No Magic, Inc.

The types __**int8**, __**int16**, and __**int32** are synonyms for the ANSI types that have the same size, and are useful for writing portable code that behaves identically across multiple platforms.

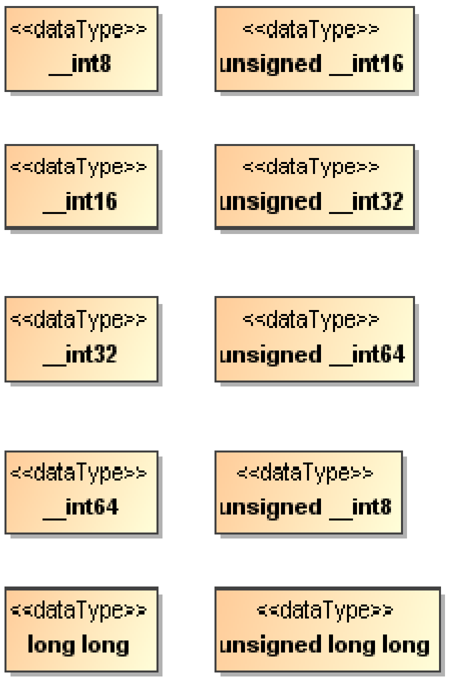


Figure 128 -- Microsoft Visual C++ Data Type

Copyright © 1998-2015 No Magic, Inc.

Stereotype

NOTE	The profile table and description in this section does not include the tagged value inherited from C++ ANSI profile.

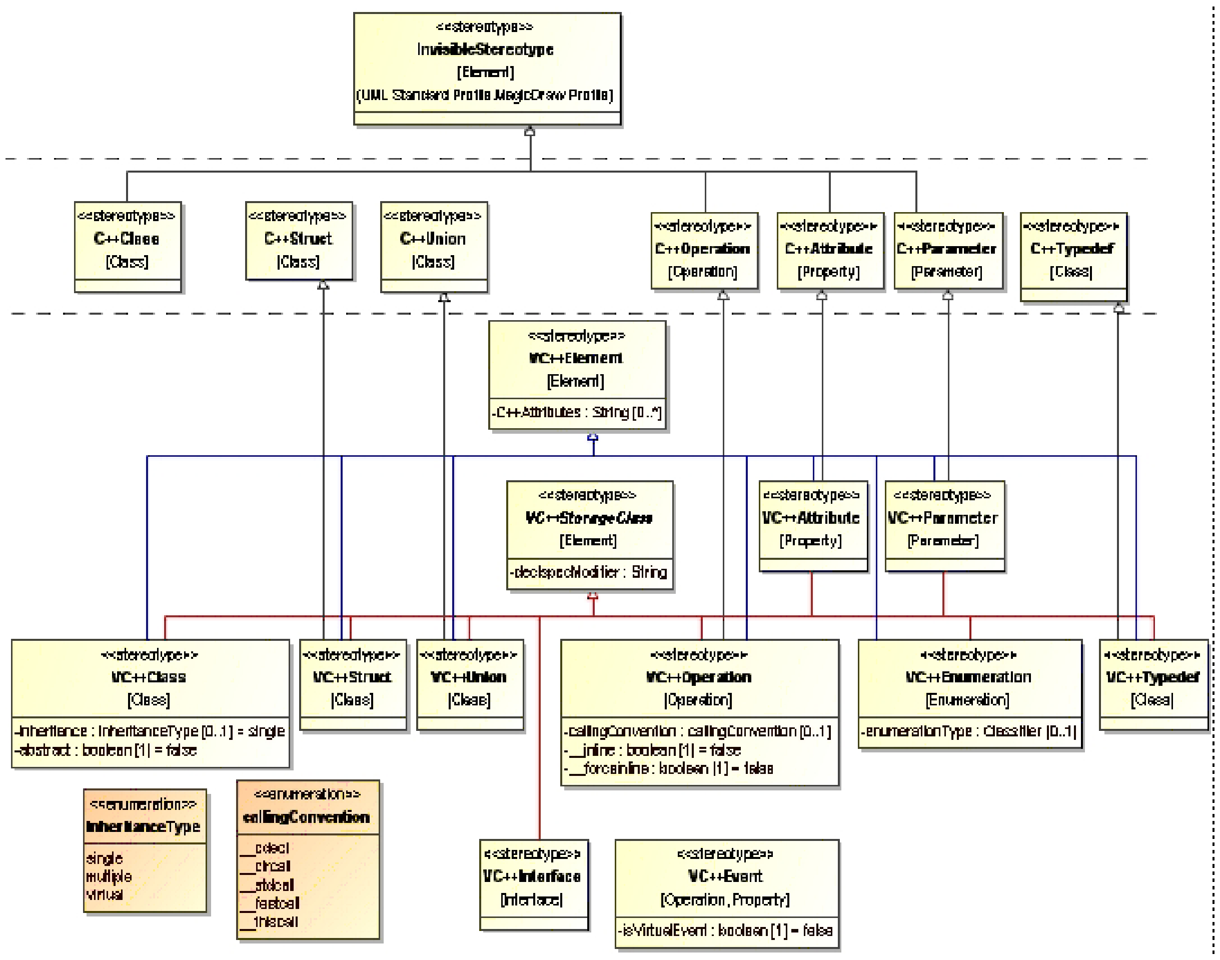


Figure 129 -- Microsoft Visual C++ Stereotypes

Copyright © 1998-2015 No Magic, Inc.

VC++Class

«VC++Class» inherits from «C++Class» and «VC++StorageClass»

Name	Meta class	Constraints
VC++Class	Class	

Tag	Type	Description
inheritance	inheritanceType[0..1] = single (Enumeration) See *inheritanceType*	Represent the utilization of VC++ keywords, **__single_inheritance**, **__multiple_inheritance** and **__virtual_inheritance**.
abstract	boolean[1] = false	Represent the utilization of keyword __abstract or abstract, depending on C++ dialects.

Inherited tag	Type	Description
declspecModifier «VC++StorageClass»	String	For keeping extended declaration modifier of __declspec See *Extended storage-class attributes with __declspec*
C++Attributes «VC++Element»	String	For keeping C++ Attributes

VC++Struct

«VC++Struct» inherits from «C++Struct» and «VC++StorageClass»

Name	Meta class	Constraints
VC++Struct	Class	

Inherited tag	Type	Description
declspecModifier «VC++StorageClass»	String	For keeping extended declaration modifier of **__declspec** See *Extended storage-class attributes with __declspec*
C++Attributes «VC++Element»	String	For keeping C++ Attributes

Copyright © 1998-2015 No Magic, Inc.

VC++Union

Name	Meta class	Constraints
VC++Union	Class	

Inherited tag	Type	Description
declspecModifier «VC++StorageClass»	String	For keeping extended declaration modifier of **__declspec** See *Extended storage-class attributes with __declspec*
C++Attributes «VC++Element»	String	For keeping C++ Attributes

VC++Enumeration

Name	Meta class	Constraints
VC++Enumeration	Enumeration	

Tag	Type	Description
Type	Classifier[0..1]	Represent the type of enumeration literal

VC++Typedef

<<VC++Typedef>> inherits from <<C++Typedef>> and <<VC++StorageClass>>

Name	Meta class	Constraints
VC++Typedef	Class	

Inherited tag	Type	Description
C++Attributes «VC++Element»	String	For keeping C++ Attributes

Copyright © 1998-2015 No Magic, Inc.

VC++Attribute

Name	Meta class	Constraints
VC++Attribute	Property	

Inherited tag	Type	Description
C++Attributes «VC++Element»	String	For keeping C++ Attributes

VC++Operation

Name	Meta class	Constraints
VC++Operation	Operation	

Tag	Type	Description
callingConvention	callingConvention [0..1] (Enumeration) See *callingConvention*	Represent the utilization of key-words. • __cdecl • __clrcall • __stdcall • __fastcall • __thiscall
__inline	boolean[1] = false	Represent the utilization of **__inline** keyword.
__forceinline	boolean[1] = false	Represent the utilization of **__forceinline** keyword.

Inherited tag	Type	Description
declspecModifier «VC++StorageClass»	String	For keeping extended declara-tion modifier of **__declspec** See *Extended storage-class attributes with __declspec*
C++Attributes «VC++Element»	String	For keeping C++ Attributes

Copyright © 1998-2015 No Magic, Inc.

VC++Parameter

Name	Meta class	Constraints
VC++Parameter	Parameter	

Inherited tag	Type	Description
C++Attributes «VC++Element»	String	For keeping C++ Attributes

VC++StorageClass

«VC++StorageClass» is an abstract stereotype corresponding to extended declaration modifier of **__declspec**.

Name	Meta class	Constraints
VC++StorageClass	Element	

Tag	Type	Description
declspecModifier	String	For keeping extended declaration modifier of **__declspec** See *Extended storage-class attributes with __declspec*

VC++Element

Name	Meta class	Constraints
VC++Element	Element	

Tag	Type	Description
C++Attributes	String	For keeping C++ Attributes

Copyright © 1998-2015 No Magic, Inc.

VC++Interface

Name	Meta class	Description
VC++Interface	Interface	Represent interface declaration with __**interface** keyword.

VC++Event

Name	Meta class	Description
VC++Event	Operation, Property	Represent interface, method or member declaration with __**event** keyword

Tag	Type	Description
isVirtualEvent	boolean[1] = false	Support virtual event as shown in the example below. // data member as event virtual __event ClickEventHandler* OnClick;

Enumeration

inheritanceType

inheritanceType will be used as value of tag named "**inheritance**" under **«VC++Class»**.

Literal	Description
single	Representation of __**single_inheritance** keyword
multiple	Representation of __**multiple_inheritance** keyword
Virtual	Representation of __**virtual_inheritance** keyword

callingConvention

callingConvention will be used as value of tag named **"calling convention"** under «VC++Operation»

Literal	Description
__cdecl	Represent the utilization of __**cdecl** modifier.
__clrcall	Represent the utilization of __**clrcall** modifier.
__stdcall	Represent the utilization of __**stdcall** modifier.
__fastcall	Represent the utilization of __**fastcall** modifier.
__thiscall	Represent the utilization of __**thiscall** modifier.

Copyright © 1998-2015 No Magic, Inc.

C++/CLI Profile

Note	The profile table and description in this section does not include the tagged value inherited from C++ ANSI profile.

Stereotype

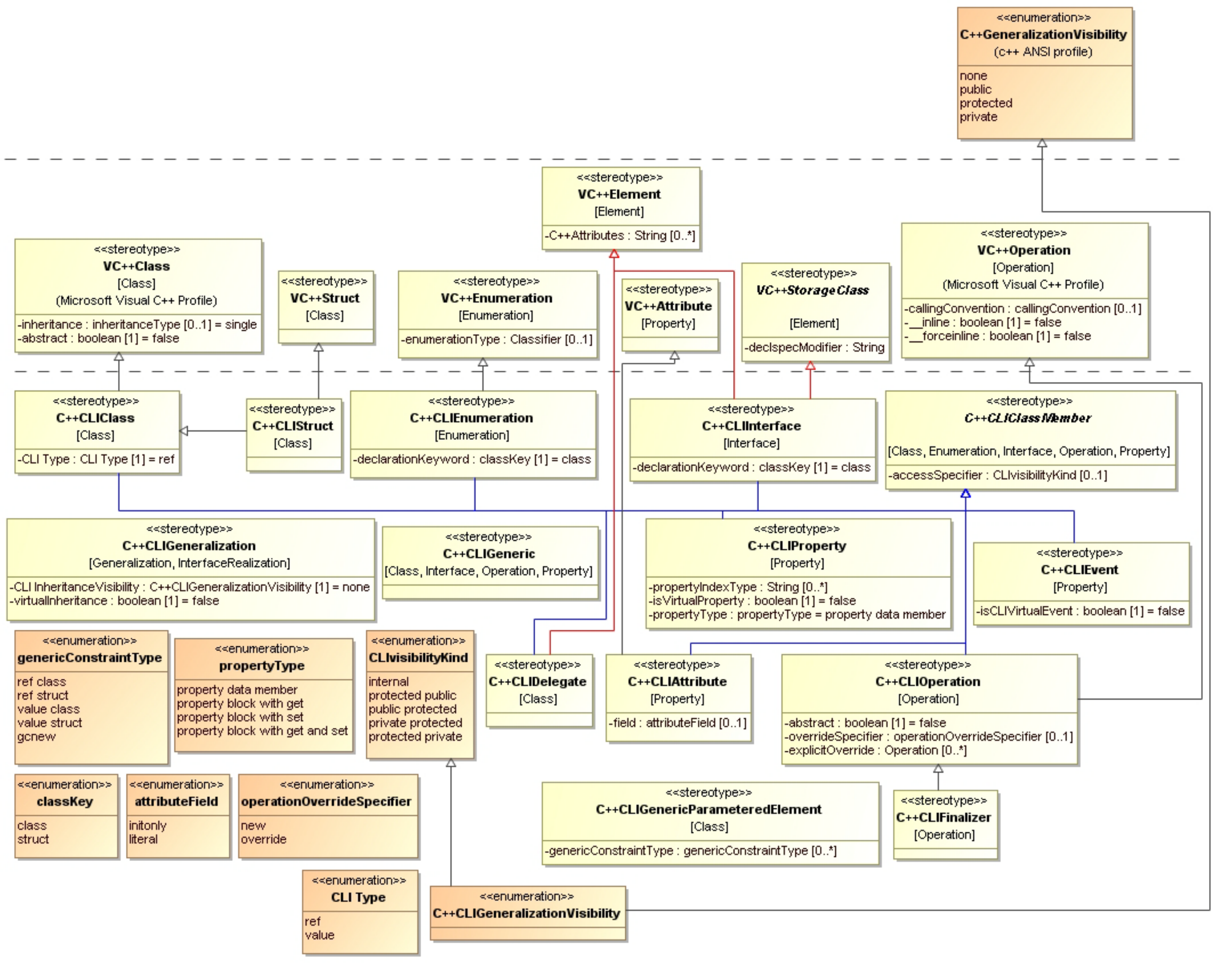


Figure 130 -- C++/CLI Stereotypes

C++CLIClass

Name	**Meta class**	**Constraints**
C++CLIClass	Class	

Tag	**Type**	**Description**
CLI Type	CLI Type[1] = ref (Enumeration) See *CLI Type*	Represent the utilization of **ref class** or **value class** keywords for definding CLR class.

Copyright © 1998-2015 No Magic, Inc.

Inherited tag	Type	Description
inheritance «VC++Class»	inheritanceType[0..1] = single (Enumeration) See *inheritanceType*	Represent the utilization of VC++ keywords, **__single_inheritance**, **__multiple_inheritance** and **__virtual_inheritance**.
declspecModifier «VC++StorageClass»	String	For keeping extended declaration modifier of **__declspec** See *Extended storage-class attributes with __declspec*
C++Attributes «VC++Element»	String	For keeping C++ Attributes
abstract «VC++Class»	boolean[1] = false	Represent the utilization of **abstract** keyword.
accessSpecifier «C++CLIClassMember»	CLIvisibilityKind[0..1] (Enumeration) See *CLIvisibilityKind*	Represent the usage of visibility kind introduced in C++/CLI including • internal • protected public • public protected • private protected • protected private

C++CLIStruct

Name	Meta class	Constraints
C++CLIStruct	Class	

Inherited tag	Type	Description
managedType «C++CLIClass»	CLI Type[1] = ref (Enumeration) See *CLI Type*	Represent the utilization of **ref class** or **value class** keywords for defining CLR class.
abstract «C++CLIClass»	boolean[1] = false	Represent the utilization of **abstract** keyword.
inheritance «VC++Class»	inheritanceType[0..1] = single (Enumeration) See *inheritanceType*	Represent the utilization of VC++ keywords, **__single_inheritance**, **__multiple_inheritance** and **__virtual_inheritance**.
declspecModifier «VC++StorageClass»	String	For keeping extended declaration modifier of **__declspec** See *Extended storage-class attributes with __declspec*

Copyright © 1998-2015 No Magic, Inc.

Inherited tag	Type	Description
VC++Attributes «VC++Element»	String	For keeping VC++ Attributes

C++CLIEnumeration

Name	Meta class	Constraints
C++CLIEnumeration	Enumeration	

Tag	Type	Description
declarationKeyword	classKey[1] = class (Enumeration) See *classKey*	Represent the utilization of **enum class** or **enum struct**. keywords. See *enum class, enum struct*

Inherited tag	Type	Description
type «VC++Enumeration»	Classifier[0..1]	Represent the type of enumeration literal

C++CLIInterface

Name	Meta class	Constraints
C++CLIInterface	Interface	

Tag	Type	Description
declarationKeyword	classKey[1] = class (Enumeration) See *classKey*	Represent the utilization of **interface class** or **interface struct**. keywords. See *ref class, ref struct, value class, value struct, interface class, interface struct*

Inherited tag	Type	Description
declspecModifier «VC++StorageClass»	String	For keeping extended declaration modifier of **__declspec** See *Extended storage-class attributes with __declspec*
C++Attributes «VC++Element»	String	For keeping C++ Attributes

Copyright © 1998-2015 No Magic, Inc.

C++CLIClassMember

Name	Meta class	Constraints
C++CLIClassMember	Operation, Property	This stereotype is an abstract stereotype for keeping access-Specifier tag definition.

Tag	Type	Description
accessSpecifier	CLIvisibilityKind[0..1] (Enumeration) See *CLIvisibilityKind*	Represent the usage of visibility kind introduce in C++/CLI including • internal • protected public • public protected • private protected • protected private

C++CLIAttribute

Name	Meta class	Constraints
C++CLIAttribute	Property	

Tag	Type	Description
field	attributeField[0..1] (Enumeration) See *attributeField*	Represent the usage of **initonly** or **literal** keywords.

Inherited tag	Type	Description
accessSpecifier «C++CLIClassMember»	CLIvisibilityKind[0..1] (Enumeration) See *CLIvisibilityKind*	Represent the usage of visibility kind introduce in C++/CLI including • internal • protected public • public protected • private protected • protected private
declspecModifier «VC++StorageClass»	String	For keeping extended declaration modifier of **__declspec** See *Extended storage-class attributes with __declspec*
C++Attributes «VC++Element»	String	For keeping C++ Attributes

Copyright © 1998-2015 No Magic, Inc.

C++CLIOperation

Name	Meta class	Constraints
C++CLIOperation	Operation	

Tag	Type	Description
abstract	boolean[1] = false	Represent the utilization of **abstract** keyword.
overrideSpecifier	operationOverrideSpecifier[0..1] (Enumeration) See *Override Specifiers*	Represent the utilization of C++/CLI operation override specifiers, **new** and **override**.
explicitOverride	Operation[0..*]	Represent the usage of explicit override feature.

Inherited tag	Type	Description
accessSpecifier «C++CLIClassMember»	CLIvisibilityKind[0..1] (Enumeration) See *CLIvisibilityKind*	Represent the usage of visibility kind introduce in C++/CLI including • internal • protected public • public protected • private protected • protected private
callingConvention «VC++Operation»	callingConvention[0..1] See *callingConvention*	Represent the utilization of keywords. • __cdecl • __clrcall • __stdcall • __fastcall • __thiscall
C++Attributes «VC++Element»	String	For keeping C++ Attributes

Copyright © 1998-2015 No Magic, Inc.

C++CLIProperty

«C++CLIProperty» is used to define a CLR property, which has the appearance of an ordinary data member, and can be written to or read from using the same syntax as a data member.

Name	Meta class	Constraints
C++CLIProperty	Property	The stereotype must not be applied to the same attribute as «C++Attribute» or other stereotypes derived from «C++Attribute»

Tag	Type	Description
propertyIndexType	String[0..*]	index type keeps the list of property index types.
isVirtualProperty	boolean[1] = false	Specify whether the property is virtual or not.
propertyType	PropertyType[1] = property data member	Specify property type between: • property data member • property block with get • property block with set • property block with get and set

Inherited tag	Type	Description
accessSpecifier «C++CLIClassMember»	CLIvisibilityKind [0..1] (Enumeration) See *CLIvisibilityKind*	Represent the usage of visibility kind introduce in C++/CLI including • internal • protected public • public protected • private protected • protected private

C++CLIDelegate

«C++CLIDelegate» is used to define a delegate, which is a reference type that can encapsulate one or more methods with a specific function prototype. Delegates provide the underlying mechanism (acting as a kind of pointer to member function) for events in the common language runtime component model.

Name	Meta class	Constraints
C++CLIDelegate	Class	Declaration of delegate can be only in a managed type. The declaration of delegate cannot have attribute and operation. Represent the utilization of **delegate** keyword.

Copyright © 1998-2015 No Magic, Inc.

C++CLIEvent

Name	**Meta class**	**Constraints**
C++CLIEvent	Property	«C++CLIEvent» can be applied only when the attribute type is «C++CLIDelegate». Represent the utilization of **event** keyword.

Tag	**Type**	**Description**
isCLIVirtualEvent	boolean[1] = false	Specify whether the event is vitual or not.

Inherited tag	**Type**	**Description**
accessSpecifier «C++CLIClassMember»	CLIvisibilityKind[0..1] (Enumeration) See *CLIvisibilityKind*	Represent the usage of visibility kind introduce in C++/CLI includ-ing • internal • protected public • public protected • private protected • protected private

C++CLIGeneric

Name	Meta class	Constraints
C++CLIGeneric	Class, Interface, Operation, Property	Represent the usage of **generic** keyword.

Copyright © 1998-2015 No Magic, Inc.

C++CLIGenericParameteredElement

Name	Meta class	Constraints
C++CLIGenericParametere-dElement	Class	

Tag	Type	Description
genericConstraintType	genericConstraintType[0..*] (Enumeration) See *genericConstraintType*	Specify generic constraint type including • ref class • ref struct • value class • value struct • gcnew

C++CLIGeneralization

Name	Meta class	Constraints
C++CLIGeneralization	Generalization	

Tag	Type	Description
CLIInheritanceVisibility	C++CLIGeneralizationVisibil-ity[1] = none (Enumeration) See *C++CLIGeneralization-Visibility*	
virtualInheritance	boolean[1] = false	

Enumeration

CLI Type

CLI Type will be used as value of tag named "**CLI Type**" under **«C++CLIClass»**.
The possible values are **ref** and **value**.

Literal	Description
ref	For defining a CLR reference class
value	For defining a CLR value class

operationOverrideSpecifier

operationOverrideSpecifier will be used as value of tag named "**overrideSpecifier**" under **«VC++Opera-tion».**

Copyright © 1998-2015 No Magic, Inc.

The possible values are **new** and **override** which are keywords that can be used to qualify override behavior for derivation.

Literal	Description
new	Indicate the use of **new** keyword to qualify override behavior for derivation. In VC++, **new** is a keyword to indicate that a virtual member will get a new slot in the vtable; that the function does not override a base class method.
override	Indicate the use of **override** keyword to qualify override behavior for derivation. In VC++, **override** is a keyword to indicate that a member of a managed type must override a base class or a base interface member. If there is no member to override, the compiler will generate an error.

attributeField

attributeField will be used as value of tag named “field” under **«C++CLIAttribute»**. The possible values are **initonly** and **literal** which are keywords that can be used to qualify field type of attribute.

Literal	Description
initonly	Represent the utilization of **initonly** keyword. **initonly** indicates that variable assignment can only occur as part of the declaration or in a static constructor in the same class.
literal	Represent the utilization of **literal** keyword. It is the native equivalent of **static const** variable. Constraint: **Is Static** and **Is Read Only** must be set to **true**.

classKey

Literal	Description
class	Represent the keyword that has the word, **class**.
struct	Represent the keyword that has the word, **struct**.

Remark: **enum class** and **enum struct** are equivalent declarations.

interface class and **interface struct** are equivalent declarations.

Copyright © 1998-2015 No Magic, Inc.

CLIvisibilityKind

Literal	**Description**
internal	Represent the **internal** visibility.
protected public	Represent the **protected public** visibility.
public protected	Represent the **public protected** visibility.
private protected	Represent the **private protected** visibility.
protected private	Represent the **protected private** visibility.

genericConstraintType

Literal	**Description**
ref class	Represent the usage of **ref class** keyword in generic constraint clause.
ref struct	Represent the usage of **ref struct** keyword in generic constraint clause.
value class	Represent the usage of **value class** keyword in generic constraint clause.
value struct	Represent the usage of **value struct** keyword in generic constraint clause.
gcnew	Represent the usage of **gcnew** keyword in generic constraint clause.

C++CLIGeneralizationVisibility

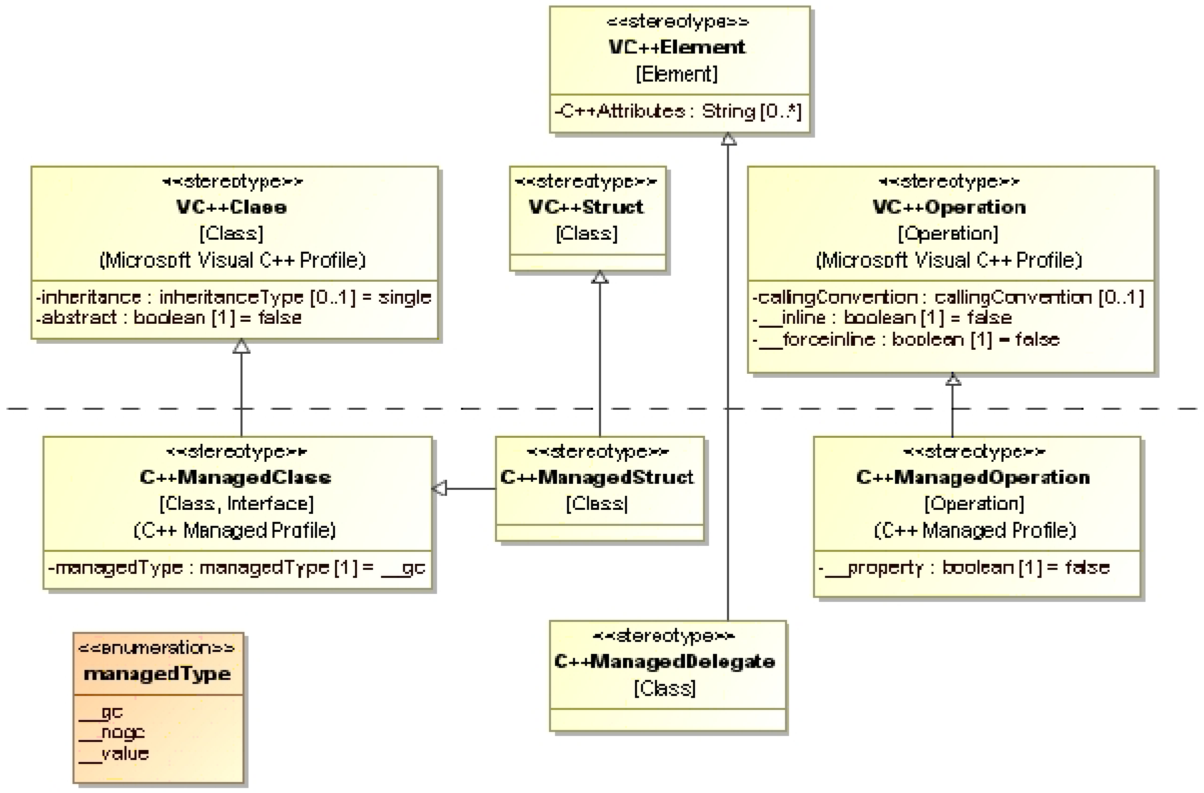
C++CLIGeneralizationVisibility is an enumeration inheriting from C++GeneralizationVisibility and CLIvisibility-Kind, so its literals include **none**, **public**, **protected**, **private**, **internal**, **protected public**, **public protected**, **private protected**, and **protected private**.

C++ Managed Profile

NOTE The profile table and description in this section does not include the tagged value inherited from other profiles.

Copyright © 1998-2015 No Magic, Inc.

Stereotype



<<stereotype>> VC++Element [Element]
-C++Attributes : String [0..*]
<<stereotype>> VC++Class [Class] (Microsoft Visual C++ Profile)
-inheritance : inheritanceType [0..1] = single
-abstract : boolean [1] = false
<<stereotype>> VC++Struct [Class]
<<stereotype>> VC++Operation [Operation] (Microsoft Visual C++ Profile)
-callingConvention : callingConvention [0..1]
-__inline : boolean [1] = false
-__forceinline : boolean [1] = false
<<stereotype>> C++ManagedClass [Class, Interface] (C++ Managed Profile)
-managedType : managedType [1] = __gc
<<stereotype>> C++ManagedStruct [Class]
<<stereotype>> C++ManagedOperation [Operation] (C++ Managed Profile)
-__property : boolean [1] = false
<<enumeration>> managedType
__gc
__nogc
__value
<<stereotype>> C++ManagedDelegate [Class]

Figure 131 -- C++ Managed Stereotypes

C++ManagedClass

Name	Meta class	Constraints
C++ManagedClass	Class	Represent the class declaration with version 1 of Managed Extension for C++.

Tag	Type	Description
managedType	managedType[1] = __gc (enumeration) See *managedType*	

C++ManagedStruct

Name	Meta class	Constraints
C++ManagedStruct	Class	Represent the struct declaration with version 1 of Managed Extension for C++.

Copyright © 1998-2015 No Magic, Inc.

C++ManagedOperation

Name	Meta class	Constraints
C++ManagedOperation	Operation	

Tag	Type	Description
__property	boolean[1] = false	Represent the usage of **__property** keyword. It is a feature in version 1 of Managed Extension for C++.

C++ManagedDelegate

Name	Meta class	Constraints
C++ManagedDelegate	Class	Represent the delegate declaration with version 1 of Managed Extension for C++. It defines a reference type that can be used to encapsulate a method with a specific signature.

Enumeration

managedType

Literal	Description
__gc	Represent managed declaration with **__gc** keyword.
__nogc	Represent the usage of **__nogc** keyword, which is used to explicitly specify that an object is allocated on the standard C++ heap.
__value	Represent managed declaration with **__value** keyword. A **__value** type differs from **__gc** types in that **__value** type variables directly contain their data, whereas managed variables point to their data, which is stored on the common language runtime heap.

Modeling with Microsoft Specific Profiles

CLR Data Type Keyword

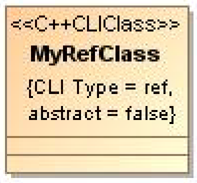
ref class, ref struct, value class, value struct, interface class, interface struct

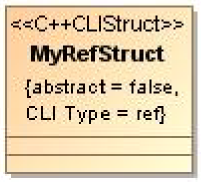
```
class_access ref class name modifier : inherit_access base_type {};
class_access ref struct name modifier : inherit_access base_type {};
class_access value class name modifier : inherit_access base_type {};
class_access value struct name modifier : inherit_access base_type {};
interface_access interface class name : inherit_access base_interface {};
```

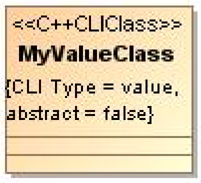
Copyright © 1998-2015 No Magic, Inc.

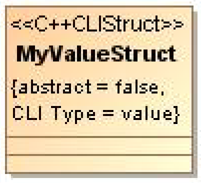
interface_access interface struct name : inherit_access base_interface {};

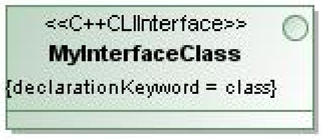
Code	MD-UML

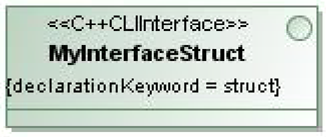
`ref class MyRefClass`
`{`
`};`
	 «C++CLIClass»
	MyRefClass
	{CLI Type = ref,
	abstract = false}
	with «C++CLIClass» and tagged value **CLI Type** = ref

`ref struct MyRefStruct`
`{`
`};`
	 «C++CLIStruct»
	MyRefStruct
	{abstract = false,
	CLI Type = ref}
	with «C++CLIStruct» and tagged value **CLI Type** = ref

`value class MyValueClass`
`{`
`};`
	 «C++CLIClass»
	MyValueClass
	{CLI Type = value,
	abstract = false}
	with «C++CLIClass» and tagged value **CLI Type** = value

`value struct`
`MyValueStruct`
`{`
`};`
	 «C++CLIStruct»
	MyValueStruct
	{abstract = false,
	CLI Type = value}
	with «C++CLIStruct» and tagged value **CLI Type** = value

`interface class`
`MyInterfaceClass`
`{`
`};`
	 «C++CLIInterface»
	MyInterfaceClass
	{declarationKeyword = class}
	with «C++CLIInterface» and tagged value **declarationKeyword** = class

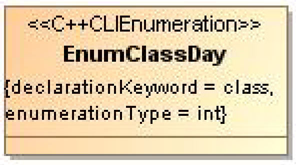
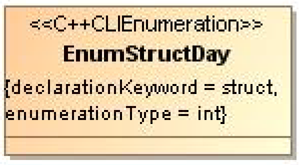
`interface struct`
`MyInterfaceStruct`
`{`
`};`
	 «C++CLIInterface»
	MyInterfaceStruct
	{declarationKeyword = struct}
	with «C++ICLIInterface» and tagged value **declarationKeyword** = struct

enum class, enum struct

access **enum class** name [: type] { enumerator-list } var;

Copyright © 1998-2015 No Magic, Inc.

access **enum struct** name [:type] { enumerator-list } var;

Code	MD-UML
`enum class EnumClassDay :int { sun, mon };`	 <<C++CLIEnumeration>> EnumClassDay {declarationKeyword = class, enumerationType = int}
`enum struct EnumStructDay :int { sun, mon };`	 <<C++CLIEnumeration>> EnumStructDay {declarationKeyword = struct, enumerationType = int}

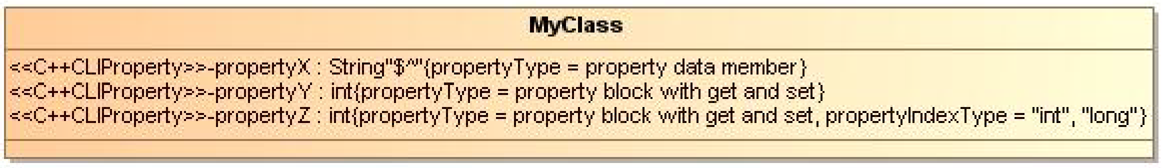
property

modifier **property** type property_name;   // property data member
modifier **property** type property_name {   // property block
      modifier void **set**(type);
      modifier type **get**();
}
modifier **property** type property_name[,] {
      modifier void **set**(type);
      modifier type **get**();
}

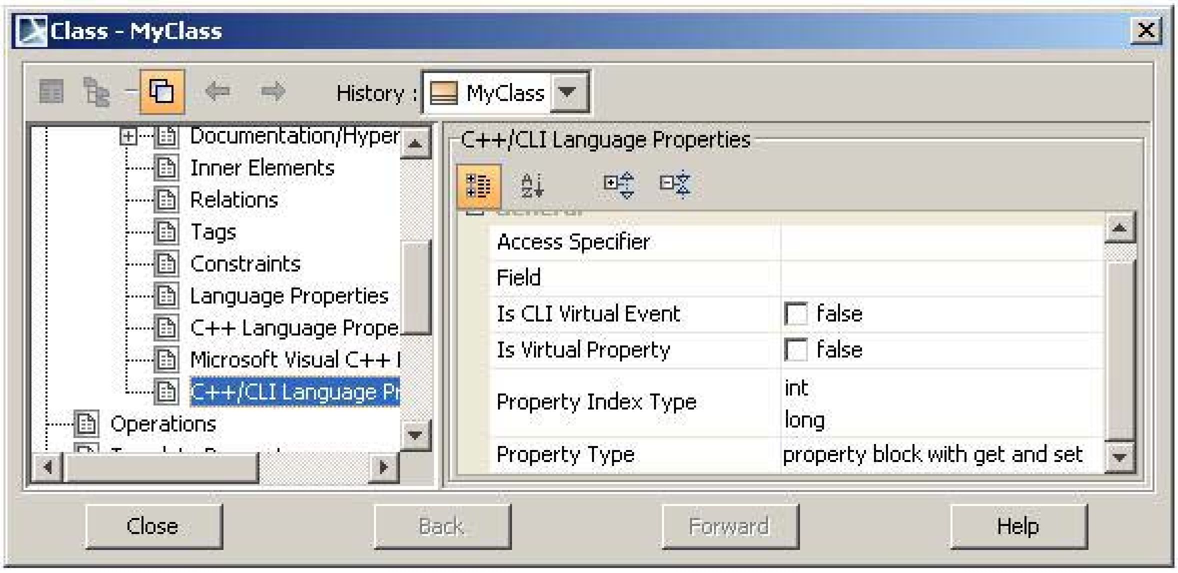
Code

```
public ref class MyClass
{
     // property data memberproperty
     property String ^ propertyX;
     // property block
     property int propertyY
     {
      int get();
      void set(int value);
     }
     //property block with index
    property int propertyZ[int, long]
    {
      int get(int index1,long index2);
      void set(int index1,long index2, int value);
    }
};
```

MD-UML


MyClass
<<C++CLIProperty>>-propertyX : String"$^"{propertyType = property data member}
<<C++CLIProperty>>-propertyY : int{propertyType = property block with get and set}
<<C++CLIProperty>>-propertyZ : int{propertyType = property block with get and set, propertyIndexType = "int", "long"}

Copyright © 1998-2015 No Magic, Inc.



Access Specifier	
Field	
Is CLI Virtual Event	☐ false
Is Virtual Property	☐ false
Property Index Type	int long
Property Type	property block with get and set

delegate

access **delegate** function_declaration

access (optional)

The accessibility of the delegate outside of the assembly can be public or private. The default is private. Inside a class, a delegate can have any accessibility.

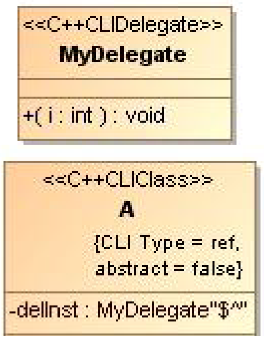
function_declaration

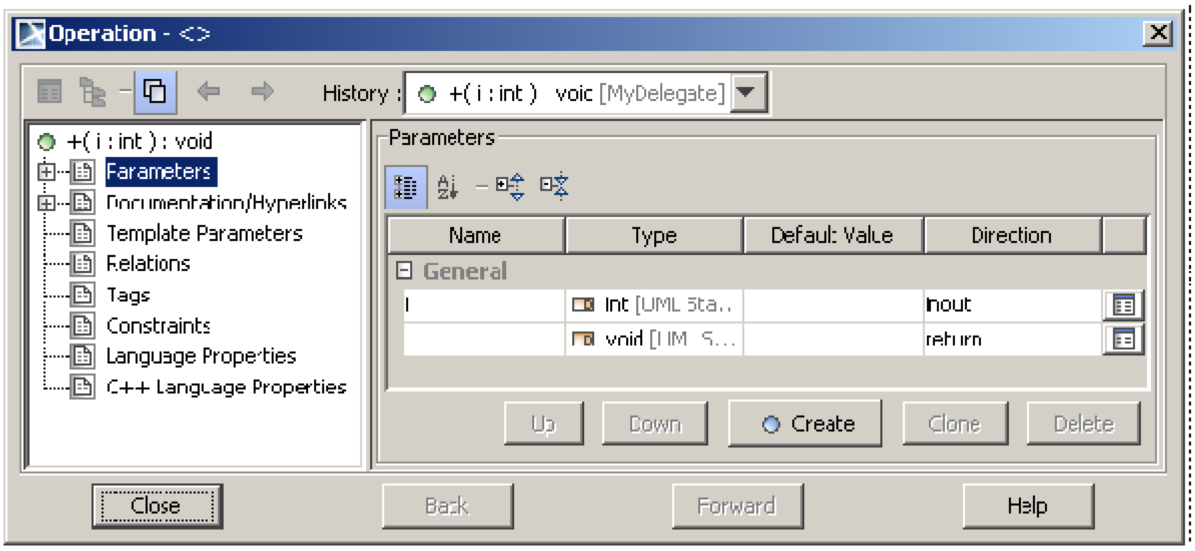
The signature of the function that can be bound to the delegate. The return type of a delegate can be any managed type. For interoperability reasons, it is recommended that the return type of a delegate be a CLS type.

To define an unbound delegate, the first parameter in function_declaration should be the type of the this pointer for the object

Copyright © 1998-2015 No Magic, Inc.

Code	MD-UML

`public delegate void MyDelegate(int i);`
`ref class A`
`{`
`    MyDelegate^ delInst;`
`};`
	 <<C++CLIDelegate>>
	MyDelegate
	+(i : int) : void
	<<C++CLIClass>>
	A
	{CLI Type = ref,
	abstract = false}
	-delInst : MyDelegate"$^"



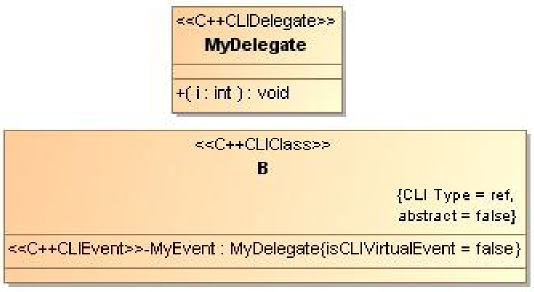
Name	Type	Default Value	Direction
General			
i	int [UML Sta...		inout
	void [UML S...		return

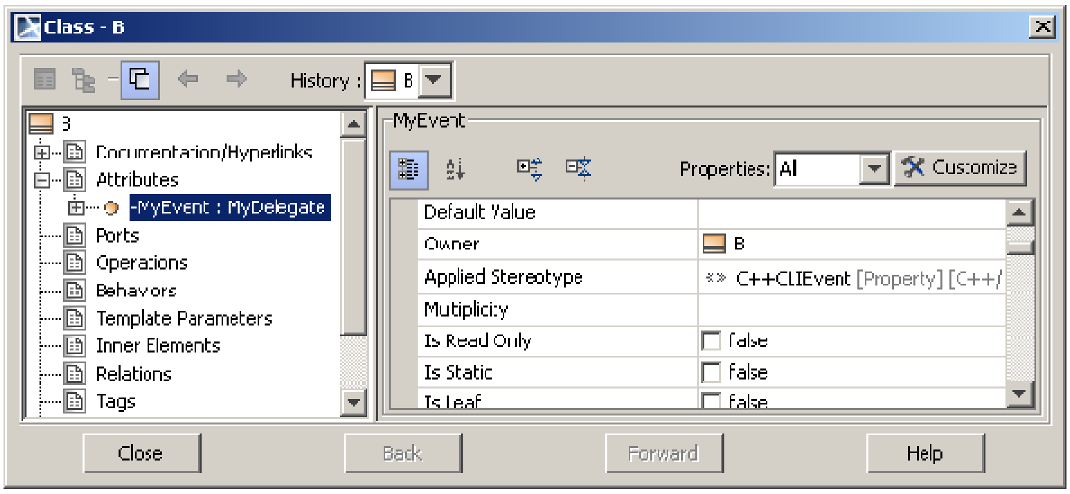
Copyright © 1998-2015 No Magic, Inc.

event

```
modifier event delegate ^ event_name;   // event data member
modifier event delegate ^ event_name
{
      modifier return_value add (delegate ^ name);
      modifier return_value remove(delegate ^ name);
      modifier return_value raise(parameters);
}   // event block
```

```
public delegate void
MyDelegate(int);
ref class B {
     event MyDelegate^ MyEvent;
};
```



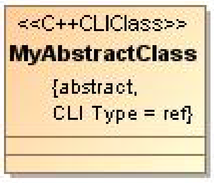


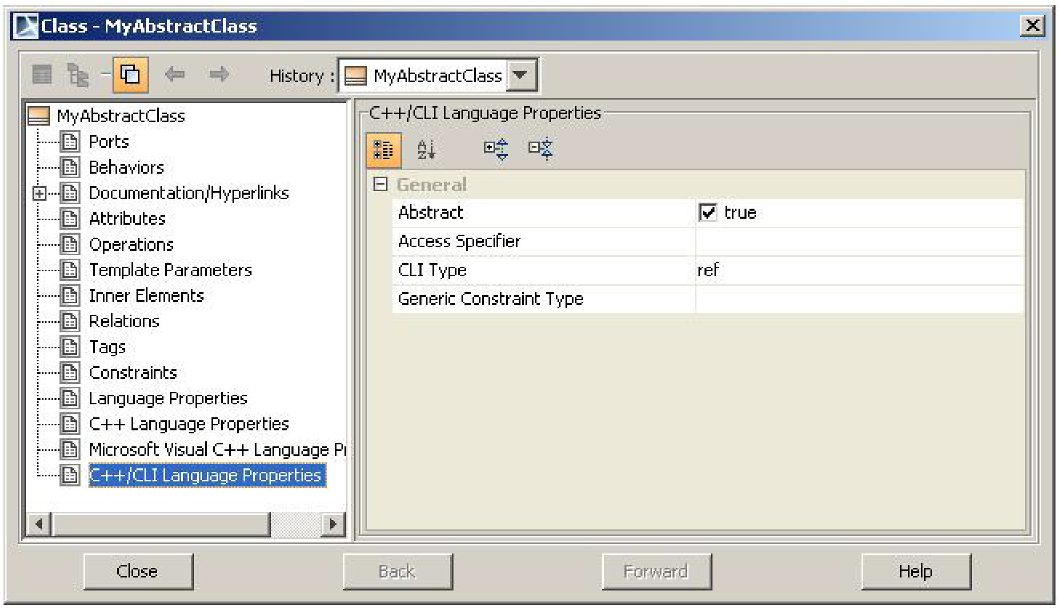
Default Value	
Owner	B
Applied Stereotype	C++CLIEvent [Property] [C++/
Multiplicity	
Is Read Only	false
Is Static	false
Is Leaf	false

Copyright © 1998-2015 No Magic, Inc.

Override Specifiers

abstract

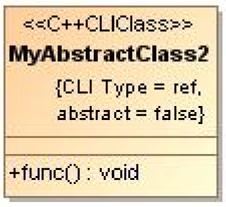
Code	MD-UML
`ref class MyAbstractClass abstract { };`	 with «C++CLIClass» and tagged value **abstract**=true



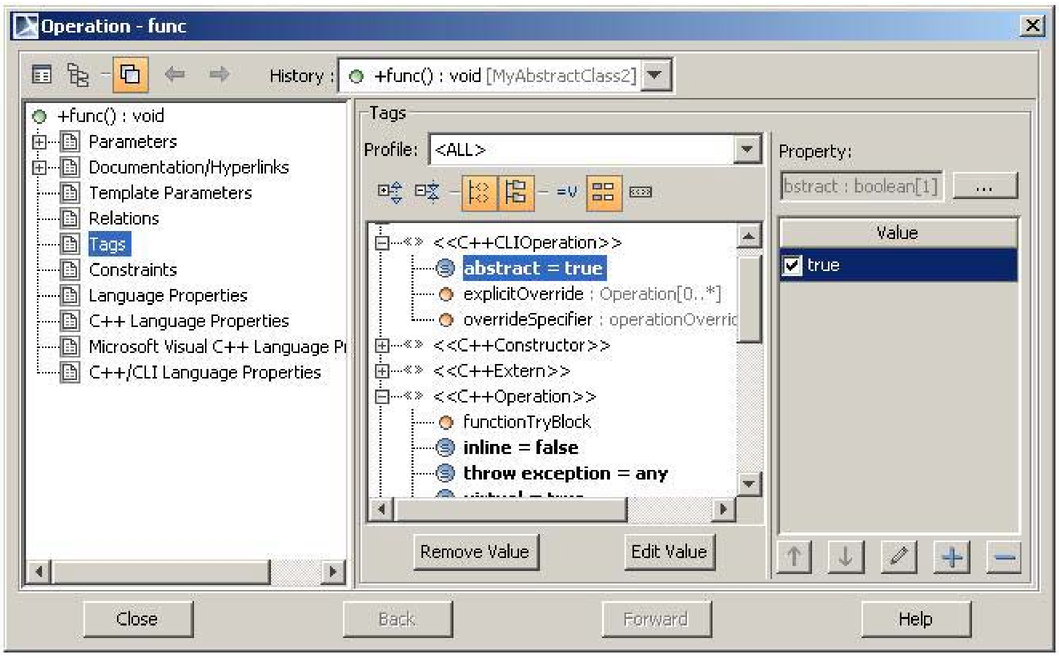
C++/CLI Language Properties

General

Abstract	☑ true
Access Specifier	
CLI Type	ref
Generic Constraint Type	

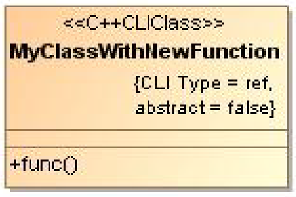
Code	MD-UML
`ref class MyAbstractClass2 { public: virtual void func() abstract; };`	 func() with «C++CLIOperation» and tagged value **abstract** = true and **virtual** = true

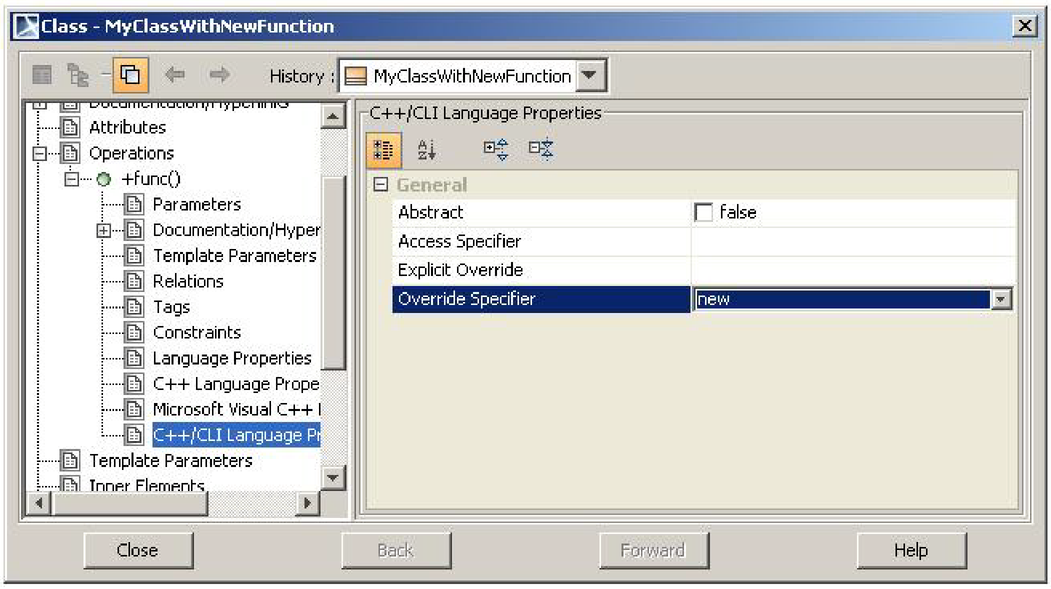
Copyright © 1998-2015 No Magic, Inc.


Operation - func
History : +func() : void [MyAbstractClass2]
+func() : void
Parameters
Documentation/Hyperlinks
Template Parameters
Relations
Tags
Constraints
Language Properties
C++ Language Properties
Microsoft Visual C++ Language Pr
C++/CLI Language Properties
Tags
Profile: <ALL>
<<C++CLIOperation>>
abstract = true
explicitOverride : Operation[0..*]
overrideSpecifier : operationOverric
<<C++Constructor>>
<<C++Extern>>
<<C++Operation>>
functionTryBlock
inline = false
throw exception = any
Property:
bstract : boolean[1]
Value
true
Remove Value
Edit Value
Close
Back
Forward
Help

new

Code	MD-UML

```
ref class MyClassWithNewFunction
 {
public:
   virtual void func() new {}
};
```
	 <<C++CLIClass>>
	MyClassWithNewFunction
	{CLI Type = ref,
	abstract = false}
	+func()
	func() with «C++CLIOperation» and tagged value **overrideSpcifier** = new

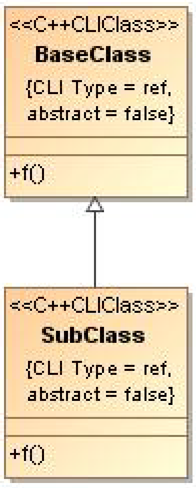


C++/CLI Language Properties

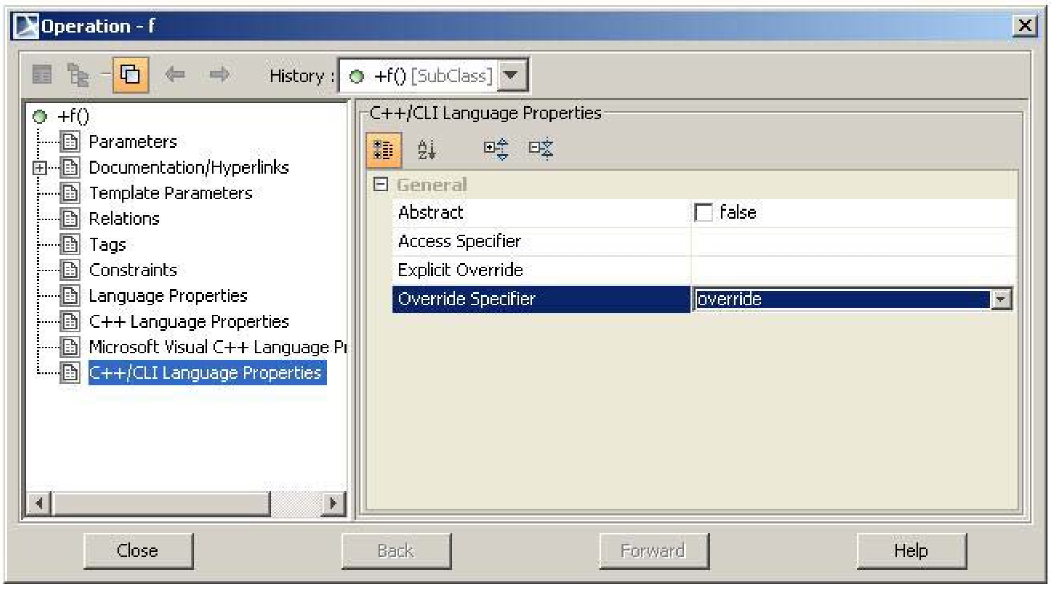
General

Abstract	☐ false
Access Specifier	
Explicit Override	
Override Specifier	new

override

```
ref class BaseClass {
   public:
virtual void f();
};

ref class SubClass : public
BaseClass {
   public:
virtual void f() override {}
};
```
	 <<C++CLIClass>>
	BaseClass
	{CLI Type = ref,
	abstract = false}
	+f()
	<<C++CLIClass>>
	SubClass
	{CLI Type = ref,
	abstract = false}
	+f()

Copyright © 1998-2015 No Magic, Inc.

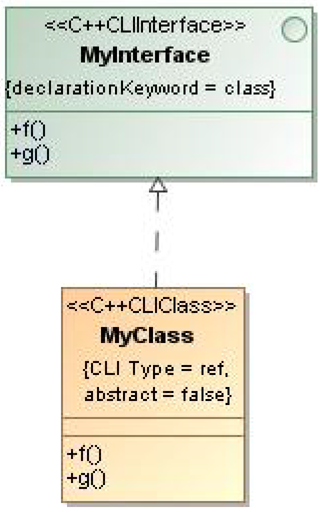


C++/CLI Language Properties

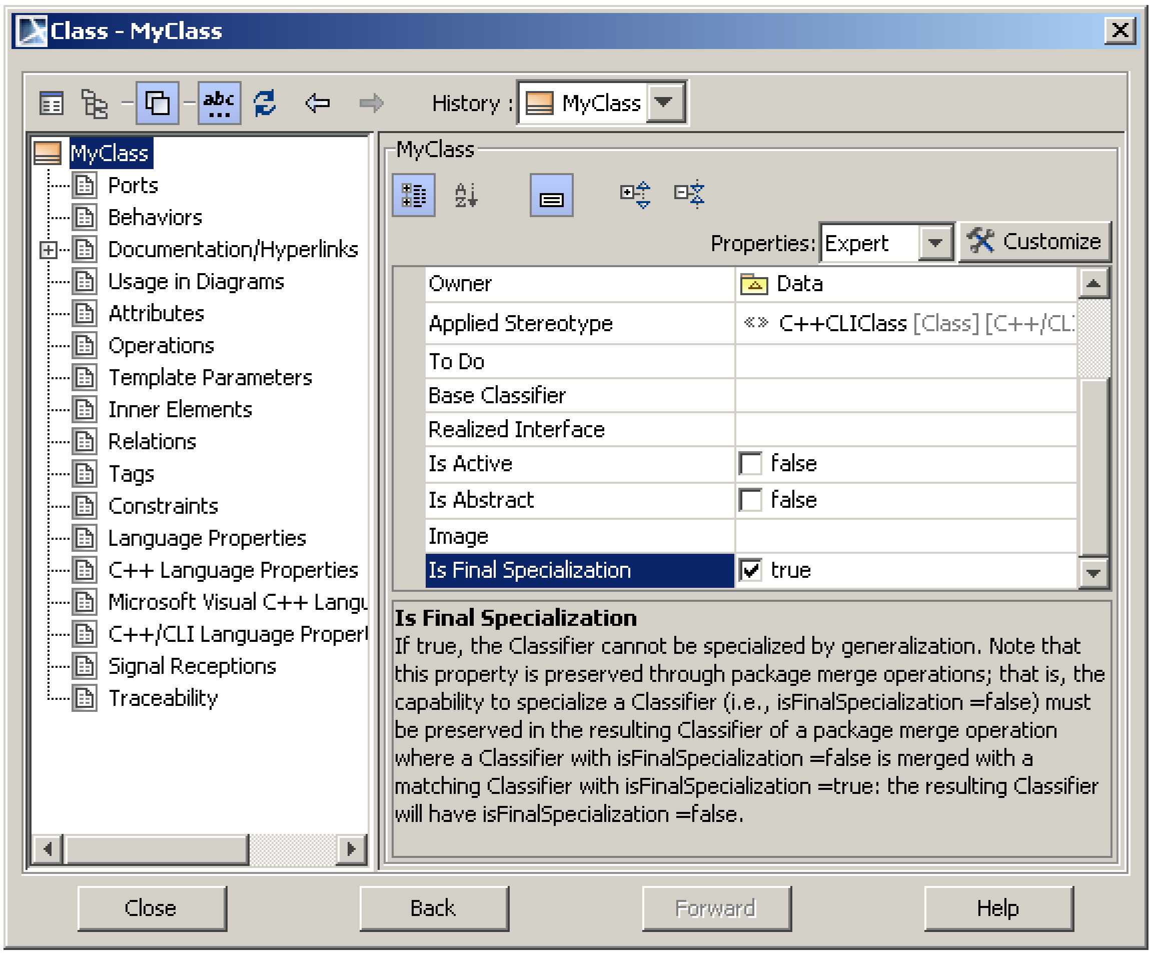
General	
Abstract	☐ false
Access Specifier	
Explicit Override	
Override Specifier	override

sealed

```
interface class MyInterface
{
    virtual void f();
    virtual void g();
};
ref class MyClass sealed: public
MyInterface
{
public:
     virtual void f(){};
     virtual void g(){};
};
```



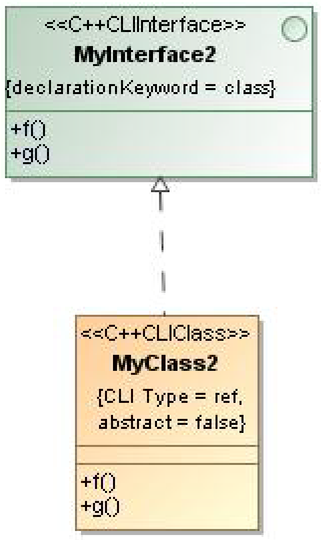
Copyright © 1998-2015 No Magic, Inc.



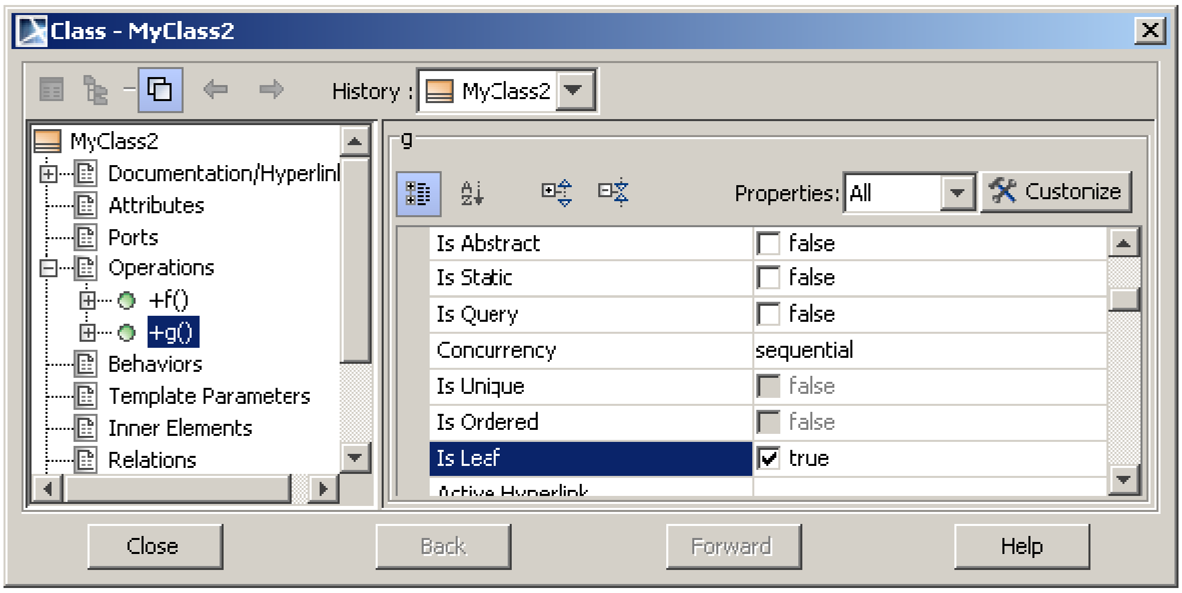
Owner	Data
Applied Stereotype	«» C++CLIClass [Class] [C++/CL
To Do	
Base Classifier	
Realized Interface	
Is Active	☐ false
Is Abstract	☐ false
Image	
Is Final Specialization	☑ true

Is Final Specialization
If true, the Classifier cannot be specialized by generalization. Note that this property is preserved through package merge operations; that is, the capability to specialize a Classifier (i.e., isFinalSpecialization =false) must be preserved in the resulting Classifier of a package merge operation where a Classifier with isFinalSpecialization =false is merged with a matching Classifier with isFinalSpecialization =true: the resulting Classifier will have isFinalSpecialization =false.

```
interface class MyInterface2 {
   public:
virtual void f();
virtual void g();
};
ref class MyClass2 : MyInterface2 {
  public:
   virtual void f() { }
   virtual void g() sealed { }
  // sub class cannot override g()
};
```



Copyright © 1998-2015 No Magic, Inc.



Is Abstract	☐ false
Is Static	☐ false
Is Query	☐ false
Concurrency	sequential
Is Unique	☐ false
Is Ordered	☐ false
Is Leaf	✓ true
Active Hyperlink	

Keywords for Generics

Generic Functions

```
[attributes] [modifiers]
return-type identifier <type-parameter identifier(s)>
[type-parameter-constraints clauses]

([formal-parameters])
{
    function-body
}
```

Parameters

attributes (Optional)

Additional declarative information. For more information on attributes and attribute classes, see attributes.

modifiers (Optional)

A modifier for the function, such as **static**. **virtual** is not allowed since virtual methods may not be generic.

return-type

The type returned by the method. If the return type is void, no return value is required.

identifier

The function name.

type-parameter identifier(s)

Comma-separated identifiers list.

Copyright © 1998-2015 No Magic, Inc.

formal-parameters (Optional)

Parameter list.

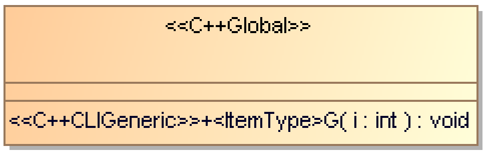
type-parameter-constraints-clauses

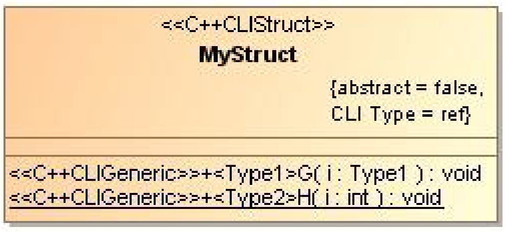
This specifies restrictions on the types that may be used as type arguments, and takes the form specified in Constraints.

function-body

The body of the method, which may refer to the type parameter identifiers.

Code	MD-UML

`generic <typename ItemType>`
`void G(int i) {}`
	
	ItemType with «C++TemplateParameter» applied and set tagged value **type keyword** = typename.

`ref struct MyStruct {`
`   generic <typename Type1>`
`   void G(Type1 i) {}`
 	
`   generic <typename Type2>`
`   static void H(int i) {}`
`};`
	 <<C++CLIStruct>>
	MyStruct
	{abstract = false,
	CLI Type = ref}
	<<C++CLIGeneric>>+<Type1>G(i : Type1) : void
	<<C++CLIGeneric>>+<Type2>H(i : int) : void

Generic Classes

[attributes]

generic <class-key type-parameter-identifier(s)**>**

[constraint-clauses]

[accessibility-modifiers] **ref class** identifier [modifiers]

[**:** base-list]

{

    class-body

} [declarators] [**;**]

Parameters

attributes (optional)

Additional declarative information. For more information on attributes and attribute classes, see Attributes.

class-key

Copyright © 1998-2015 No Magic, Inc.

Either class or typename

type-parameter-identifier(s)

Comma-separated list of identifiers specifying the names of the type parameters.

constraint-clauses

A list (not comma-separated) of where clauses specifying the constraints for the type parameters. Takes the form:

where type-parameter-identifier : constraint-list ...

constraint-list

class-or-interface[, ...]

accessibility-modifiers

Allowed accessibility modifiers include public and private.

identifier

The name of the generic class, any valid C++ identifier.

modifiers (optional)

Allowed modifiers include sealed and abstract.

base-list

A list that contains the one base class and any implemented interfaces, all separated by commas.

class-body

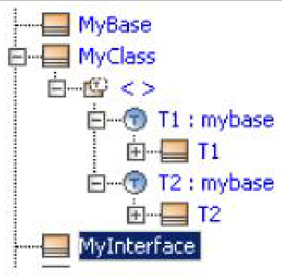
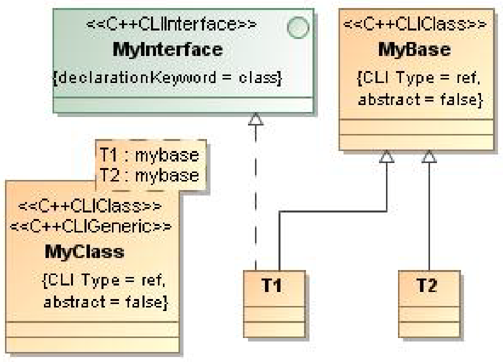
The body of the class, containing fields, member functions, etc.

declarators

Declarations of any variables of this type. For example: ^identifier[, ...]

Copyright © 1998-2015 No Magic, Inc.

Code	MD-UML

`interface class MyInterface {};`
`ref class MyBase{};`
 	
`generic <class T1, class T2>`
`where T1 : MyInterface, MyBase`
`where T2 : MyBase`
`ref class MyClass {};`
	 MyBase
	MyClass
	< >
	T1 : mybase
	T1
	T2 : mybase
	T2
	MyInterface
	 <<C++CLIInterface>>
	MyInterface
	{declarationKeyword = class}
	<<C++CLIClass>>
	MyBase
	{CLI Type = ref, abstract = false}
	T1 : mybase
	T2 : mybase
	<<C++CLIClass>>
	<<C++CLIGeneric>>
	MyClass
	{CLI Type = ref, abstract = false}
	T1
	T2

Generic Interfaces

[attributes] **generic <**class-key type-parameter-identifier[, ...]**>**

[type-parameter-constraints-clauses]

[accesibility-modifiers] **interface class** identifier [: base-list] **{** interface-body**}** [declarators] ;

Parameters

attributes (optional)

Additional declarative information. For more information on attributes and attribute classes, see Attributes.

class-key

class or typename

type-parameter-identifier(s)

Comma-separated identifiers list.

type-parameter-constraints-clauses

Takes the form specified in Constraints

accessibility-modifiers (optional)

Copyright © 1998-2015 No Magic, Inc.

Accessibility modifiers (e.g. public, private).

identifier

The interface name.

base-list (optional)

A list that contains one or more explicit base interfaces separated by commas.

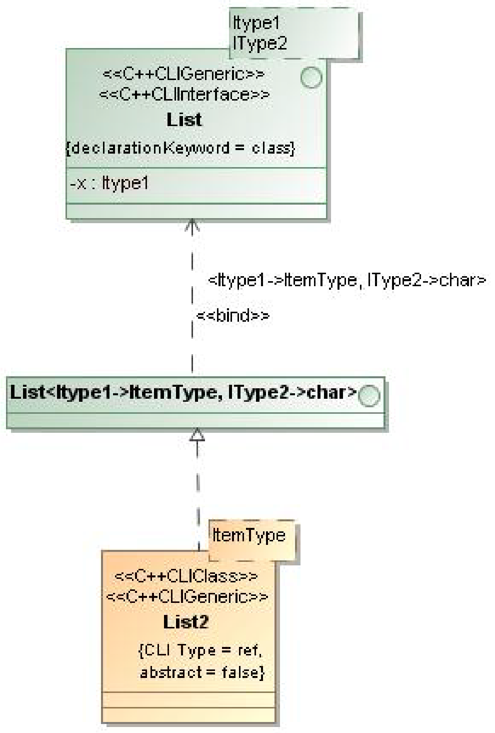
interface-body

Declarations of the interface members.

declarators (optional)

Declarations of variables based on this type.

Code	MD-UML

`generic <typename Itype1,`
`typename Itype2>`
`public interface class List {`
`  Itype1 x;`
`};`
 	
`generic <typename ItemType>`
`ref class List2 : public`
`List<ItemType, char>`
`{`
`};`
	

Generic Delegates

[attributes]
generic < [**class** | **typename**] type-parameter-identifiers >
[type-parameter-constraints-clauses]
[accessibility-modifiers] **delegate** result-type identifier
([formal-parameters]**)**;

Parameters

attributes (Optional)

Copyright © 1998-2015 No Magic, Inc.

Additional declarative information. For more information on attributes and attribute classes, see Attributes.

type-parameter-identifier(s)

Comma-separated list of identifiers for the type parameters.

type-parameter-constraints-clauses

Takes the form specified in Constraints

accessibility-modifiers (Optional)

Accessibility modifiers (e.g., **public**, **private**).

result-type

The return type of the delegate.

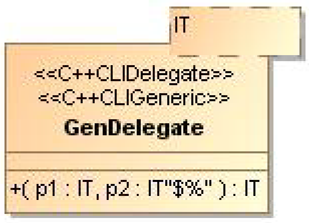
identifier

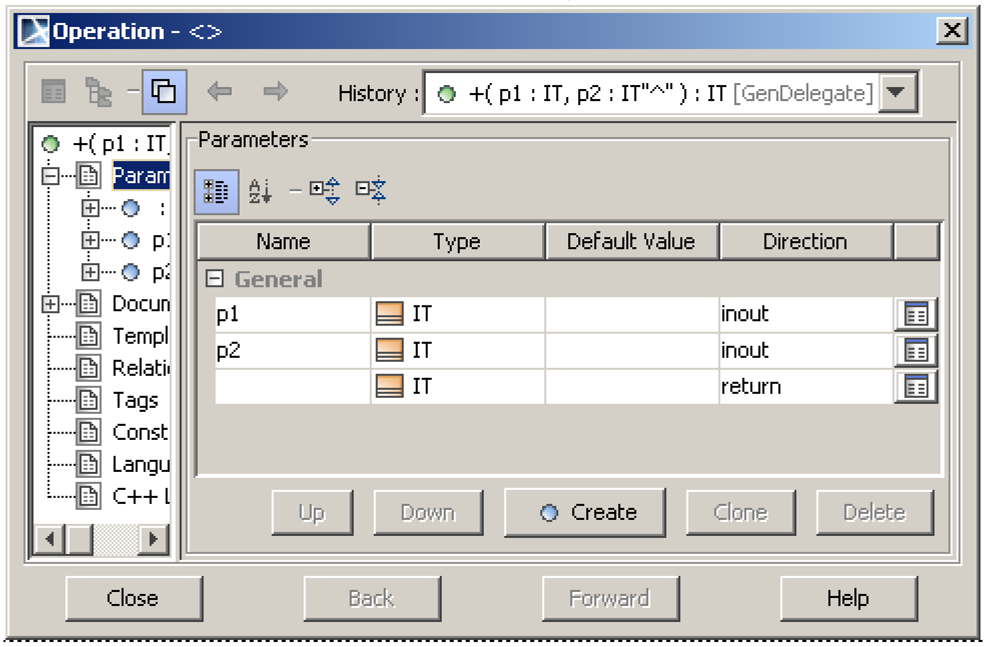
The name of the delegate.

formal-parameters (Optional)

The parameter list of the delegate.

Code	MD-UML

`generic < class IT>`
`delegate IT GenDelegate(IT p1, IT%`
`p2);`
	 IT
	<<C++CLIDelegate>>
	<<C++CLIGeneric>>
	GenDelegate
	+(p1 : IT, p2 : IT"$%") : IT



Operation - <>

History : +(p1 : IT, p2 : IT"^") : IT [GenDelegate]

Parameters

Name	Type	Default Value	Direction
General			
p1	IT		inout
p2	IT		inout
	IT		return

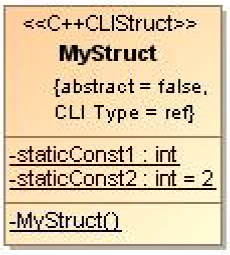
Copyright © 1998-2015 No Magic, Inc.

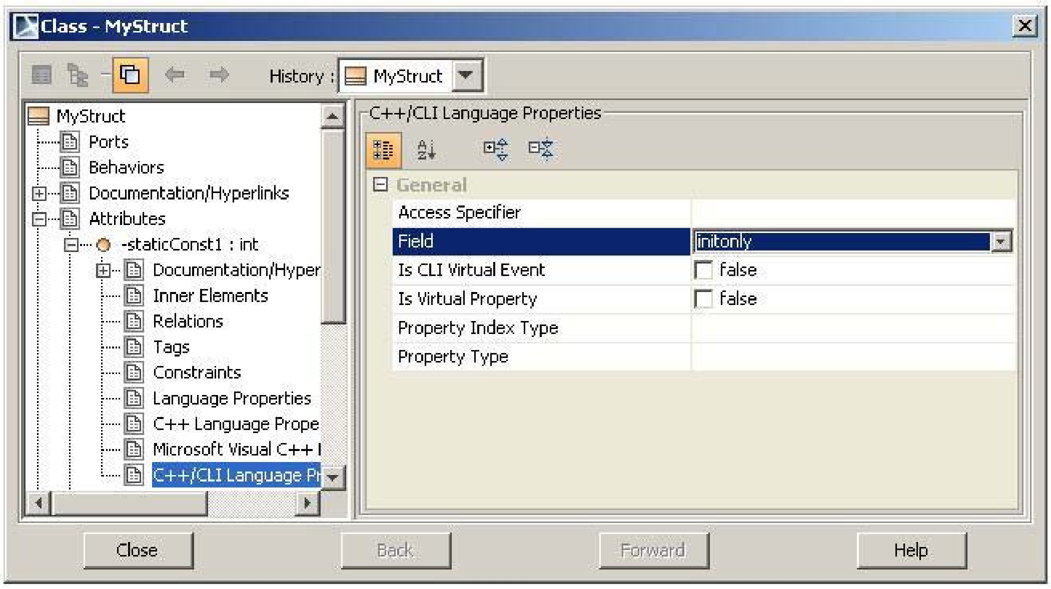
Clr Data Member option

initonly

initonly indicates that variable assignment can only occur as part of the declaration or in a static constructor in the same class.

Code	**MD-UML**

```
ref struct MyStruct {
   initonly
   static int staticConst1;

   initonly
   static int staticConst2 = 2;

   static MyStruct() {
      staticConst1 = 1;
   }
};
```
	 <<C++CLIStruct>>
	MyStruct
	{abstract = false,
	CLI Type = ref}
	-staticConst1 : int
	-staticConst2 : int = 2
	-MyStruct()
	MyStruct with «C++CLIStruct»
	staticConst1 and staticConst2 have «C++CLIAttribute» applied with tagged value **field** = initonly and **Is Static** = true



Access Specifier	
Field	initonly
Is CLI Virtual Event	false
Is Virtual Property	false
Property Index Type	
Property Type	

literal

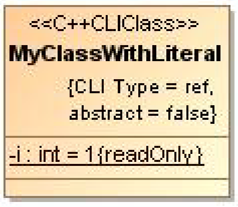
A variable (data member) marked as literal in a **/clr** compilation is the native equivalent of a **static const** variable.

A data member marked as **literal** must be initialized when declared and the value must be a constant integral, enum, or string type. Conversion from the type of the initialization expression to the type of the static const data-member must not require a user-defined conversion.

Code	MD-UML

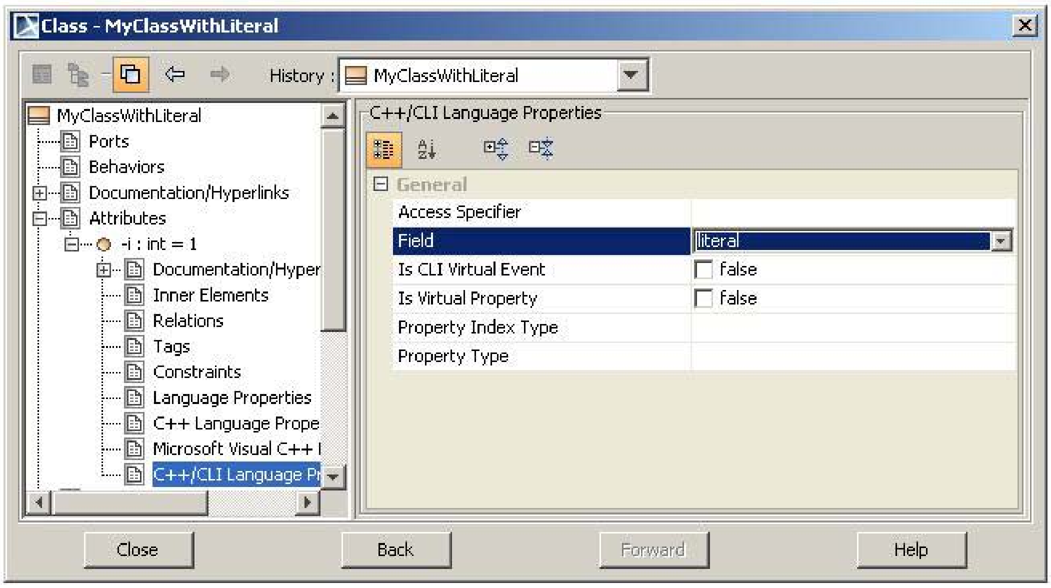
Copyright © 1998-2015 No Magic, Inc.

```
ref class MyClassWithLiteral
{
   literal int i = 1;
};
```



MyClassWithLiteral with «C++CLIClass»

Attribute i has «C++CLIAttribute» applied with tagged value **field** = literal and set to **Is Static =** true and **Is Read Only** = true



C++/CLI Language Properties

General

Access Specifier	
Field	literal
Is CLI Virtual Event	false
Is Virtual Property	false
Property Index Type	
Property Type	

Inheritance Keywords

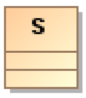
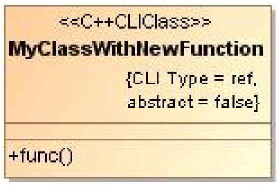
__single_inheritance, __multiple_inheritance, __virtual_inheritance
Grammar:
class [__single_inheritance] *class-name*;
class [__multiple_inheritance] *class-name*;
class [__virtual_inheritance] *class-name*;

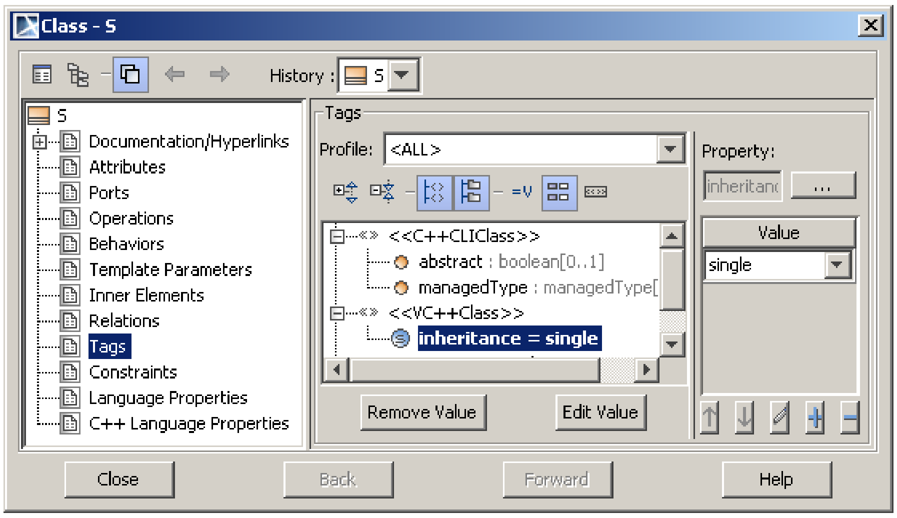
Parameter

class-name

Copyright © 1998-2015 No Magic, Inc.

The name of the class being declared.

Code	MD-UML
`class __single_inheritance S;`	 Class with «VC++Class» 



Class - S

History : S

S
- Documentation/Hyperlinks
- Attributes
- Ports
- Operations
- Behaviors
- Template Parameters
- Inner Elements
- Relations
- Tags
- Constraints
- Language Properties
- C++ Language Properties

Tags

Profile: <ALL>

```
<<C++CLIClass>>
    abstract : boolean[0..1]
    managedType : managedType[
<<VC++Class>>
    inheritance = single
```

Property: inheritanc

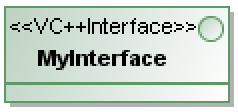
Value: single

Microsoft-Specific Native declaration keywords

__interface

Grammar:
modifier **__interface** interface-name **{**interface-definition**};**
A Visual C++ interface can be defined as follows:

- Can inherit from zero or more base interfaces.
- Cannot inherit from a base class.
- Can only contain public, pure virtual methods.
- Cannot contain constructors, destructors, or operators.
- Cannot contain static methods.
- Cannot contain data members; properties are allowed.

Code	MD-UML
`__interface MyInterface {};`	

Copyright © 1998-2015 No Magic, Inc.

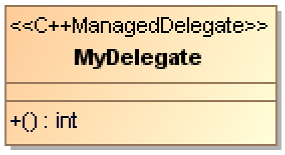
__delegate

Grammar:
__delegate function-declarator
A delegate is roughly equivalent to a C++ function pointer except for the following difference:

- A delegate can only be bound to one or more methods within a __gc class.

When the compiler encounters the **__delegate** keyword, a definition of a __gc class is generated. This __gc class has the following characteristics:

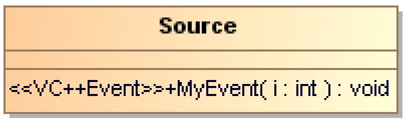
- It inherits from **System::MulticastDelegate**.
- It has a constructor that takes two arguments: a pointer to a __gc class or **NULL** (in the case of binding to a static method) and a fully qualified method of the specified type.
- It has a method called **Invoke**, whose signature matches the declared signature of the delegate.

Code	MD-UML
`__delegate int MyDelegate();`	

__event
Grammar:
__event method-declarator**;**
__event __interface interface-specifier**;**
__event member-declarator**;**

Native Events

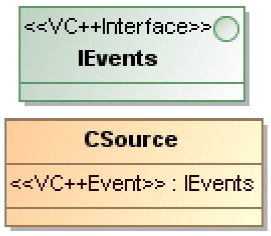
Code	MD-UML

```
class Source {
public:
    __event void MyEvent(int i);
};
```
	

Com Events

The **__interface** keyword is always required after **__event** for a COM event source.

Code	MD-UML

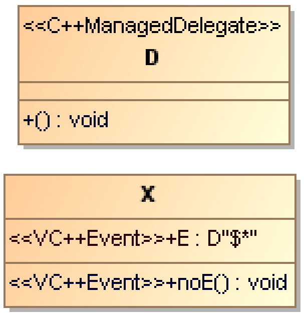
```
__interface IEvents {
  };
class CSource {
public:
    __event __interface IEvents;
};
```
	 <<VC++Interface>>
	IEvents
	CSource
	<<VC++Event>> : IEvents
	__event __interface IEvents;
	is mapped to attribute without name

Copyright © 1998-2015 No Magic, Inc.

NOTE This mapping produces the exceptional case for syntax checker. The syntax checker has to allow attribute without name for this case.

Managed Events

Code	MD-UML

`public __delegate void D();`
`public __gc class X {`
`public:`
`    __event D* E;`
`    __event void noE();`
`};`
	 <<C++ManagedDelegate>>
	D
	+() : void
	X
	<<VC++Event>>+E : D"$*"
	<<VC++Event>>+noE() : void

NOTE __delegate is in C++ Managed Profile whereas __event is in Microsoft Visual C++ Profile.

Microsoft-Specific Modifiers

Based addressing with __based

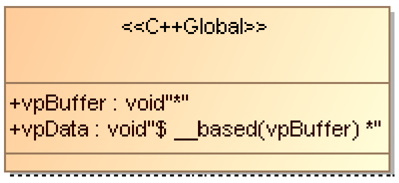
The **__based** keyword allows you to declare pointers based on pointers (pointers that are offsets from existing pointers).

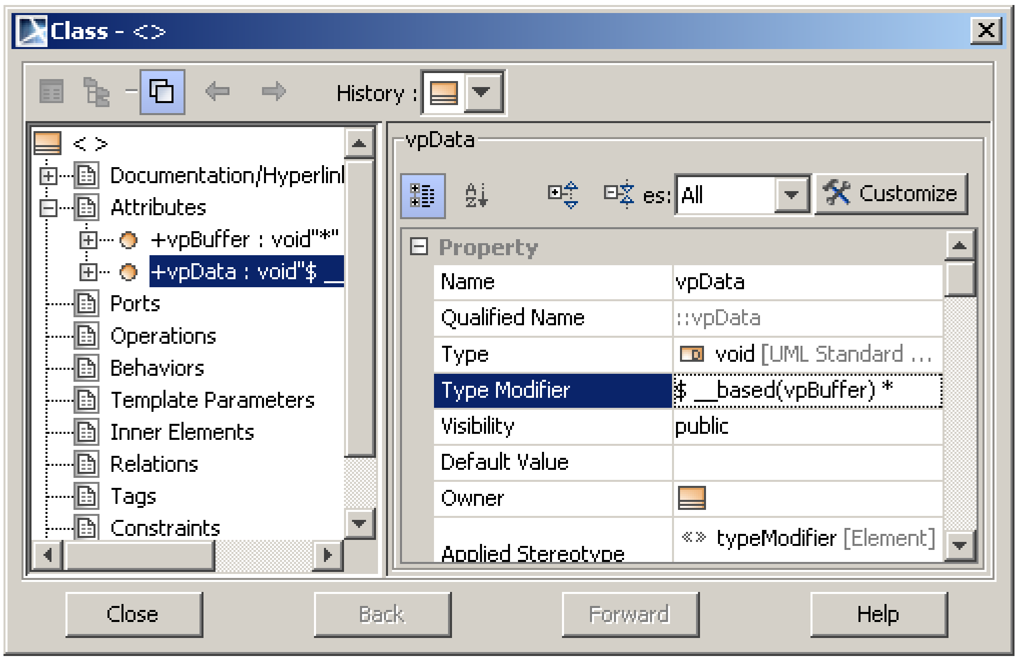
type __**based(** base **)** declarator

Example

```
// based_pointers1.cpp
// compile with: /c
void *vpBuffer;
struct llist_t {
   void __based( vpBuffer ) *vpData;
   struct llist_t __based( vpBuffer ) *llNext;
};
```

Code	MD-UML

`void *vpBuffer;`
`void __based(vpBuffer) *vpData;`
	 <<C++Global>>
	+vpBuffer : void"*"
	+vpData : void"$ __based(vpBuffer) *"



Property	
Name	vpData
Qualified Name	::vpData
Type	void [UML Standard ...
Type Modifier	$ __based(vpBuffer) *
Visibility	public
Default Value	
Owner	
Applied Stereotype	«» typeModifier [Element]

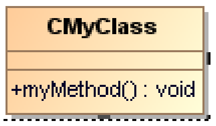
Function calling conventions

The Visual C/C++ compiler provides several different conventions for calling internal and external functions.

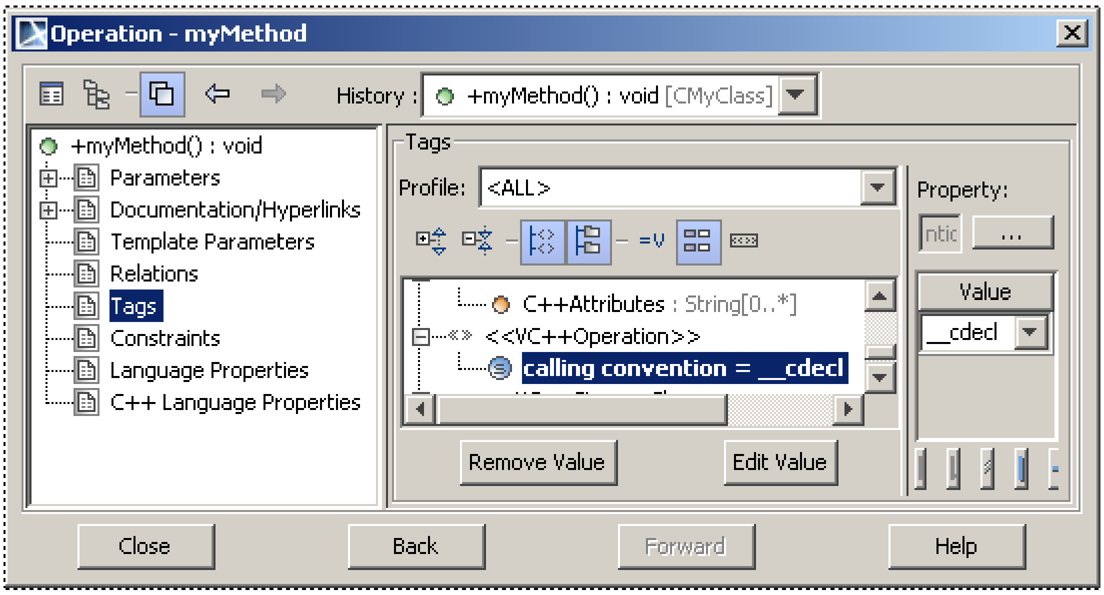
__cdecl

return-type **__cdecl** function-name[(argument-list)]
This is the default calling convention for C and C++ programs. Because the stack is cleaned up by the caller, it can do **vararg** functions. The **__cdecl** calling convention creates larger executables than **__stdcall**, because it requires each function call to include stack cleanup code. The following list shows the implementation of this calling convention.

Code	MD-UML

```
class CMyClass {
    void __cdecl myMethod();
};
```
	

Copyright © 1998-2015 No Magic, Inc.



__clrcall

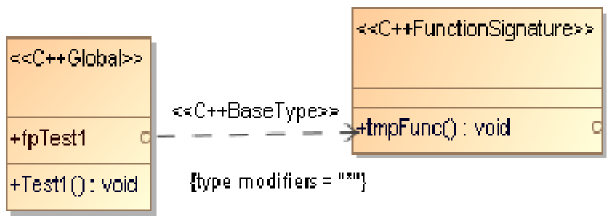
return-type **__clrcall** function-name[(argument-list)]

Specifies that a function can only be called from managed code. Use **__clrcall** for all virtual functions that will only be called from managed code. However, this calling convention cannot be used for functions that will be called from native code.

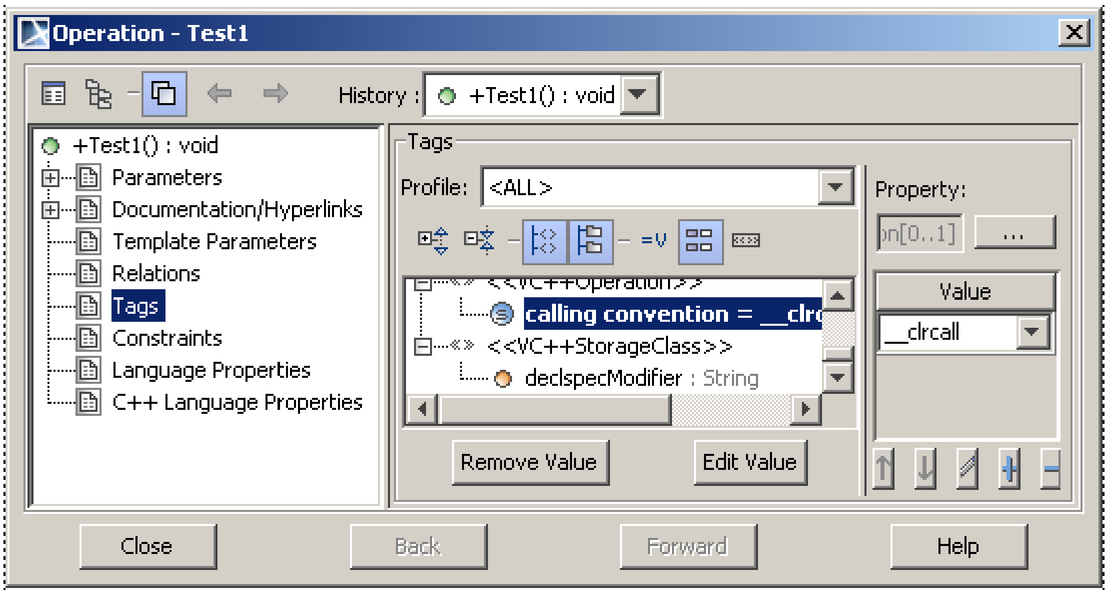
Example

```
// compile with: /clr:oldSyntax /LD
void __clrcall Test1( ) {}
void (__clrcall *fpTest1)( ) = &Test1;
```

Code	MD-UML

`void __clrcall Test1( ) {}`
`void (__clrcall *fpTest1)( ) = &Test1;`
	

Copyright © 1998-2015 No Magic, Inc.



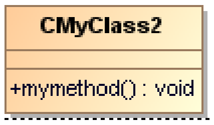
__stdcall

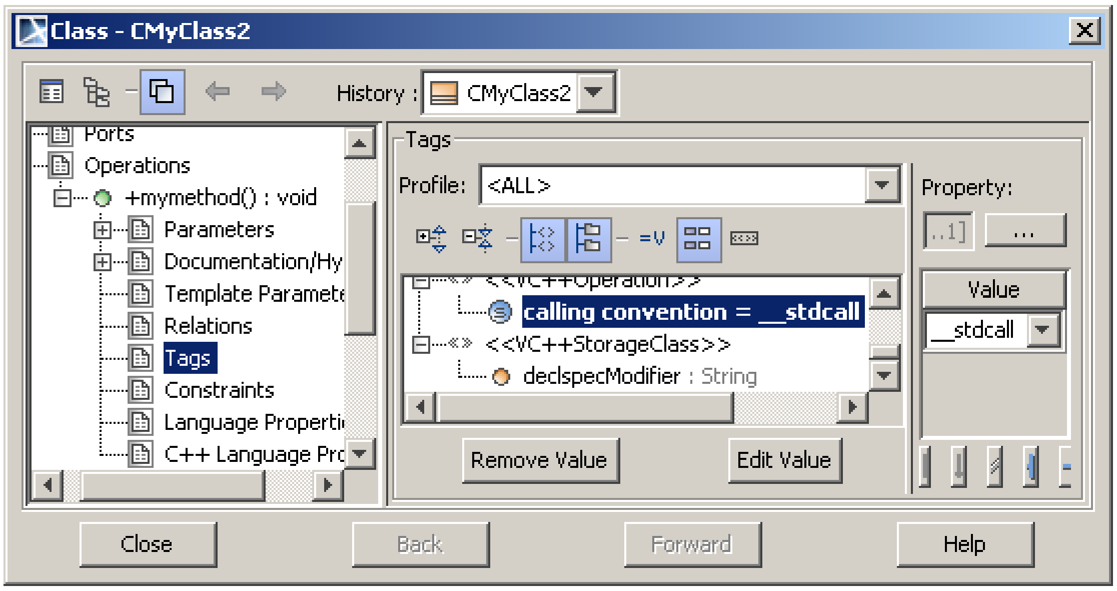
return-type __**stdcall** function-name[(argument-list)]

The __**stdcall** calling convention is used to call Win32 API functions. The callee cleans the stack, so the compiler makes **vararg** functions __**cdecl**. Functions that use this calling convention require a function prototype.

Copyright © 1998-2015 No Magic, Inc.

Code	MD-UML

`class CmyClass2 {`
`    void __stdcall mymethod();`
`};`
	


Class - CMyClass2

History : CMyClass2

- Ports
- Operations
  - +mymethod() : void
    - Parameters
    - Documentation/Hy
    - Template Paramete
    - Relations
    - Tags
    - Constraints
    - Language Properti
    - C++ Language Pro

Tags

Profile: <ALL>

- <<VC++Operation>>
  - calling convention = __stdcall
- <<VC++StorageClass>>
  - declspecModifier : String

Property:

Value: __stdcall

Remove Value | Edit Value

Close | Back | Forward | Help

__fastcall

return-type __**fastcall** function-name[(argument-list)]

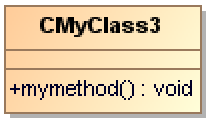
The __**fastcall** calling convention specifies that arguments to functions are to be passed in registers, when possible. The following list shows the implementation of this calling convention.

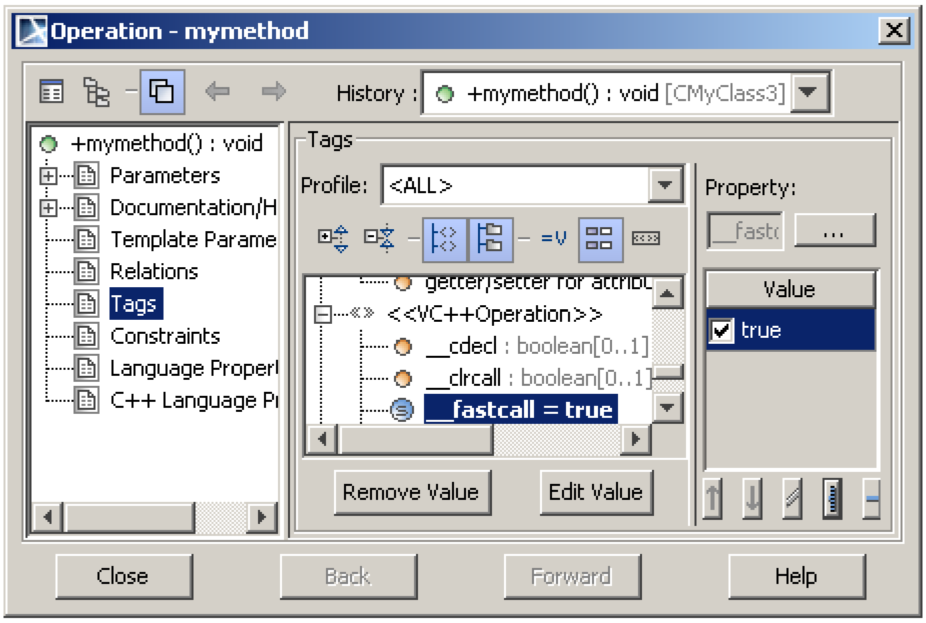
Example

```
class CmyClass3 {
   void __fastcall mymethod();
};
```

Copyright © 1998-2015 No Magic, Inc.

Code	MD-UML

`class CmyClass3 {`
`    void __fastcall mymethod();`
`};`
	



Operation - mymethod

History : +mymethod() : void [CMyClass3]

+mymethod() : void
- Parameters
- Documentation/H
- Template Parame
- Relations
- Tags
- Constraints
- Language Propert
- C++ Language Pr

Tags

Profile: <ALL>

```
getter/setter for attribu
<<VC++Operation>>
    __cdecl : boolean[0..1]
    __clrcall : boolean[0..1]
    __fastcall = true
```

Property: __fastc

Value
✓ true

__thiscall

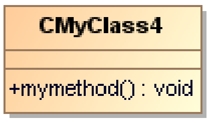
return-type __**thiscall** function-name[(argument-list)]

The __**thiscall** calling convention is used on member functions and is the default calling convention used by C++ member functions that do not use variable arguments. Under __**thiscall**, the callee cleans the stack, which is impossible for **vararg** functions. Arguments are pushed on the stack from right to left, with the **this** pointer being passed via register ECX, and not on the stack, on the x86 architecture.

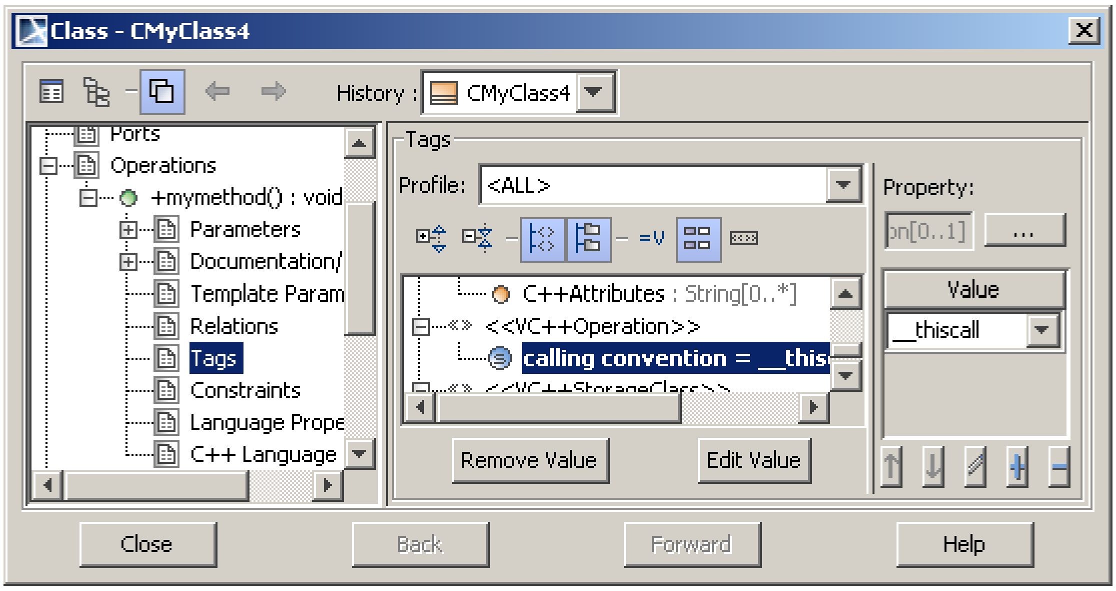
Example

```
// compile with: /c /clr:oldSyntax
class CmyClass4 {
    void __thiscall mymethod();
    void __clrcall mymethod2();
};
```

Code	MD-UML

`class CmyClass4 {`
`    void __thiscall mymethod();`
`};`
	

Copyright © 1998-2015 No Magic, Inc.



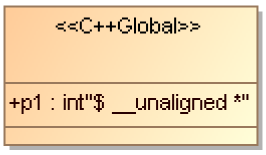
__unaligned

type **__unaligned** pointer_identifier

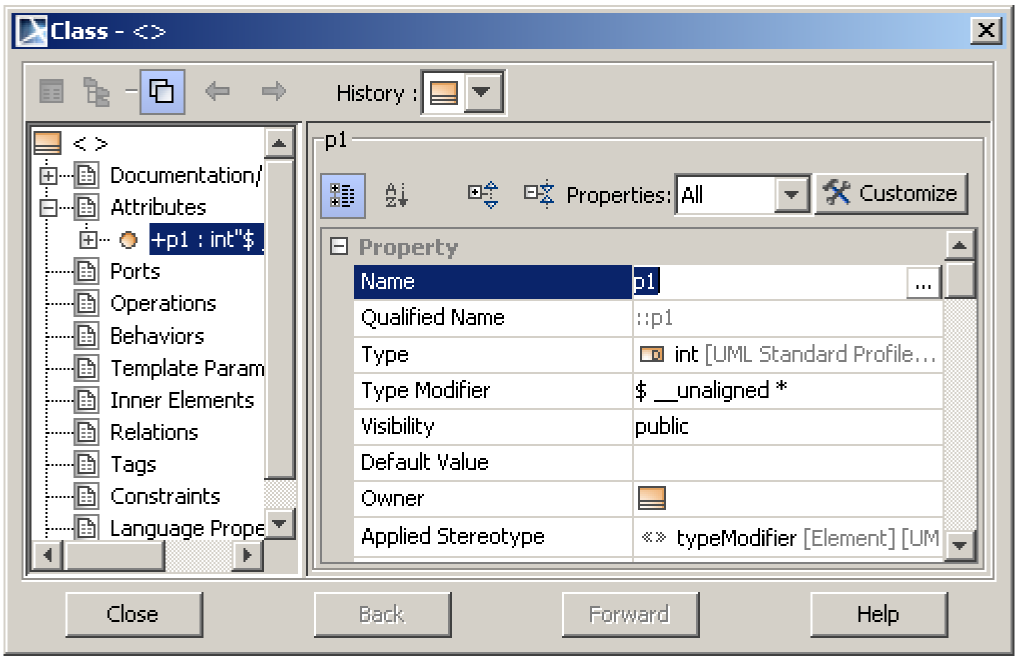
When a pointer is declared as **__unaligned**, the compiler assumes that the type or data pointed to is not aligned. **__unaligned** is only valid in compilers for x64 and the Itanium Processor Family (IPF).

Example

```
// compile with: /c
// processor: x64 IPF
#include <stdio.h>
int main() {
   char buf[100];
   int __unaligned *p1 = (int*)(&buf[37]);
   int *p2 = (int *)p1;
   *p1 = 0;   // ok
   __try {
      *p2 = 0;  // throws an exception
   }
   __except(1) {
      puts("exception");
   }}
```

Code	MD-UML
`int __unaligned *p1;`	

Copyright © 1998-2015 No Magic, Inc.



Property	
Name	p1
Qualified Name	::p1
Type	int [UML Standard Profile…
Type Modifier	$ __unaligned *
Visibility	public
Default Value	
Owner	
Applied Stereotype	«» typeModifier [Element] [UM

__w64

type **__w64** identifier

Parameters

type

One of the three types that might cause problems in code being ported from a 32-bit to a 64-bit compiler: **int**, **long**, or a pointer.

identifier

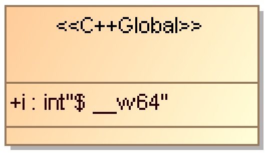
The identifier for the variable you are creating.

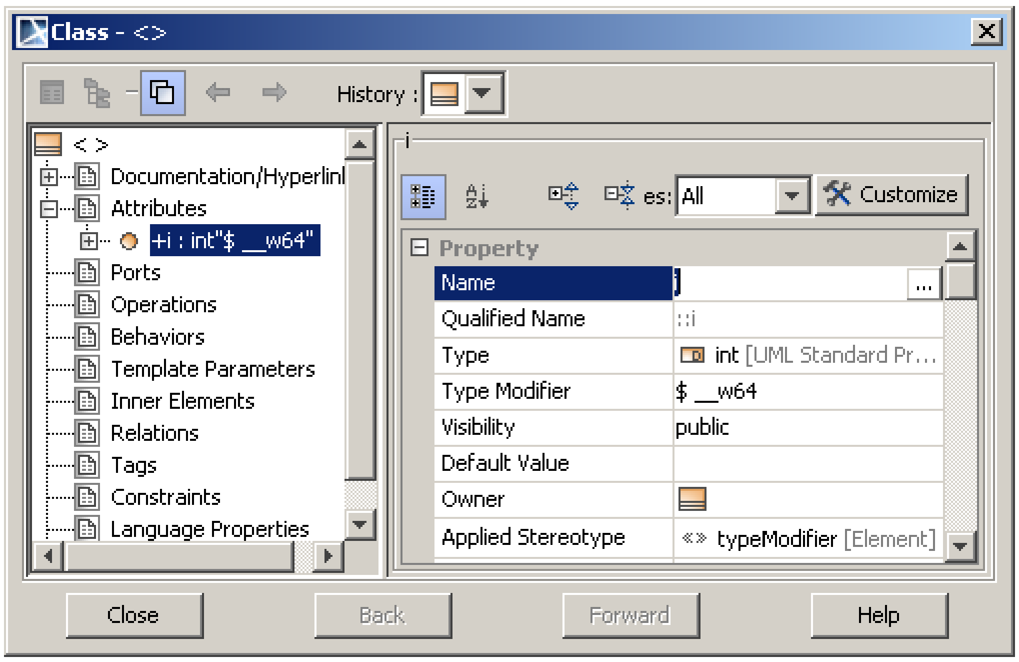
__w64 lets you mark variables, such that when you compile with /Wp64 the compiler will report any warnings that would be reported if you were compiling with a 64-bit compiler.

Example

```
// compile with: /W3 /Wp64
typedef int Int_32;
#ifdef _WIN64
typedef __int64 Int_Native;
#else
typedef int __w64 Int_Native;
#endif
int main() {
   Int_32 i0 = 5;
   Int_Native i1 = 10;
   i0 = i1;   // C4244 64-bit int assigned to 32-bit int

   // char __w64 c;  error, cannot use __w64 on char

}
```

Copyright © 1998-2015 No Magic, Inc.

Code	MD-UML
`int __w64 i;`	



< >
- Documentation/Hyperlinks
- Attributes
  - +i : int"$ __w64"
- Ports
- Operations
- Behaviors
- Template Parameters
- Inner Elements
- Relations
- Tags
- Constraints
- Language Properties

Property	
Name	i
Qualified Name	::i
Type	int [UML Standard Pr...
Type Modifier	$ __w64
Visibility	public
Default Value	
Owner	
Applied Stereotype	typeModifier [Element]

Extended storage-class attributes with __declspec

The extended attribute syntax for specifying storage-class information uses the **__declspec** keyword, which specifies that an instance of a given type is to be stored with a Microsoft-specific storage-class attribute listed below. Examples of other storage-class modifiers include the **static** and **extern** keywords. However, these key-words are part of the ANSI specification of the C and C++ languages, and as such are not covered by extended attribute syntax. The extended attribute syntax simplifies and standardizes Microsoft-specific extensions to the C and C++ languages.

decl-specifier:

__declspec (extended-decl-modifier-seq **)**

extended-decl-modifier-seq:

extended-decl-modifieropt

extended-decl-modifier extended-decl-modifier-seq

extended-decl-modifier:

align(#**)**

allocate("segname**")**

appdomain

deprecated

dllimport

dllexport

jitintrinsic

naked

Copyright © 1998-2015 No Magic, Inc.

noalias
noinline
noreturn
nothrow
novtable
process
property({get=get_func_name**|,put**=put_func_name**})**
restrict
selectany
thread
uuid("ComObjectGUID**")**

White space separates the declaration modifier sequence.

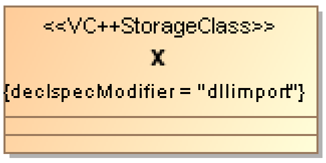
Extended attribute grammar supports these Microsoft-specific storage-class attributes: **align**, **allocate**, **appdomain**, **deprecated**, **dllexport**, **dllimport**, **jitintrinsic**, **naked**, **noalias**, **noinline**, **noreturn**, **nothrow**, **novtable**, **process**, **restrict**, **selectany**, and **thread**. It also supports these COM-object attributes: **property** and **uuid**.

The **dllexport**, **dllimport**, **naked**, **noalias**, **nothrow**, **property**, **restrict**, **selectany**, **thread**, and **uuid** storage-class attributes are properties only of the declaration of the object or function to which they are applied. The **thread** attribute affects data and objects only. The **naked** attribute affects functions only. The **dllimport** and **dllexport** attributes affect functions, data, and objects. The **property**, **selectany**, and **uuid** attributes affect COM objects.

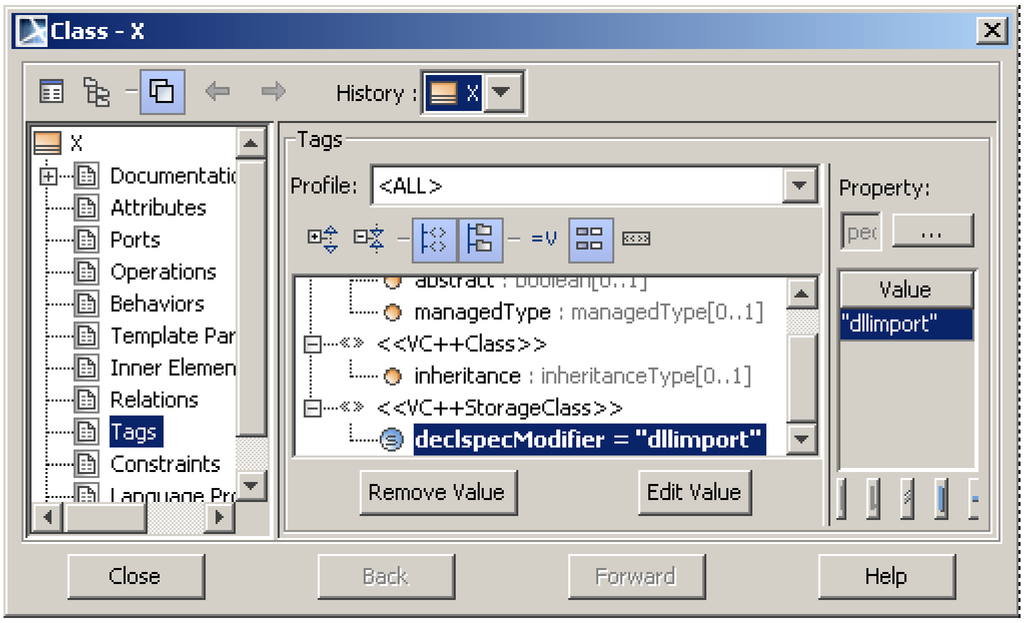
The **__declspec** keywords should be placed at the beginning of a simple declaration. The compiler ignores, without warning, any **__declspec** keywords placed after * or & and in front of the variable identifier in a declaration.

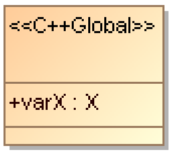
Note:

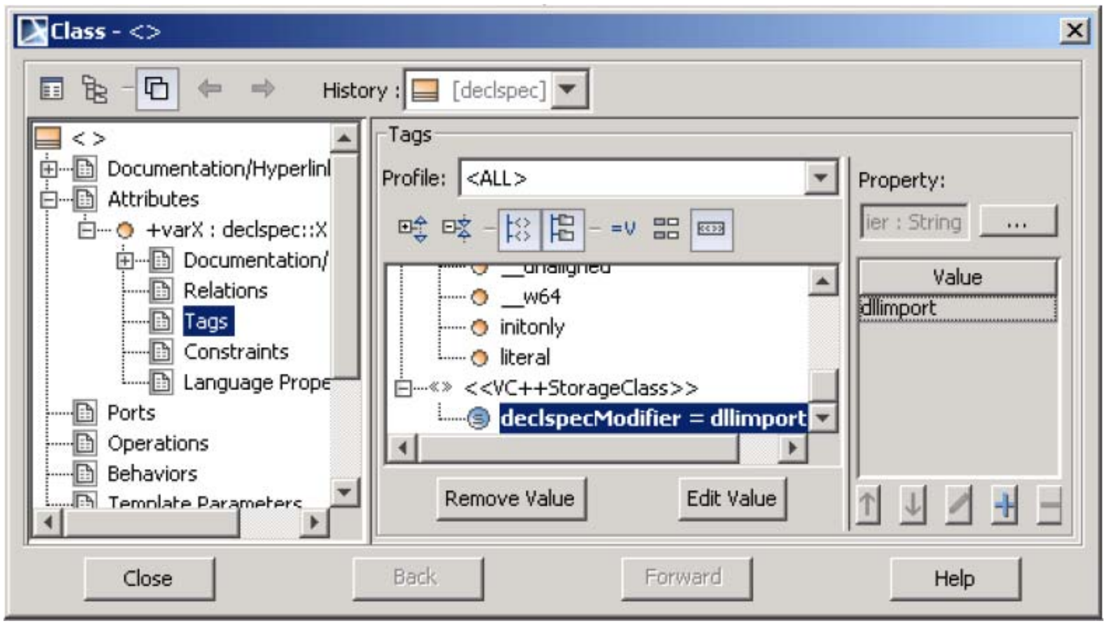
- A **__declspec** attribute specified in the beginning of a user-defined type declaration applies to the variable of that type.
- A **__declspec** attribute placed after the **class** or **struct** keyword applies to the user-defined type.

Code	MD-UML
`class __declspec(dllimport) X {};`	 <<VC++StorageClass>> X {declspecModifier = "dllimport"}

Copyright © 1998-2015 No Magic, Inc.



Code	MD-UML
`__declspec(dllimport) class X {} varX;`	



__restrict

The __**restrict** keyword is valid only on variables, and __**declspec(restrict)** is only valid on function declarations and definitions.

Copyright © 1998-2015 No Magic, Inc.

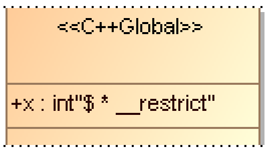
When **__restrict** is used, the compiler will not propagate the no-alias property of a variable. That is, if you assign a **__restrict** variable to a non-**__restrict** variable, the compiler will not imply that the non-**__restrict** variable is not aliased.

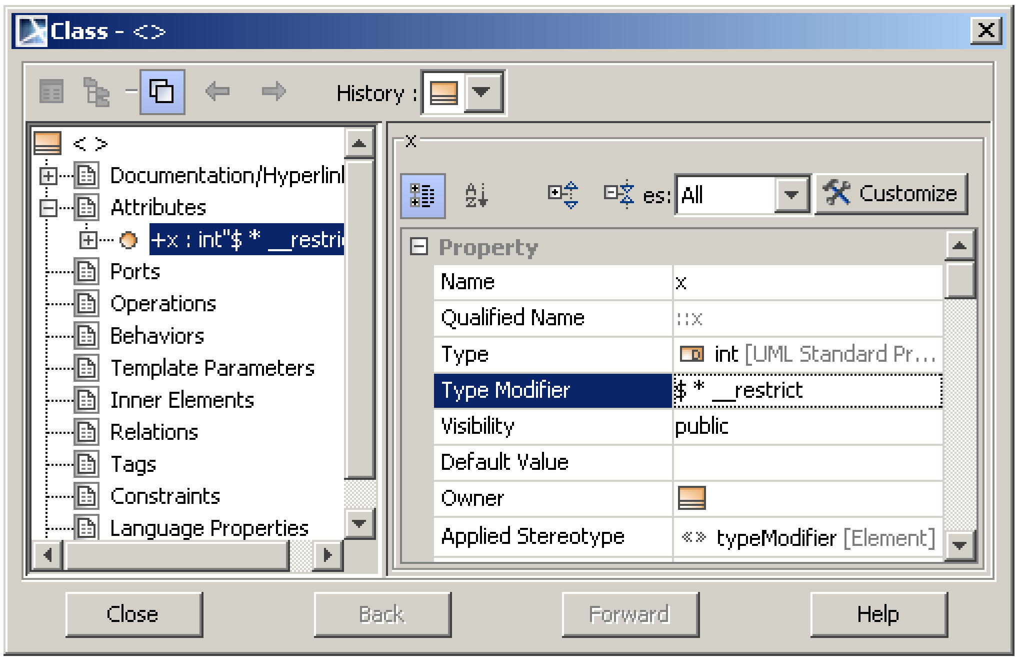
Generally, if you affect the behavior of an entire function, it is better to use the **__declspec** than the keyword.

__restrict is similar to **restrict** from the C99 spec, but **__restrict** can be used in C++ or C programs.

Example

```
// __restrict_keyword.c
// compile with: /LD
// In the following function, declare a and b as disjoint arrays
// but do not have same assurance for c and d.
void sum2(int n, int * __restrict a, int * __restrict b,
          int * c, int * d) {
   int i;
   for (i = 0; i < n; i++) {
      a[i] = b[i] + c[i];
      c[i] = b[i] + d[i];
    }
}

// By marking union members as __restrict, tells the compiler that
// only z.x or z.y will be accessed in any given scope.
union z {
   int * __restrict x;
   double * __restrict y;
};
```

Code	MD-UML
`int * __restrict x;`	



Property	
Name	x
Qualified Name	::x
Type	int [UML Standard Pr...
Type Modifier	$ * __restrict
Visibility	public
Default Value	
Owner	
Applied Stereotype	«» typeModifier [Element]

Copyright © 1998-2015 No Magic, Inc.

__forceinline, __inline

The insertion (called inline expansion or inlining) occurs only if the compiler's cost/benefit analysis show it to be profitable. Inline expansion alleviates the function-call overhead at the potential cost of larger code size.

The **__forceinline** keyword overrides the cost/benefit analysis and relies on the judgment of the programmer instead. Exercise caution when using **__forceinline**. Indiscriminate use of **__forceinline** can result in larger code with only marginal performance gains or, in some cases, even performance losses (due to increased paging of a larger executable, for example).

Using inline functions can make your program faster because they eliminate the overhead associated with function calls. Functions expanded inline are subject to code optimizations not available to normal functions.

The compiler treats the inline expansion options and keywords as suggestions. There is no guarantee that functions will be inlined. You cannot force the compiler to inline a particular function, even with the **__forceinline** keyword. When compiling with **/clr**, the compiler will not inline a function if there are security attributes applied to the function.

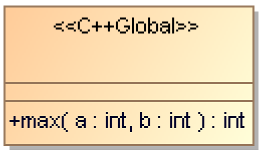
The **inline** keyword is available only in C++. The **__inline** and **__forceinline** keywords are available in both C and C++. For compatibility with previous versions, **_inline** is a synonym for **__inline**.

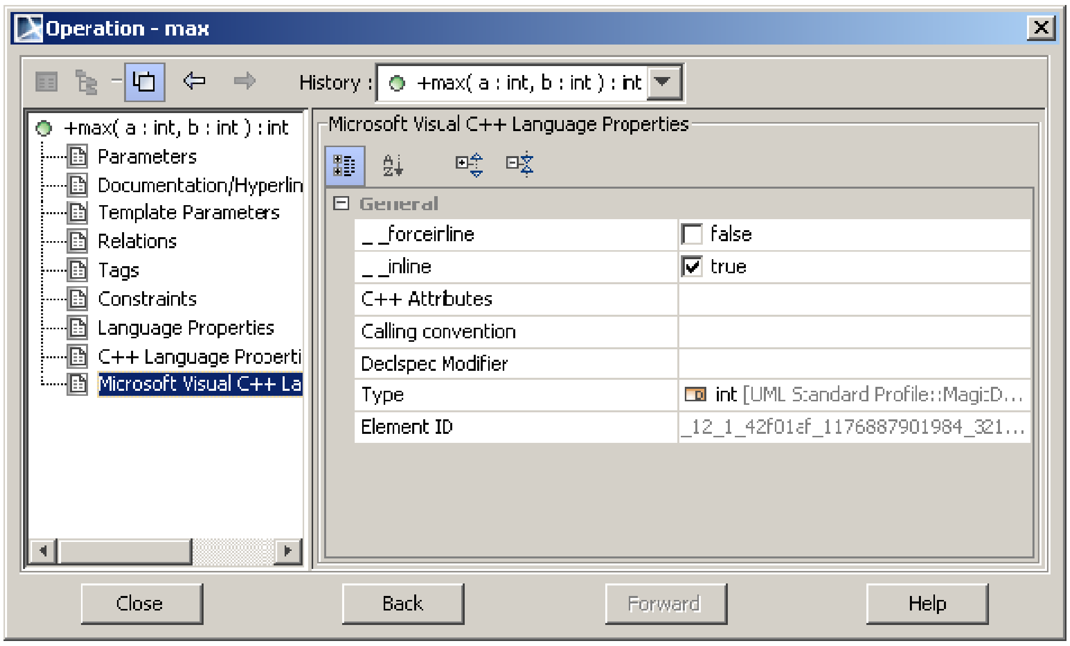
Grammar:

__inline function_declarator**;**

__forceinline function_declarator**;**

Code	MD-UML

```
__inline int max( int a , int b ) {
    if( a > b )
        return a;
    return b;
}
```
	



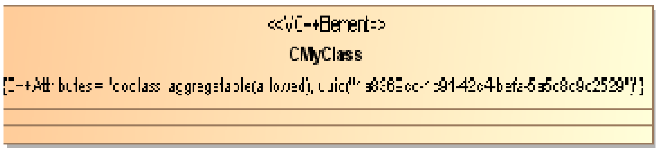
__forceinline	☐ false
__inline	☑ true
C++ Attributes	
Calling convention	
Declspec Modifier	
Type	int [UML Standard Profile::MagicD...
Element ID	_12_1_42f01af_1176887901984_321...

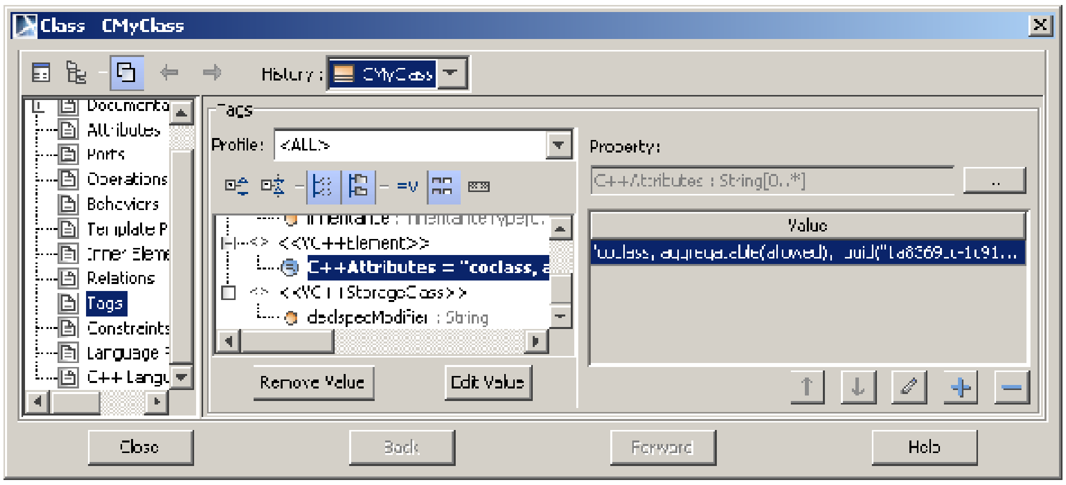
Copyright © 1998-2015 No Magic, Inc.

C++ Attributes

Attributes are designed to simplify COM programming and .NET Framework common language runtime development. When you include attributes in your source files, the compiler works with provider DLLs to insert code or modify the code in the generated object files.

Code	MD-UML

`[ coclass,`
`aggregatable(allowed),`
`  uuid("1a8369cc-1c91-`
`42c4-befa-5a5d8c9d2529")]`
`class CMyClass {};`
	 <<VC++Element>>
	CMyClass
	[C++Attributes='coclass, aggregatable(allowed), uuid("1a8369cc-1c91-42c4-befa-5a5d8c9d2529")']



Profile: <ALL>

Property: C++Attributes : String[0..*]

```
inheritance : InheritanceType[C...
<<VC++Element>>
    C++Attributes = "coclass, a
<<VC++StorageClass>>
    declspecModifier : String
```

Value
"coclass, aggregatable(allowed), uuid("1a8369cc-1c91...

Copyright © 1998-2015 No Magic, Inc.

C# 2.0 Description

The MagicDraw UML C# Code Engineering Project is responsible for providing round-trip functionality between The MagicDraw UML and C# codes. In the current version of this project, it supports up to C# version 3.0.

Generics

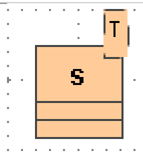
Generics permit classes, structs, interfaces, delegates, and methods to be parameterized by the types of data they store and manipulate. Generic class declaration should be mapped to the UML classifier (class or interface) with a template parameter. (See the detail of Generics in C# Specification chapter 20)

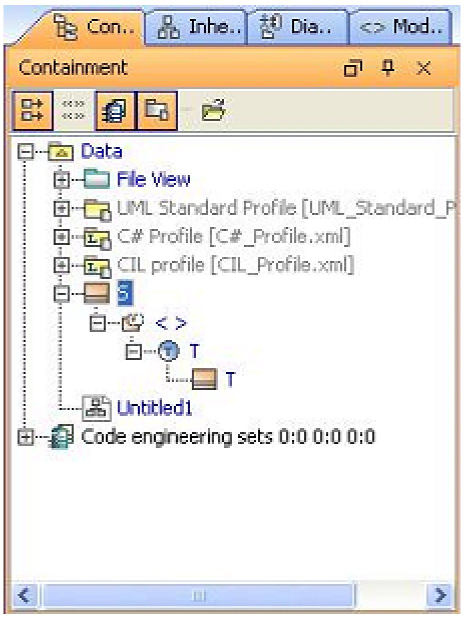
Additionally, Generics still affect other parts of the program structure such as attribute, operation, parameter, parent class, and overloading operators. The mappings of these are shown in the next part of this document.

Generic Class

Class S has one template parameter named T. The default type of T is Class.

Code	MD-UML

`public class S<T>`
`{`
 	
`}`
	

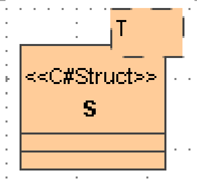

Con..
Inhe..
Dia..
<> Mod..
Containment
Data
File View
UML Standard Profile [UML_Standard_P
C# Profile [C#_Profile.xml]
CIL profile [CIL_Profile.xml]
S
< >
T
T
Untitled1
Code engineering sets 0:0 0:0 0:0

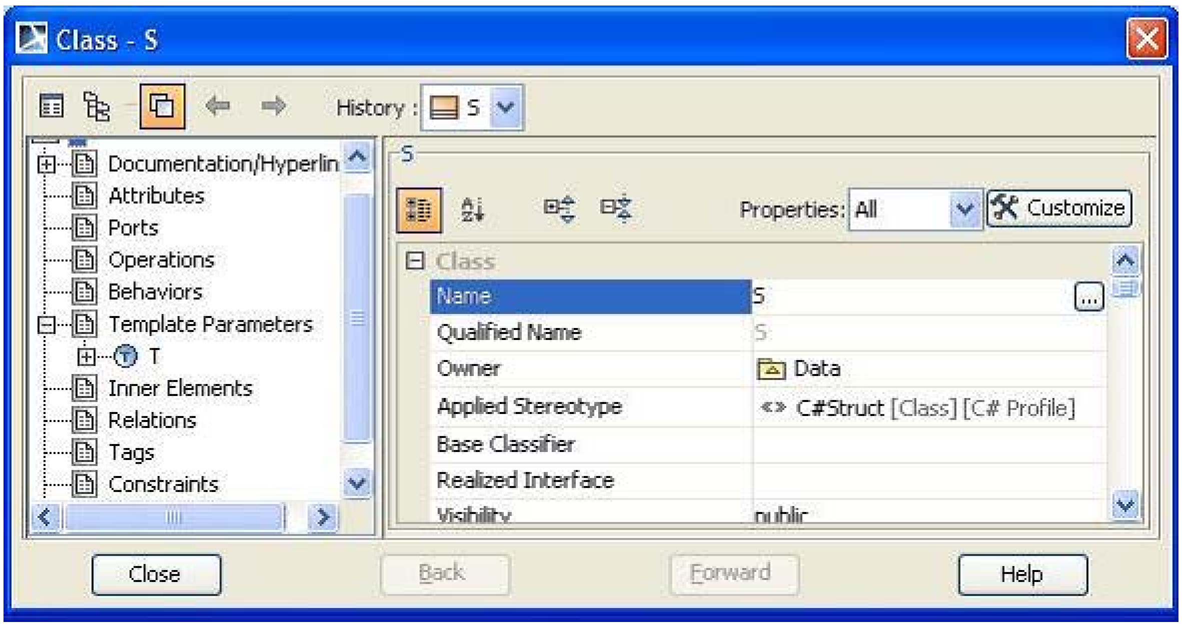
Copyright © 1998-2015 No Magic, Inc.

Generic Struct

The type parameter of generic struct is created the same as generic class, but we apply «C#Struct» stereotype to the model.

Code	MD-UML

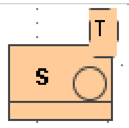
`struct S<T>`
`{`
 	
`}`
	

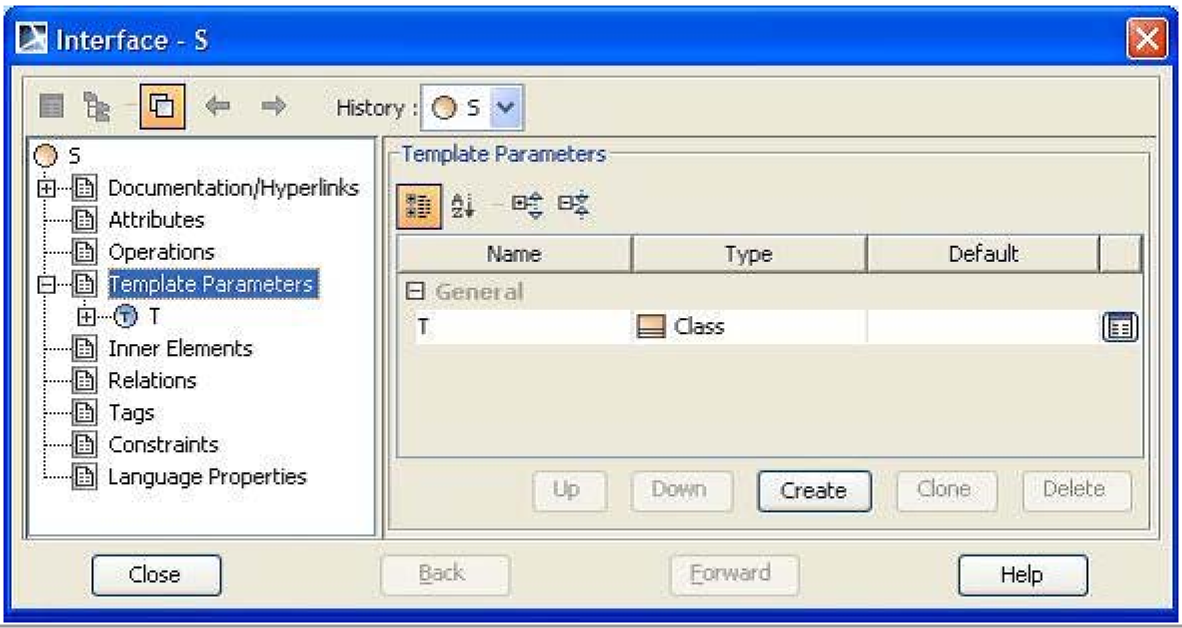


Class	
Name	S
Qualified Name	S
Owner	Data
Applied Stereotype	C#Struct [Class] [C# Profile]
Base Classifier	
Realized Interface	
Visibility	public

Copyright © 1998-2015 No Magic, Inc.

Generic Interface

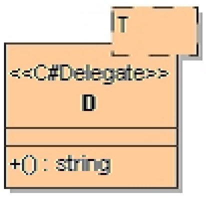
Code	MD-UML
`interface S<T>` `{` `}`	



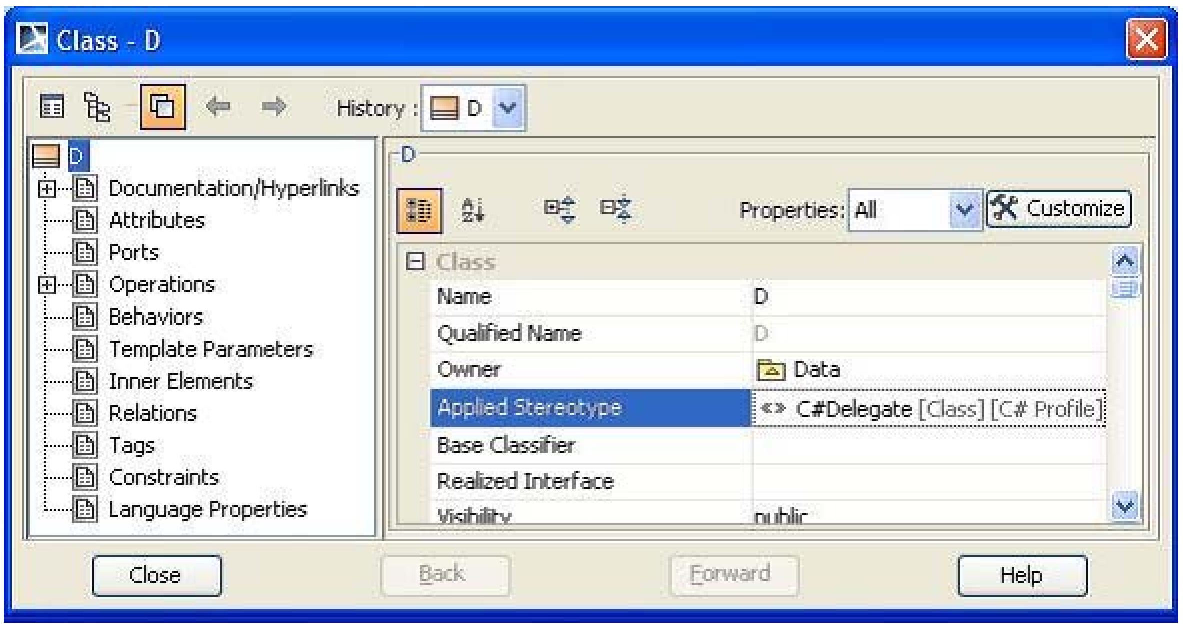
Name	Type	Default
General		
T	Class	

Generic Delegate

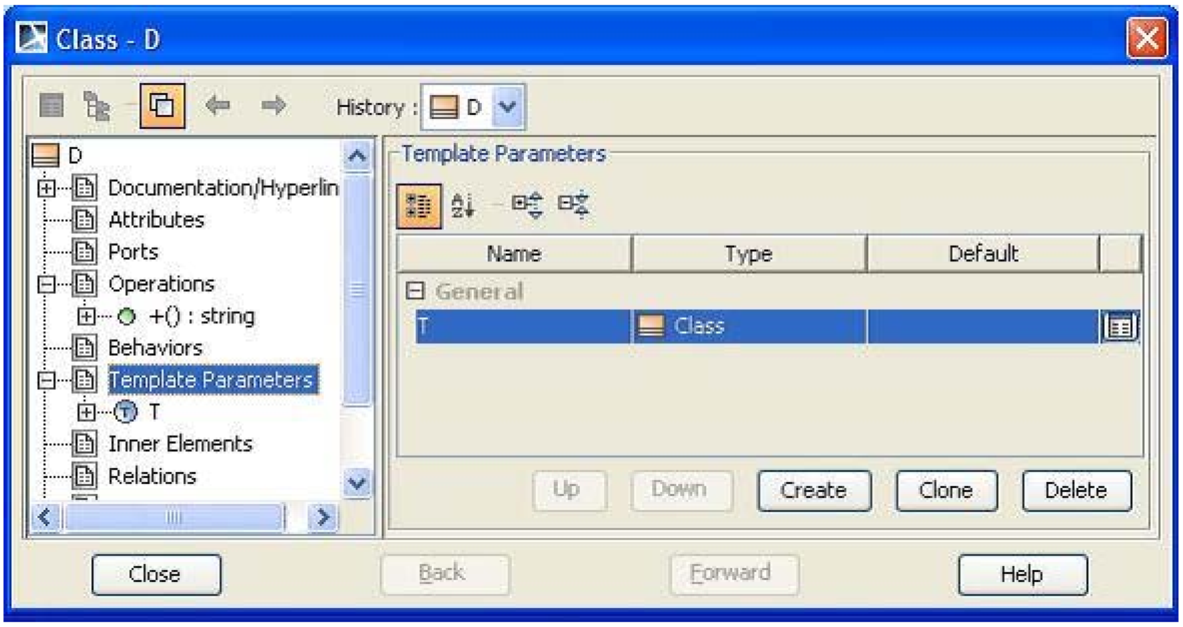
To create a generic delegate, we create a class model and apply the «C#Delegate» to the model. We, then, create an empty named method with delegate return type, and add template parameter like a normal generic class.

Code	MD-UML
`delegate string D<T>();`	

Copyright © 1998-2015 No Magic, Inc.



Class	
Name	D
Qualified Name	D
Owner	Data
Applied Stereotype	«» C#Delegate [Class] [C# Profile]
Base Classifier	
Realized Interface	
Visibility	public



Template Parameters

Name	Type	Default
General		
T	Class	

Generic Attribute

The type of attributes in the class can be generic binding (template binding) or template parameter of owner class.

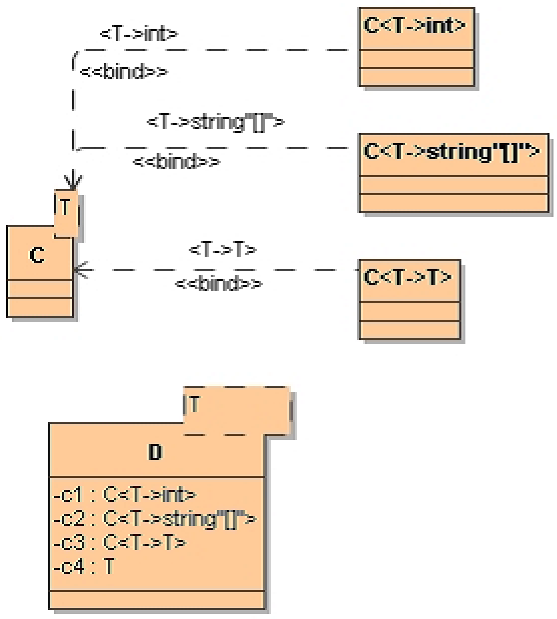
The example code shows attributes that use template binding as its type

Copyright © 1998-2015 No Magic, Inc.

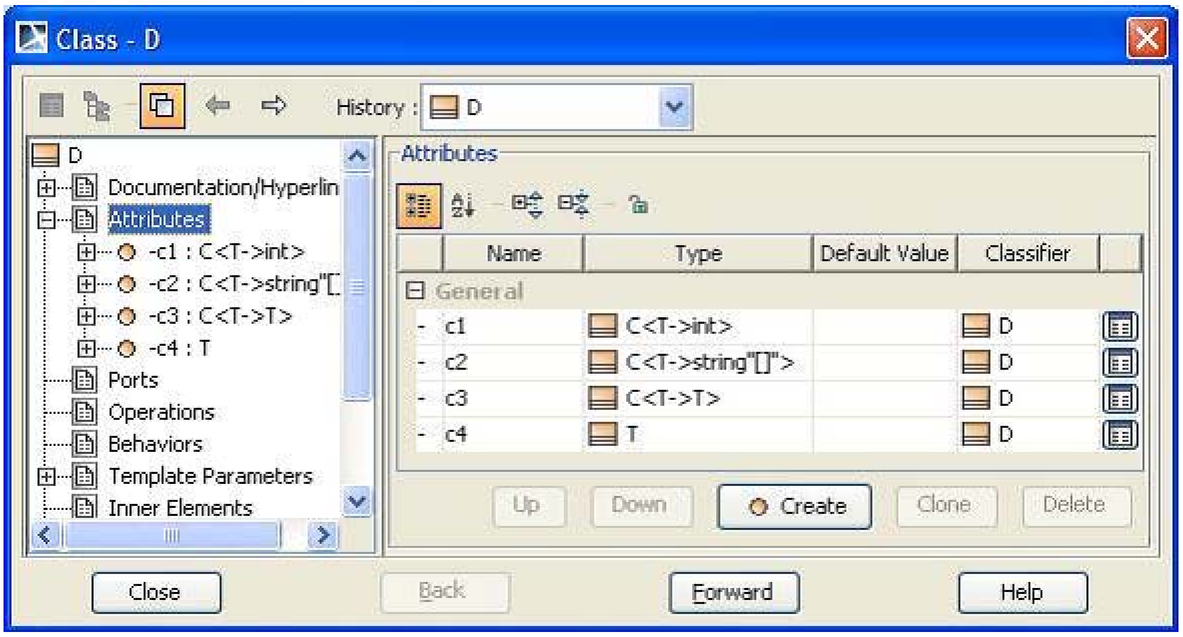
<table>
<tr><th>Code</th><th>MD-UML</th></tr>
<tr><td>

```
class C<T>
{

}
class D<T>
{
        private C<int> c1;
        private C<string[]> c2;
        private C<T> c3;
        private T c4;
}
```

</td><td>



</td></tr>
</table>



	Name	Type	Default Value	Classifier
-	c1	C<T->int>		D
-	c2	C<T->string"[]">		D
-	c3	C<T->T>		D
-	c4	T		D

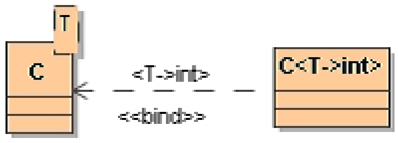
The following shows how to create each attribute.

<table>
<tr><td>

```
..
Private C<int> c1;
```

</td><td>

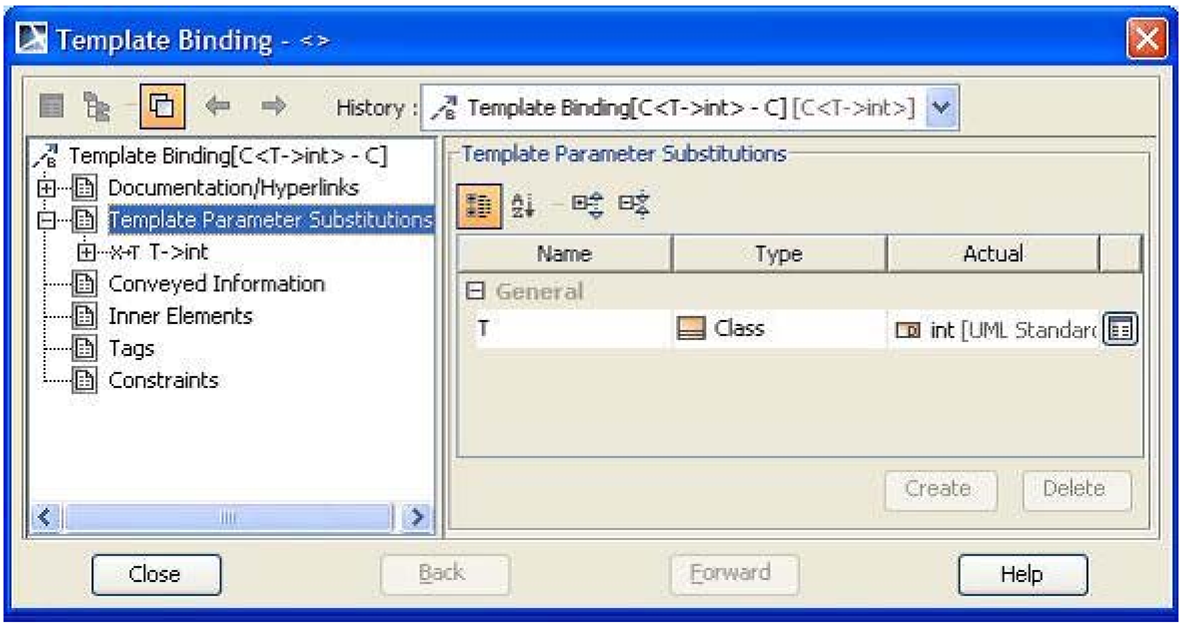
Template binding for C<int>



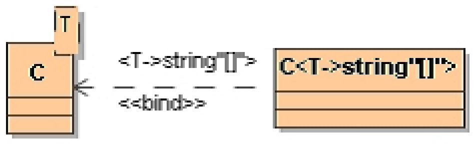
</td></tr>
</table>

Copyright © 1998-2015 No Magic, Inc.

Open the specification of template binding link and add the binding type in Template Parameter Substitutions section. For binding with int datatype, select data type int in the Actual.

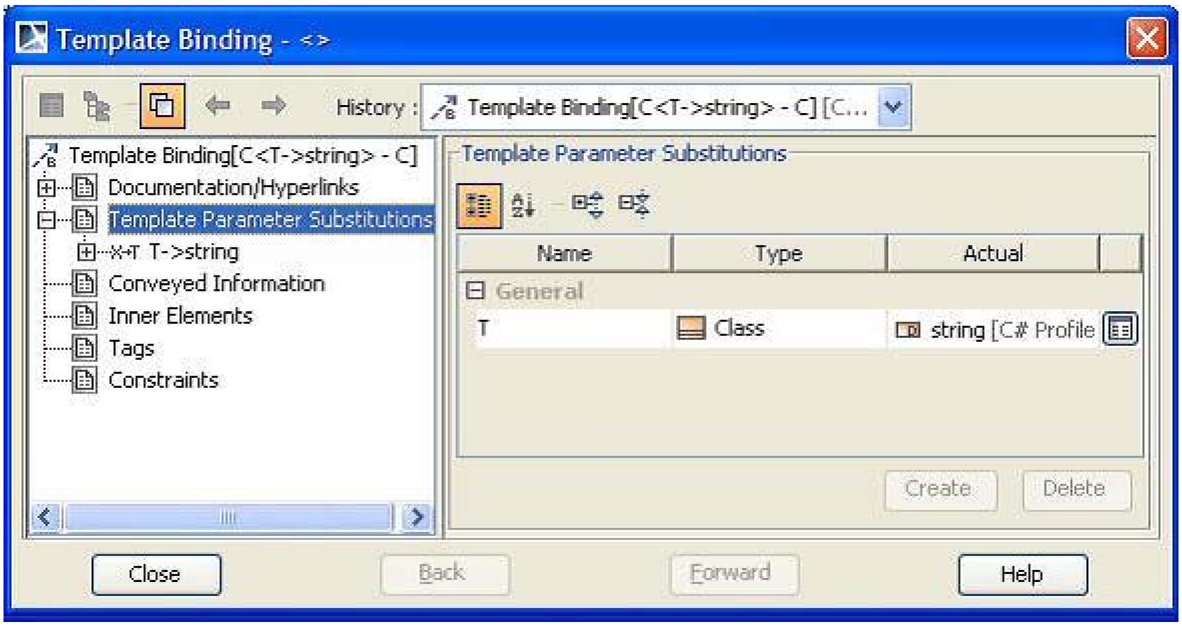


Name	Type	Actual
General		
T	Class	int [UML Standard

```..\nprivate C<string[]> c2;```	Template binding for C<string[]> 

Copyright © 1998-2015 No Magic, Inc.

Create binding type as usual.


Template Binding - <>

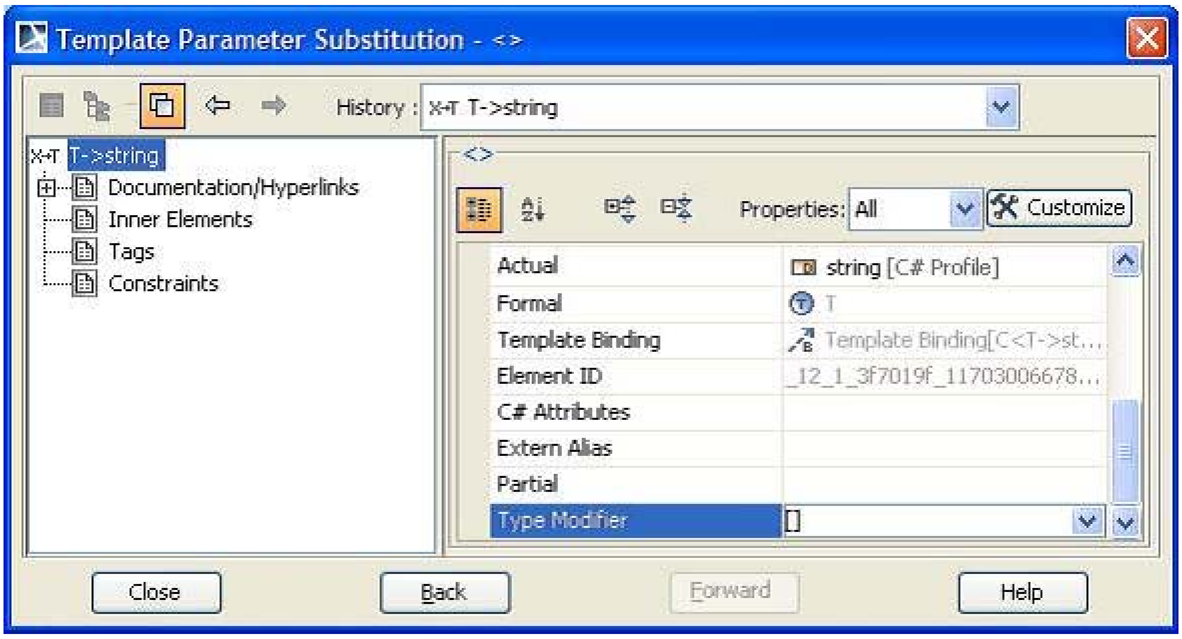
History : Template Binding[C<T->string> - C] [C...

- Template Binding[C<T->string> - C]
  - Documentation/Hyperlinks
  - Template Parameter Substitutions
    - T->string
  - Conveyed Information
  - Inner Elements
  - Tags
  - Constraints

Template Parameter Substitutions

Name	Type	Actual
General		
T	Class	string [C# Profile

To add type modifier [] to string, open the specification of template parameter substitution, and then add [] to Type Modifier property.


Template Parameter Substitution - <>

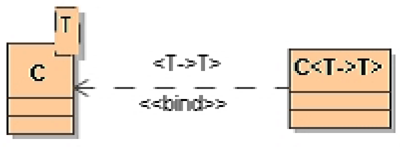
History : T->string

- T->string
  - Documentation/Hyperlinks
  - Inner Elements
  - Tags
  - Constraints

<>

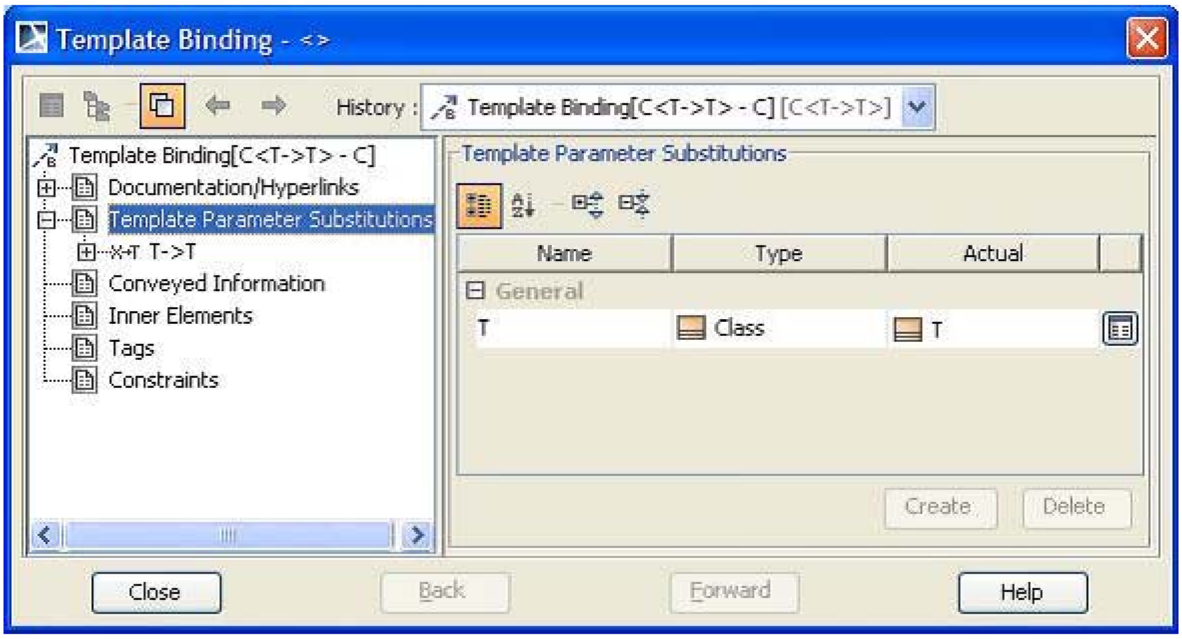
Properties: All

Actual	string [C# Profile]
Formal	T
Template Binding	Template Binding[C<T->st...
Element ID	_12_1_3f7019f_11703006678...
C# Attributes	
Extern Alias	
Partial	
Type Modifier	[]

	Template binding for C<T>
.. `private C<T> c3;`	

Copyright © 1998-2015 No Magic, Inc.

To create template binding for C<T>, we have to select the correct T element for the Actual. In this case, the binding type T is the template parameter of its owner class that is D<T>.	


Template Binding - < >

History : Template Binding[C<T->T> - C] [C<T->T>]

- Template Binding[C<T->T> - C]
  - Documentation/Hyperlinks
  - Template Parameter Substitutions
    - T->T
  - Conveyed Information
  - Inner Elements
  - Tags
  - Constraints

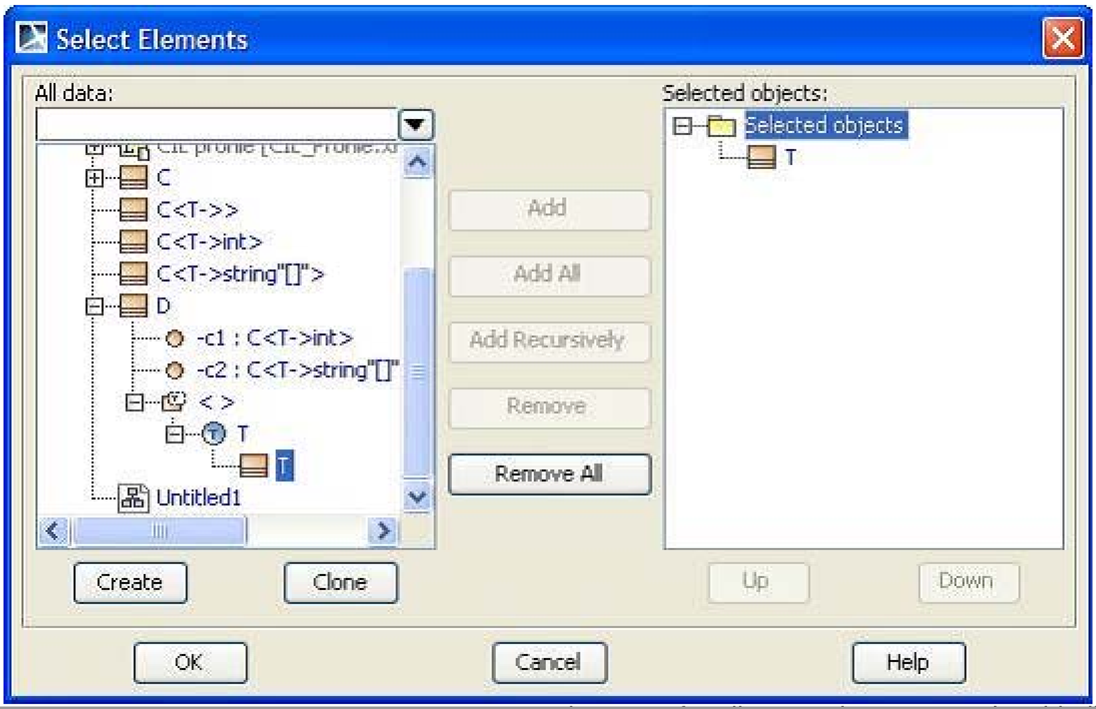
Template Parameter Substitutions

Name	Type	Actual
General		
T	Class	T

Create Delete

Close Back Forward Help

The following panel shows the correct type for creating template binding type T.


Select Elements

All data:

- CIL profile [CIL_Profile.x
- C
- C<T->>
- C<T->int>
- C<T->string"[]">
- D
  - -c1 : C<T->int>
  - -c2 : C<T->string"[]"
  - < >
    - T
      - T
- Untitled1

Add / Add All / Add Recursively / Remove / Remove All

Selected objects:

- Selected objects
  - T

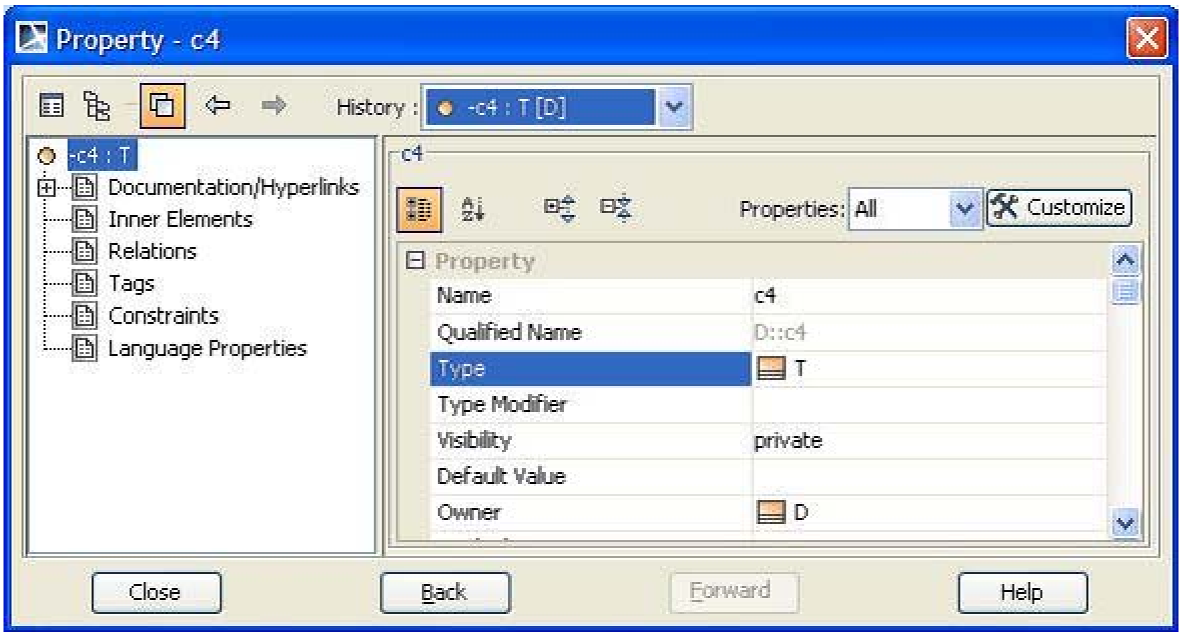
Create Clone Up Down

OK Cancel Help

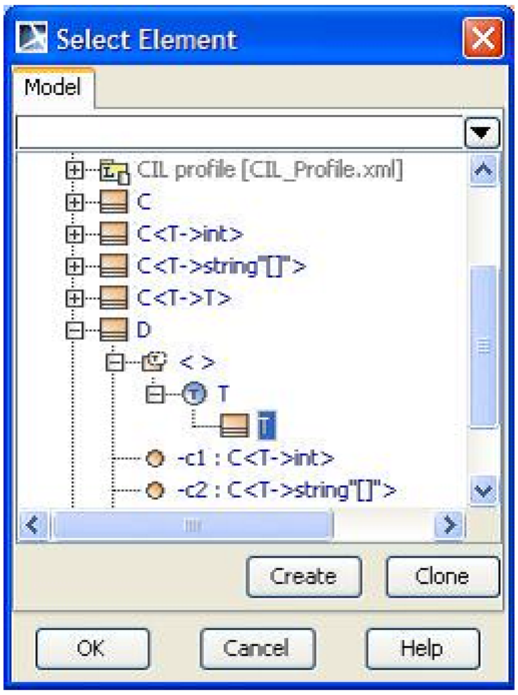
`..`
`private T c4;`
	The type of attribute, c4 is not a template binding class. Its type is a template parameter of the owner class D<T>

Copyright © 1998-2015 No Magic, Inc.

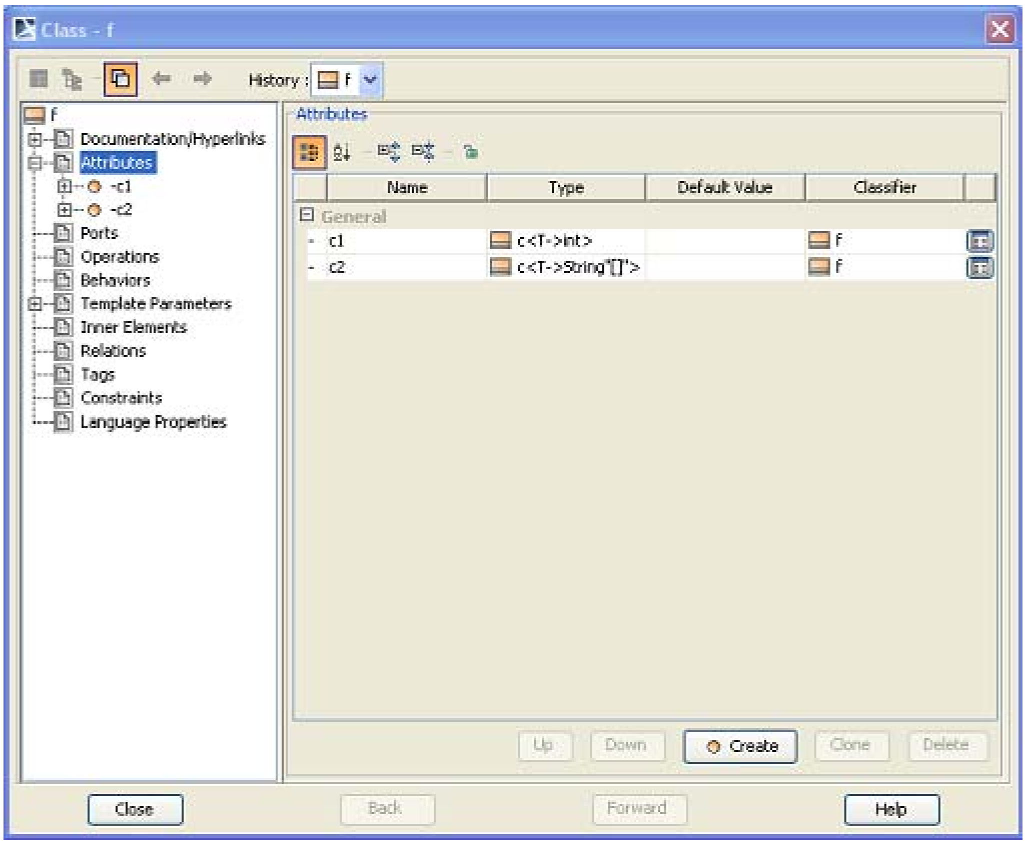
We create the attribute, c4 as usual, but select the correct T type. In this case, it is template parameter of the owner class, D<T>.



Property	
Name	c4
Qualified Name	D::c4
Type	T
Type Modifier	
Visibility	private
Default Value	
Owner	D

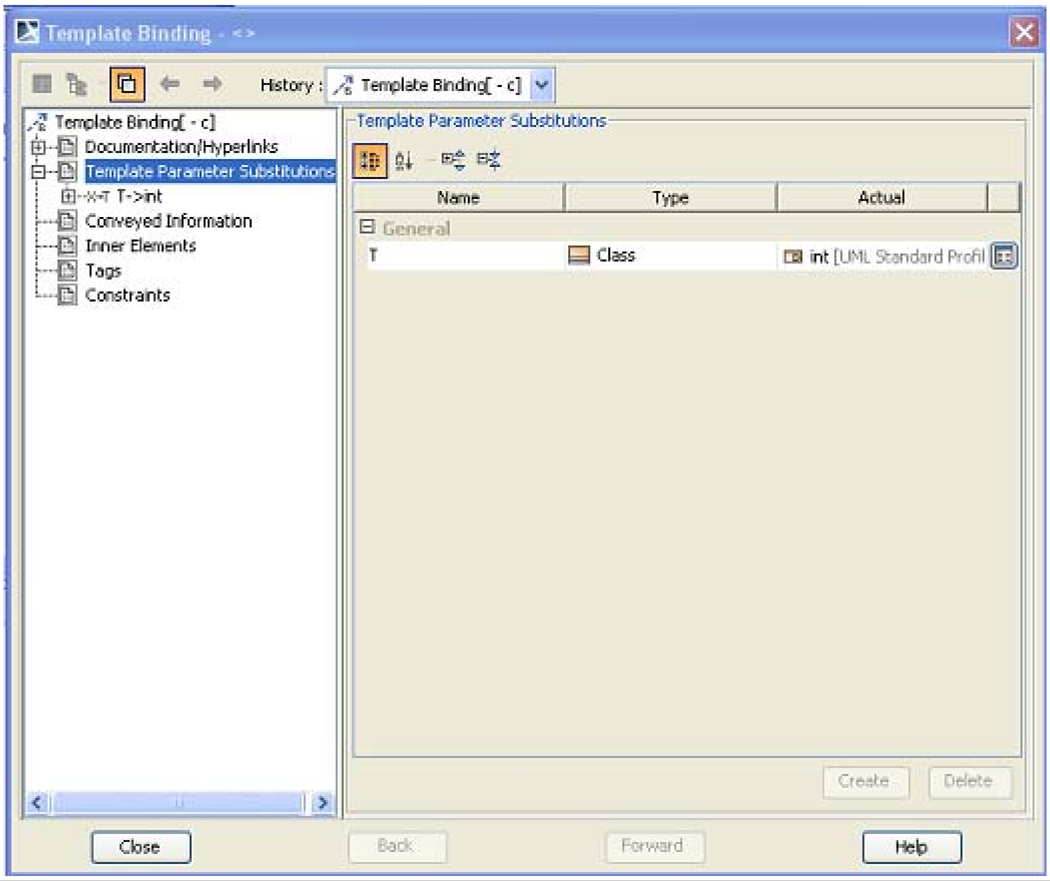


Copyright © 1998-2015 No Magic, Inc.



Name	Type	Default Value	Classifier
General			
c1	c<T->int>		f
c2	c<T->String'[]'>		f

The property of template binding is shown below

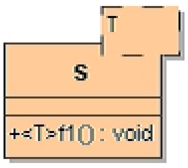


Name	Type	Actual
General		
T	Class	int [UML Standard Profil

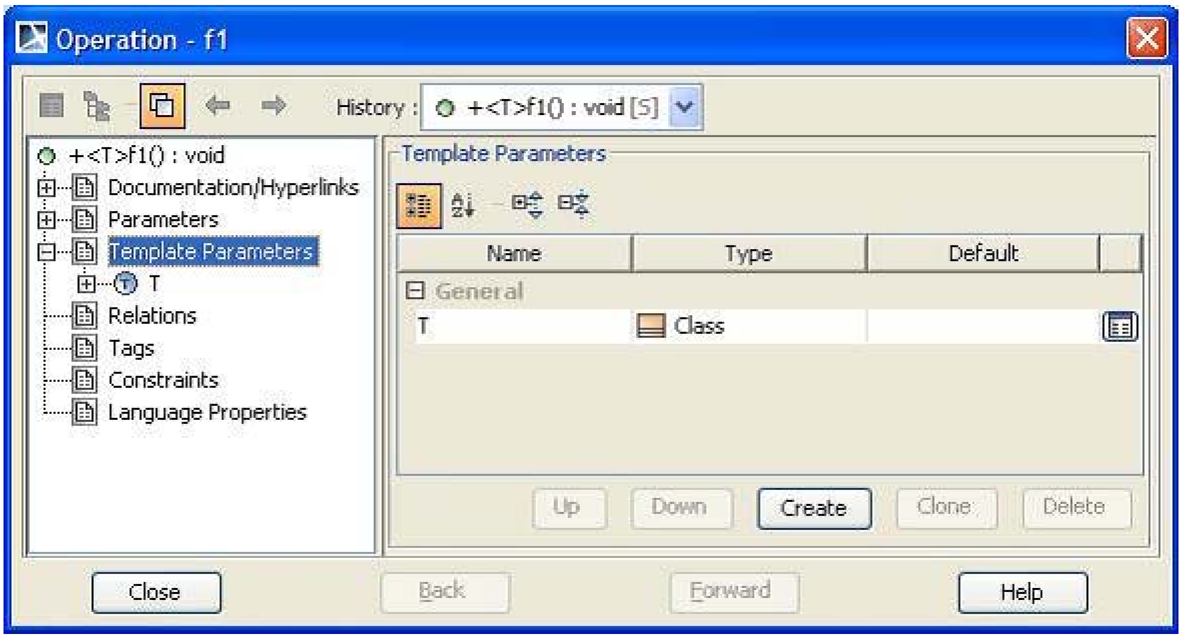
Generic Operation

Generic can be applied to operation such as operation name, return type and operation parameter.

Code	MD-UML

`public class S<T>`
`{`
`        public void f1<T>() {}`
`}`
	

To create generic operation, open the specification of operation, and create template parameter as usual.



Name	Type	Default
General		
T	Class	

For the return type and parameters of the operation, we create them like a generic attribute creation.

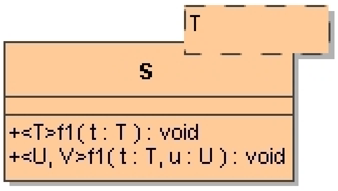
Important: We have to create or select the correct binding types.

In the example code, the type of method parameter t in the first method, public void f1<T> (T t), must be the template parameter of the owner method, f1<T>, not the template parameter of the owner class, S<T>.

For the second method, public void f1<U, V> (T t, U u), the type of parameter t must be the template parameter of the owner class, S<T>

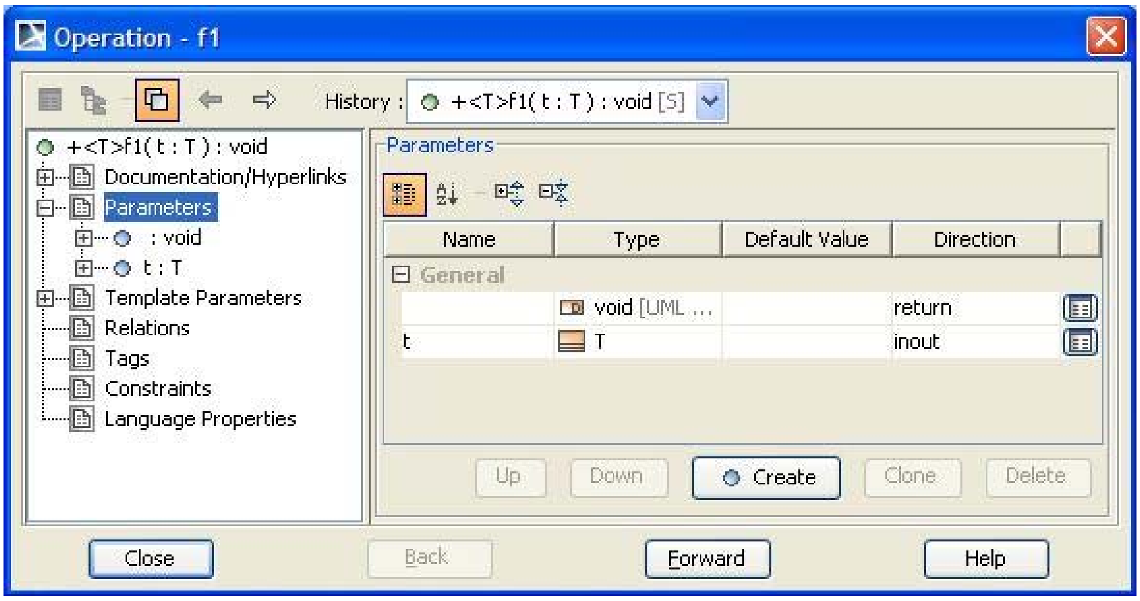
.

Code	MD-UML

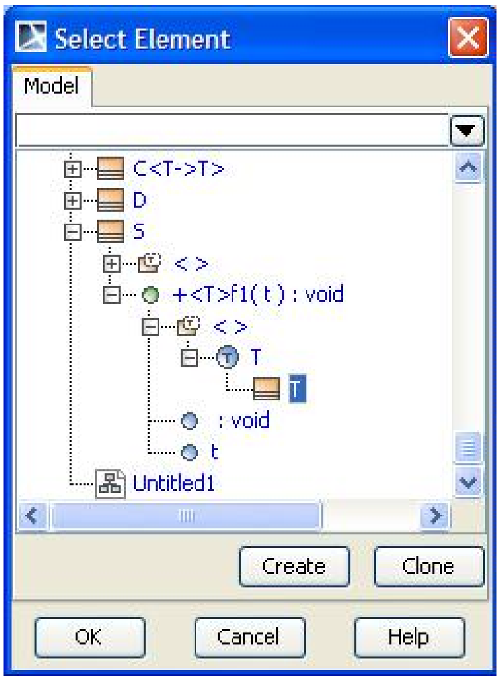
`public class S<T>`
`{`
`public void f1<T> (T t)`
`{`
`}`
`public void f1<U, V> (T t, U u)`
`{`
`}`
`}`
	 T
	S
	+<T>f1(t : T) : void
	+<U, V>f1(t : T, u : U) : void

Copyright © 1998-2015 No Magic, Inc.

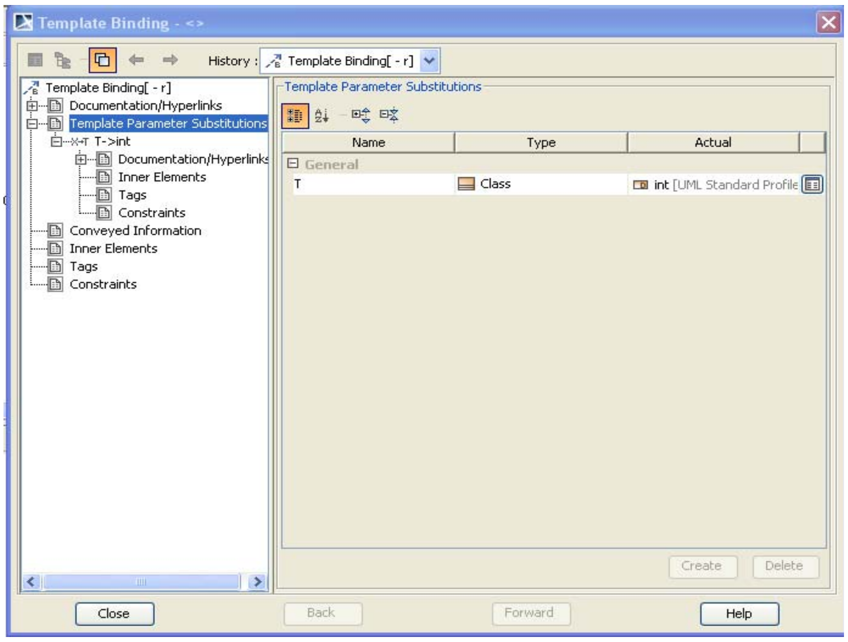
The following panels show the selection of method parameter t of the first method.



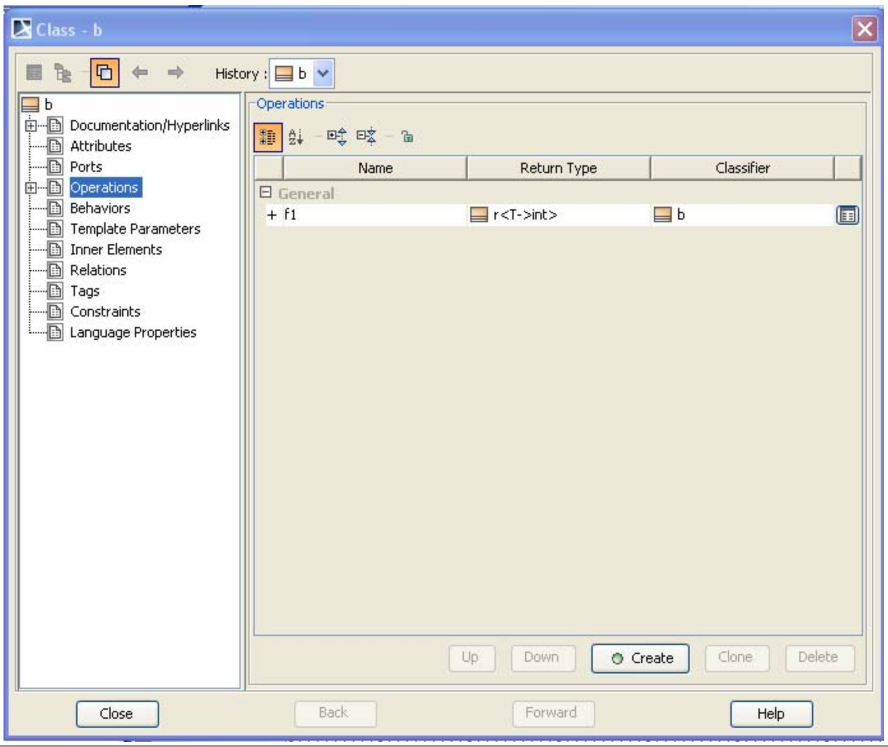
Name	Type	Default Value	Direction
General			
	void [UML ...		return
t	T		inout



Copyright © 1998-2015 No Magic, Inc.



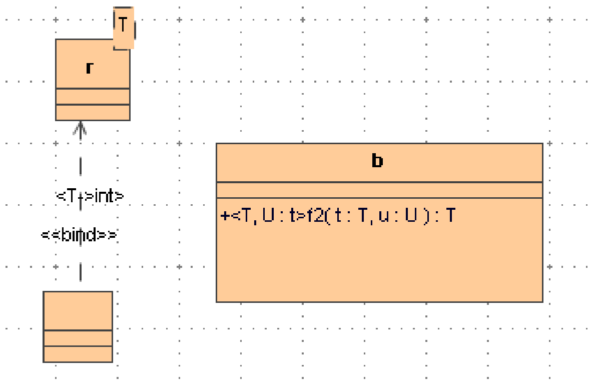
Name	Type	Actual
General		
T	Class	int [UML Standard Profile

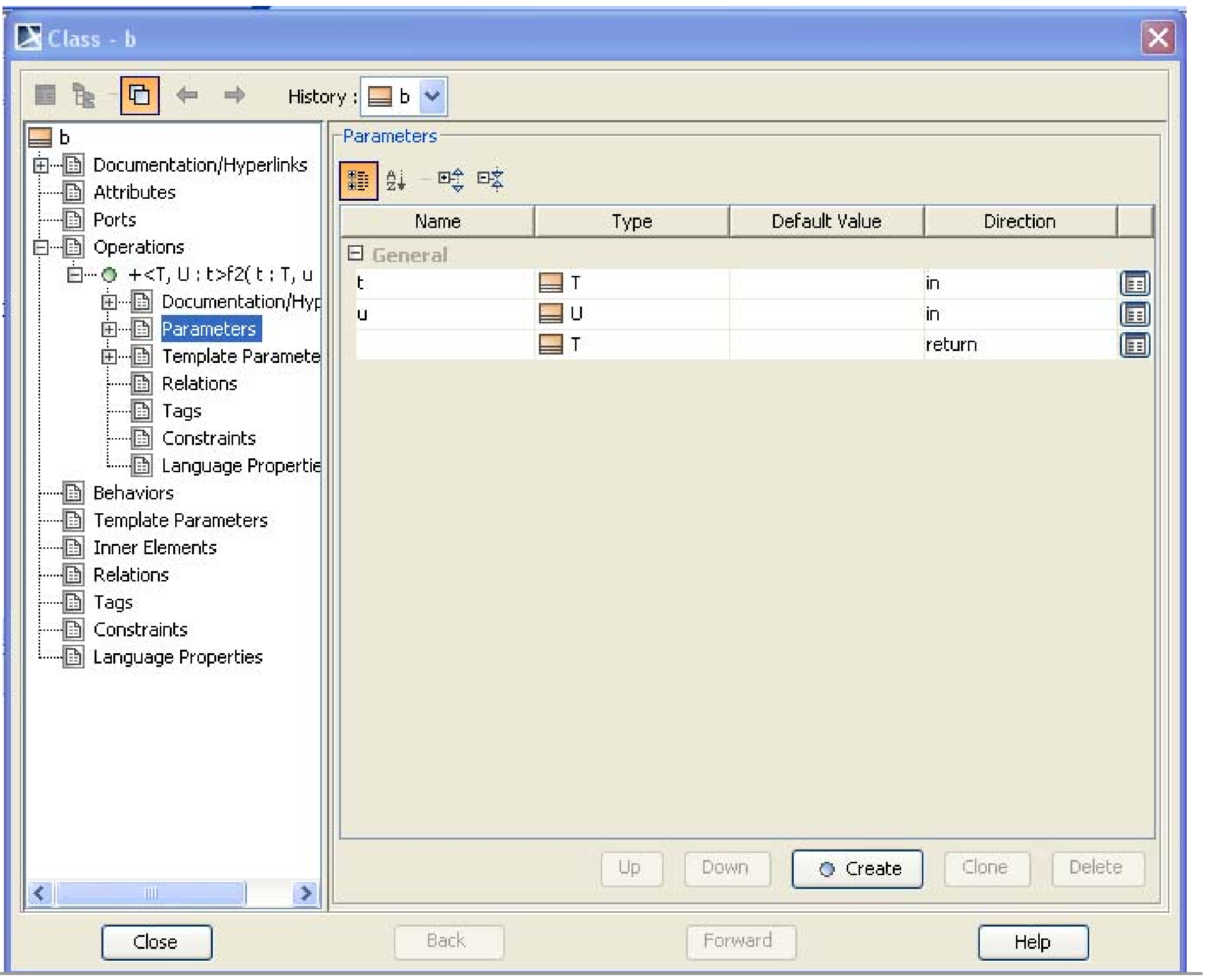


Name	Return Type	Classifier
General		
+ f1	r<T->int>	b

Copyright © 1998-2015 No Magic, Inc.

```
class b
{
        public T f2<T, U>(T t, U u)
        where U : T { return t; }
}
```





- b
  - Documentation/Hyperlinks
  - Attributes
  - Ports
  - Operations
    - +<T, U : t>f2(t : T, u
      - Documentation/Hyp
      - Parameters
      - Template Paramete
      - Relations
      - Tags
      - Constraints
      - Language Propertie
  - Behaviors
  - Template Parameters
  - Inner Elements
  - Relations
  - Tags
  - Constraints
  - Language Properties

Parameters

Name	Type	Default Value	Direction
General			
t	T		in
u	U		in
	T		return

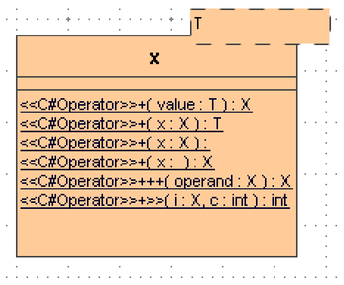
Copyright © 1998-2015 No Magic, Inc.

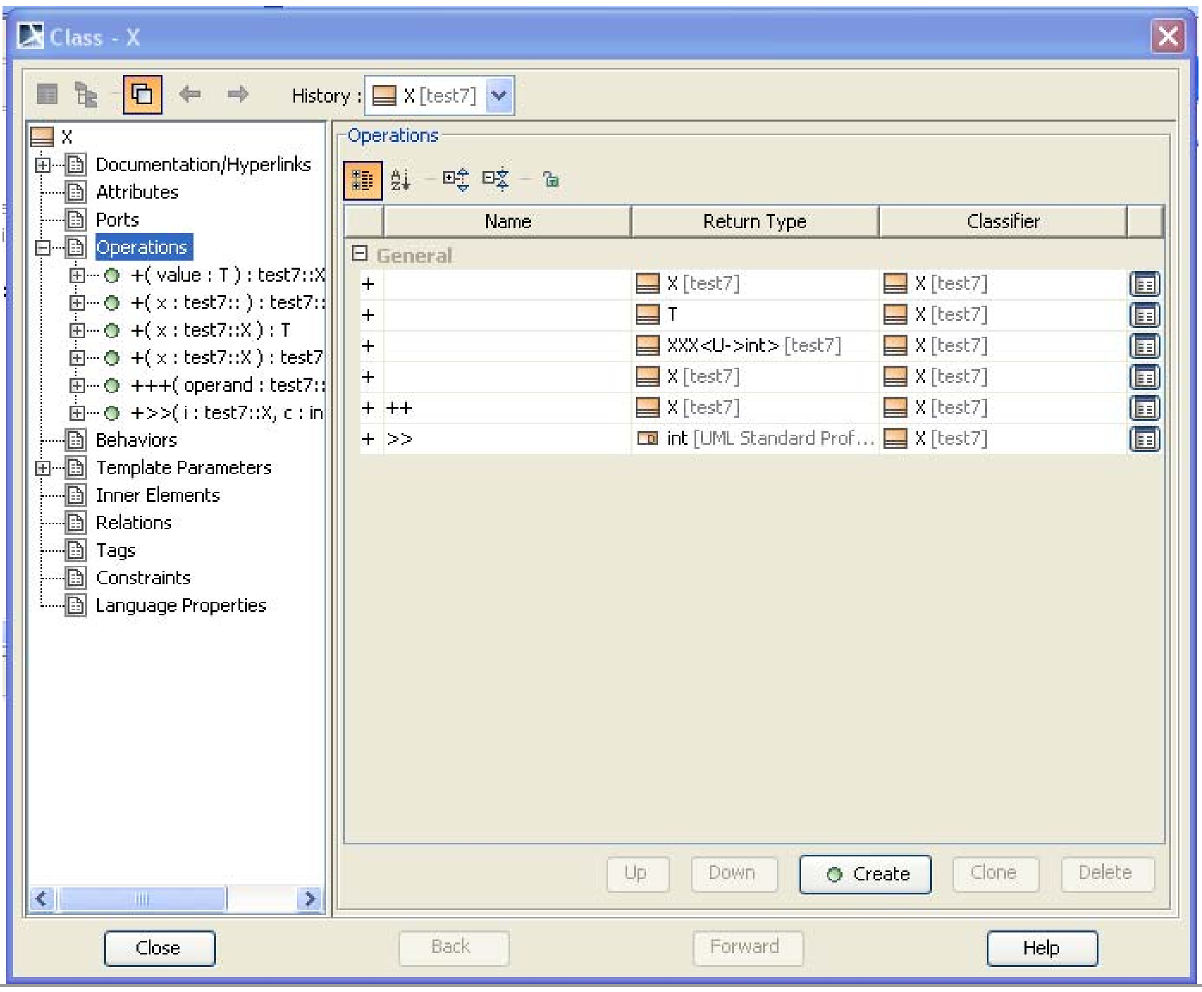
Generic Overloading

Code

```
class X<T>
{
        public static explicit operator X<T>(T
value)
        { return null; }
        public static implicit operator T(X<T> x)
        { return x.item; }
        public static explicit operator
XXX<int>(X<T> x)
        { return null; }
        public static explicit operator
X<T>(XXX<int> x)
        { return null; }
        public static X<T> operator ++(X<T> operand)

        { return null; }
        public static int operator >>(X<T> i, int c)

        { return c; }

}
```

MD-UML


T
X
<<C#Operator>>+(value : T) : X
<<C#Operator>>+(x : X) : T
<<C#Operator>>+(x : X) :
<<C#Operator>>+(x :) : X
<<C#Operator>>+++(operand : X) : X
<<C#Operator>>+>>(i : X, c : int) : int



X
- Documentation/Hyperlinks
- Attributes
- Ports
- Operations
  - +(value : T) : test7::X
  - +(x : test7::) : test7::
  - +(x : test7::X) : T
  - +(x : test7::X) : test7
  - +++(operand : test7::
  - +>>(i : test7::X, c : in
- Behaviors
- Template Parameters
- Inner Elements
- Relations
- Tags
- Constraints
- Language Properties

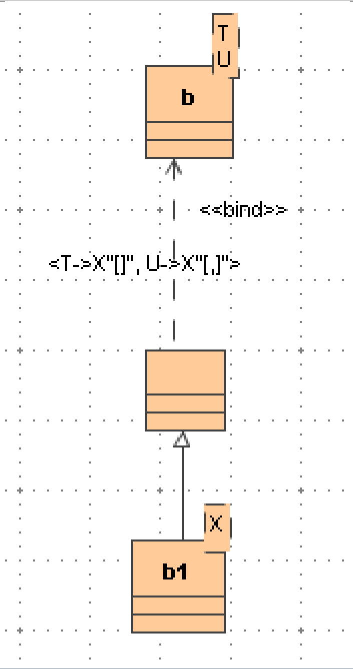
Operations

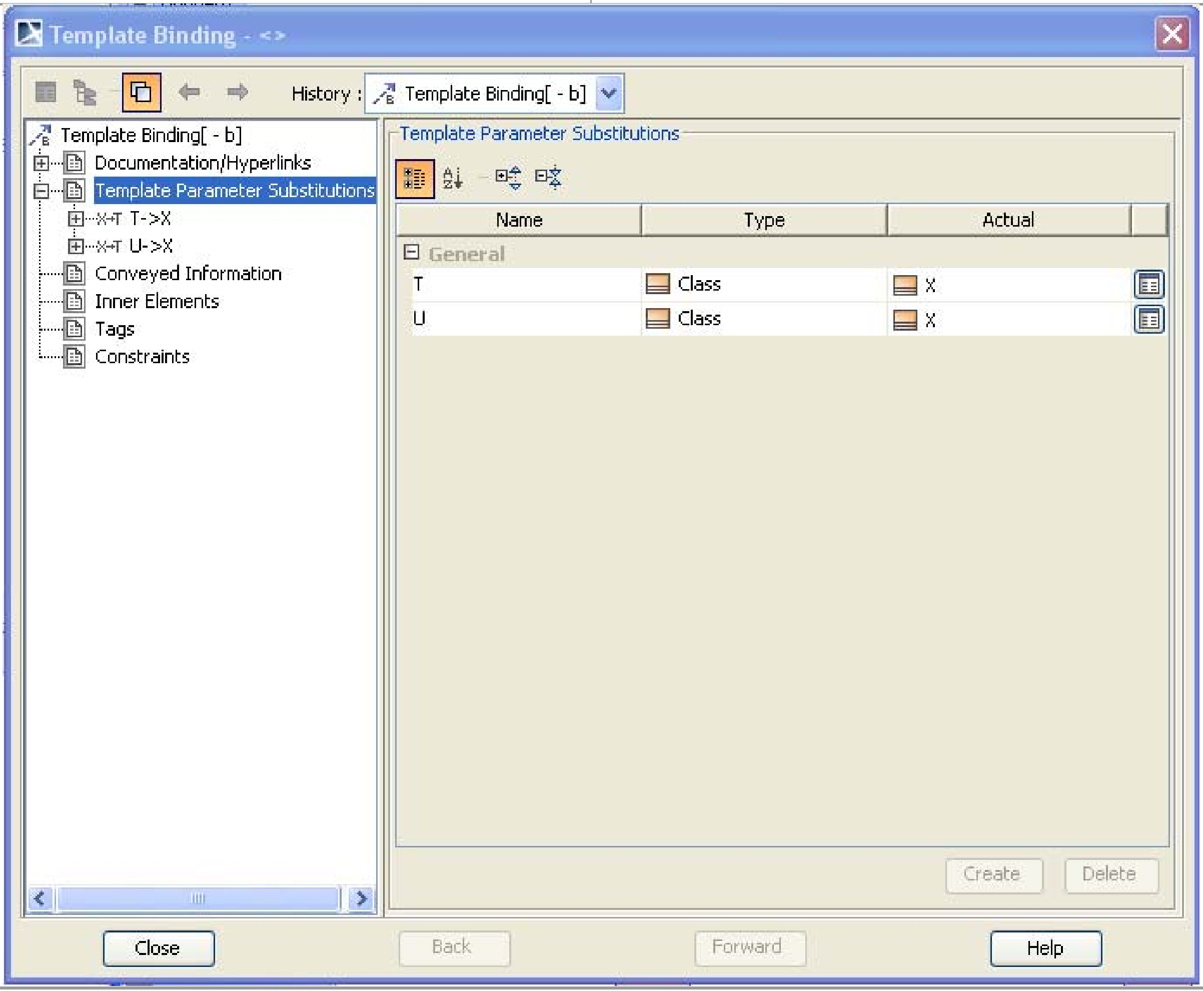
	Name	Return Type	Classifier
General			
+		X [test7]	X [test7]
+		T	X [test7]
+		XXX<U->int> [test7]	X [test7]
+		X [test7]	X [test7]
+	++	X [test7]	X [test7]
+	>>	int [UML Standard Prof...	X [test7]

Copyright © 1998-2015 No Magic, Inc.

Generic Parent Class

Code	MD-UML

`class b<T, U>`
`{}`
`class b1<X> : b<X[], X[,]>`
`{}`
	



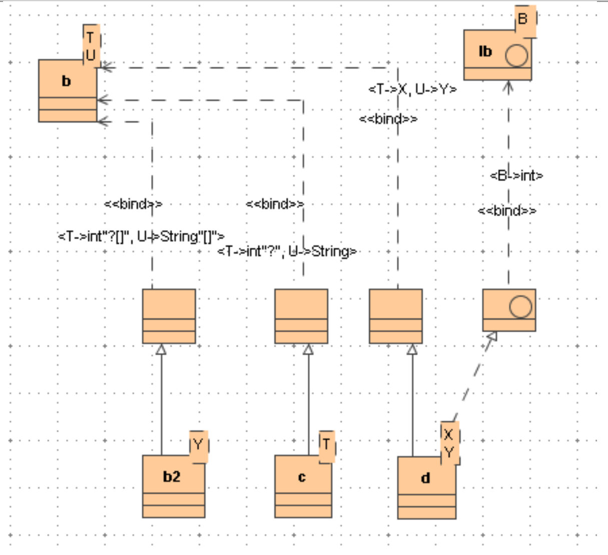
Template Binding[- b]
- Documentation/Hyperlinks
- Template Parameter Substitutions
  - T->X
  - U->X
- Conveyed Information
- Inner Elements
- Tags
- Constraints

Template Parameter Substitutions

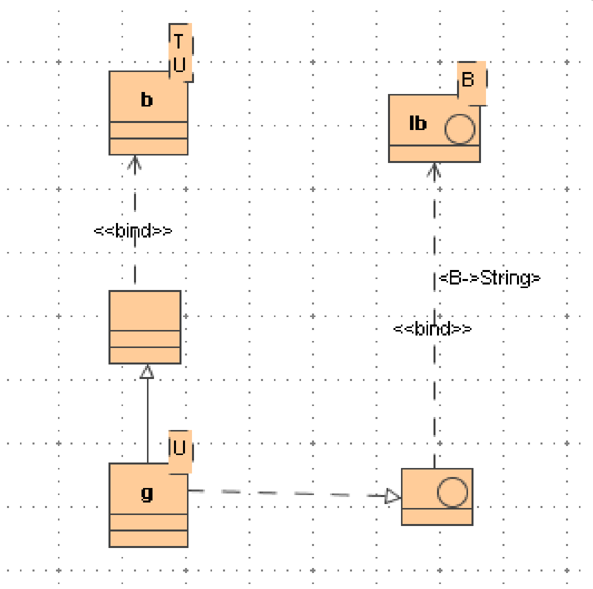
Name	Type	Actual
General		
T	Class	X
U	Class	X

Copyright © 1998-2015 No Magic, Inc.

Code	MD-UML

`class b<T, U>`
`{}`
`interface Ib<B>`
`{}`
`class b2<Y> : b<int?[], string[]>`
`{}`
`class c<T> : b<int?, string>`
`{}`
`class d<X, Y> : b<X, Y>, Ib<int>`
`{}`
	

Code	MD-UML

`class b<T, U>`
`{}`
`interface Ib<B>`
`{}`
`class g<U> : b<b<int, object>, U>,`
`Ib<string>`
`{}`
	

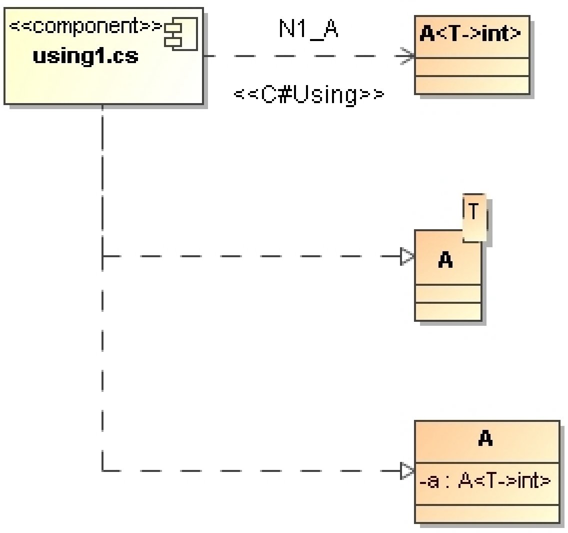
Generic Using Alias

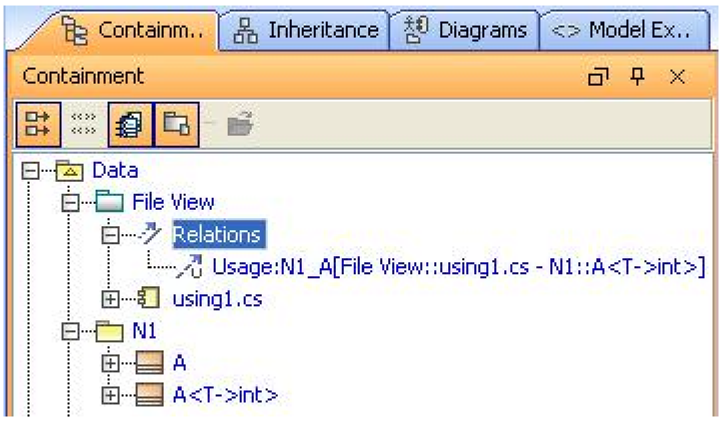
For example, using N1_A = N1.A<int>, we create template binding for A<int> in namespace N1, and then we create the Usage dependency from the parent component (in this case it is file component) to the template binding class.

For more information about the mapping of Using Directive, Using Directive Mapping.

Copyright © 1998-2015 No Magic, Inc.

Code	MD-UML

`using N1_A = N1.A<int>;`
`namespace N1`
`{`
`public class A<T> {}`
`}`
`class A`
`{`
`N1_A a;`
`}`
	 <<component>>
	using1.cs
	N1_A
	<<C#Using>>
	A<T->int>
	T
	A
	A
	-a : A<T->int>



Data
File View
Relations
Usage:N1_A[File View::using1.cs - N1::A<T->int>]
using1.cs
N1
A
A<T->int>

Generic Constraints

Generic type and method declarations can optionally specify type parameter constraints by including *type parameter-constraints-clauses.*

type-parameter-constraints-clauses:
    type-parameter-constraints-clause
    type-parameter-constraints-clauses type-parameter-constraints-clause

type-parameter-constraints-clause:
    `where` *type-parameter* `:` *type-parameter-constraints*

type-parameter-constraints:
    primary-constraint
    secondary-constraints
    constructor-constraint
    primary-constraint , *secondary-constraints*
    primary-constraint , *constructor-constraint*
    secondary-constraints , *constructor-constraint*
    primary-constraint , *secondary-constraints* , *constructor-constraint*

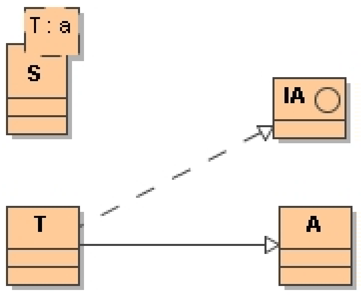
Copyright © 1998-2015 No Magic, Inc.

primary-constraint:
  class-type
  `class`
  `struct`

secondary-constraints:
  interface-type
  type-parameter
  secondary-constraints , *interface-type*
  secondary-constraints , *type-parameter*

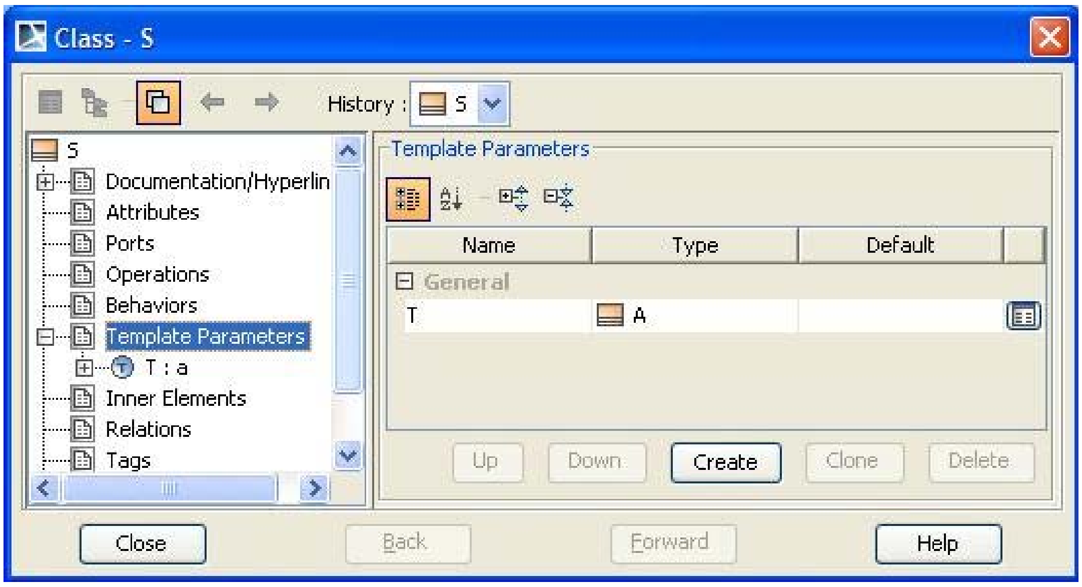
constructor-constraint:
  `new ( )`

Code	MD-UML

`public class A { }`
`interface IA {}`
`public class S<T> where T: A, IA{}`
	

Copyright © 1998-2015 No Magic, Inc.

To create the first template constraint for template parameter T, we select the type of template parameter T as class A.



Class - S

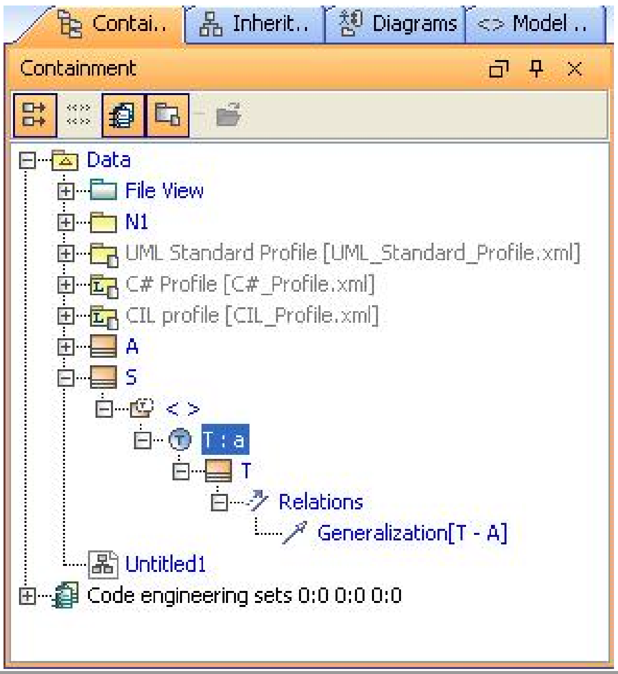
History : S

- S
  - Documentation/Hyperlin
  - Attributes
  - Ports
  - Operations
  - Behaviors
  - Template Parameters
    - T : a
  - Inner Elements
  - Relations
  - Tags

Template Parameters

Name	Type	Default
General		
T	A	

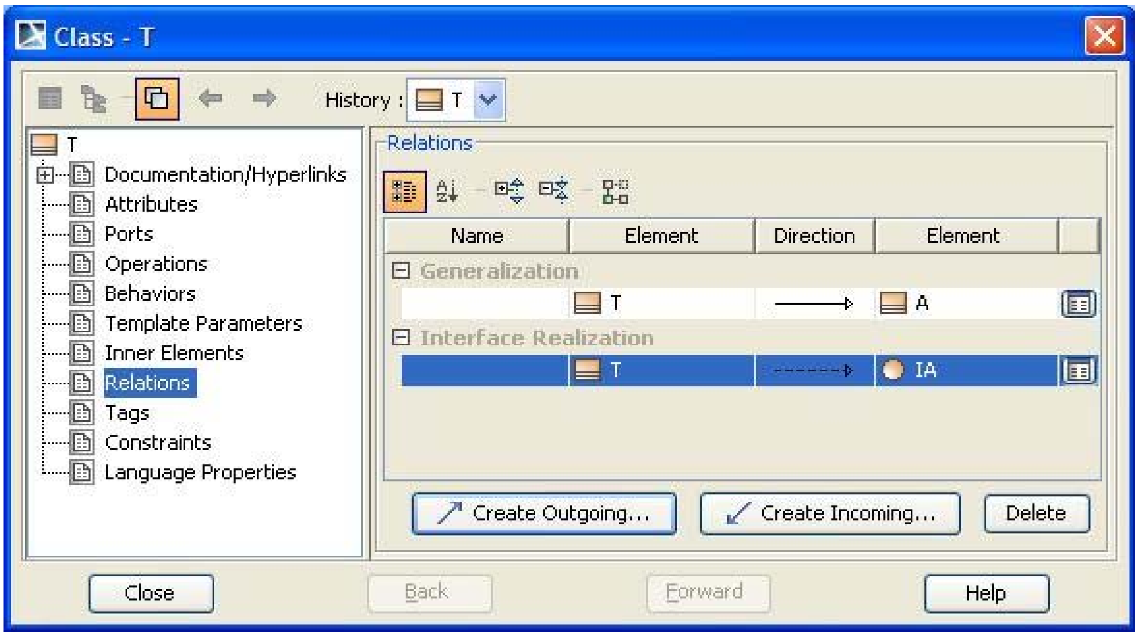
The constraint A is shown as generalization relation of T.



- Data
  - File View
  - N1
  - UML Standard Profile [UML_Standard_Profile.xml]
  - C# Profile [C#_Profile.xml]
  - CIL profile [CIL_Profile.xml]
  - A
  - S
    - < >
      - T : a
        - T
          - Relations
            - Generalization[T - A]
  - Untitled1
- Code engineering sets 0:0 0:0 0:0

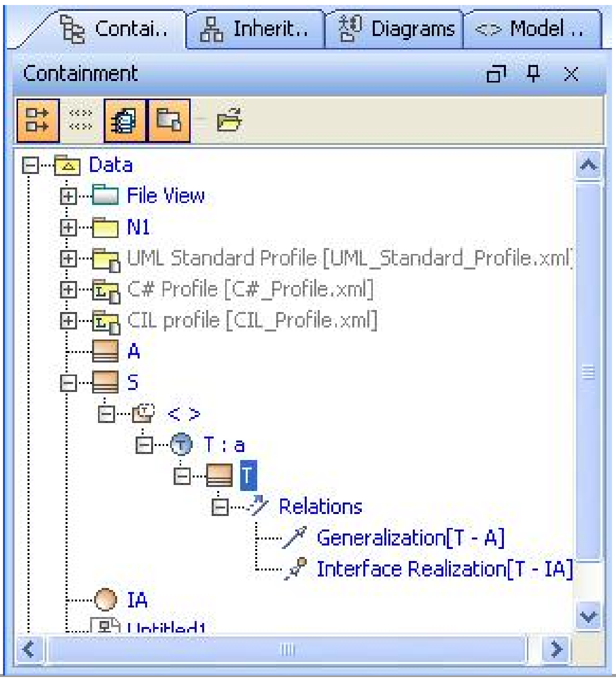
Copyright © 1998-2015 No Magic, Inc.

To create the second template constraint for template parameter T, Open the specification of template parameter T, and then create Interface Realization as Outgoing relation to the interface IA.



Relations

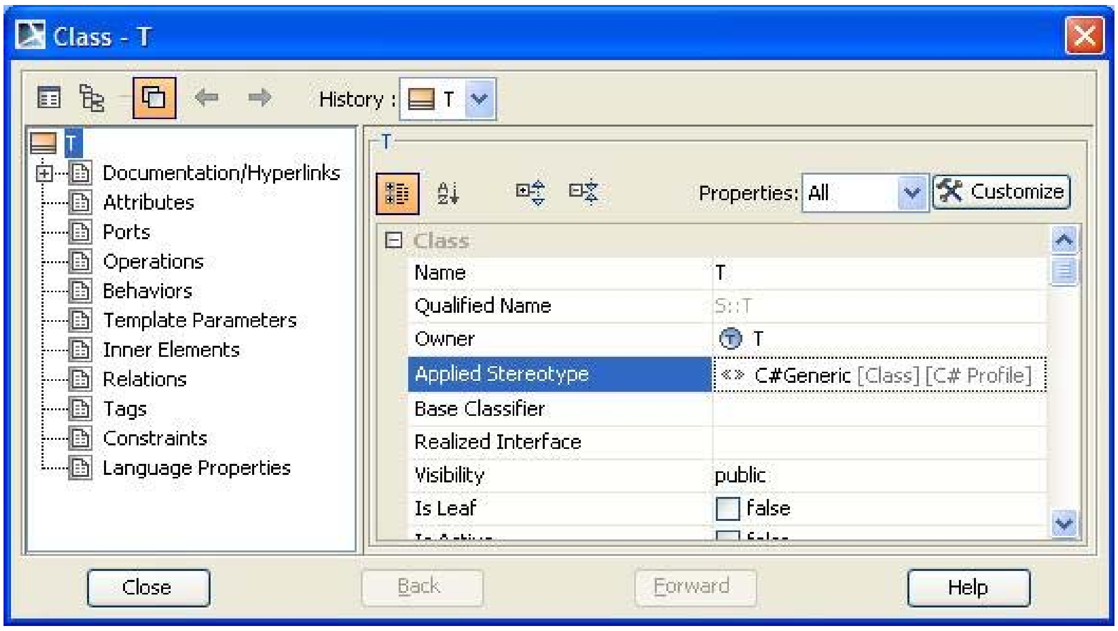
Name	Element	Direction	Element
Generalization			
	T	——▷	A
Interface Realization			
	T	- - - - ▷	IA



```
Data
  File View
  N1
  UML Standard Profile [UML_Standard_Profile.xml]
  C# Profile [C#_Profile.xml]
  CIL profile [CIL_Profile.xml]
  A
  S
    < >
      T : a
        T
          Relations
            Generalization[T - A]
            Interface Realization[T - IA]
  IA
  Untitled1
```

Copyright © 1998-2015 No Magic, Inc.

To create class, struct, and new() constraints, we apply «C#Generic» stereotype to the template parameter, and then we create the tag values.


Class - T

History : T

- T
  - Documentation/Hyperlinks
  - Attributes
  - Ports
  - Operations
  - Behaviors
  - Template Parameters
  - Inner Elements
  - Relations
  - Tags
  - Constraints
  - Language Properties

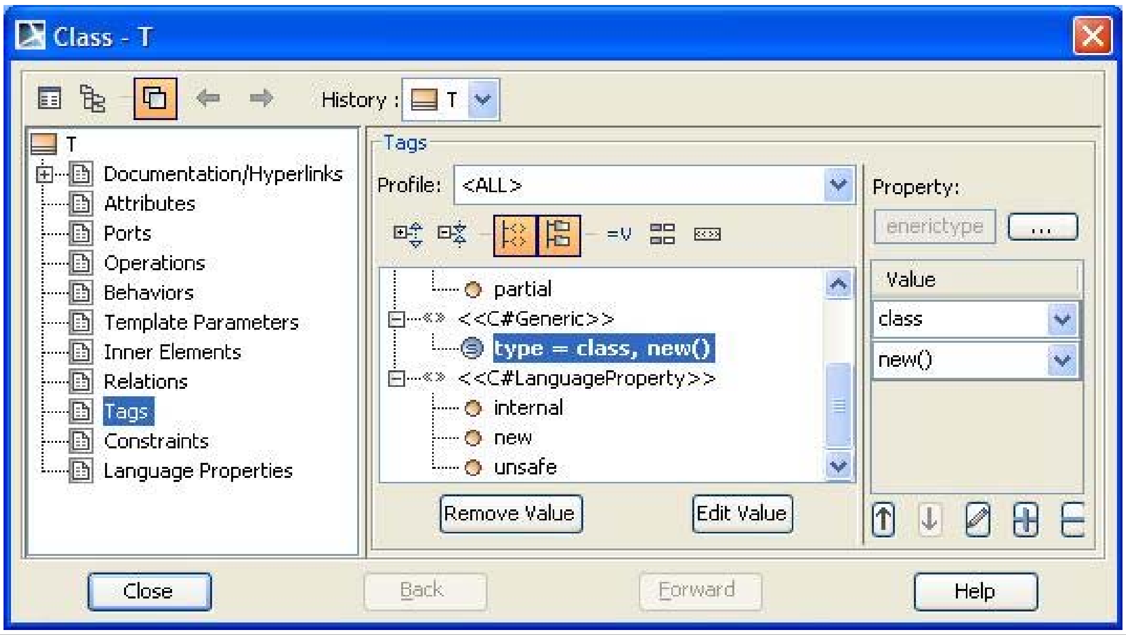
T

Properties: All | Customize

Class

Name	T
Qualified Name	S::T
Owner	T
Applied Stereotype	«» C#Generic [Class] [C# Profile]
Base Classifier	
Realized Interface	
Visibility	public
Is Leaf	false

Close | Back | Forward | Help


Class - T

History : T

- T
  - Documentation/Hyperlinks
  - Attributes
  - Ports
  - Operations
  - Behaviors
  - Template Parameters
  - Inner Elements
  - Relations
  - Tags
  - Constraints
  - Language Properties

Tags

Profile: <ALL>

```
    partial
«» <<C#Generic>>
    type = class, new()
«» <<C#LanguageProperty>>
    internal
    new
    unsafe
```

Remove Value | Edit Value

Property: enerictype

Value
class
new()

Close | Back | Forward | Help

Anonymous Methods

In versions of C# previous to 2.0, the only way to declare a delegate was to use named methods. C# 2.0 introduces anonymous methods.

Creating anonymous methods is essentially a way to pass a code block as a delegate parameter.

For example:

```
// Create a delegate instance
delegate void Del(int x);

// Instantiate the delegate using an anonymous method
```

Copyright © 1998-2015 No Magic, Inc.

```
Del d = delegate(int k) { /* ... */ };
```

By using anonymous methods, you reduce the coding overhead in instantiating delegates by eliminating the need to create a separate method.

From now, there is no direct mapping for Anonymous Methods but it uses the mapping for delegate method feature.

Partial Types

A new type modifier, `partial`, is used when defining a type in multiple parts. To ensure compatibility with existing programs, this modifier is different than other modifiers: like `get` and `set`, it is not a keyword, and it must appear immediately before one of the keywords `class`, `struct`, or `interface`.

class-declaration:
attributes$_{opt}$ *class-modifiers*$_{opt}$ `partial`$_{opt}$ `class` *identifier* *type-parameter-list*$_{opt}$
class-base$_{opt}$ *type-parameter-constraints-clauses*$_{opt}$ *class-body* `;`$_{opt}$

struct-declaration:
attributes$_{opt}$ *struct-modifiers*$_{opt}$ `partial`$_{opt}$ `struct` *identifier* *type-parameter-list*$_{opt}$
struct-interfaces$_{opt}$ *type-parameter-constraints-clauses*$_{opt}$ *struct-body* `;`$_{opt}$

interface-declaration:
attributes$_{opt}$ *interface-modifiers*$_{opt}$ `partial`$_{opt}$ `interface` *identifier* *type-parameter-list*$_{opt}$
interface-base$_{opt}$ *type-parameter-constraints-clauses*$_{opt}$ *interface-body* `;`$_{opt}$

Each part of a partial type declaration must include a `partial` modifier and must be declared in the same namespace as the other parts. The `partial` modifier indicates that additional parts of the type declaration may exist elsewhere, but the existence of such additional parts is not a requirement. It is valid for just a single declaration of a type to include the `partial` modifier.

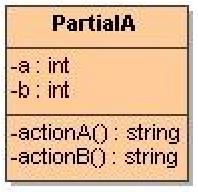
All parts of a partial type must be compiled together such that the parts can be merged at compile-time. Partial types specifically do not allow already compiled types to be extended.

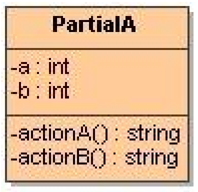
Nested types may be declared in multiple parts by using the `partial` modifier. Typically, the containing type is declared using `partial` as well, and each part of the nested type is declared in a different part of the containing type.

The `partial` modifier is not permitted on delegate or enum declarations.

Copyright © 1998-2015 No Magic, Inc.

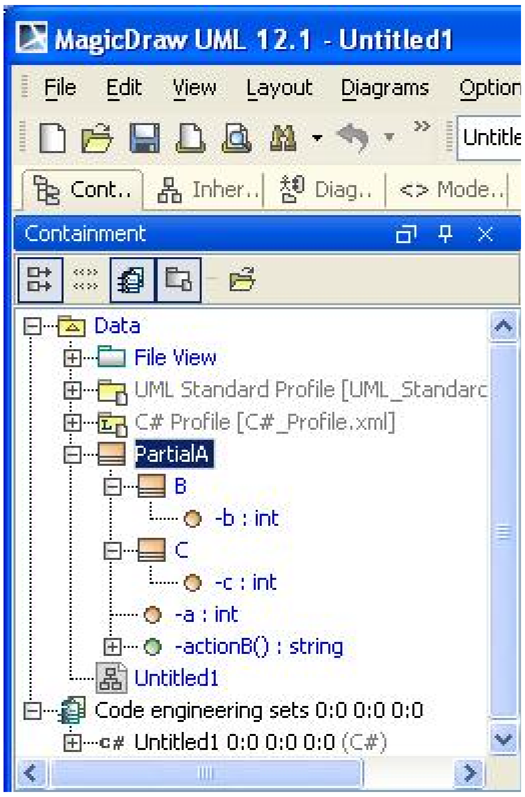
Code	MD-UML

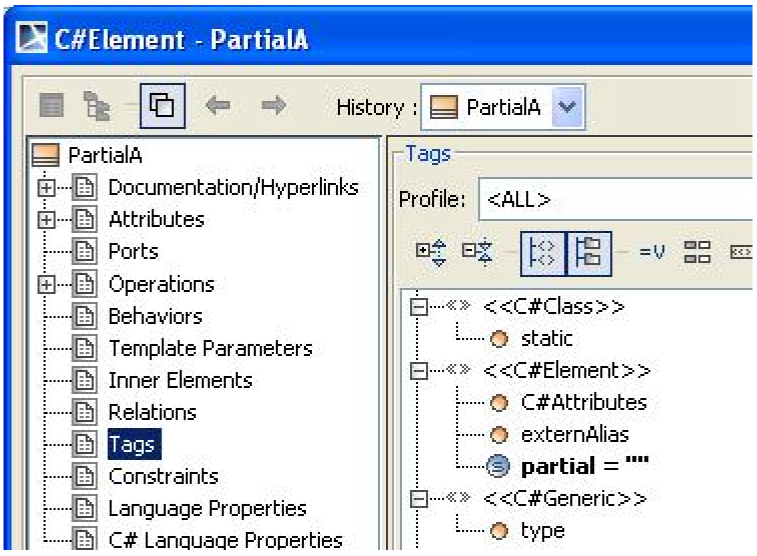
`//Case #1`
`//The partial class is written into one class file.`
`public partial class PartialA{`
`        int a;`
`        string actionA()`
`        {`
`        }`
`}`
`public partial class PartialA{`
`        int b;`
`        string actionB()`
`        {`
`        }`
`}`
	 PartialA
	-a : int
	-b : int
	-actionA() : string
	-actionB() : string
	All child elements will be merged into one Class element.

`//Case #2`
`//The partial class is written into separate class file.`
`//PartialA1.cs`
`public partial class PartialA{`
`        int a;`
`        string actionA()`
`{`
`        }`
`}`
 	
`//PartialA2.cs`
`public partial class PartialA{`
`        int b;`
`        string actionB()`
`        {`
`        }`
`}`
	 PartialA
	-a : int
	-b : int
	-actionA() : string
	-actionB() : string
	All child elements will be merged into one Class element, the same as both classes are written into one class file.

Copyright © 1998-2015 No Magic, Inc.

Code	MD-UML

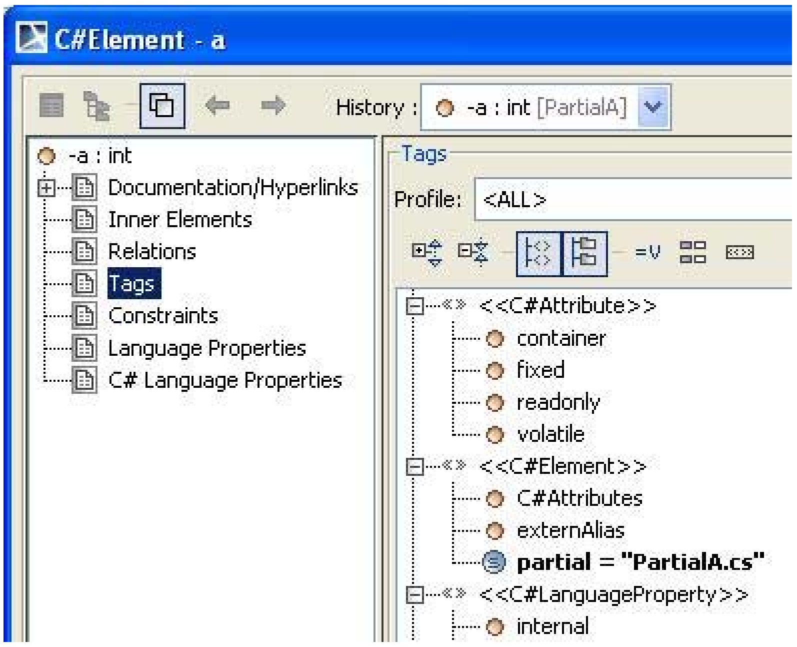
`//Case #3`
`//The partial class with inner class`
`public partial class PartialA{`
`        public class B`
`        {`
`                int b;`
`        }`
`}`
`public partial class PartialA{`
`        int a;`
`        string actionB()`
`        {`
`        }`
`        public class C`
`        {`
`                int c;`
`        }`
`}`
	 Data / File View / UML Standard Profile [UML_Standard / C# Profile [C#_Profile.xml] / PartialA / B / -b : int / C / -c : int / -a : int / -actionB() : string / Untitled1 / Code engineering sets 0:0 0:0 0:0 / c# Untitled1 0:0 0:0 0:0 (C#)
	
	All inner classes have to be located as inner class of the parent class (partial class).



```
<<C#Class>>
    static
<<C#Element>>
    C#Attributes
    externAlias
    partial = ""
<<C#Generic>>
    type
```

The "partial" tag will have a blank tagged value for Partial Class Element.

Copyright © 1998-2015 No Magic, Inc.

Code	MD-UML
 C#Element - a History : -a : int [PartialA] -a : int Documentation/Hyperlinks Inner Elements Relations Tags Constraints Language Properties C# Language Properties Tags Profile: <ALL> <<C#Attribute>> container fixed readonly volatile <<C#Element>> C#Attributes externAlias partial = "PartialA.cs" <<C#LanguageProperty>> internal The "partial" tag will have a value of the file name that the class belongs to for each child element in partial class element.	

Nullable Types

A nullable type is classified as a value type:

value-type:
    struct-type
    enum-type

struct-type:
    type-name
    simple-type
    nullable-type

nullable-type:
    non-nullable-value-type ?

non-nullable-value-type:
    type

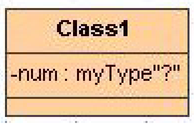
The type specified before the ? modifier in a nullable type is called the ***underlying type*** of the nullable type. The underlying type of a nullable type can be any non-nullable value type or any type parameter that is constrained to non-nullable value types (that is, any type parameter with a `struct` constraint). The underlying type of a nullable type cannot be a nullable type or a reference type.

A nullable type can represent all values of its underlying type plus an additional null value.

Copyright © 1998-2015 No Magic, Inc.

The syntax `T?` is shorthand for `System.Nullable<T>`, and the two forms can be used interchangeably.

Code	MD-UML

```
class class1
{
        int? a = null;
        System.Nullable a = null;
}
```
	

Add “Nullable” class type to C# profile.

Accessor Declarations

The syntax for property accessors and indexer accessors is modified to permit an optional *accessor-modifier*:

get-accessor-declaration:
  attributes$_{opt}$ *accessor-modifier*$_{opt}$ `get` *accessor-body*

set-accessor-declaration:
  attributes$_{opt}$ *accessor-modifier*$_{opt}$ `set` *accessor-body*

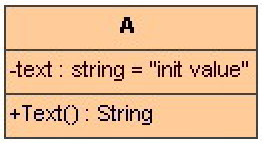
accessor-modifier:
```
protected
internal
private
protected internal
internal protected
```

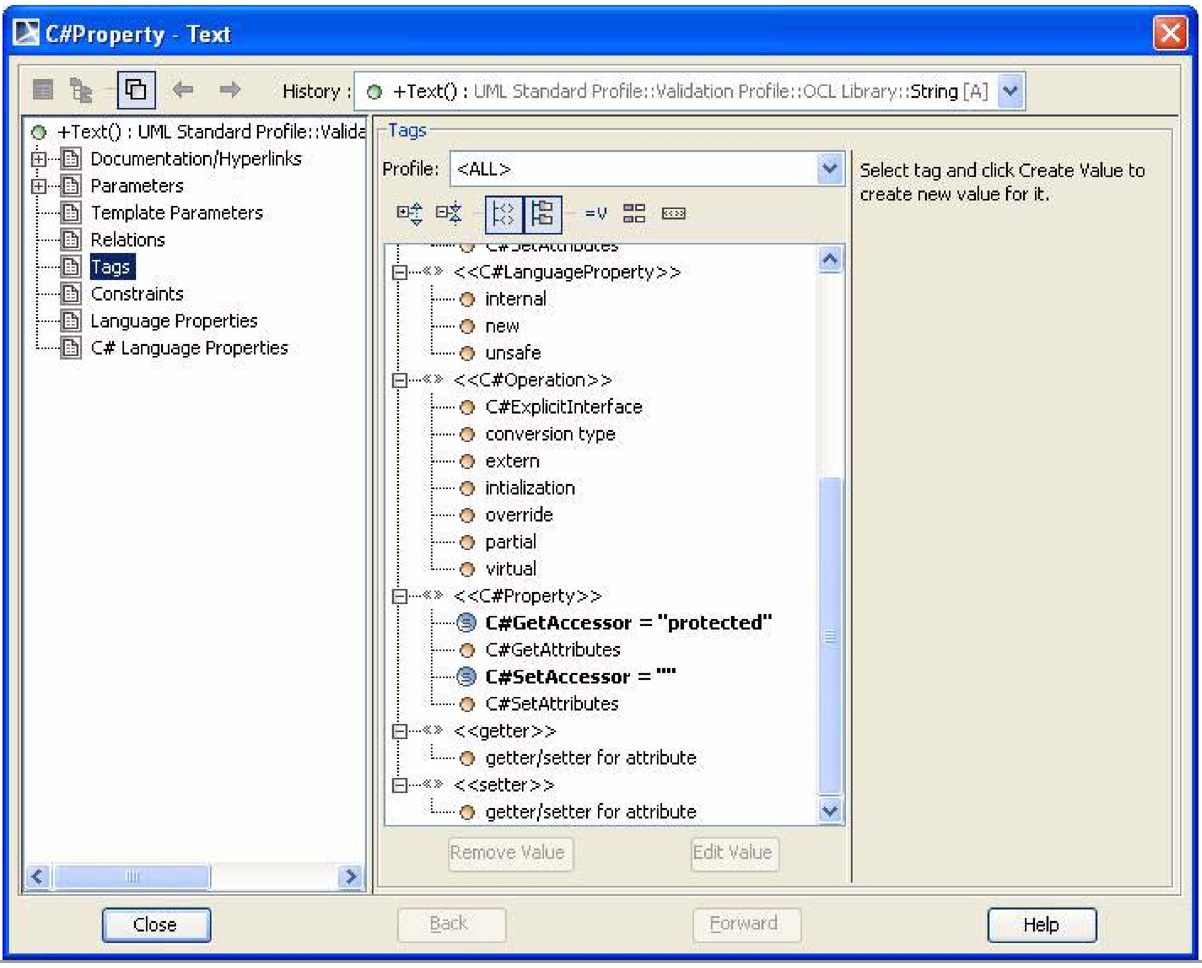
The use of *accessor-modifiers* is governed by the following restrictions:

- An *accessor-modifier* may not be used in an interface or in an explicit interface member implementation.
- For a property or indexer that has no `override` modifier, an *accessor-modifier* is permitted only if the property or indexer has both a `get` and `set` accessor, and then is permitted only on one of those accessors.
- For a property or indexer that includes an `override` modifier, an accessor must match the *accessor-modifier*, if any, of the accessor being overridden.
- The *accessor-modifier* must declare an accessibility that is strictly more restrictive than the declared accessibility of the property or indexer itself. To be precise:
- If the property or indexer has a declared accessibility of `public`, any *accessor-modifier* may be used.
- If the property or indexer has a declared accessibility of `protected internal`, the *accessor-modifier* may be either `internal`, `protected`, or `private`.
- If the property or indexer has a declared accessibility of `internal` or `protected`, the *accessor-modifier* must be `private`.
- If the property or indexer has a declared accessibility of `private`, no *accessor-modifier* may be used.

Copyright © 1998-2015 No Magic, Inc.

Code	MD-UML

`Class A`
`{`
`private string text = “init value”;`
`public String Text`
`{`
`        protected get{ return text;}`
`        set{ text = value;}`
`}`
`}`
	



C#Property - Text

History : +Text() : UML Standard Profile::Validation Profile::OCL Library::String [A]

- +Text() : UML Standard Profile::Valida
  - Documentation/Hyperlinks
  - Parameters
  - Template Parameters
  - Relations
  - Tags
  - Constraints
  - Language Properties
  - C# Language Properties

Tags

Profile: <ALL>

Select tag and click Create Value to create new value for it.

- C#SetAttributes
- <<C#LanguageProperty>>
  - internal
  - new
  - unsafe
- <<C#Operation>>
  - C#ExplicitInterface
  - conversion type
  - extern
  - intialization
  - override
  - partial
  - virtual
- <<C#Property>>
  - **C#GetAccessor = "protected"**
  - C#GetAttributes
  - **C#SetAccessor = ""**
  - C#SetAttributes
- <<getter>>
  - getter/setter for attribute
- <<setter>>
  - getter/setter for attribute

Remove Value | Edit Value

Close | Back | Forward | Help

Static Class

Static classes are classes that are not intended to be instantiated and which contain only static members. When a class declaration includes a static modifier, the class being declared is said to be a static class.

When a class declaration includes a static modifier, the class being declared is said to be a static class.

class-declaration:

attributes$_{opt}$ *class-modifiers*$_{opt}$ `partial`$_{opt}$ `class` *identifier* *type-parameter-list*$_{opt}$

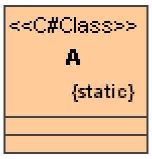
class-base$_{opt}$ *type-parameter-constraints-clauses*$_{opt}$ *class-body* `;`$_{opt}$

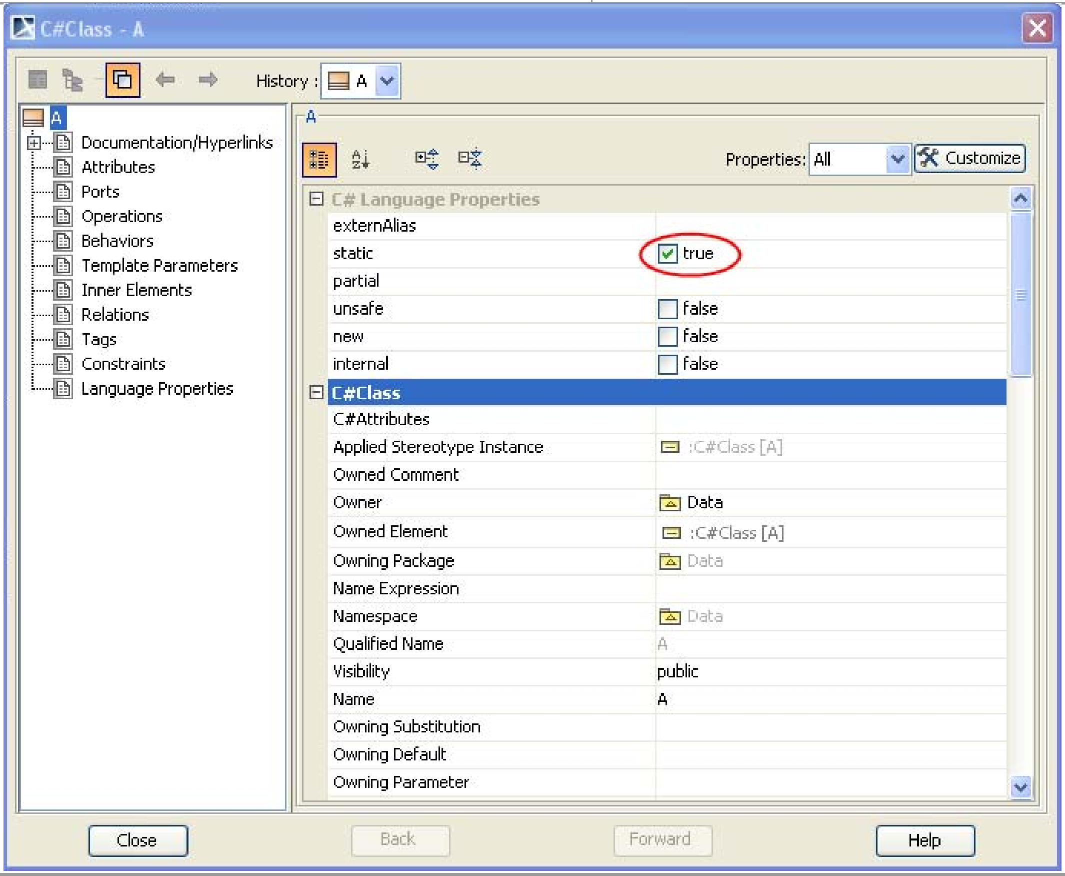
Copyright © 1998-2015 No Magic, Inc.

class-modifiers:
  class-modifier
  class-modifiers class-modifier

class-modifier:
```
new
public
protected
internal
private
abstract
sealed
static
```

Code	MD-UML

`static class A`
`{`
`…`
`}`
	



A
- Documentation/Hyperlinks
- Attributes
- Ports
- Operations
- Behaviors
- Template Parameters
- Inner Elements
- Relations
- Tags
- Constraints
- Language Properties

C# Language Properties	
externAlias	
static	✓ true
partial	
unsafe	false
new	false
internal	false
C#Class	
C#Attributes	
Applied Stereotype Instance	:C#Class [A]
Owned Comment	
Owner	Data
Owned Element	:C#Class [A]
Owning Package	Data
Name Expression	
Namespace	Data
Qualified Name	A
Visibility	public
Name	A
Owning Substitution	
Owning Default	
Owning Parameter	

Extern Alias Directive

Until now, C# has supported only a single namespace hierarchy into which types from referenced assemblies and the current program are placed. Because of this design, it has not been possible to reference types with the same fully qualified name from different assemblies, a situation that arises when types are independently given the same name, or when a program needs to reference several versions of the same assembly. Extern aliases make it possible to create and reference separate namespace hierarchies in such situations.

An *extern-alias-directive* introduces an identifier that serves as an alias for a namespace hierarchy.

Copyright © 1998-2015 No Magic, Inc.

compilation-unit:
  extern-alias-directives$_{opt}$ *using-directives*$_{opt}$ *global-attributes*$_{opt}$
    namespace-member-declarations$_{opt}$

namespace-body:
  `{` *extern-alias-directives*$_{opt}$ *using-directives*$_{opt}$ *namespace-member-declarations*$_{opt}$ `}`

extern-alias-directives:
  extern-alias-directive
  extern-alias-directives *extern-alias-directive*

extern-alias-directive:
  `extern alias` *identifier* `;`

Consider the following two assemblies:

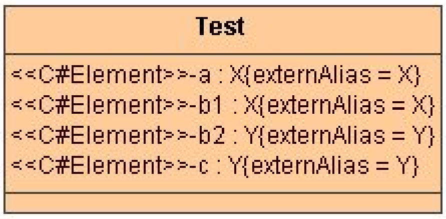
```
Assembly a1.dll:
namespace N
{
	public class A {}
	public class B {}
}
Assembly a2.dll:
namespace N
{
	public class B {}
	public class C {}
}
and the following program:
class Test
{
	N.A a;
	N.C c;
}
```

The following program declares and uses two extern aliases, X and Y, each of which represent the root of a distinct namespace hierarchy created from the types contained in one or more assemblies.

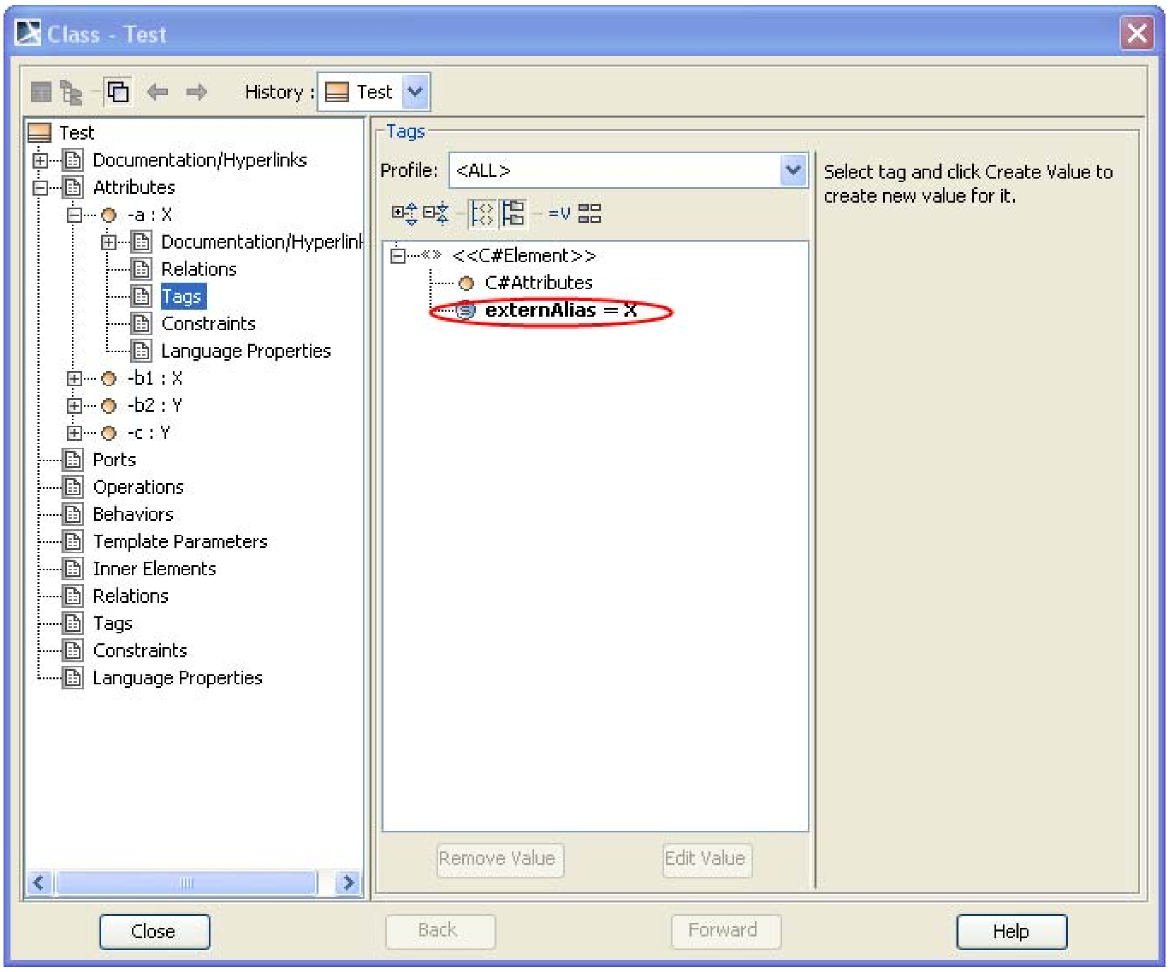
```
extern alias X;
extern alias Y;
class Test
{
	X::N.A a;
	X::N.B b1;
	Y::N.B b2;
	Y::N.C c;
}
```

Copyright © 1998-2015 No Magic, Inc.

Code	MD-UML

`extern alias X;`
`extern alias Y;`
`class Test`
`{`
`        X::N.A a;`
`        X::N.B b1;`
`        Y::N.B b2;`
`        Y::N.C c;`
`}`
	

Test
<<C#Element>>-a : X{externAlias = X}
<<C#Element>>-b1 : X{externAlias = X}
<<C#Element>>-b2 : Y{externAlias = Y}
<<C#Element>>-c : Y{externAlias = Y}



Profile: <ALL>

<<C#Element>>
C#Attributes
externAlias = X

Select tag and click Create Value to create new value for it.

Pragma Directives

The #pragma preprocessing directive is used to specify optional contextual information to the compiler. The information supplied in a #pragma directive will never change program semantics.

pp-directive:
  …
  pp-pragma

Copyright © 1998-2015 No Magic, Inc.

pp-pragma:
   whitespace$_{opt}$ `#` *whitespace*$_{opt}$ `pragma` *whitespace* *pragma-body* *pp-new-line*

pragma-body:
   pragma-warning-body

C# 2.0 provides #pragma directives to control compiler warnings. Future versions of the language may include additional #pragma directives.

Pragma Warning

The #pragma warning directive is used to disable or restore all or a particular set of warning messages during compilation of the subsequent program text.

pragma-warning-body:
   `warning` *whitespace* *warning-action*
   `warning` *whitespace* *warning-action* *whitespace* *warning-list*

warning-action:
   `disable`
   `restore`

warning-list:
   decimal-digits
   warning-list *whitespace*$_{opt}$ `,` *whitespace*$_{opt}$ *decimal-digits*

A `#pragma warning` directive that omits the warning list affects all warnings. A `#pragma warning` directive that includes a warning list affects only those warnings that are specified in the list.

A `#pragma warning disable` directive disables all or the given set of warnings.

A `#pragma warning restore` directive restores all or the given set of warnings to the state that was in effect at the beginning of the compilation unit. Note that if a particular warning was disabled externally, a `#pragma warning restore` will not re-enable that warning.

The following example shows use of `#pragma` warning to temporarily disable the warning reported when obsolete members are referenced.

```
using System;
class Program
{
	[Obsolete]
	static void Foo() {}
	static void Main() {
#pragma warning disable 612
	Foo();
#pragma warning restore 612
	}
}
```

There is no code engineering mapping for Pragma Directives now.

Copyright © 1998-2015 No Magic, Inc.

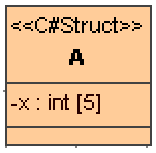
Fix Size Buffer

Fixed size buffers are used to declare “C style” in-line arrays as members of structs. Fixed size buffers are primarily useful for interfacing with unmanaged APIs. Fixed size buffers are an unsafe feature, and fixed size buffers can only be declared in unsafe contexts.

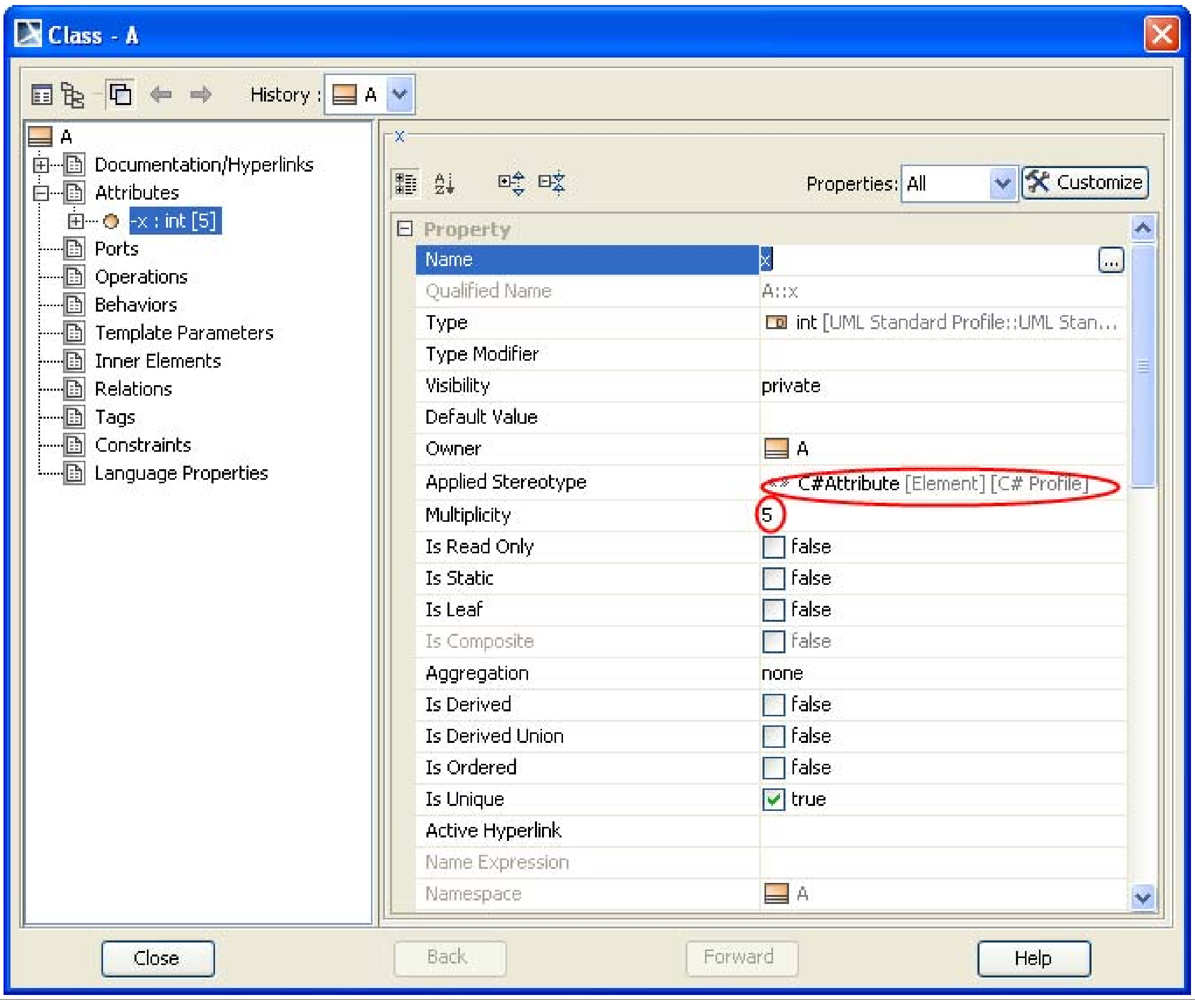
A fixed size buffer is a member that represents storage for a fixed length buffer of variables of a given type. A fixed size buffer declaration introduces one or more fixed size buffers of a given element type. Fixed size buffers are only permitted in struct declarations and can only occur in unsafe contexts.

```
struct-member-declaration:
…
fixed-size-buffer-declaration
fixed-size-buffer-declaration:
attributesopt   fixed-size-buffer-modifiersopt   fixed   buffer-element-type
                         fixed-sized-buffer-declarators   ;
fixed-size-buffer-modifiers:
fixed-size-buffer-modifier
fixed-sized-buffer-modifier   fixed-size-buffer-modifiers
fixed-size-buffer-modifiers:
new
public
protected
internal
private
unsafe
buffer-element-type:
type
fixed-sized-buffer-declarators:
fixed-sized-buffer-declarator
fixed-sized-buffer-declarator   fixed-sized-buffer-declarators
fixed-sized-buffer-declarator:
identifier   [   const-expression   ]
```

Code	MD-UML

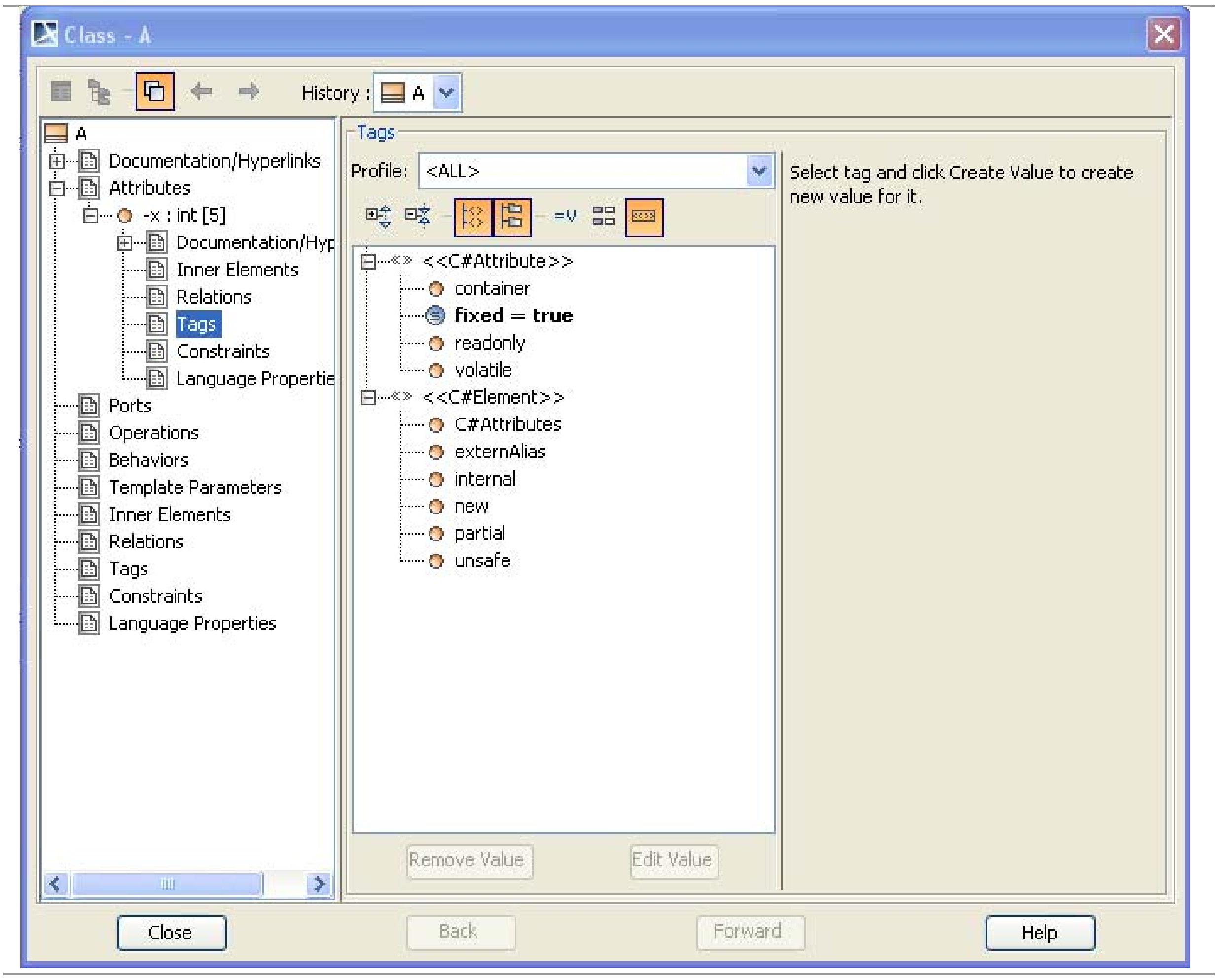
`unsafe struct A`
`{`
`   public fixed int x[5];`
`}`
	

Copyright © 1998-2015 No Magic, Inc.



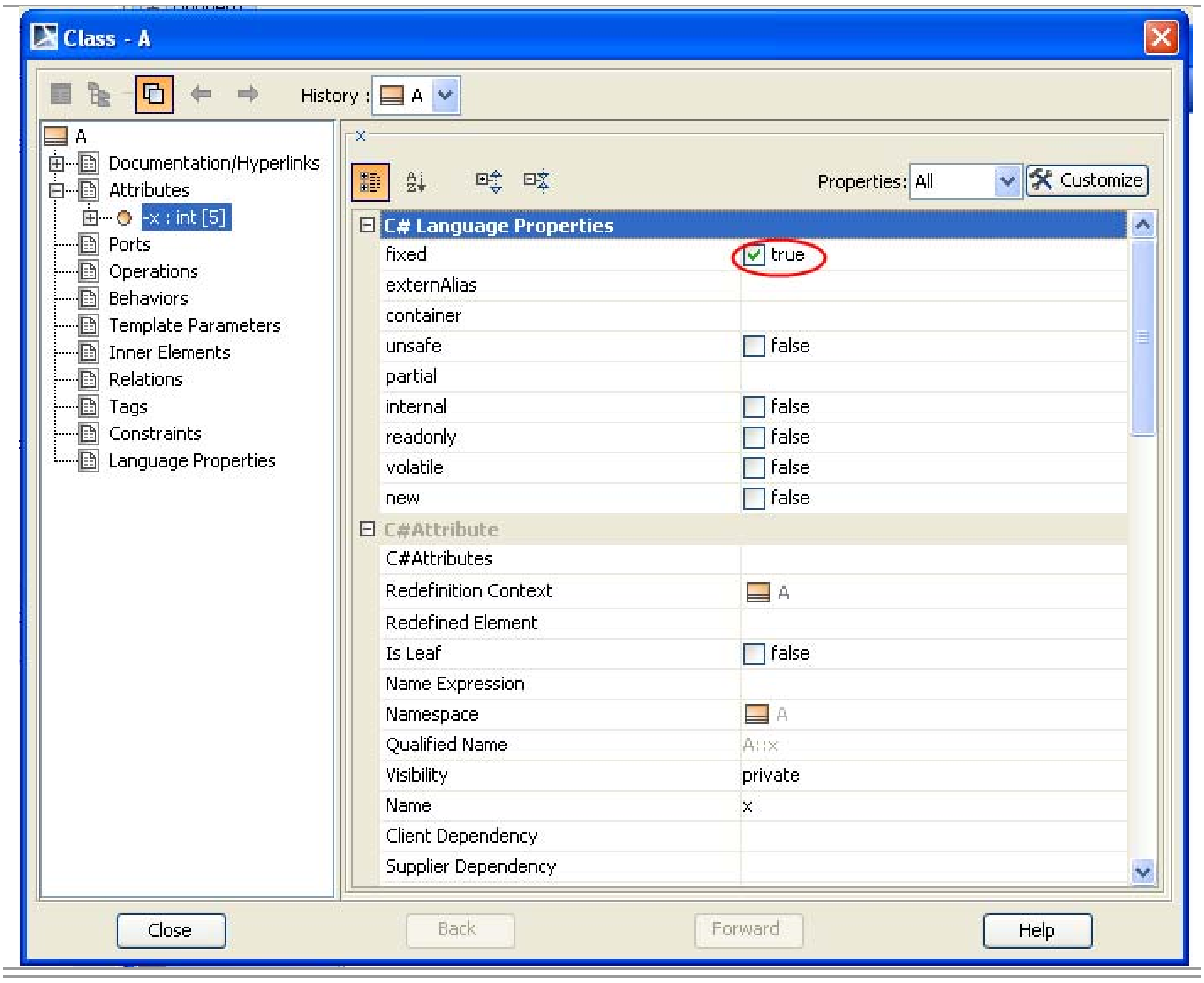
Property	
Name	x
Qualified Name	A::x
Type	int [UML Standard Profile::UML Stan...
Type Modifier	
Visibility	private
Default Value	
Owner	A
Applied Stereotype	C#Attribute [Element] [C# Profile]
Multiplicity	5
Is Read Only	false
Is Static	false
Is Leaf	false
Is Composite	false
Aggregation	none
Is Derived	false
Is Derived Union	false
Is Ordered	false
Is Unique	✓ true
Active Hyperlink	
Name Expression	
Namespace	A

Copyright © 1998-2015 No Magic, Inc.



Select tag and click Create Value to create new value for it.

Copyright © 1998-2015 No Magic, Inc.



A
Documentation/Hyperlinks
Attributes
-x : int [5]
Ports
Operations
Behaviors
Template Parameters
Inner Elements
Relations
Tags
Constraints
Language Properties

C# Language Properties	
fixed	✓ true
externAlias	
container	
unsafe	false
partial	
internal	false
readonly	false
volatile	false
new	false
C#Attribute	
C#Attributes	
Redefinition Context	A
Redefined Element	
Is Leaf	false
Name Expression	
Namespace	A
Qualified Name	A::x
Visibility	private
Name	x
Client Dependency	
Supplier Dependency	

C# 3.0 Description

Extension Methods

Extension methods are static methods that can be invoked using instance method syntax. In effect, extension methods make it possible to extend existing types and constructed types with additional methods.

Extension methods are declared by specifying the keyword `this` as a modifier on the first parameter of the methods. Extension methods can only be declared in static classes.

The sample code :

```
public static class Extensions
{
      public static int ToInt32(this string s) {
            return Int32.Parse(s);
      }
      public static T[] Slice<T>(this T[] source, int index, int count) {
            if (index < 0 || count < 0 || source.Length – index < count)
                  throw new ArgumentException();
            T[] result = new T[count];
            Array.Copy(source, index, result, 0, count);
```

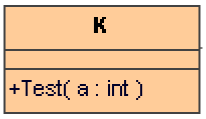
Copyright © 1998-2015 No Magic, Inc.

```
                return result;
            }
    }
```

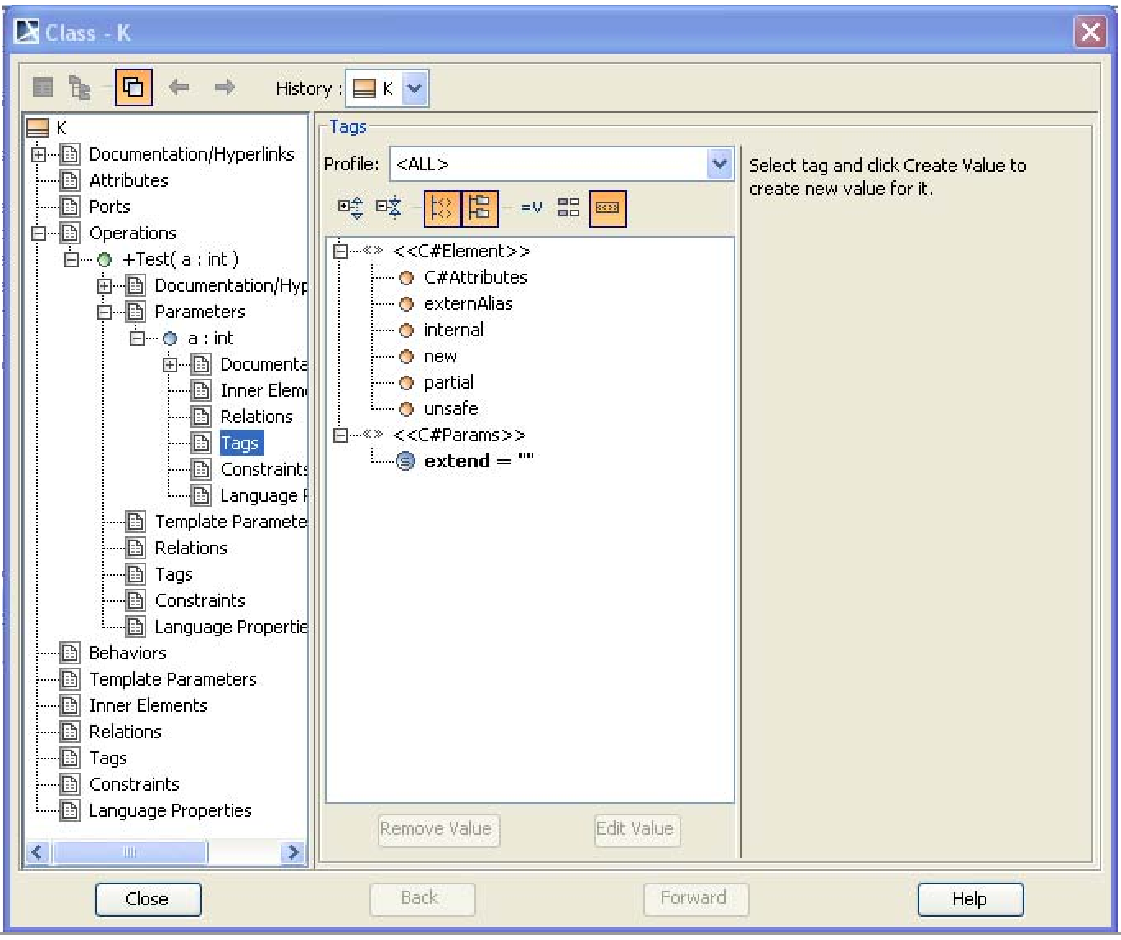
It becomes possible to invoke the extension methods in the static class `Extensions` using instance method syntax:

```
string s = "1234";
int i = s.ToInt32();// Same as Extensions.ToInt32(s)

int[] digits = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};
int[] a = digits.Slice(4, 3);// Same as Extensions.Slice(digits, 4, 3)
```

Code	MD-UML

`class K`
`{`
`        Test(this int a)`
`        {}`
`}`
	

The value is created to tag “extend” in «C#Params»



Tags

K
Documentation/Hyperlinks
Attributes
Ports
Operations
+Test(a : int)
Documentation/Hyp
Parameters
a : int
Documenta
Inner Elem
Relations
Tags
Constraints
Language P
Template Paramete
Relations
Tags
Constraints
Language Propertie
Behaviors
Template Parameters
Inner Elements
Relations
Tags
Constraints
Language Properties

<<C#Element>>
C#Attributes
externAlias
internal
new
partial
unsafe
<<C#Params>>
extend = ""

Select tag and click Create Value to create new value for it.

Lambda Expression Conversion

A lambda expression is written as a parameter list, followed by the => token, followed by an expression or a statement block.

Copyright © 1998-2015 No Magic, Inc.

The parameters of a lambda expression can be explicitly or implicitly typed. In an explicitly typed parameter list, the type of each parameter is explicitly stated. In an implicitly typed parameter list, the types of the parameters are inferred from the context in which the lambda expression occurs.

Some examples of lambda expressions are below:

```
x => x + 1                              // Implicitly typed, expression body
x => { return x + 1; }         // Implicitly typed, statement body
(int x) => x + 1                       // Explicitly typed, expression body
(int x) => { return x + 1; }// Explicitly typed, statement body
(x, y) => x * y                         // Multiple parameters
() => Console.WriteLine()// No parameters
```

There is no mapping for this feature.

Copyright © 1998-2015 No Magic, Inc.

C# Profile

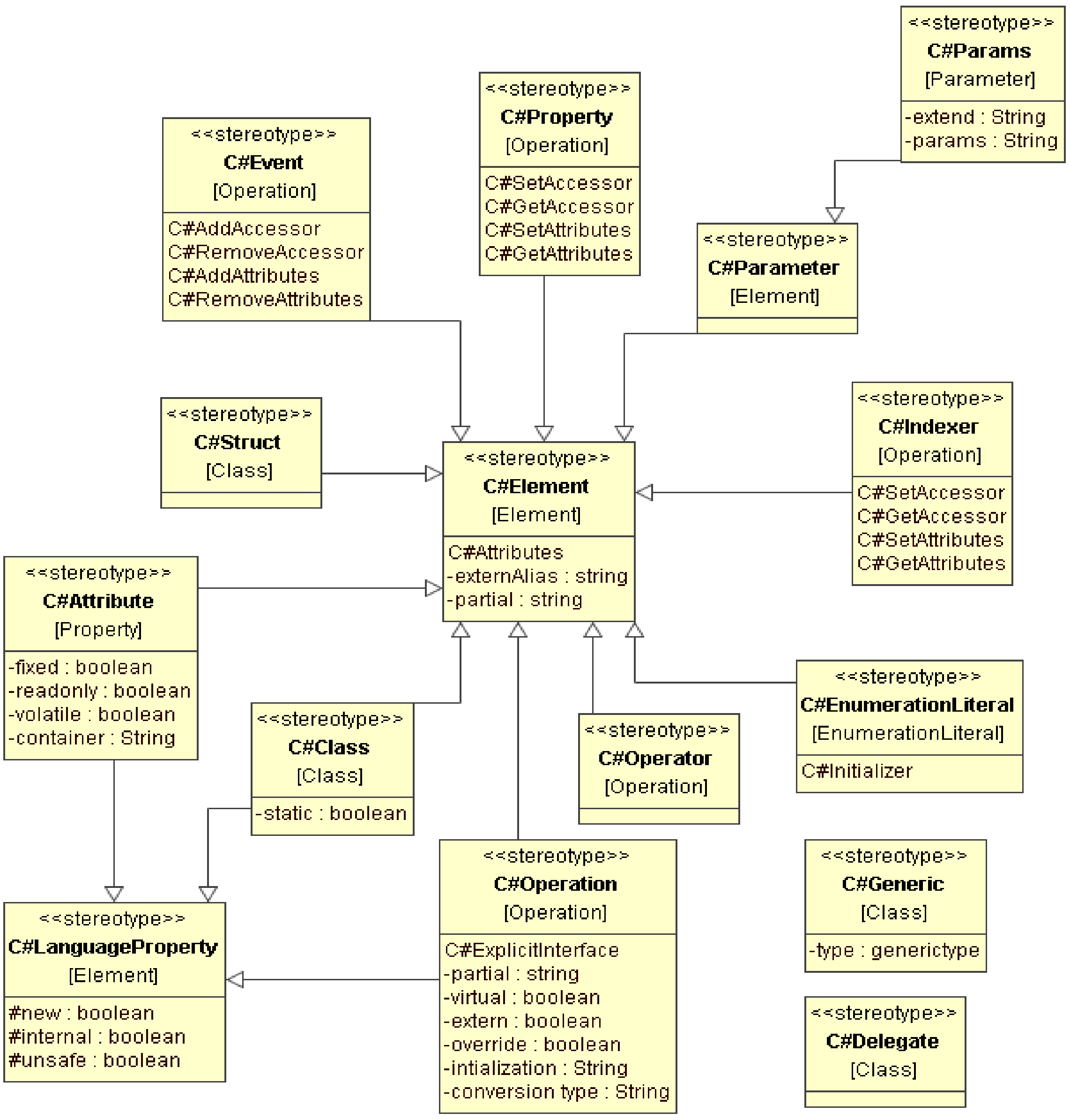


Figure 132 -- C# Profile

The above diagram shows the class diagram of C# profile. The C# profile package contains many stereotype and related tagged value to contain the properties of C# language usage. The new stereotypes and tagged value are shown in the following part.

Stereotype

C# Profile contains many stereotypes and tagged definitions. When the mapping begins, there will be tagged value applied to tagged definition in stereotype. Since MagicDraw 12.0 the information of language properties is moved to tagged value in stereotype. The information of language properties of:

- Class is moved to «C#Class»
- Attribute is moved to «C#Attribute»

Copyright © 1998-2015 No Magic, Inc.

- Operation is moved to «C#Operation»
- The common information of the old language properties (Class, Attribute and Operation) is moved to C#LanguageProperty

This document represents only newly stereotyped and tagged definitions, the details are shown below.

C#Attribute

«C#Attribute» is an invisible stereotype that is used to define C# attribute properties. This is used to store the language properties of the attribute in C#.

Name	Meta class	Constraints
C#Attribute	Property	

Tag	Type	Description
container	String	
fixed	Boolean	Represents the usage of Fixed Size Buffer unsafe struct A { public **fixed** int x[5]; }
readonly	Boolean	Represents the usage of read-only attribute Class A { public **readonly** int x; }
volatile	Boolean	Represents the usage of volatile attribute Class A { public **volatile** int x; }

C#Class

«C#Class» is a stereotype that is used to define property of C# class. This is used to store the language properties of the class in C#. Moreover this class is used for store the information about the static class too.

Name	Meta class	Constraints
C#Class	Class	

Tag	Type	Description
Static	boolean	Represents the usage of static class **static** Class A { public int x; }

Copyright © 1998-2015 No Magic, Inc.

C#Delegate

«C#Delegate» is a stereotype that indicates that model class represents C# delegate.

Name	Meta class	Constraints
`C#Delegate`	`Class`	

C#Element

«C#Element» is a stereotype that is used to define properties of all element in model.

Name	Meta class	Constraints
`C#Element`	`Element`	This stereotype can be applied to all elements such as class, attribute, operation and parameter.

Tag	Type	Description

`externAlias`	`String`	Represents the usage of extern alias
		extern alias X;
		class Test
		{
		    X::N.A a;
		}

`partial`	`String`	
`C#Attributes`	`String`	Represents C# attributes for element.

C#EnumerationLiteral

«C#EnumeraltionLiteral» is a stereotype that is used to define enumeration.

Name	Meta class	Constraints
`C#EnumerationLiteral`	`EnumerationLiteral`	

Tag	Type	Description
`C#Initializer`	`String`	Represents enumeration member's constant value.

C#Event

«C#Event» is a stereotype that is used to indicate that operation represents C# event.

Name	Meta class	Constraints
`C#Event`	`Operation`	

Tag	Type	Description
`C#AddAccessor`	`String`	Adds *add* accessor for event
`C#AddAttributes`	`String`	Defines C# attributes for *add* accessor.
`C#RemoveAccessor`	`String`	Adds *remove* accessor for event
`C#RemoveAttributes`	`String`	Defines C# attributes for *remove* accessor.

Copyright © 1998-2015 No Magic, Inc.

C#Generic

«C#Generic» is a stereotype that is used to define generic properties.

Name	Meta class	Constraints
`C#Generic`	`Class`	

Tag	Type	Description
`type`	`generictype`	Defines type for generic constraint parameter, class, struct, and new()

C#Indexer

«C#Indexer» is a stereotype that is used to indicate that operation represents C# indexer.

Name	Meta class	Constraints
`C#Indexer`	`Operation`	

Tag	Type	Description
`C#GetAccessor`	`String`	Adds *get* accessor for indexer.
`C#GetAttributes`	`String`	Defines C# attributes for *get* accessor.
`C#SetAccessor`	`String`	Adds *set* accessor for indexer.
`C#SetAttributes`	`String`	Defines C# attributes for *set* accessor.

C#LanguageProperty

«C#LanguageProperty» is the parent of «C#Class», «C#Attribute», and «C#Operation», So the «C#Class», «C#Attribute» and «C#Operation» also have it's tag definition.

Name	Meta class	Constraints
`C#LanguageProperty`	`Element`	

Tag	Type	Description
`internal`	`boolean`	
`new`	`boolean`	

`unsafe`	`boolean`	Represents the usage of unsafe element
		`unsafe struct A`
		`{`
		`   public fixed int x[5];`
		`}`

C#Operation

«C#Operation» is a stereotype which is used to define properties of the operation. This is used to store the language properties of the operation in C#.

Name	Meta class	Constraints
`C#Operation`	`Operation`	

Copyright © 1998-2015 No Magic, Inc.

Tag	Type	Description
`conversion type`	`String`	

`extern`	`boolean`	Represent the usage of extern operation
		`Class A`
		`{`
		`   public extern int x();`
		`}`

`initialization`	`String`	
`override`	`boolean`	
`partial`	`String`	
`virtual`	`boolean`	
`C#ExplicitInterface`	`Classifier`	Defines C# explicit interface for explicit interface member implementation.

C#Operator

«C#Operator» is a stereotype that is used to indicate that the operation represents C# operator.

Name	Meta class	Constraints
`C#Operator`	`Operation`	

C#Parameter

«C#Parameter» is a stereotype that is used to indicate that the element represents C#Parameter.

Name	Meta class	Constraints
`C#Parameter`	`Element`	

C#Params

«C#Params» is a stereotype that is used to indicate parameter arrays.

Name	Meta class	Constraints
`C#Params`	`Parameter`	

Tag	Type	Description
`extend`	`String`	
`params`	`String`	

C#Property

«C#Property» is a stereotype that is used to indicate that the operation represents C# property.

Name	Meta class	Constraints
`C#Property`	`Operation`	

Tag	Type	Description
`C#GetAccessor`	`String`	Adds *get* accessor for indexer.
`C#GetAttributes`	`String`	Defines C# attributes for *get* accessor.
`C#SetAccessor`	`String`	Adds *set* accessor for indexer.
`C#SetAttributes`	`String`	Defines C# attributes for *set* accessor.

Copyright © 1998-2015 No Magic, Inc.

C#Struct

«C#Struct» is a stereotype that is used to indicate that the model class represents C# structure.

Name	Meta class	Constraints
`C#Struct`	`Class`	

C#Using

«C#Using» is a stereotype that is used to indicate that the usage dependency is C# Using directive.

Name	Meta class	Constraints
`C#Using`	`Association, Realization, Usage`	The clients of dependency are Component, and Namespace.

Data Type

In addition to stereotype and tagged value, C# datatype should be created in the profile. The following diagram shows the data type of C# profile.

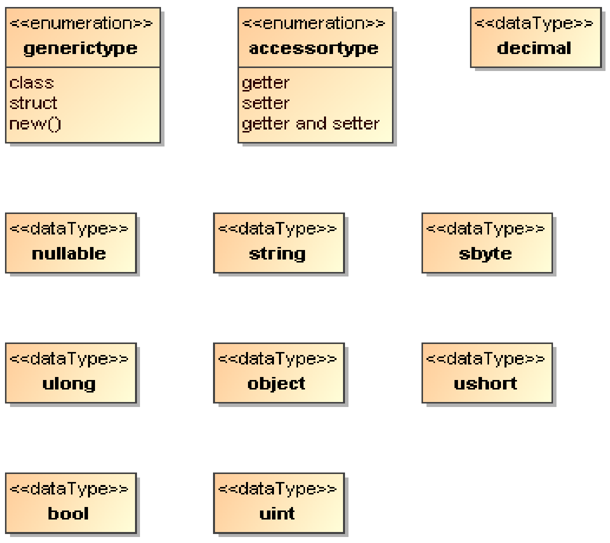


Figure 133 -- Data Type of C# Profile

Conversion from old project version

This chapter describes the project converter of MD project version less than 11.6.

Translation Activity Diagram

There are projects that use C# language properties, which need to be translated with version of MagicDraw project less than or equal to 11.6.

Copyright © 1998-2015 No Magic, Inc.

Open local project

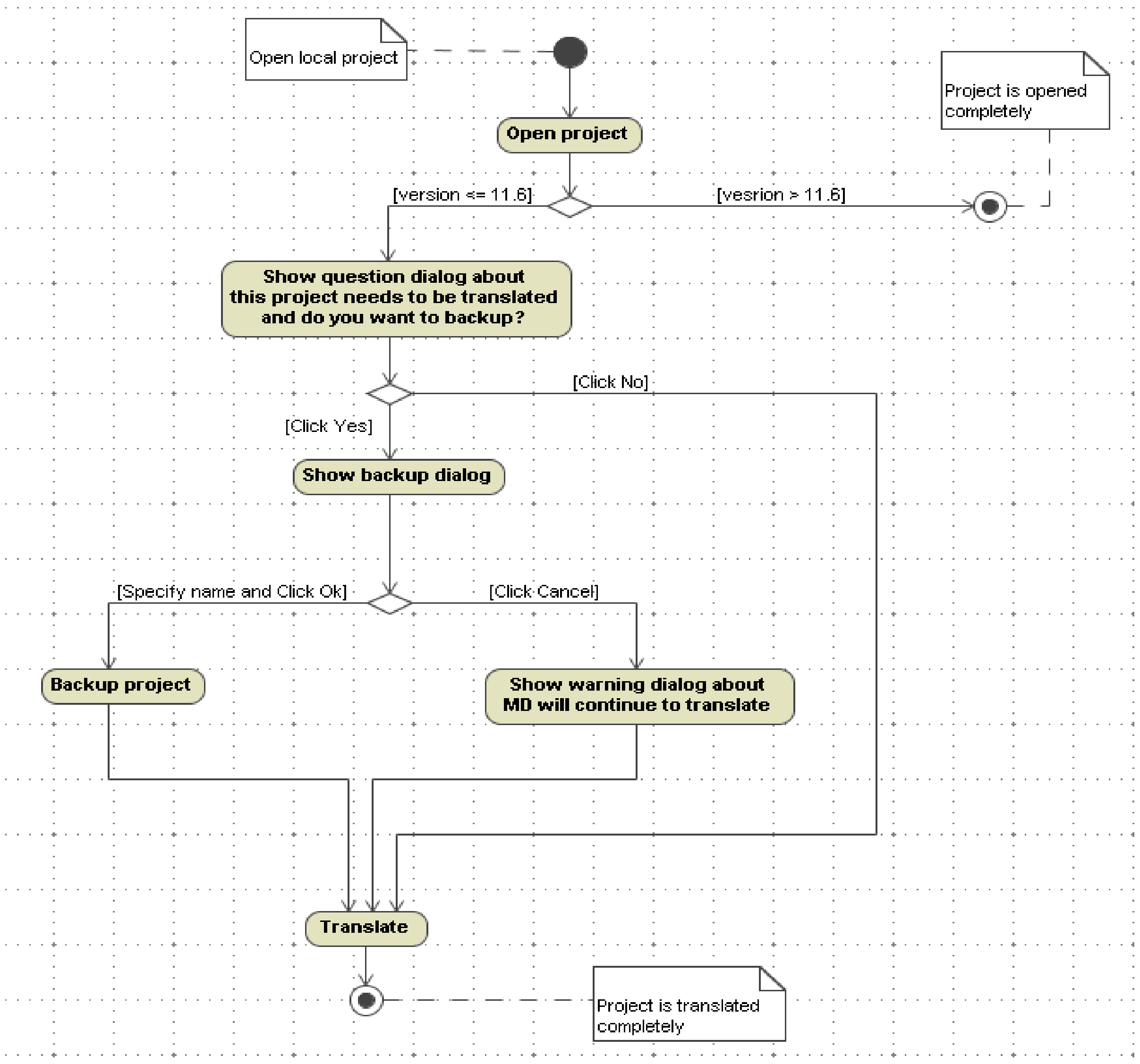

Open local project
Open project
Project is opened completely
[version <= 11.6]
[vesrion > 11.6]
Show question dialog about this project needs to be translated and do you want to backup?
[Click No]
[Click Yes]
Show backup dialog
[Specify name and Click Ok]
[Click Cancel]
Backup project
Show warning dialog about MD will continue to translate
Translate
Project is translated completely

Figure 134 -- Open local project Activity Diagram

Copyright © 1998-2015 No Magic, Inc.

Open teamwork project

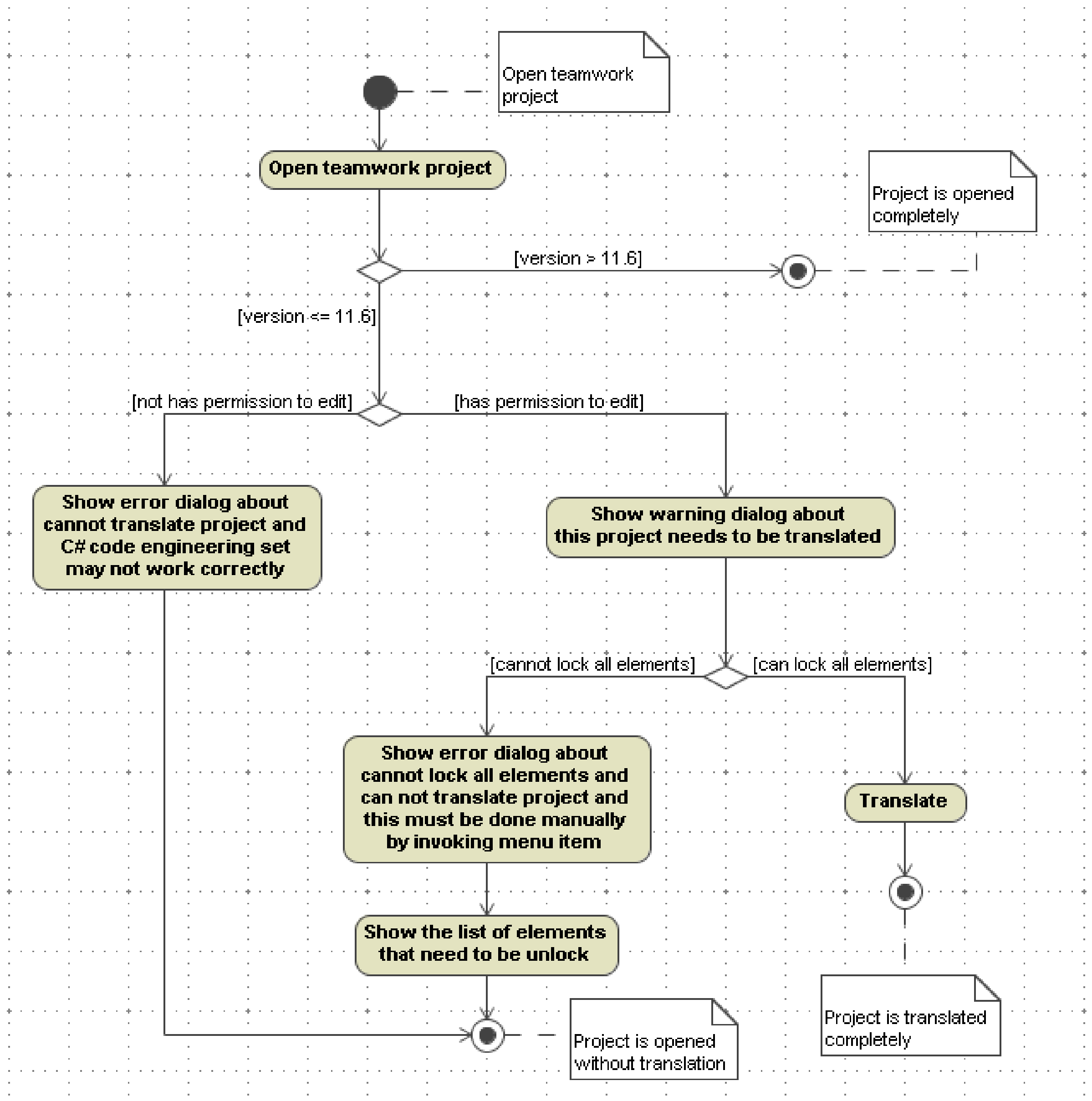

Open teamwork project
Open teamwork project
Project is opened completely
[version > 11.6]
[version <= 11.6]
[not has permission to edit]
[has permission to edit]
Show error dialog about cannot translate project and C# code engineering set may not work correctly
Show warning dialog about this project needs to be translated
[cannot lock all elements]
[can lock all elements]
Show error dialog about cannot lock all elements and can not translate project and this must be done manually by invoking menu item
Translate
Show the list of elements that need to be unlock
Project is translated completely
Project is opened without translation

Figure 135 -- Open teamwork project Activity Diagram

Copyright © 1998-2015 No Magic, Inc.

Import MagicDraw project

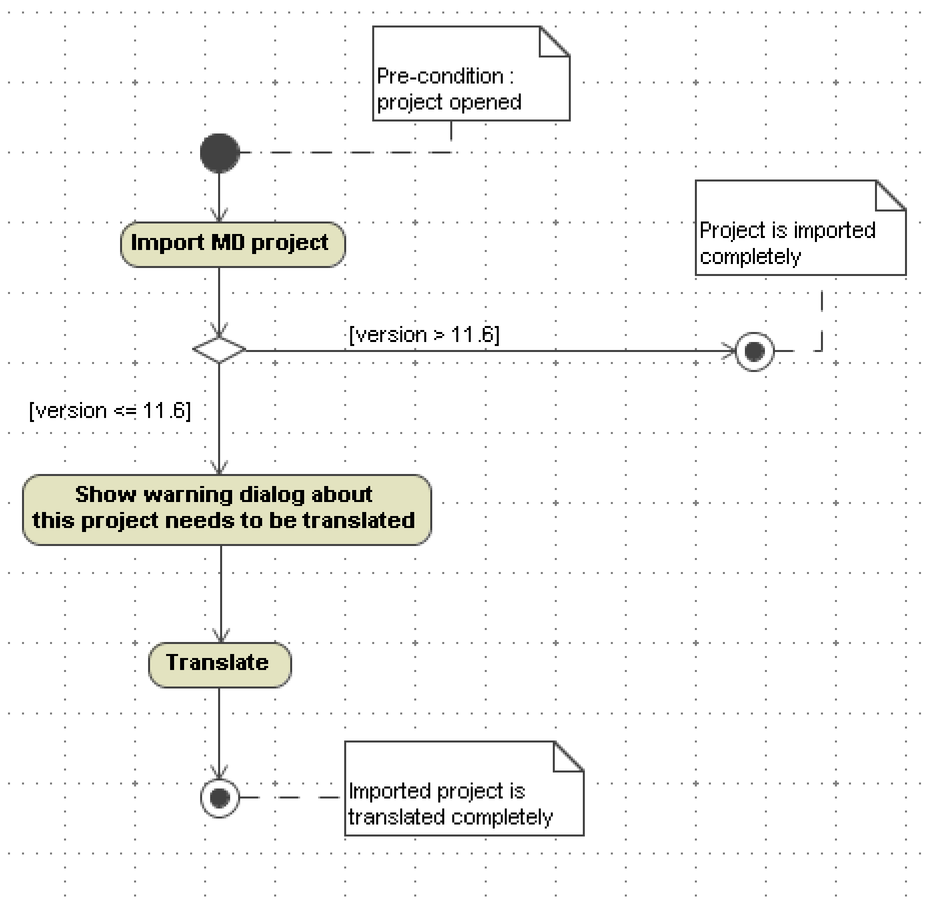

Pre-condition : project opened
Import MD project
Project is imported completely
[version > 11.6]
[version <= 11.6]
Show warning dialog about this project needs to be translated
Translate
Imported project is translated completely

Figure 136 -- Import MagicDraw project Activity Diagram

Copyright © 1998-2015 No Magic, Inc.

Use module

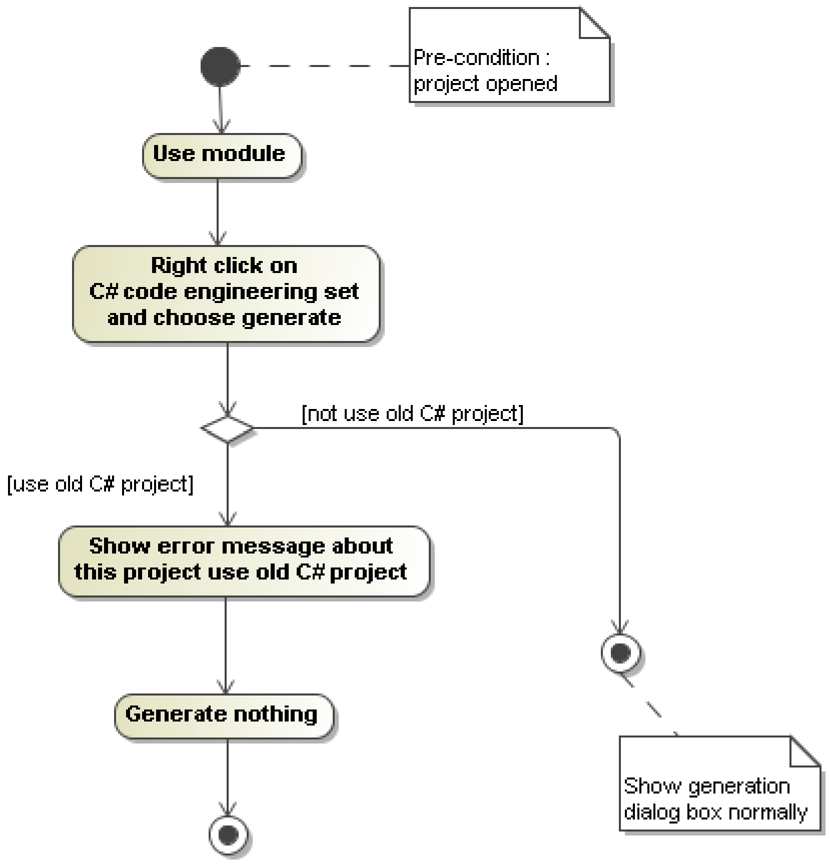

Pre-condition : project opened
Use module
Right click on C# code engineering set and choose generate
[not use old C# project]
[use old C# project]
Show error message about this project use old C# project
Generate nothing
Show generation dialog box normally

Figure 137 -- Use module Activity Diagram

Copyright © 1998-2015 No Magic, Inc.

Update C++ Language Properties and Profiles

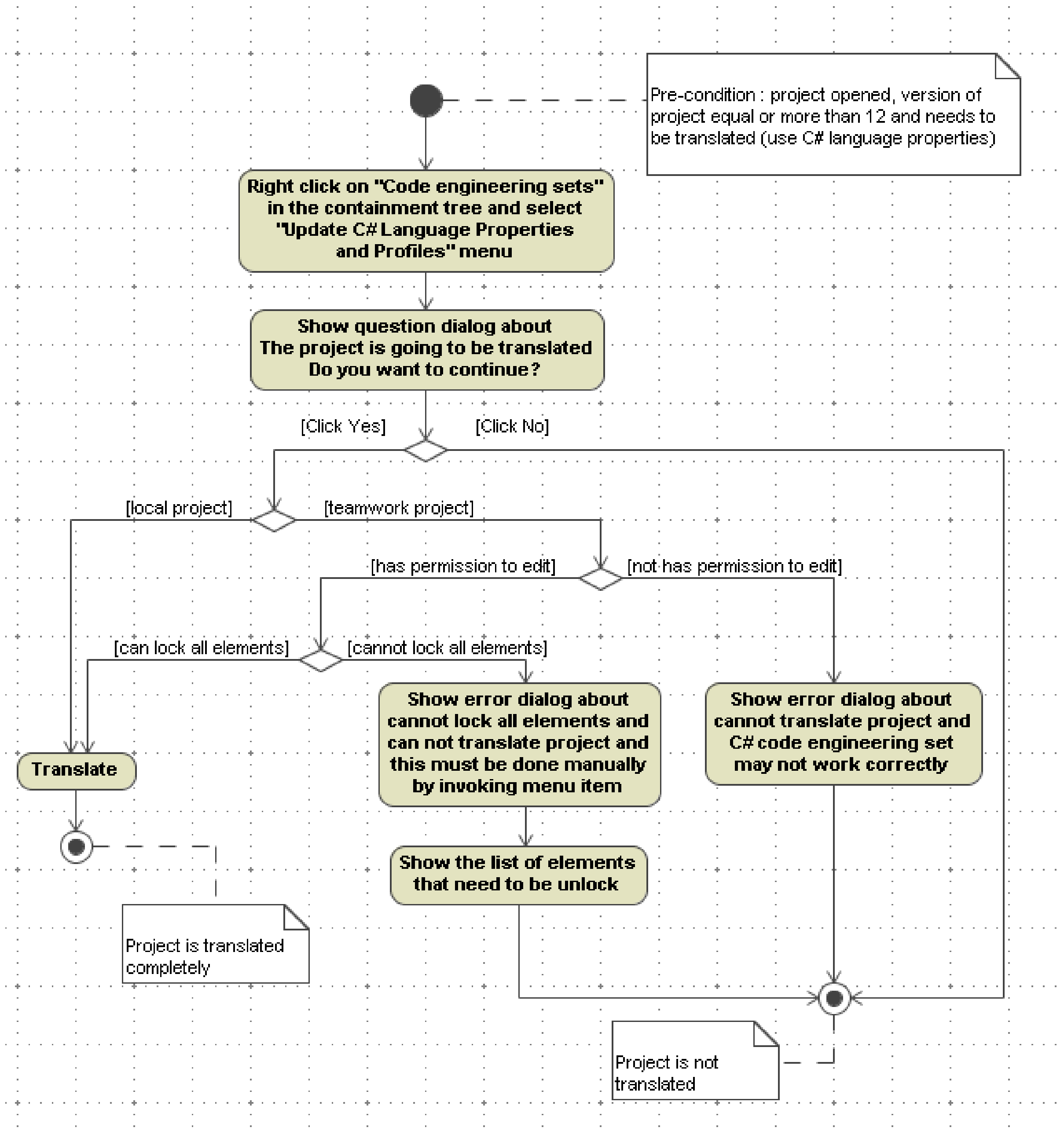

Pre-condition : project opened, version of project equal or more than 12 and needs to be translated (use C# language properties)
Right click on "Code engineering sets" in the containment tree and select "Update C# Language Properties and Profiles" menu
Show question dialog about The project is going to be translated Do you want to continue?
[Click Yes]
[Click No]
[local project]
[teamwork project]
[has permission to edit]
[not has permission to edit]
[can lock all elements]
[cannot lock all elements]
Show error dialog about cannot lock all elements and can not translate project and this must be done manually by invoking menu item
Show error dialog about cannot translate project and C# code engineering set may not work correctly
Translate
Show the list of elements that need to be unlock
Project is translated completely
Project is not translated

Figure 138 -- Update C# Language Properties and Profiles Activity Diagram

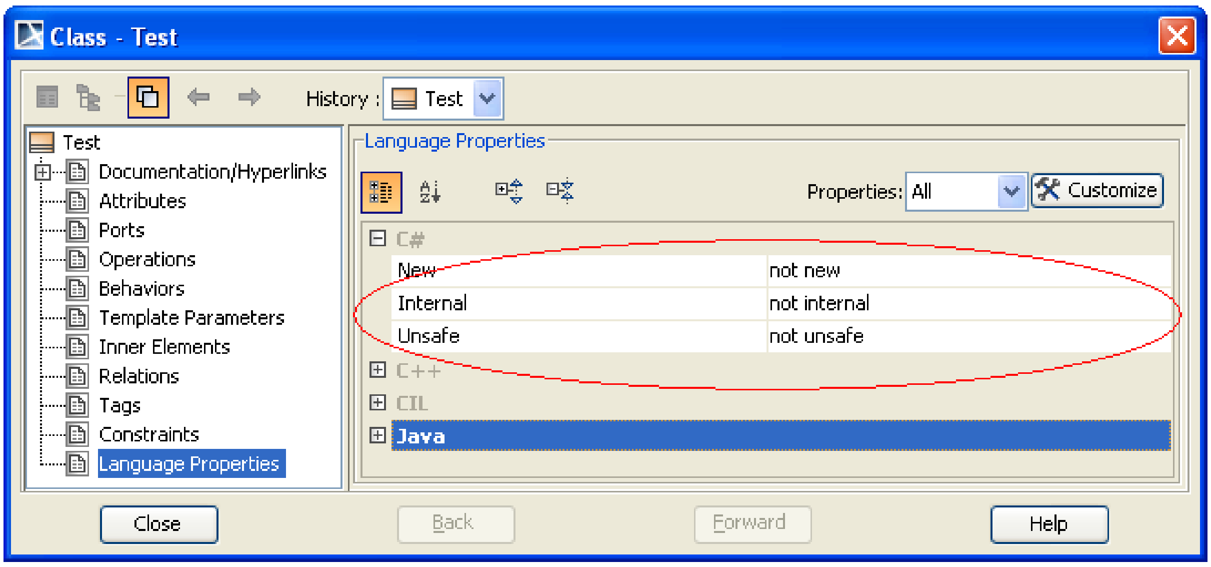
Mapping

Language Properties Mapping

Until MD version 11.6, language properties are stored in a specific format, since MD version 12 language properties are moved to stereotype tag value and using DSL to customize to C# Language Properties. (The language properties will move to C# Language Properties)

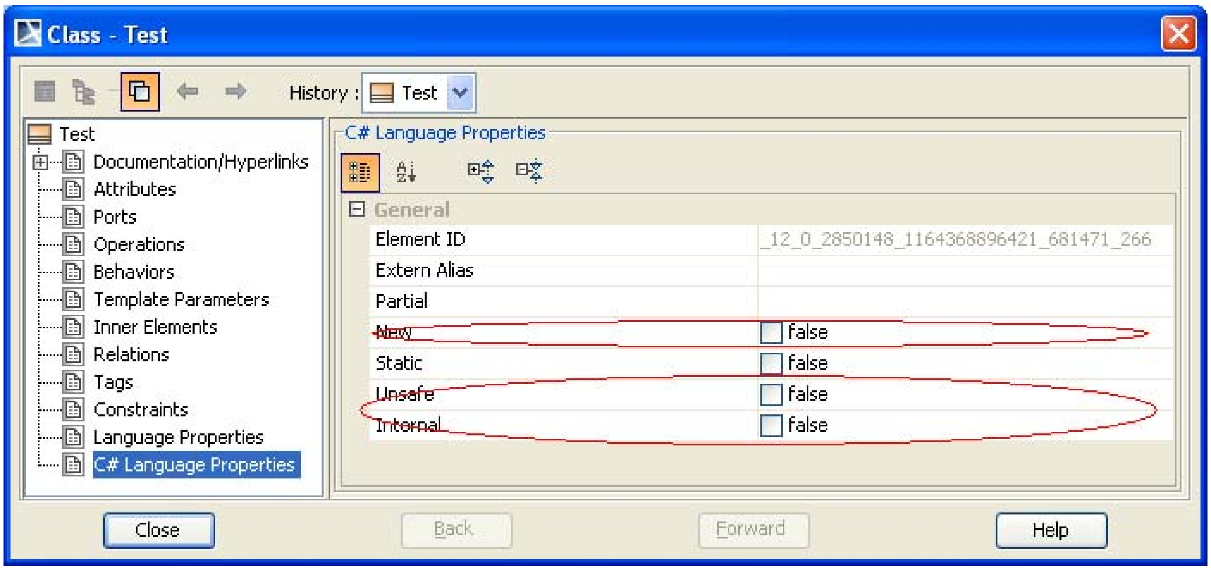
Copyright © 1998-2015 No Magic, Inc.

Class



Language Properties

C#	
New	not new
Internal	not internal
Unsafe	not unsafe



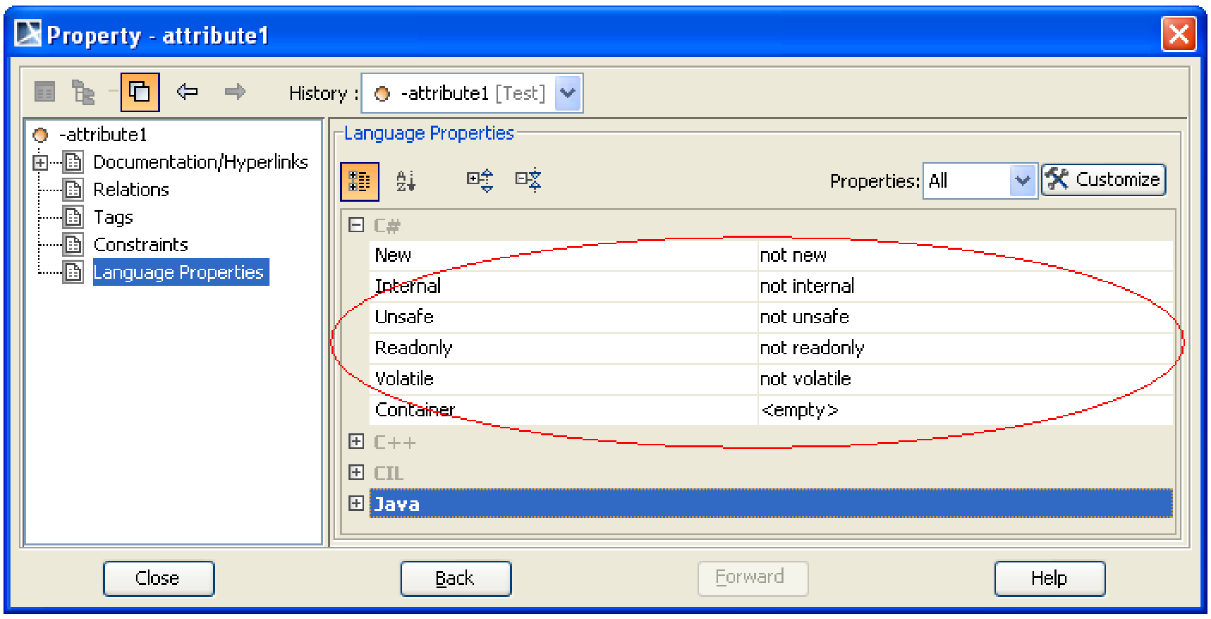
C# Language Properties

General	
Element ID	_12_0_2850148_1164368896421_681471_266
Extern Alias	
Partial	
New	false
Static	false
Unsafe	false
Internal	false

Old Value	Translation
New	Mapped to tag “new” of «C#LanguageProperty» (The usage for this is to apply «C#Class» and set value to tag “new”)
Internal	Mapped to tag “internal” of «C#LanguageProperty» (The usage for this is to apply «C#Class» and set value to tag “internal”)
Unsafe	Mapped to tag “unsafe” of «C#LanguageProperty» (The usage for this is to apply «C#Class» and set value to tag “unsafe”)

Copyright © 1998-2015 No Magic, Inc.

Attribute



Property - attribute1

History : -attribute1 [Test]

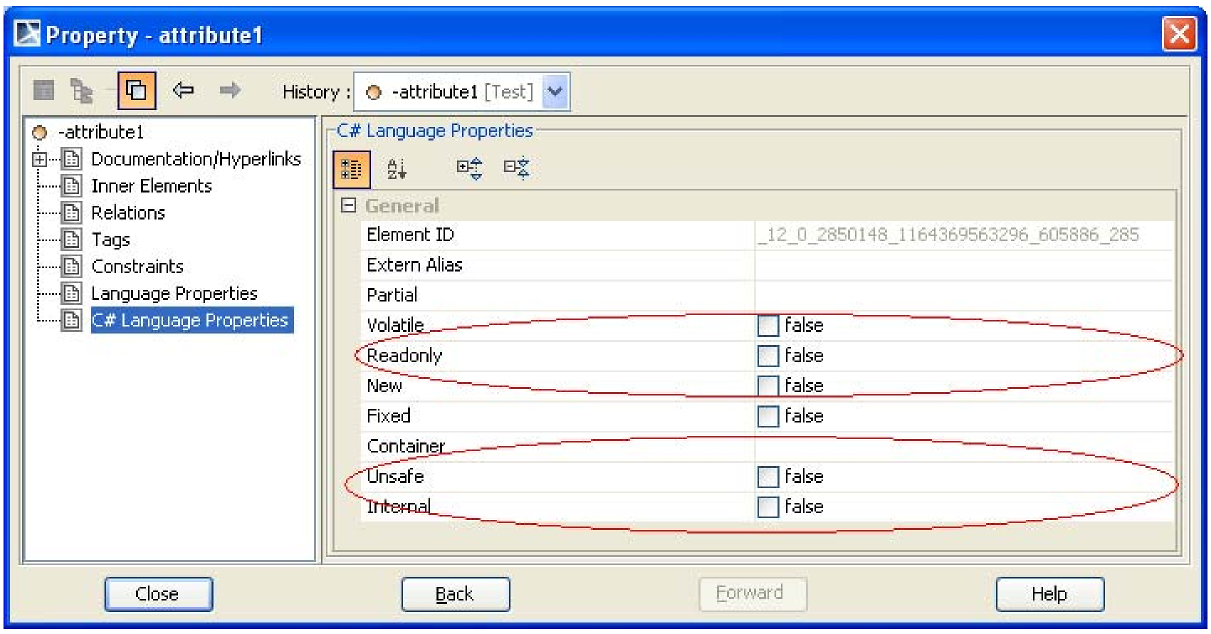
- -attribute1
  - Documentation/Hyperlinks
  - Relations
  - Tags
  - Constraints
  - Language Properties

Language Properties

Properties: All

C#	
New	not new
Internal	not internal
Unsafe	not unsafe
Readonly	not readonly
Volatile	not volatile
Container	<empty>

C++
CIL
Java



Property - attribute1

History : -attribute1 [Test]

- -attribute1
  - Documentation/Hyperlinks
  - Inner Elements
  - Relations
  - Tags
  - Constraints
  - Language Properties
  - C# Language Properties

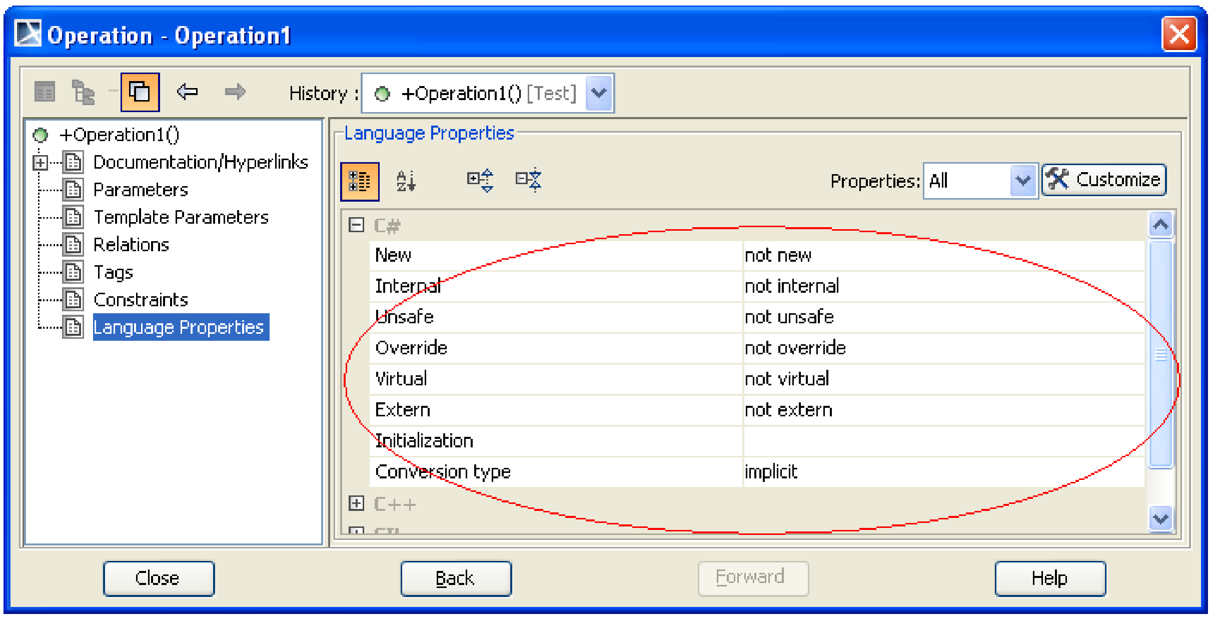
C# Language Properties

General	
Element ID	_12_0_2850148_1164369563296_605886_285
Extern Alias	
Partial	
Volatile	false
Readonly	false
New	false
Fixed	false
Container	
Unsafe	false
Internal	false

Old Value	Translation
New	Mapped to tag “new” of «C#LanguageProperty» (The usage for this is to apply «C#Attribute» and set value to tag “new”)
Internal	Mapped to tag “internal” of «C#LanguageProperty» (The usage for this is to apply «C#Attribute» and set value to tag “internal”)
Unsafe	Mapped to tag “unsafe” of «C#LanguageProperty» (The usage for this is to apply «C#Attribute» and set value to tag “unsafe”)
Readonly	Mapped to “readonly” of «C#Attribute»
Volatile	Mapped to “volatile” of «C#Attribute»
Container	Mapped to “container” of «C#Attribute»

Copyright © 1998-2015 No Magic, Inc.

Operation



Operation - Operation1

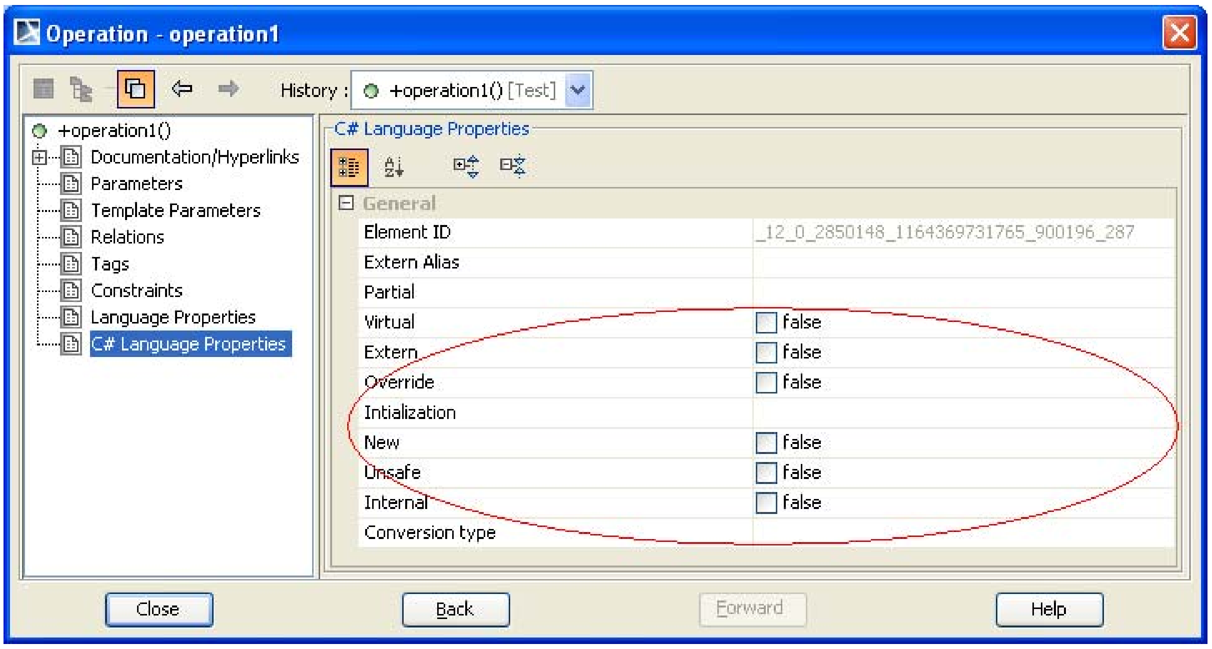
History : +Operation1() [Test]

+Operation1()
Documentation/Hyperlinks
Parameters
Template Parameters
Relations
Tags
Constraints
Language Properties

Language Properties

Properties: All

C#	
New	not new
Internal	not internal
Unsafe	not unsafe
Override	not override
Virtual	not virtual
Extern	not extern
Initialization	
Conversion type	implicit
C++	



Operation - operation1

History : +operation1() [Test]

+operation1()
Documentation/Hyperlinks
Parameters
Template Parameters
Relations
Tags
Constraints
Language Properties
C# Language Properties

C# Language Properties

General	
Element ID	_12_0_2850148_1164369731765_900196_287
Extern Alias	
Partial	
Virtual	false
Extern	false
Override	false
Intialization	
New	false
Unsafe	false
Internal	false
Conversion type	

Old Value	Translation
New	Mapped to tag “new” of «C#LanguageProperty» (The usage for this is to apply «C#Operation» and set value to tag “new”)
Internal	Mapped to tag “internal” of «C#LanguageProperty» (The usage for this is to apply «C#Operation» and set value to tag “internal”)
Unsafe	Mapped to tag “unsafe” of «C#LanguageProperty» (The usage for this is to apply «C#Operation» and set value to tag “unsafe”)
Override	Mapped to tag “override” of «C#Operation»
Virtual	Mapped to tag “virtual” of «C#Operation»
Extern	Mapped to tag “virtual” of «C#Operation»
Initialization	Mapped to tag “initialization” of «C#Operation»
Conversion type	Mapped to tag “conversion type” of «C#Operation»

Copyright © 1998-2015 No Magic, Inc.

C# Properties Customization

DSL Customization is adopted to hide UML mapping, so extensions in MagicDraw should look like they are standard elements. DSL properties should appear in specifications as regular properties, not tags.

So we disable the extension C# language properties and then move them to tagged value in stereotype of C# profile and DSL properties. The DSL properties will conform to tagged value of stereotype.

For an old project containing old C# language properties, after performing open or import, they will be translated to the appropriate stereotype and tagged value.

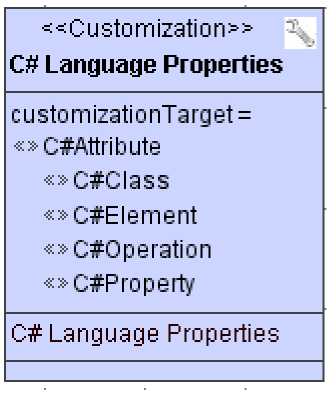
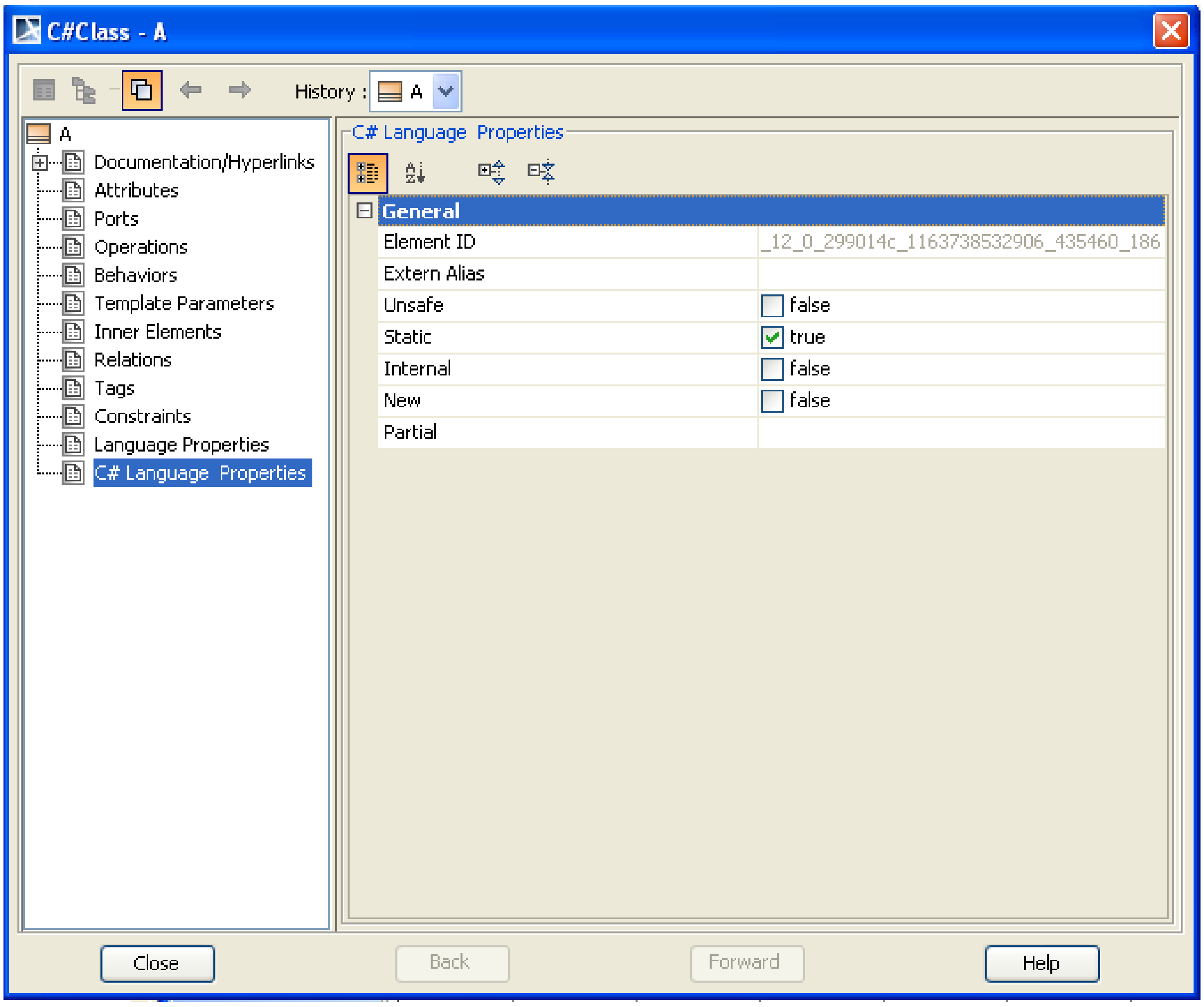


Figure 139 -- Customization Class

All DSL specific custom rules should be stored as tag values in Classes, marked with «Customization» stereotype.

This stereotype contains special tags that should be interpreted by “MD Customization Engine” in special ways, to enable many possible customizations in MD GUI and behavior.

Copyright © 1998-2015 No Magic, Inc.



A
- Documentation/Hyperlinks
- Attributes
- Ports
- Operations
- Behaviors
- Template Parameters
- Inner Elements
- Relations
- Tags
- Constraints
- Language Properties
- C# Language Properties

C# Language Properties

General	
Element ID	_12_0_299014c_1163738532906_435460_186
Extern Alias	
Unsafe	☐ false
Static	✓ true
Internal	☐ false
New	☐ false
Partial	

Figure 140 -- DSL Properties

Using Directive Mapping

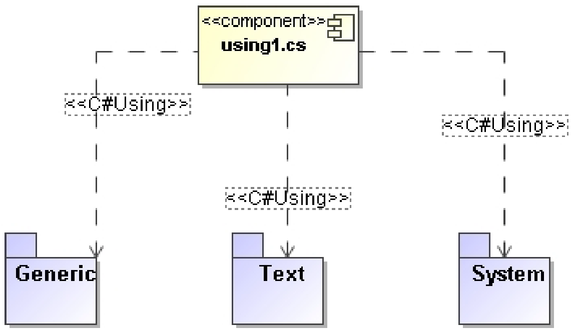
In MagicDraw version 12.1, the C# Using directive is mapped to the model as usage dependency with «C#Using» stereotype.

The following example shows the mapping of C# Using Namespace. The usage dependency for C# Using namespace declaration that is not in the namespace will be created under File View component.

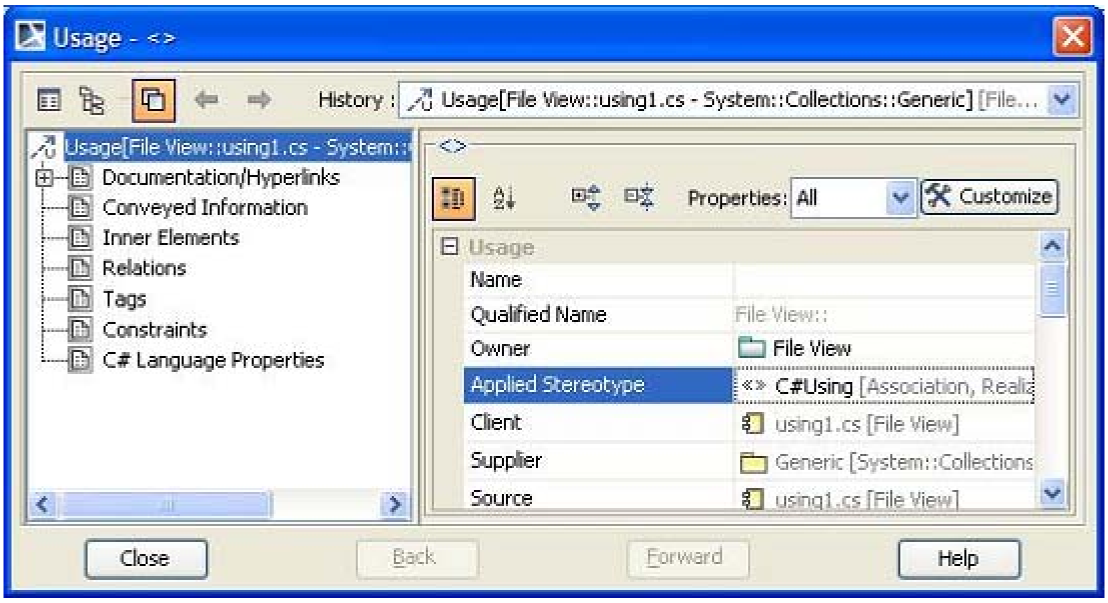
Copyright © 1998-2015 No Magic, Inc.

.

Code	MD-UML

`using System;`
`using System.Collections.Generic;`
`using System.Text;`
`namespace n0`
`{`
`class using1 {}`
`}`
	 <<component>>
	using1.cs
	<<C#Using>>
	<<C#Using>>
	<<C#Using>>
	Generic
	Text
	System

```
Open usage dependency specification
apply «C#Using» stereotype.
For using namespace, leave the name empty.
```


Usage - <>

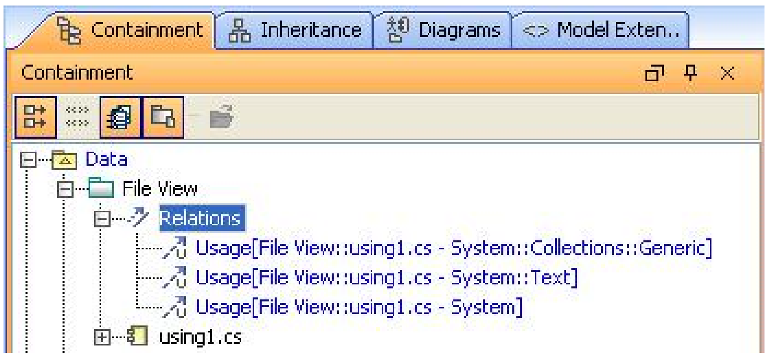
History : Usage[File View::using1.cs - System::Collections::Generic] [File...

Usage[File View::using1.cs - System::
- Documentation/Hyperlinks
- Conveyed Information
- Inner Elements
- Relations
- Tags
- Constraints
- C# Language Properties

Properties: All | Customize

Usage	
Name	
Qualified Name	File View::
Owner	File View
Applied Stereotype	«» C#Using [Association, Realiz
Client	using1.cs [File View]
Supplier	Generic [System::Collections
Source	using1.cs [File View]

Close | Back | Forward | Help

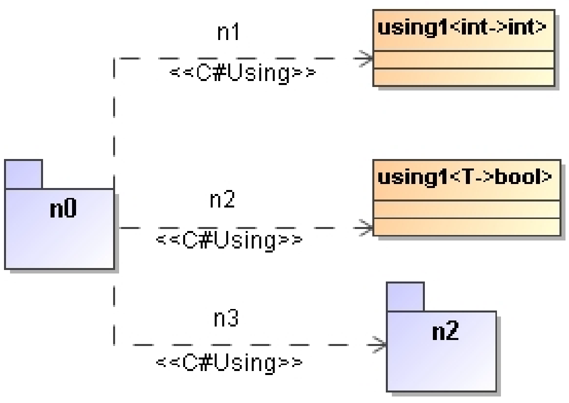

Containment | Inheritance | Diagrams | Model Exten..

Containment

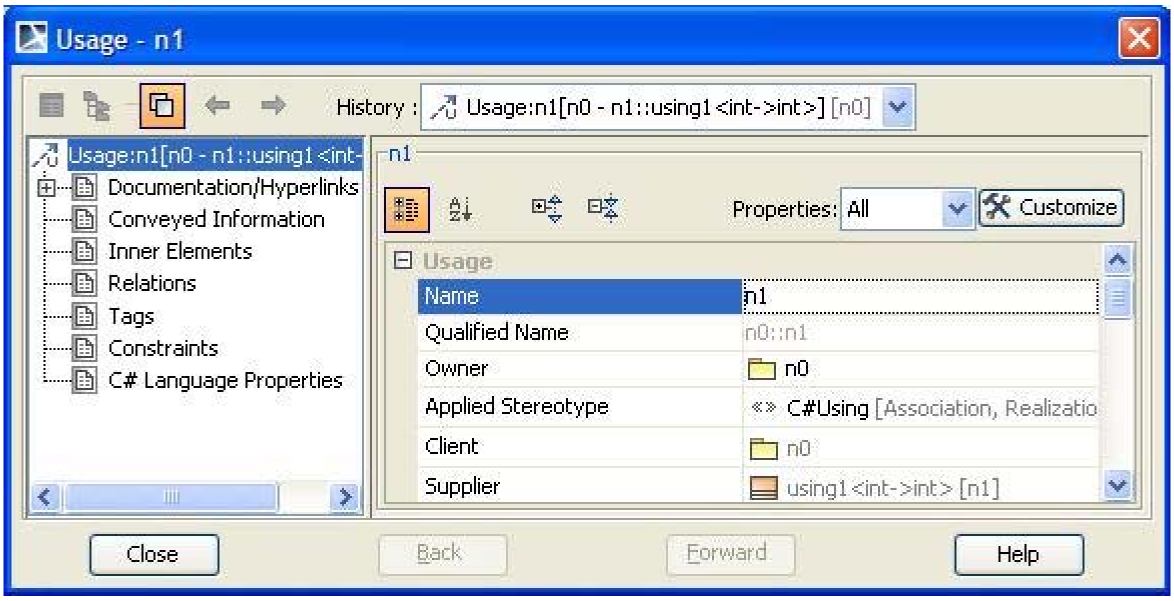
```
Data
  File View
    Relations
      Usage[File View::using1.cs - System::Collections::Generic]
      Usage[File View::using1.cs - System::Text]
      Usage[File View::using1.cs - System]
    using1.cs
```

The following example shows the mapping of C# Using alias. The usage dependency for C# Using alias declared in the namespace will be created under that namespace.

Copyright © 1998-2015 No Magic, Inc.

Code	MD-UML
`namespace n0{     using n1 = n1.using1<int>;     using n2 = n1.n2.using1<bool>;     using n3 = n1.n2;    class using1 : n2    {         n1 a;           n2 b;         n3.using1<float> c;     }}namespace n1{    namespace n2    {         class using1<T> { }      }     class using1<T>     {     }}`	 n1 \<\<C#Using\>\> using1<int->int> n0 n2 \<\<C#Using\>\> using1<T->bool> n3 \<\<C#Using\>\> n2

```
Open usage dependency specification
Apply «C#Using» stereotype.
For using alias, enter the alias name.
```



Usage - n1

History : Usage:n1[n0 - n1::using1<int->int>] [n0]

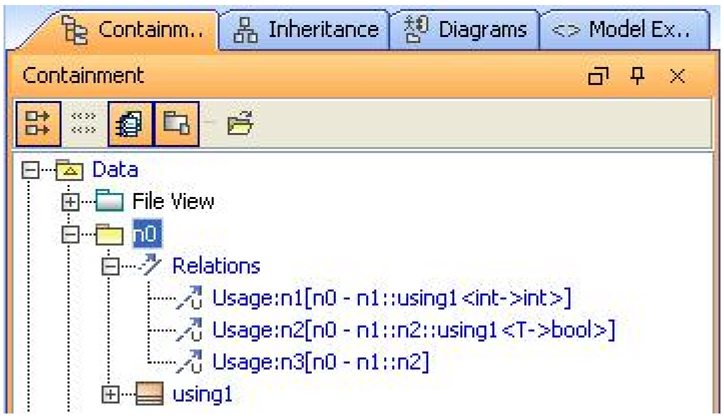
Usage:n1[n0 - n1::using1<int-
- Documentation/Hyperlinks
- Conveyed Information
- Inner Elements
- Relations
- Tags
- Constraints
- C# Language Properties

n1

Properties: All — Customize

Usage	
Name	n1
Qualified Name	n0::n1
Owner	n0
Applied Stereotype	«» C#Using [Association, Realizatio
Client	n0
Supplier	using1<int->int> [n1]

Close — Back — Forward — Help

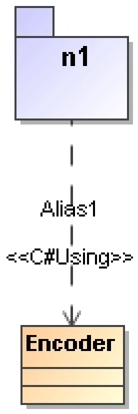


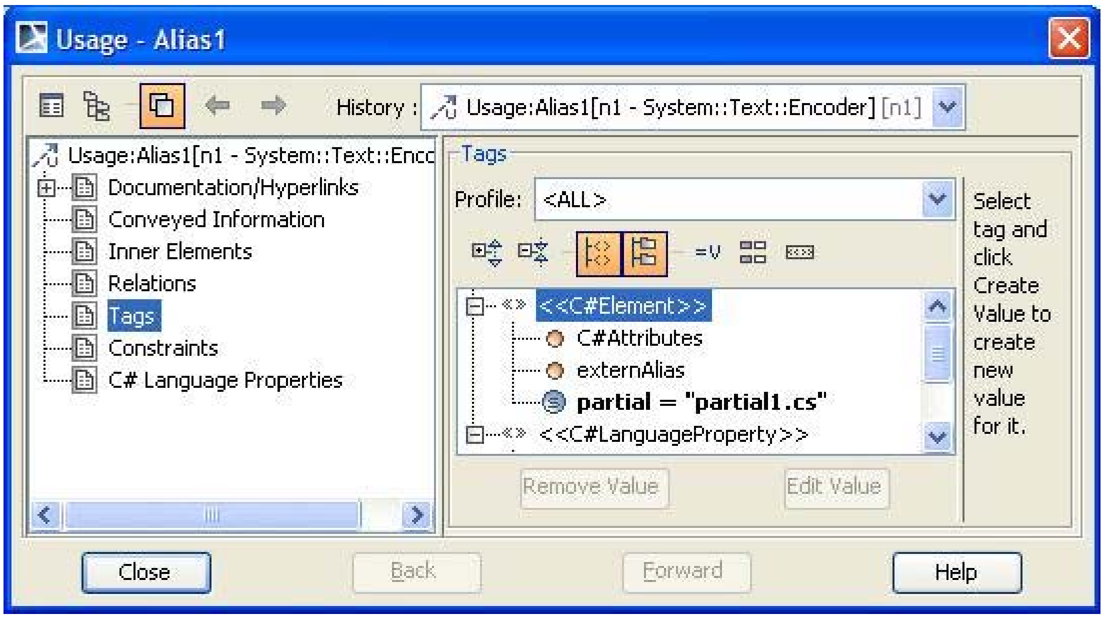
- Data
  - File View
  - n0
    - Relations
      - Usage:n1[n0 - n1::using1<int->int>]
      - Usage:n2[n0 - n1::n2::using1<T->bool>]
      - Usage:n3[n0 - n1::n2]
    - using1

Copyright © 1998-2015 No Magic, Inc.

For mapping C# Using directive in the C# Partial feature, we have to add file component name into Partial Tag value of usage link specification.

Code	MD-UML

```
//partial1.cs
namespace n1 {
        using LL =
System.Text.Encoder;
        partial class using1
        {
        LL a;
        }
}
```
	



Usage - Alias1

History : Usage:Alias1[n1 - System::Text::Encoder] [n1]

Usage:Alias1[n1 - System::Text::Enco
- Documentation/Hyperlinks
- Conveyed Information
- Inner Elements
- Relations
- Tags
- Constraints
- C# Language Properties

Tags

Profile: <ALL>

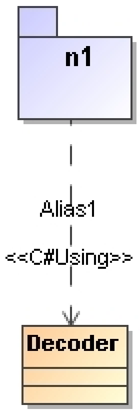
<<C#Element>>
- C#Attributes
- externAlias
- partial = "partial1.cs"

<<C#LanguageProperty>>

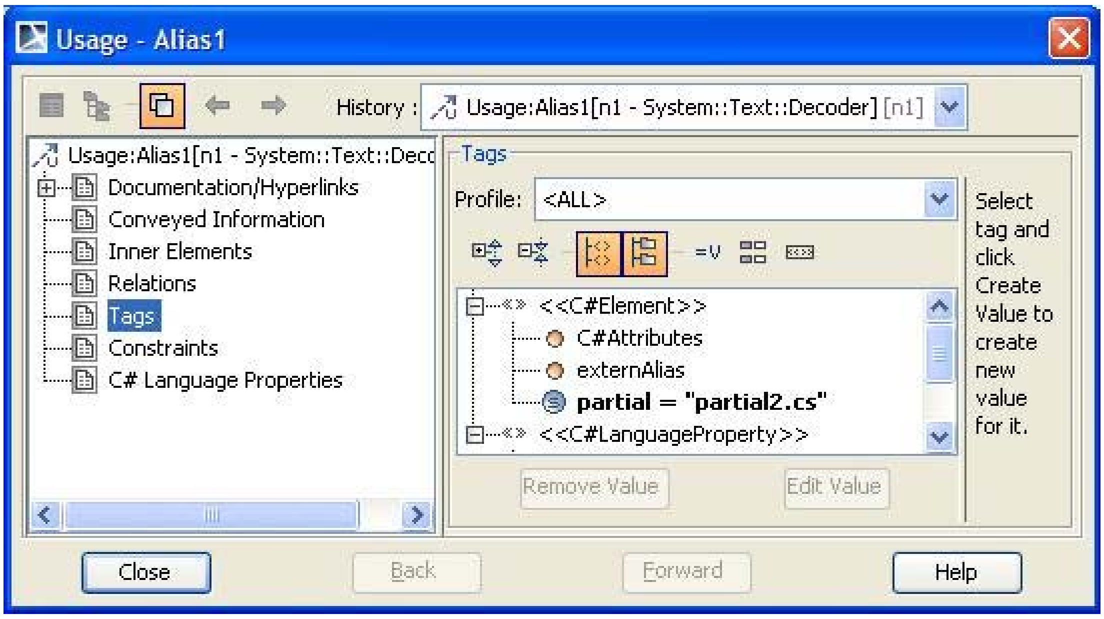
Remove Value | Edit Value

Select tag and click Create Value to create new value for it.

Close | Back | Forward | Help

```
//partial2.cs
namespace n1 {
        using LL =
System.Text.Decoder;
        partial class using1
        {
                LL b;
        }
}
```
	

Copyright © 1998-2015 No Magic, Inc.



Constraints

Mapping Constraints

No Mapping Cases

This section will show some cases which have no mapping in C# reverse engineering.

UML Constraints

Multiple Generic Class

In C# syntax, we can declare classes with the same name but different generic type parameters in the same package. For example;

```
namespace N {

class A { .. }

class A<T> { .. }

class A<T, U> { .. }

}
```

The user cannot create the model for these three classes in MagicDraw manually. They can only be created in C# reverse engineering.

Copyright © 1998-2015 No Magic, Inc.

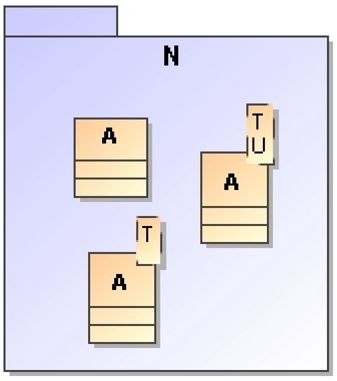


Figure 141 -- Generic Classes

How to manually create class A in the model, and Figure 12 shows the error message when trying to create class A.

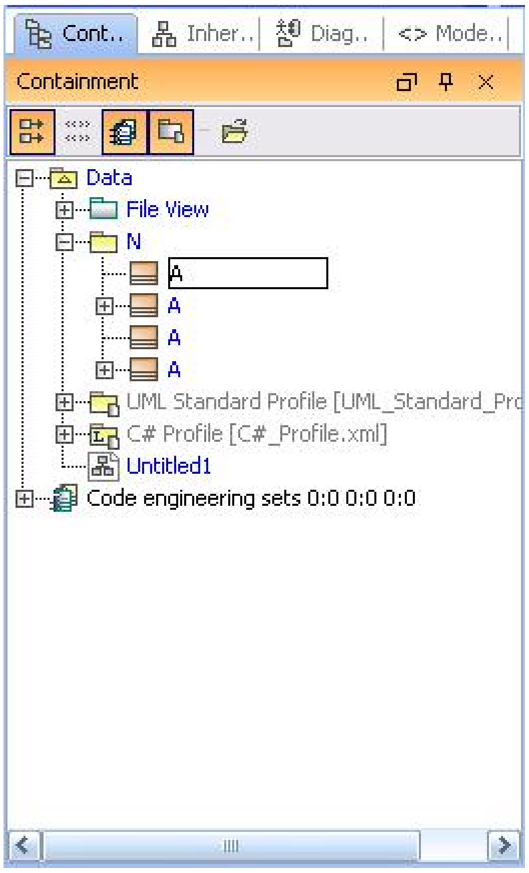
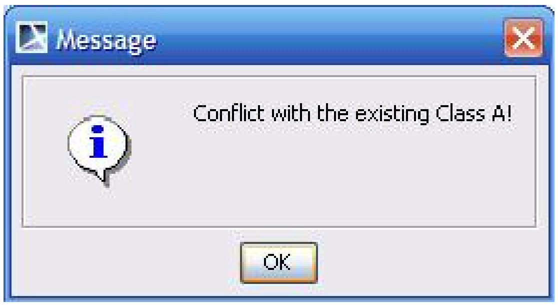

Cont..
Inher..
Diag..
<> Mode..
Containment
Data
File View
N
A
A
A
A
UML Standard Profile [UML_Standard_Pro
C# Profile [C#_Profile.xml]
Untitled1
Code engineering sets 0:0 0:0 0:0

Figure 142 -- Creating Class A in the Model

Copyright © 1998-2015 No Magic, Inc.



Conflict with the existing Class A!

Figure 143 -- Conflict Message Dialog

Code Generation for Partial Class

Generate the code without the round trip feature. Separated partial classes and all its child elements will be generated into only one class code and one class file as the example below.

Code used for reverse engineer	Code after generation

```
//Case #1
//The partial class is written into
one class file.

public partial class PartialA{
        int a;
        string actionA()
        {
        }
}
public partial class PartialA{
        int b;
        string actionB()
        {
        }
}
```
	public partial class PartialA{
	
	int a;
	
	int b;
	
	string actionA( )
	
	{return ;}
	
	string actionB( )
	
	{return ;}
	
	}

```
//Case #2
//The partial class is written into
separate class file.

//PartialA1.cs
public partial class PartialA{
        int a;
        string actionA()
        {
        }
}

//PartialA2.cs
public partial class PartialA{
        int b;
        string actionB()
        {
        }
}
```
	public partial class PartialA{
	
	int a;
	
	int b;
	
	string actionA( )
	
	{return ;}
	
	string actionB( )
	
	{return ;}
	
	}

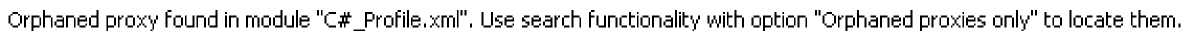
Copyright © 1998-2015 No Magic, Inc.

Code used for reverse engineer	Code after generation
`//Case #3`	public partial class PartialA{
`//The partial class with inner class`	
	int a;
`public partial class PartialA{`	
`        public class B`	string actionB( )
`        {`	
`                int b;`	{return ;}
`        }`	
`}`	public class B
`public partial class PartialA{`	
`        int a;`	{int b;}
`        string actionB()`	
`        {`	public class C
`        }`	
`        public class C`	{int c;}
`        {`	
`                int c;`	}
`        }`	
`}`	

Translation Constraints

Property Translation

As you may know that after version 12.1 MD C# will translate C#Property from operation to attribute. After the translation is finished, the message window will show the following message:



Orphaned proxy found in module "C#_Profile.xml". Use search functionality with option "Orphaned proxies only" to locate them.

This message occurred because there is no Operation as a metaclass of C#Property (changing of profile), but this will not affect the functionality of reverse engineering and translation.

After the project has been saved and reopened, the message will not be appear any more.

Copyright © 1998-2015 No Magic, Inc.

CORBA IDL MAPPING TO UML

NOTE: This functionality is available in Enterprise edition only.

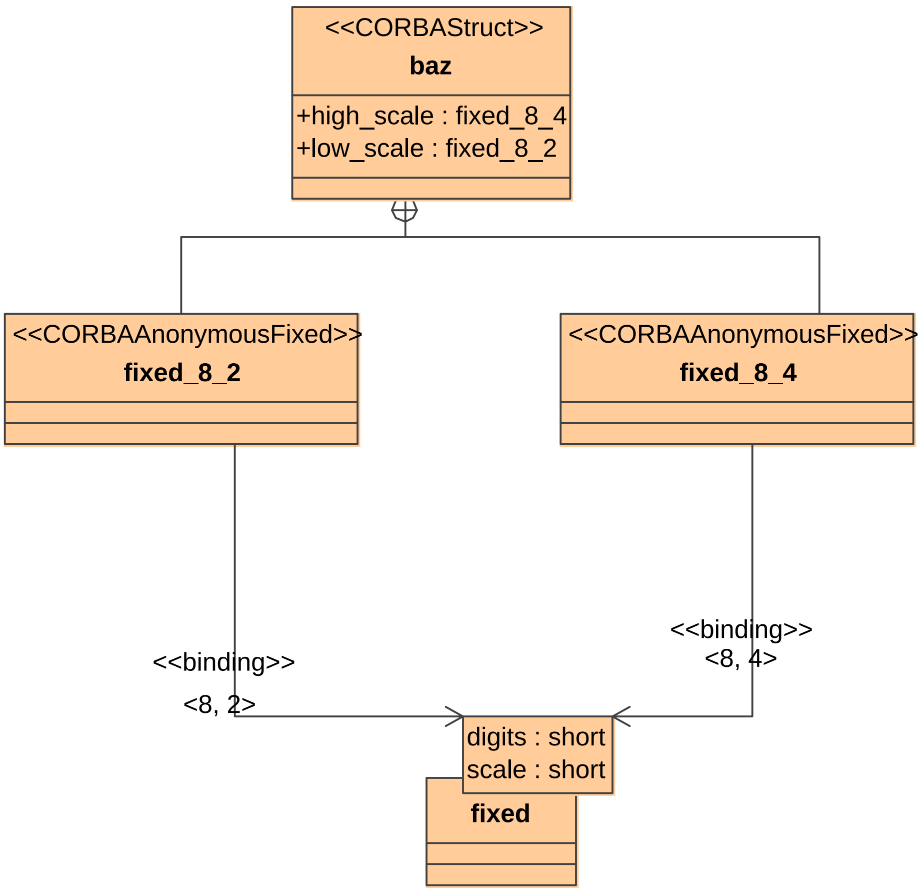
CORBA IDL mapping to UML is based on UML^{TM} Profile for CORBA^{TM} Specification, Version 1.0, April 2002. http://www.omg.org/technology/documents/profile_catalog.htm#UML_for_CORBA.

Differences between UML Profile for CORBA specification and mapping in MagicDraw is listed below.

- Constraints defined in UML Profile for CORBA specification are not checked in MagicDraw.
- Stereotype CORBAAnonymousFixed is introduced. It is used to represent anonymous fixed types. Fixed types without names are mapped to inner classes with stereotype CORBAAnonymousFixed. These classes are bound to CORBA::fixed classes.IDL Code:

```
struct baz
{
fixed <8, 4> high_scale;
fixed <8, 2> low_scale;
};
```

This code is mapped to the following diagram:



MagicDraw presents a CORBA IDL diagram, which simplifies the creation of standard CORB IDL elements. The loffowing elements are available in the CORBA IDL diagram:

Copyright © 1998-2015 No Magic, Inc.

Button	Shortcut key	Model Element
	M	**CORBA IDL Module**
	I	**CORBA IDL Interface**
	V	**CORBA IDL Value**
	SHIFT+P	**Class by Pattern**
	G	**Generalization**
		Truncatable Generalization
		Value Support Generalization
		CORBA IDL Association
	I	**Interface**

CORBA Interface Implementation

You can select either the UML Interface or the UML Class as a base element for CORBA Interface. The default element as a base element for CORBA Interfaces is as follows:

- UML Class. A class as a base element for CORBA Interfaces is in projects created on earlier than MagicDraw version 17.0.1. The CORBA Interfaces are modeled as classes in these projects.

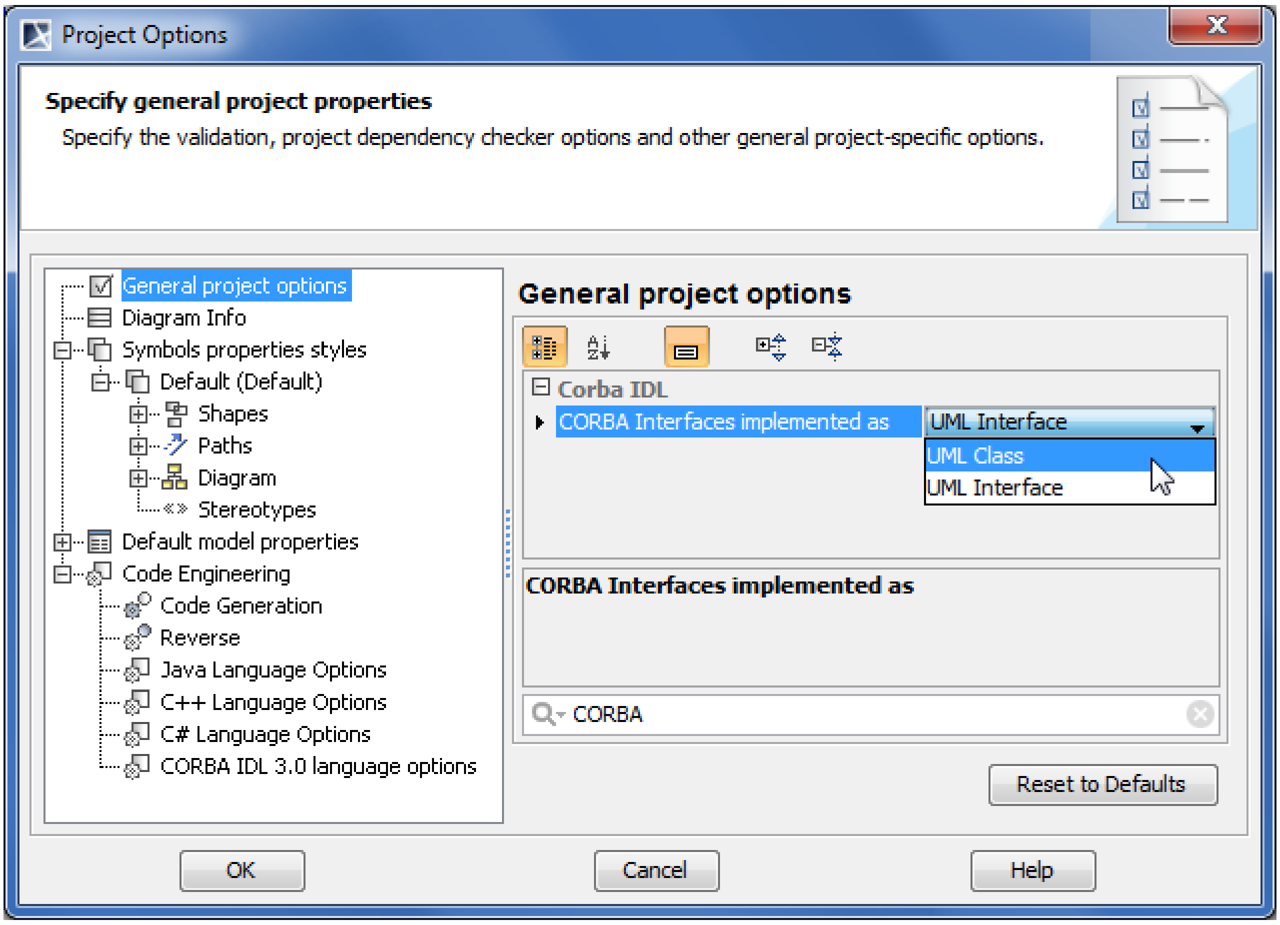
Copyright © 1998-2015 No Magic, Inc.

- UML Interface. An interface as a base element for CORBA Interfaces is in projecs created on MagicDraw version 17.0.1 and later. The CORBA Interfaces are modeled as interfaces in these projects.

IMPORTANT!	The generalization relationship can be used only between CORBA Interfaces based on the same element. That is, you can use the generalization relationship between CORBA Intarfaces based on a class or between CORBA Interfaces based on an interface but not between CORBA Interface based on a class and CORBA Interface based on an interface.

To set a base element for CORBA Interfaces

1. On the **Options** menu, click **Project**. The **Project Options** dialog opens.
2. In the Tab tree, select **General project options**.
3. In the **General project options** list, click the **CORBA Interfaces implemented as** value cell. The list of available values opens.
4. Do one of the following:
   - Select **UML Class**, to set a class as the base element for the CORBA Interface.
   - Select **UML Interface**, to set an interface as the base element for the CORBA Interface.
5. Click **OK** when you are done.



To change a base element for CORBA Interfaces

1. Select the CORBA Interface element in the Model Browser or its symbol on the diagram pane.

Copyright © 1998-2015 No Magic, Inc.

2. On the shortcut menu of the selected CORBA Interface, point to **Refactor** > **Convert To**, and then do the following:
   - Click **Class**, if an interface is the current base element for CORBA Interface.
   - Click **Interface**, if a class is the current base element for CORBA Interface.

IMMPORTANT!	Convert selected CORBA Interface only to the class or interface element. Conversion to other elements changes the CORBA Interface element to the selected element but not the element CORBA Interface is based on.

For more information about refactoring, see “Refactoring” in “MagicDraw UserManual.pdf”.

Copyright © 1998-2015 No Magic, Inc.

WSDL

NOTE: This functionality is available in Enterprise edition only.

Reference http://www.w3.org/TR/2001/NOTE-wsdl-20010315

WSDL is an XML format for describing network services as a set of endpoints operating on messages containing either document-oriented or procedure-oriented information. The operations and messages are described abstractly, and then bound to a concrete network protocol and message format to define an endpoint. Related concrete endpoints are combined into abstract endpoints (services).

MagicDraw UML supports WSDL code engineering: code generation, reverse and syntax checking. WSDL diagram is dedicated for WSDL modeling.

WSDL Profile and XML Schema Profile are used in WSDL code engineering.

WSDL Services are defined using six major elements:

- **types**, which provides data type definitions used to describe the messages exchanged.
- **message**, which represents an abstract definition of the data being transmitted. A message consists of logical parts, each of which is associated with a definition within some type system.
- **portType**, which is a set of abstract operations. Each operation refers to an input message and output messages.
- **binding**, which specifies concrete protocol and data format specifications for the operations and messages defined by a particular portType.
- **port**, which specifies an address for a binding, thus defining a single communication endpoint.
- **service**, which is used to aggregate a set of related ports.

Copyright © 1998-2015 No Magic, Inc.

WSDL Mapping to UML elements

Defined stereotypes

Element	Stereotype name	Applies to	Defined TagDefinitions	Details
Definition	**WSDLdefinitions**	Component	extension - string name - string targetNamespace - string	
Message	**WSDLmessage**	Class		
Port Type	**WSDLporrtype**	Interface		
Binding	**WSDLbinding**	Class	extension - string	
Port	**WSDLport**	Instance Specification	extension - string	
Service	**WSDLservice**	Component	extension - string	
Type	**WSDLtypes**	Component	extension - string	
	WSDLimport	ElementImport, PackageImport		
	xmlns	ElementImport, PackageImport		From the XML Schema Profile
	XSDnamespace	Package		From the XML Schema Profile
	WSDLresponse	Parameter	extension - string	
	WSDLoperation	Operation	extension - string	
	WSDLpart	Property	typing Attribute - string	
	WSDLfault	Parameter	extension - string	
	WSDLrequest	Parameter	extension - string	

Definitions

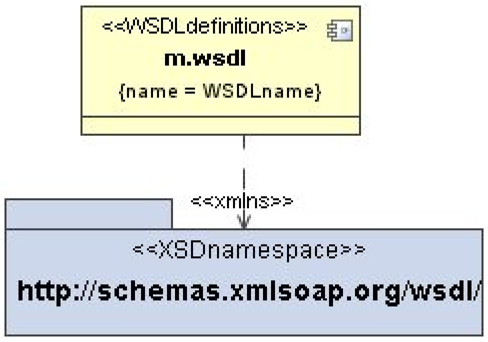
A WSDL document is simply a set of definitions. There is a definitions element at the root, and definitions inside.

Example:

```
<definitions name="WSDLname" xmlns="http://schemas.xmlsoap.org/wsdl/"/>
```

Copyright © 1998-2015 No Magic, Inc.

Reversed UML model example:

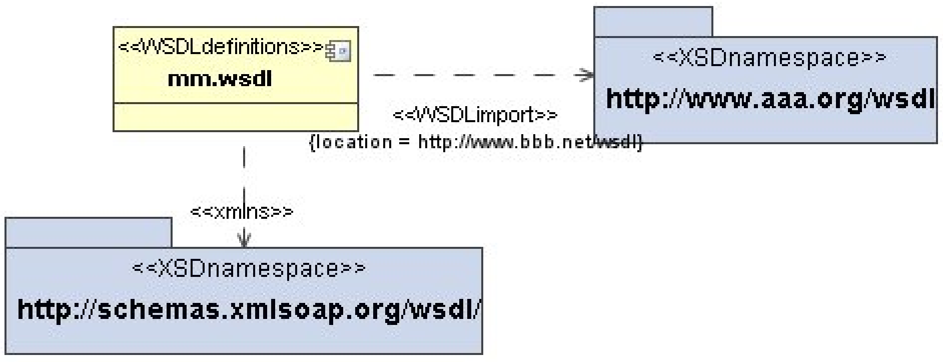


Import, namespace

Example:

```
<definitions xmlns="http://schemas.xmlsoap.org/wsdl/">
     <import location="http://www.bbb.net/wsdl" namespace="http://www.aaa.org/
wsdl"/>
</definitions>
```

Reversed UML model example:



Messages

Messages consist of one or more logical parts. Each part is associated with a type from some type system using a message-typing attribute. The set of message-typing attributes is extensible.

WSDL defines several such message-typing attributes for use with XSD:

- **element**. Refers to an XSD element using a QName.
- **type**. Refers to an XSD simpleType or complexType using a QName.

Other message-typing attributes may be defined as long as they use a namespace different from that of WSDL. Binding extensibility elements may also use message-typing attributes.

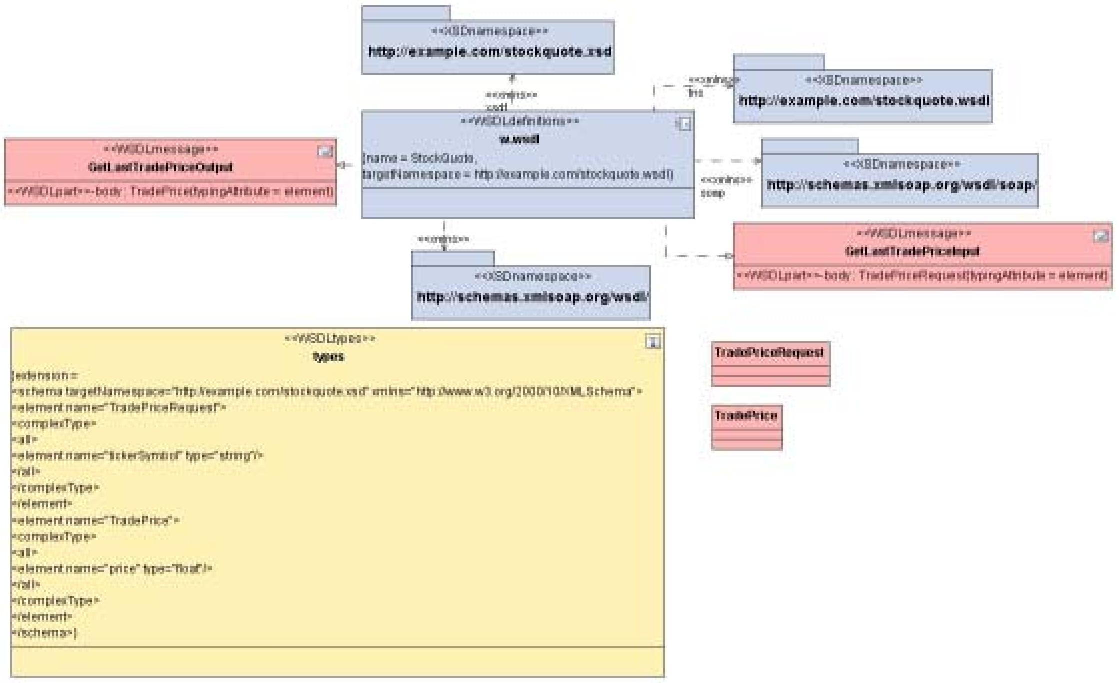
Example:

```
<definitions name="StockQuote"
targetNamespace="http://example.com/stockquote.wsdl"
           xmlns:tns="http://example.com/stockquote.wsdl"
           xmlns:xsd1="http://example.com/stockquote.xsd"
           xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
```

Copyright © 1998-2015 No Magic, Inc.

```
            xmlns="http://schemas.xmlsoap.org/wsdl/">

      <types>
         <schema targetNamespace="http://example.com/stockquote.xsd"
                xmlns="http://www.w3.org/2000/10/XMLSchema">
             <element name="TradePriceRequest">
                <complexType>
                    <all>
                        <element name="tickerSymbol" type="string"/>
                    </all>
                </complexType>
             </element>
             <element name="TradePrice">
                <complexType>
                    <all>
                        <element name="price" type="float"/>
                    </all>
                </complexType>
             </element>
         </schema>
      </types>

      <message name="GetLastTradePriceInput">
          <part name="body" element="xsd1:TradePriceRequest"/>
      </message>

      <message name="GetLastTradePriceOutput">
          <part name="body" element="xsd1:TradePrice"/>
      </message>
  </definitions>
```

Reversed UML model example:



<<XSDnamespace>>
http://example.com/stockquote.xsd
<<xmlns>>
xsd1
<<xmlns>>
tns
<<XSDnamespace>>
http://example.com/stockquote.wsdl
<<WSDLdefinitions>>
w.wsdl
{name = StockQuote,
targetNamespace = http://example.com/stockquote.wsdl}
<<WSDLmessage>>
GetLastTradePriceOutput
<<WSDLpart>>-body : TradePrice{typingAttribute = element}
<<xmlns>>
soap
<<XSDnamespace>>
http://schemas.xmlsoap.org/wsdl/soap/
<<xmlns>>
<<XSDnamespace>>
http://schemas.xmlsoap.org/wsdl/
<<WSDLmessage>>
GetLastTradePriceInput
<<WSDLpart>>-body : TradePriceRequest{typingAttribute = element}
<<WSDLtypes>>
types
TradePriceRequest
TradePrice

```
{extension =
<schema targetNamespace="http://example.com/stockquote.xsd" xmlns="http://www.w3.org/2000/10/XMLSchema">
<element name="TradePriceRequest">
<complexType>
<all>
<element name="tickerSymbol" type="string"/>
</all>
</complexType>
</element>
<element name="TradePrice">
<complexType>
<all>
<element name="price" type="float"/>
</all>
</complexType>
</element>
</schema>}
```

Copyright © 1998-2015 No Magic, Inc.

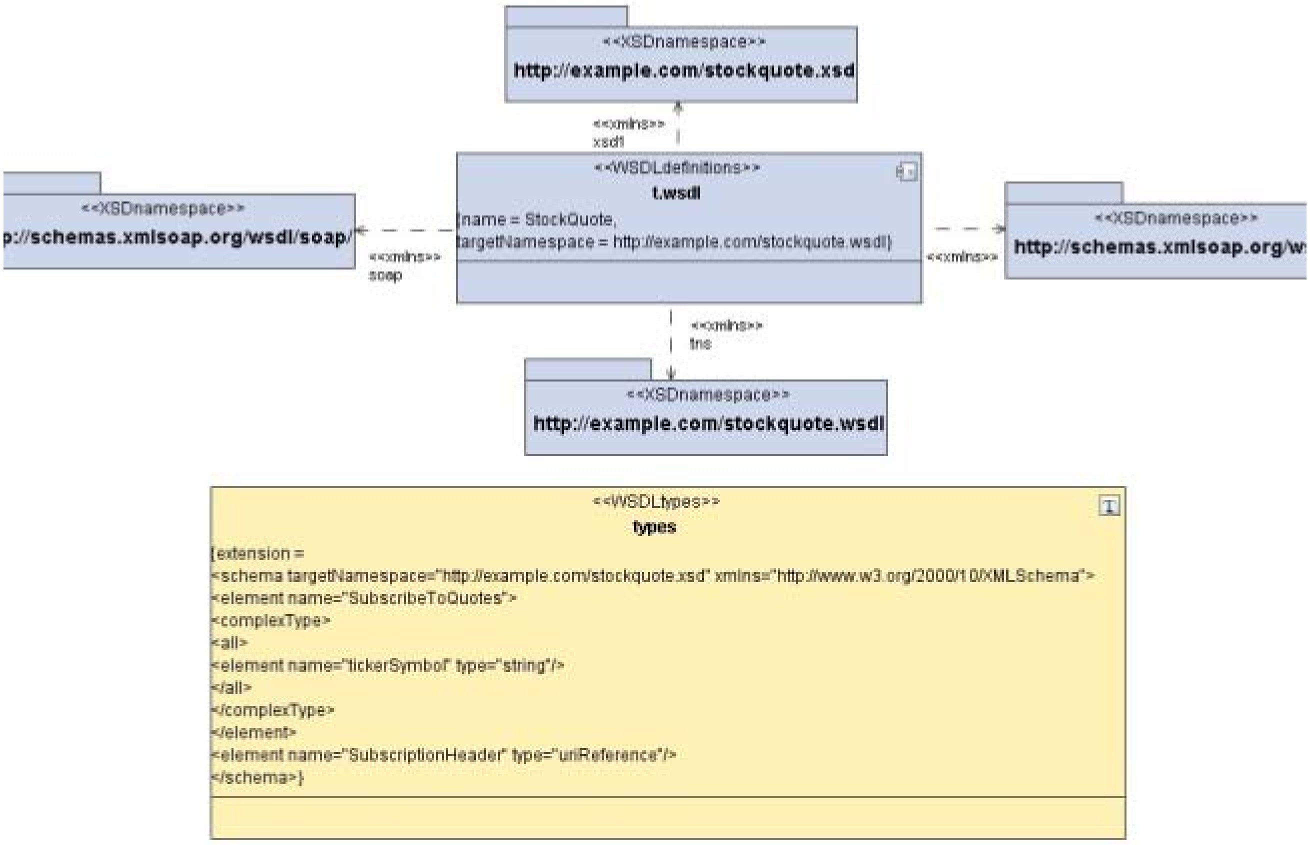
Types

The **types** element encloses data type definitions that are relevant for the exchanged messages. For maximum interoperability and platform neutrality, WSDL prefers the use of XSD as the canonical type system, and treats it as the intrinsic type system.

Example:

```
<definitions name="StockQuote"
          targetNamespace="http://example.com/stockquote.wsdl"
          xmlns:tns="http://example.com/stockquote.wsdl"
          xmlns:xsd1="http://example.com/stockquote.xsd"
          xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
          xmlns="http://schemas.xmlsoap.org/wsdl/">
<types>
        <schema targetNamespace="http://example.com/stockquote.xsd"
               xmlns="http://www.w3.org/2000/10/XMLSchema">
           <element name="SubscribeToQuotes">
               <complexType>
                   <all>
                       <element name="tickerSymbol" type="string"/>
                   </all>
               </complexType>
           </element>
           <element name="SubscriptionHeader" type="uriReference"/>
        </schema>
    </types>
</definitions>
```

Reversed UML model example:


«XSDnamespace»
http://example.com/stockquote.xsd
«xmlns»
xsd1
«XSDnamespace»
p://schemas.xmlsoap.org/wsdl/soap/
«xmlns»
soap
«WSDLdefinitions»
t.wsdl
{name = StockQuote,
targetNamespace = http://example.com/stockquote.wsdl}
«xmlns»
«XSDnamespace»
http://schemas.xmlsoap.org/w
«xmlns»
tns
«XSDnamespace»
http://example.com/stockquote.wsdl
«WSDLtypes»
types

```
{extension =
<schema targetNamespace="http://example.com/stockquote.xsd" xmlns="http://www.w3.org/2000/10/XMLSchema">
<element name="SubscribeToQuotes">
<complexType>
<all>
<element name="tickerSymbol" type="string"/>
</all>
</complexType>
</element>
<element name="SubscriptionHeader" type="uriReference"/>
</schema>}
```

Copyright © 1998-2015 No Magic, Inc.

Port types

A **port type** is a named set of abstract operations and the abstract messages involved.

The port type **name** attribute provides a unique name among all port types defined within in the enclosing WSDL document.

An operation is named via the **name** attribute.

WSDL has four transmission primitives that an endpoint can support:

- **One-way**. The endpoint receives a message.
- **Request-response**. The endpoint receives a message, and sends a correlated message.
- **Solicit-response**. The endpoint sends a message, and receives a correlated message.
- **Notification.** The endpoint sends a message.

WSDL refers to these primitives as **operations**. Although request/response or solicit/response can be modeled abstractly using two one-way messages, it is useful to model these as primitive operation types.

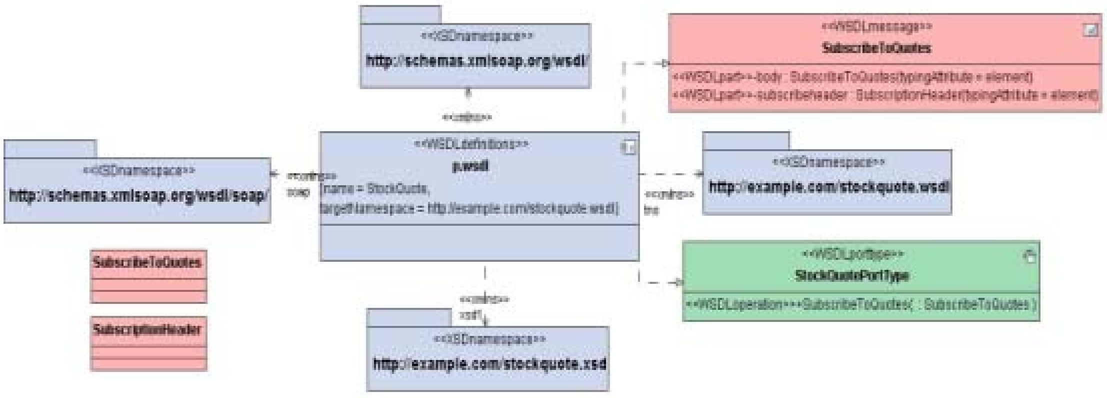
Example:

```
<definitions name="StockQuote"
          targetNamespace="http://example.com/stockquote.wsdl"
          xmlns:tns="http://example.com/stockquote.wsdl"
          xmlns:xsd1="http://example.com/stockquote.xsd"
          xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
          xmlns="http://schemas.xmlsoap.org/wsdl/">

    <message name="SubscribeToQuotes">
        <part name="body" element="xsd1:SubscribeToQuotes"/>
        <part name="subscribeheader" element="xsd1:SubscriptionHeader"/>
    </message>

    <portType name="StockQuotePortType">
        <operation name="SubscribeToQuotes">
           <input message="tns:SubscribeToQuotes"/>
        </operation>
    </portType>

    <binding name="StockQuoteSoap" type="tns:StockQuotePortType">
        <soap:binding style="document" transport="http://example.com/smtp"/>
        <operation name="SubscribeToQuotes">
           <input message="tns:SubscribeToQuotes">
               <soap:body parts="body" use="literal"/>
               <soap:header message="tns:SubscribeToQuotes" part="subscribeheader"
use="literal"/>
           </input>
        </operation>
    </binding>

    <service name="StockQuoteService">
        <port name="StockQuotePort" binding="tns:StockQuoteSoap">
           <soap:address location="mailto:subscribe@example.com"/>
        </port>
    </service>

    <types>
        <schema targetNamespace="http://example.com/stockquote.xsd"
               xmlns="http://www.w3.org/2000/10/XMLSchema">
           <element name="SubscribeToQuotes">
               <complexType>
```

Copyright © 1998-2015 No Magic, Inc.

```
                    <all>
                        <element name="tickerSymbol" type="string"/>
                    </all>
                </complexType>
            </element>
            <element name="SubscriptionHeader" type="uriReference"/>
         </schema>
     </types>
 </definitions>
```

Reversed UML model example:


<<XSDnamespace>>
http://schemas.xmlsoap.org/wsdl/
<<WSDLmessage>>
SubscribeToQuotes
<<WSDLpart>>-body : SubscribeToQuotes{typingAttribute = element}
<<WSDLpart>>-subscribeheader : SubscriptionHeader{typingAttribute = element}
<<XSDnamespace>>
http://schemas.xmlsoap.org/wsdl/soap/
<<imports>>
soap
<<WSDLdefinitions>>
p.wsdl
{name = StockQuote,
targetNamespace = http://example.com/stockquote.wsdl}
<<imports>>
tns
<<XSDnamespace>>
http://example.com/stockquote.wsdl
<<WSDLporttype>>
StockQuotePortType
<<WSDLoperation>>+SubscribeToQuotes(: SubscribeToQuotes)
SubscribeToQuotes
SubscriptionHeader
<<imports>>
xsd1
<<XSDnamespace>>
http://example.com/stockquote.xsd

Bindings

A **binding** defines message format and protocol details for operations and messages defined by a particular portType. There may be any number of bindings for a given portType.

The **name** attribute provides a unique name among all bindings defined within in the enclosing WSDL document.

A binding references the portType that it binds using the **type** attribute. Binding extensibility elements are used to specify the concrete grammar for the input, output, and fault messages. Per-operation binding information as well as per-binding information may also be specified.

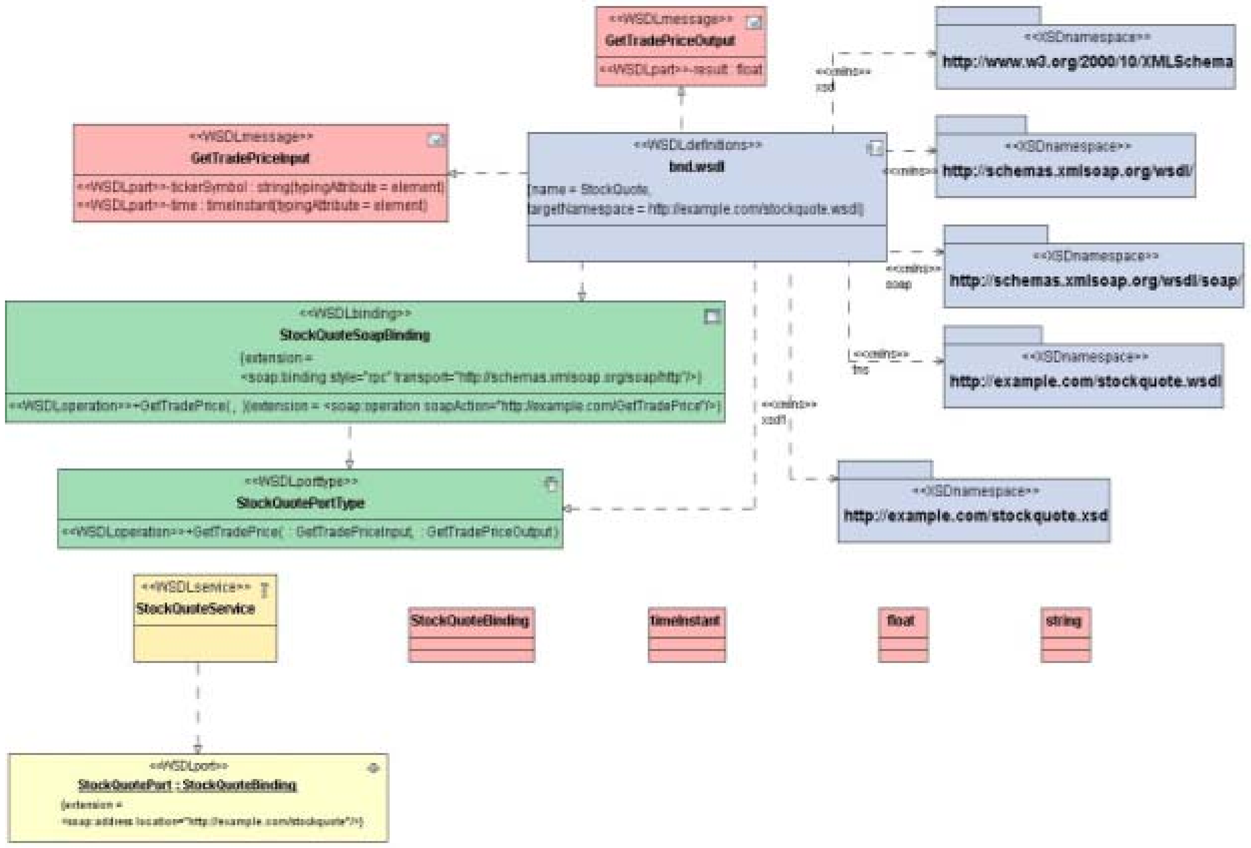
Example:

```
 <definitions name="StockQuote"
           targetNamespace="http://example.com/stockquote.wsdl"
           xmlns:tns="http://example.com/stockquote.wsdl"
           xmlns:xsd="http://www.w3.org/2000/10/XMLSchema"
           xmlns:xsd1="http://example.com/stockquote.xsd"
           xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
           xmlns="http://schemas.xmlsoap.org/wsdl/">

     <message name="GetTradePriceInput">
         <part name="tickerSymbol" element="xsd:string"/>
         <part name="time" element="xsd:timeInstant"/>
     </message>

     <message name="GetTradePriceOutput">
         <part name="result" type="xsd:float"/>
     </message>

     <portType name="StockQuotePortType">
         <operation name="GetTradePrice">
            <input message="tns:GetTradePriceInput"/>
```

Copyright © 1998-2015 No Magic, Inc.

```
            <output message="tns:GetTradePriceOutput"/>
         </operation>
    </portType>

    <binding name="StockQuoteSoapBinding" type="tns:StockQuotePortType">
        <soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/
>
        <operation name="GetTradePrice">
            <soap:operation soapAction="http://example.com/GetTradePrice"/>
            <input>
                <soap:body use="encoded" namespace="http://example.com/stockquote"
                         encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
            </input>
            <output>
                <soap:body use="encoded" namespace="http://example.com/stockquote"
                         encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
            </output>
        </operation>>
    </binding>

    <service name="StockQuoteService">
        <documentation>My first service</documentation>
        <port name="StockQuotePort" binding="tns:StockQuoteBinding">
            <soap:address location="http://example.com/stockquote"/>
        </port>
    </service>
</definitions>
```

Reversed UML model example:


«WSDLmessage»
GetTradePriceOutput
«WSDLpart»-result : float
«XSDnamespace»
http://www.w3.org/2000/10/XMLSchema
«xmlns»
xsd
«WSDLmessage»
GetTradePriceInput
«WSDLpart»-tickerSymbol : string{typingAttribute = element}
«WSDLpart»-time : timeInstant{typingAttribute = element}
«WSDLdefinitions»
bnd.wsdl
{name = StockQuote,
targetNamespace = http://example.com/stockquote.wsdl}
«xmlns»
«XSDnamespace»
http://schemas.xmlsoap.org/wsdl/
«xmlns»
soap
«XSDnamespace»
http://schemas.xmlsoap.org/wsdl/soap/
«xmlns»
tns
«XSDnamespace»
http://example.com/stockquote.wsdl
«WSDLbinding»
StockQuoteSoapBinding
{extension =
<soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>}
«WSDLoperation»+GetTradePrice( ){extension = <soap:operation soapAction="http://example.com/GetTradePrice"/>}
«xmlns»
xsd1
«WSDLporttype»
StockQuotePortType
«WSDLoperation»+GetTradePrice(: GetTradePriceInput, : GetTradePriceOutput)
«XSDnamespace»
http://example.com/stockquote.xsd
«WSDLservice»
StockQuoteService
StockQuoteBinding
timeInstant
float
string
«WSDLport»
StockQuotePort : StockQuoteBinding
{extension =
<soap:address location="http://example.com/stockquote"/>}

Copyright © 1998-2015 No Magic, Inc.

Services

A **service** groups a set of related ports together.

The **name** attribute provides a unique name among all services defined within in the enclosing WSDL document.

Ports within a service have the following relationship:

- None of the ports communicate with each other (e.g. the output of one port is not the input of another).
- If a service has several ports that share a port type, but employ different bindings or addresses, the ports are alternatives. Each port provides semantically equivalent behavior (within the transport and message format limitations imposed by each binding).
- By examining it's ports, we can determine a service's port types. This allows a consumer of a WSDL document to determine if it wishes to communicate to a particular service based whether or not it supports several port types.

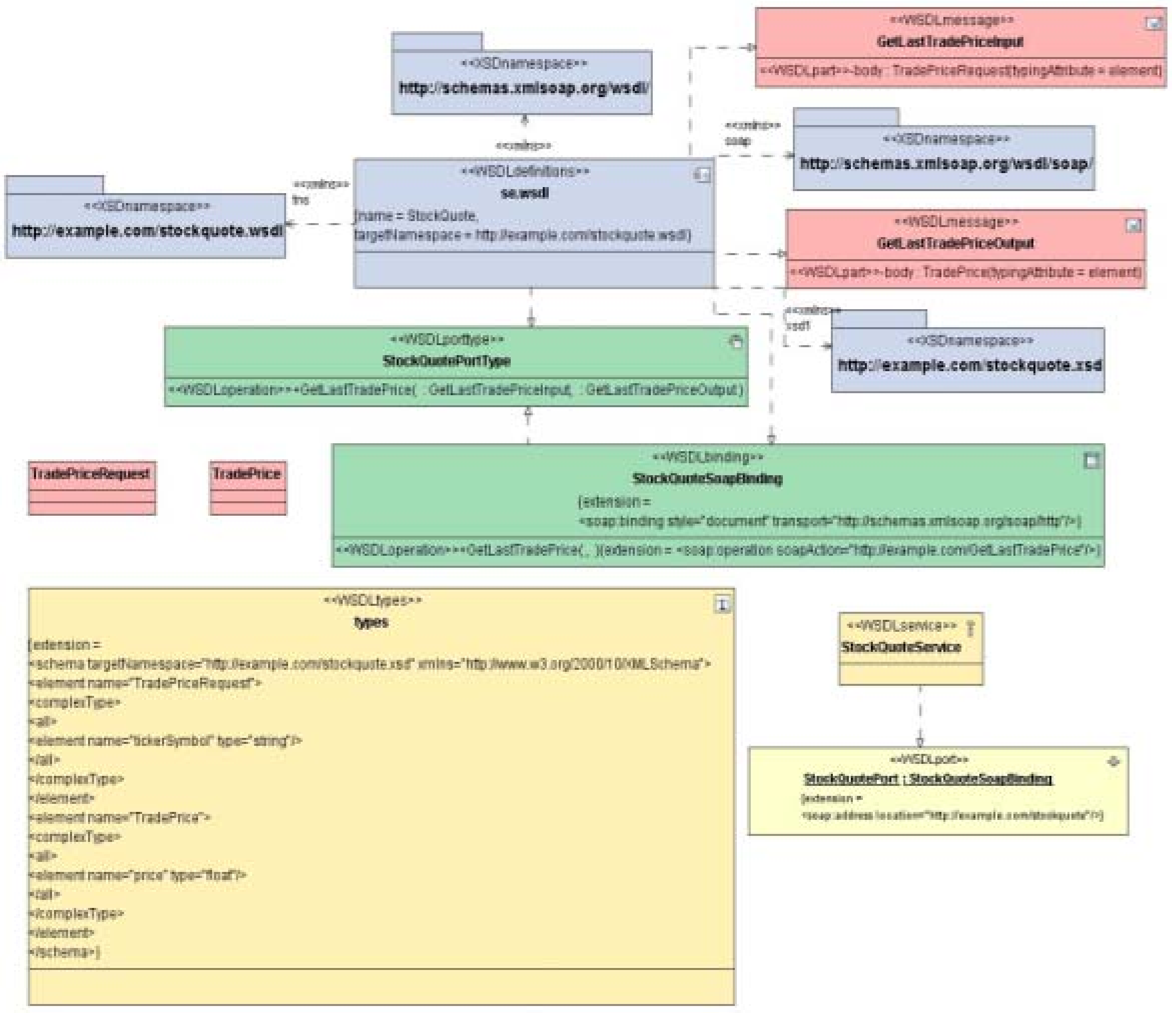
Example:

```
<definitions name="StockQuote"
             targetNamespace="http://example.com/stockquote.wsdl"
             xmlns:tns="http://example.com/stockquote.wsdl"
             xmlns:xsd1="http://example.com/stockquote.xsd"
             xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
             xmlns="http://schemas.xmlsoap.org/wsdl/">

  <types>
    <schema targetNamespace="http://example.com/stockquote.xsd"
           xmlns="http://www.w3.org/2000/10/XMLSchema">
      <element name="TradePriceRequest">
        <complexType>
          <all>
            <element name="tickerSymbol" type="string"/>
          </all>
        </complexType>
      </element>
      <element name="TradePrice">
         <complexType>
           <all>
             <element name="price" type="float"/>
           </all>
         </complexType>
      </element>
    </schema>
  </types>

  <message name="GetLastTradePriceInput">
    <part name="body" element="xsd1:TradePriceRequest"/>
  </message>

  <message name="GetLastTradePriceOutput">
    <part name="body" element="xsd1:TradePrice"/>
  </message>

  <portType name="StockQuotePortType">
    <operation name="GetLastTradePrice">
      <input message="tns:GetLastTradePriceInput"/>
      <output message="tns:GetLastTradePriceOutput"/>
    </operation>
  </portType>

  <binding name="StockQuoteSoapBinding" type="tns:StockQuotePortType">
```

Copyright © 1998-2015 No Magic, Inc.

```
    <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/
http"/>
    <operation name="GetLastTradePrice">
      <soap:operation soapAction="http://example.com/GetLastTradePrice"/>
      <input>
        <soap:body use="literal"/>
      </input>
      <output>
        <soap:body use="literal"/>
      </output>
    </operation>
  </binding>

  <service name="StockQuoteService">
    <documentation>My first service</documentation>
    <port name="StockQuotePort" binding="tns:StockQuoteSoapBinding">
      <soap:address location="http://example.com/stockquote"/>
    </port>
  </service>

</definitions>
```

Reversed UML model example:


«XSDnamespace» http://schemas.xmlsoap.org/wsdl/
«WSDLmessage» GetLastTradePriceInput
«WSDLpart»-body : TradePriceRequest{typingAttribute = element}
«xmlns»
«xmlns» soap
«XSDnamespace» http://schemas.xmlsoap.org/wsdl/soap/
«XSDnamespace» http://example.com/stockquote.wsdl
«xmlns» tns
«WSDLdefinitions» se.wsdl
{name = StockQuote, targetNamespace = http://example.com/stockquote.wsdl}
«WSDLmessage» GetLastTradePriceOutput
«WSDLpart»-body : TradePrice{typingAttribute = element}
«xmlns» xsd1
«WSDLporttype» StockQuotePortType
«WSDLoperation»+GetLastTradePrice(: GetLastTradePriceInput, : GetLastTradePriceOutput)
«XSDnamespace» http://example.com/stockquote.xsd
TradePriceRequest
TradePrice
«WSDLbinding» StockQuoteSoapBinding
{extension = <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>}
«WSDLoperation»+GetLastTradePrice(,){extension = <soap:operation soapAction="http://example.com/GetLastTradePrice"/>}
«WSDLtypes» types
«WSDLservice» StockQuoteService
«WSDLport» StockQuotePort : StockQuoteSoapBinding
{extension = <soap:address location="http://example.com/stockquote"/>}

```
{extension =
<schema targetNamespace="http://example.com/stockquote.xsd" xmlns="http://www.w3.org/2000/10/XMLSchema">
<element name="TradePriceRequest">
<complexType>
<all>
<element name="tickerSymbol" type="string"/>
</all>
</complexType>
</element>
<element name="TradePrice">
<complexType>
<all>
<element name="price" type="float"/>
</all>
</complexType>
</element>
</schema>}
```

Copyright © 1998-2015 No Magic, Inc.

Ports

A port defines an individual endpoint by specifying a single address for a binding.

The **name** attribute provides a unique name among all ports defined within in the enclosing WSDL document.

The **binding** attribute refers to the binding using the linking rules defined by WSDL.

Binding extensibility elements are used to specify the address information for the port.

A port must not specify more than one address.

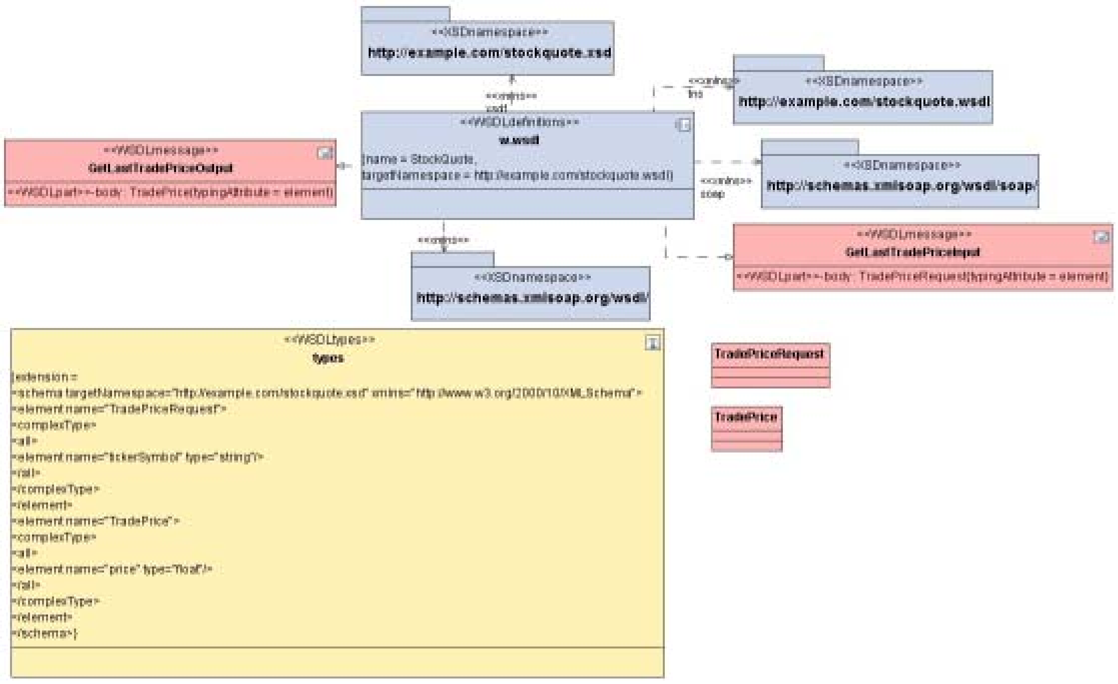
A port must not specify any binding information other than address information.

Example:

```
<definitions name="HelloService"
   targetNamespace="http://www.ecerami.com/wsdl/HelloService.wsdl"
   xmlns="http://schemas.xmlsoap.org/wsdl/"
   xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
   xmlns:tns="http://www.ecerami.com/wsdl/HelloService.wsdl"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema">

   <message name="SayHelloRequest">
      <part name="firstName" type="xsd:string"/>
   </message>
   <message name="SayHelloResponse">
      <part name="greeting" type="xsd:string"/>
   </message>

   <portType name="Hello_PortType">
      <operation name="sayHello">
         <input message="tns:SayHelloRequest"/>
         <output message="tns:SayHelloResponse"/>
      </operation>
   </portType>

   <binding name="Hello_Binding" type="tns:Hello_PortType">
      <soap:binding style="rpc"
         transport="http://schemas.xmlsoap.org/soap/http"/>
      <operation name="sayHello">
         <soap:operation soapAction="sayHello"/>
         <input>
            <soap:body
               encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
               namespace="urn:examples:helloservice"
               use="encoded"/>
         </input>
         <output>
            <soap:body
               encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
               namespace="urn:examples:helloservice"
               use="encoded"/>
         </output>
      </operation>
   </binding>

   <service name="Hello_Service">
      <documentation>WSDL File for HelloService</documentation>
      <port binding="tns:Hello_Binding" name="Hello_Port">
         <soap:address
            location="http://localhost:8080/soap/servlet/rpcrouter"/>
      </port>
   </service>
</definitions>
```

Copyright © 1998-2015 No Magic, Inc.

Reversed UML model example:


«XSDnamespace»
http://example.com/stockquote.xsd
«xmlns»
xsd1
«xmlns»
tns
«XSDnamespace»
http://example.com/stockquote.wsdl
«WSDLdefinitions»
w.wsdl
«WSDLmessage»
GetLastTradePriceOutput
«WSDLpart»-body : TradePrice{typingAttribute = element}
«xmlns»
soap
«XSDnamespace»
http://schemas.xmlsoap.org/wsdl/soap/
«WSDLmessage»
GetLastTradePriceInput
«WSDLpart»-body : TradePriceRequest{typingAttribute = element}
«xmlns»
«XSDnamespace»
http://schemas.xmlsoap.org/wsdl/
«WSDLtypes»
types
TradePriceRequest
TradePrice

```
{name = StockQuote,
targetNamespace = http://example.com/stockquote.wsdl}
```

```
{extension =
<schema targetNamespace="http://example.com/stockquote.xsd" xmlns="http://www.w3.org/2000/10/XMLSchema">
<element name="TradePriceRequest">
<complexType>
<all>
<element name="tickerSymbol" type="string"/>
</all>
</complexType>
</element>
<element name="TradePrice">
<complexType>
<all>
<element name="price" type="float"/>
</all>
</complexType>
</element>
</schema>}
```

Copyright © 1998-2015 No Magic, Inc.